

Models in Software Architecture Derivation and Evaluation: Challenges and Opportunities

Javier Gonzalez-Huerta, Emilio Insfran, and Silvia Abrahão^(✉)

ISSI Research Group, Universitat Politècnica de València,
Camino de Vera, s/n, 46022 Valencia, Spain
{jagonzalez, einsfran, sabrahao}@dsic.upv.es

Abstract. Software architecture derivation and evaluation are complex and error prone activities that still represent an open problem with many challenges and opportunities where model-driven software development can play a leading role. In software product line development, the use of model-driven principles could help by providing a richer semantic representation of a product line and by capturing the architectural design decisions and its impact on the product quality attributes. In this chapter, we analyze the main challenges and opportunities surrounding the product architecture derivation and evaluation and introduce QuaDAI, a method for the derivation, evaluation, and improvement of product architectures in model-driven software product line development environments. The method comprises a multimodel, which represents the different viewpoints of a software product line, and a process conducted by model transformations that automate the derivation, evaluation, and improvement of product architectures.

Keywords: Software architectures · Software product lines · Model-driven development · Quality assurance

1 Introduction

Software architecture derivation and evaluation in Software Product Line (SPL) development environments is a complex and error-prone process [1] that still represents an open problem with many challenges and opportunities where Model-Driven Software Development (MDSD) can play a leading role. MDSD advocates the use of models not only to document the software development lifecycle but also to obtain the final product as a result of a model transformation chain. MDSD has been traditionally applied in the development of SPLs, especially for solving the configuration and product architecture derivation problem. A SPL is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [2]. SPLs emerged as a promising approach to improve software development processes so as to reduce costs and enhance productivity and product quality.

Quality assurance is a crucial activity for the success of any software development effort, but is even more important in SPL development since a defect in a core asset may impact negatively on the quality of the whole set of products within the product line. This fact is especially relevant when dealing with the software architecture. Software architecture is a key asset in SPL development and plays a dual role: on the

one hand, the product line architecture (PLA) should provide variation mechanisms that help to achieve a set of explicitly allowed variations and, on the other hand, the product architecture (PA) is derived from the PLA by exercising its built-in architectural variation points [2]. Software architectures are the means for the attainment of the non-functional requirements (NFRs) of the products that will be derived from the product line, and thus assuring the achievement of those NFRs during the architecture derivation process is a critical activity in the development process.

In the last few years, MDSD has been applied in several research works to face the product architecture derivation problem in SPL environments (e.g., [3–17]) although the majority of these approaches do not properly integrate NFRs in the derivation process. It is surprising that being quality one of the main reasons for the adoption of the SPL approach, it has been often neglected in such a critical and complex process [18]. In addition, in those cases in which the derived PA is evaluated after its derivation, this evaluation is carried out by using software architecture evaluation methods that have not been specially defined for SPLs (e.g., ATAM [19], SAAM [20]). We believe that the use of model-driven principles could help by providing a richer semantic representation of a software product line and by capturing the architectural design decisions and its impact on the product quality attributes.

In this chapter, we first discuss the challenges identified in the area of architecture derivation and evaluation in Model-Driven Software Product Line Engineering (MD-SPLE), and then introduce QuaDAI [21, 22], an integrated method for the derivation, evaluation, and improvement of product architectures in MD-SPLE environments. The method comprises a multimodel, which represents the different viewpoints of the software product line, and a process conducted by model transformations that automate the derivation, evaluation, and improvement of product architectures.

The remainder of the chapter is structured as follows. Section 2 discusses existing approaches that deal with the derivation and evaluation of architectures in SPL development. Section 3 introduces QuaDAI, a method to support the derivation, evaluation and improvement of product architectures in MD-SPLE environments. Finally, Sect. 4 provides our conclusions and final remarks.

2 Existing Approaches for Architecture Derivation and Evaluation in MD-SPLE

In this section, we analyze the approaches that support the derivation and quality evaluation of product architectures in MD-SPLE. Section 2.1 analyzes existing approaches for product architecture derivation in MD-SPLE. Section 2.2 analyzes existing software architecture evaluation methods that allow the quality evaluation and analysis of product architectures in SPL development. Finally, Sect. 2.3 summarizes the main findings.

2.1 Product Architecture Derivation in MD-SPLE

Despite the huge number of research work dealing with architecture derivation in SPL development, the introduction of quality concerns in this process has not received a

proper coverage. In Table 2, we show a summary of the classification of these approaches by applying an extension of the classification criteria defined by Rabiser et al. [1] (see Table 1).

Table 1. Architecture derivation classification criteria.

Criteria	Description
C1*	Non-functional requirements (NFRs) support
C2*	Explicit representation of NFRs/quality attributes and their relationships with the features (SPL external variability) or the architectural variants
C3**	Configuration support
C4**	Automated derivation support
C5***	Adaptability and extensibility (i.e., metamodel support, extension points for the integration of domain specific generators)
C6***	Flexible and user-specific visualizations of variability (filtering, classification and ordering support based on tasks, users, roles etc.)
C7	Explicit representation of architectural variability
C8	Architectural views support
C9	ADL/Modeling language support
C10	Configuration consistency checking

*C1 and C2: Adapted from the “Application requirements management support” criterion described in [1]

**C3 and C4: Adapted from the “Automated and interactive variability resolution” criterion described in [1]

***C5 and C6: Proposed at the systematic review by Rabiser et al. [1]

Given a set of architectural variation points for a product line architecture one of the main challenges is how to decide which variation points should be selected and which ones should not taking into account not only functional but also non-functional requirements. For these reason, we have suggested the criteria on Table 1.

The analysis show that there is a lack of approaches that: (i) can be applied regardless the architectural description language or architectural viewpoint; (ii) allow the explicitly representation of the architectural variability; (iii) allow the explicit representation of the NFRs as well as the relationships among the NFRs and the features that represent the SPL external variability but also the architectural variants that realize this external variability; (iv) allow to configure the product by considering both the features and the NFRs that the product must satisfy; (v) solve the architectural variability automatically by using model transformations.

2.2 Architecture Evaluation in MD-SPLE

Over the last years, several approaches that allow the quality evaluation and assessment of SPL product architectures have been proposed (e.g., [19, 23, 25–37]). In Table 4, we show the summary of the classification of these approaches by applying the classification criteria shown in Table 3.

Table 2. Classification of architecture derivation approaches.

Approach	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Koalish [4]	–	–	+	–	+	–	+	C&C	Own	–
Cabello et al. [6]	–	–	+	+	+	–	+	+	Own	+
Botterweck et al. [5]	–	–	+	+	+	–	+(FM/C)	C&C	+	+(FM/C)
Duran-Limon et al. [8]	–	–	+	+	–	–	+(OWL and FM)	C&C	+	+(FM)
Guana and Correal [23]	+	–	+	+	+	–	+	C&C	+	–
Czarnecki and Antkiewicz [9]	–	–	+	+	+	–	+	+	+	+(FM)
Ziadi and Jézéquel [10]	+	–	+	+	+	–	Model	+	UML	+
PLUS-EE [11]	–	–	+	+	–	–	Model	Multiple viewpoints	UML	+
Perrouin et al. [12]	–	–	+	+	+	–	+	–	UML	+
Schaefer et al. [13]	–	–	+	+	+	–	+	CoBoxes	CoBoxes	–
Tawhid and Petriu [14]	–	–	–	+	–	–	Model	Structure	Marte	–
Sánchez et al. [15]	–	–	–	+	+	–	+	+	+	–
FeatureMapper [16]	–	–	+	+	+	–	+(FM and Model)	+	+	+
Haugen et al. [24]	–	–	+	+	+	–	+	+	+	+

Legend: FM: Feature Model; C&C: Component and Connector; FM/C: Feature Model and Component Model; +: Supported; –: Not Supported; ± Partially Supported

Table 3. Architecture evaluation classification criteria.

Criteria	Description
C1	Defined for evaluating product architectures (PA)
C2	Multi-attribute support
C3	Objective evaluation (e.g., metric-based evaluation)/Subjective evaluation (e.g., scenario-based evaluation)
C4	Multi-architectural viewpoint/Multi-architectural description language
C5	Derivation-time evaluation

An analysis of these approaches reveals that the majority of them have not been proposed specifically for SPL development and only few of them provide coverage to the evaluation of product architecture at derivation-time (e.g., [23, 33–37]). The majority

Table 4. Classification of architecture evaluation approaches.

Method	C1	C2	C3	C4	C5
ATAM [19]	–	+	–	+	–
FAAM [25]	–	+	–	+	–
D-SAAM [26]	–	+	–	+	–
ALMA [27]	–	– (Modifiability)	–	+	–
AQA [28]	–	+	–	+	–
Alves et al. [29]	–	+	+	+	–
Gannod and Lutz [30]	–	+	–	+	–
Maccari [31]	–	– (Evolution)	–	+	–
Riva and Rosso [32]	–	– (Evolution)	+	+	–
Tawhid and Petriu [33]	+	– (Performance)	+	–	+
Alonso et al. [34]	+	– (Performance)	+	+	+
Guana and Correal [20]	+	+	–	+	+
E-ATAM [35]	+ (PLA/PA)	+	–	+	+
HoPLAA [36]	+ (PLA/PA)	+	–	+	+
CaLiPro [37]	+ (PLA/PA)	+	–	+	+

only provide scenario-based subjective evaluation (e.g., [19, 23, 25–28, 30, 31, 35–37]) and only few of them provide software metrics that allow to perform an objective evaluation of the derived product architecture through measurement processes (e.g., [29, 32–34]). However, these approaches only cover performance metrics (e.g., [32–34]) or do not cover the evaluation of product architectures (e.g., [29]). None of the approaches allow the explicit representation of design decisions and their impact on the product quality attributes.

The main finding of this analysis is that the architecture evaluation in MD-SPLE is not sufficiently covered by methods that allow the evaluation of product architectures regardless the set of quality attributes or the nature of the architecture being evaluated. In addition, we observed a lack of metric-based product architecture evaluation methods that can be applied at derivation time. In SPLE development, variability in quality attribute levels is also possible and thus the application of metric-based evaluation methods at derivation-time will allow us to analyze whether the measured values for a specific configuration are within the limits established for the product line or not. The evaluation of quality attributes after the derivation (or during derivation time) allows us the early detection of potential problems, reducing costs and enhancing productivity and product quality.

2.3 Discussion

The main finding of the analysis of existing works in the field is that there is lack of methods that support the derivation, evaluation and improvement of product architectures in an integrated manner, by means of evaluation mechanisms that allow us to ensure the fulfillment of the desired quality attribute levels at derivation and evaluation time.

A lot of effort have been spent in obtaining optimal solutions for the configuration problem [38–40], but these efforts are meaningless if the product obtained after the derivation do not fulfill the quality attribute levels that it is supposed to have. Furthermore, in many occasions the evaluations do not take into account the unpredictability of certain quality attributes [41] which makes that certain properties could not be modeled as the sum of the properties of their parts. This introduces a degree of uncertainty that can only be solved through the measurement of the actual values of these properties once the software artifacts have been obtained. Finally, the majority of the approaches have been tailored for a specific modeling language or architectural description language, or for a specific architectural viewpoint.

All the problems described above is what has motivated us to define QuaDAI as an integrated product architecture derivation, evaluation and improvement method that is applicable regardless the quality attributes to be evaluated or the architectural description languages used to specify the architecture or the architectural viewpoints of interest. We have also faced the problem of empirically validate the usefulness of the method through a family of experiments reported in [22].

3 A Multimodel Approach for the Derivation, Evaluation and Improvement of Product Architectures

QuaDAI is a generic, integrated method for the derivation, evaluation and improvement of product architectures regardless the architectural description language in which they are expressed or the domain. It is based in a multimodel [42]) that represents the different SPL viewpoints and a process consisting of a set of activities conducted by model transformations.

The approach is supported by a prototype¹ that gives support to the configuration, consistency checking and generation of the product architecture. The prototype allows to import feature models and specifications defined using third party tools and to establish the relationships among them so as to automate the product architecture derivation.

The rest of the section is structured as follows: Sect. 3.1 introduces the example we use to illustrate the method; Sect. 3.2 presents the multimodel for representing SPLs; Sect. 3.3 introduces the main activities of the QuaDAI process; Sect. 3.4 describes the details of the product architecture derivation; and finally, Sect. 3.5 describes the product architecture derivation and improvement activities.

3.1 An Illustrative Example

The different activities of the approach are illustrated through the use of a running example: a SPL from the automotive domain that comprises the safety critical embedded software systems responsible for controlling a car. This SPL comprises

¹ The prototype is available for download at: <http://users.dsic.upv.es/~jagonzalez/CarCarSPL/index.html>.

several features such as Antilock Braking System, Traction Control System, Stability Control System or Cruise Control System².

The Cruise Control System feature incorporates variability. This variability is resolved depending on other selections made on a feature model (i.e., the selection of the cruise control together with the park assistant implies the positive resolution of an extended version of the cruise control). Figure 1 shows an excerpt of the feature model that represents the SPL external variability.

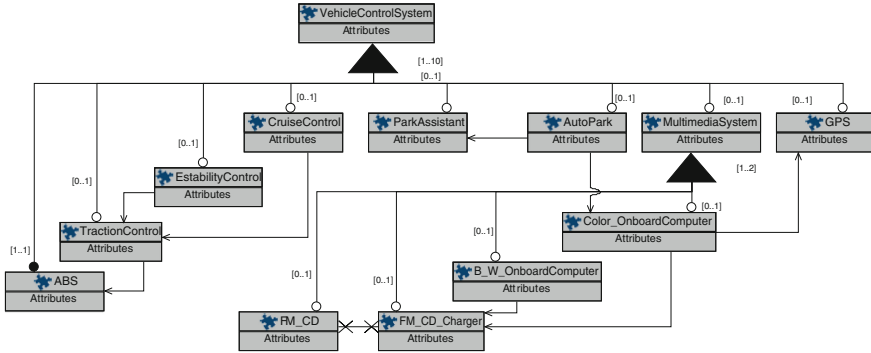


Fig. 1. Feature model representing the SPL external variability.

3.2 A Multimodel for Specifying SPLs

Traditional product line development process are based on: (1) the reuse of software assets (e.g., components, web services) that have been previously developed and stored; and (2) the realization of a production plan addressed to a product family which share a common functionality (product line architecture) but that vary in some features (variants). This approach can only be realized by assuming that we have a limited amount of variants, perfectly defined, and by assuming that these variants can be captured as instances of a feature model. However this is not realistic: variants go beyond the monotonic addition/removal of functionality grains from the product line architecture. For instance, changes in the structure or behavior of the application that is being produced can impact its quality, even for the same functionality, thus making the product unfeasible. Moreover, different properties of the application domain, design decisions, usability and user requirements, etc. are difficult to capture by means of only one feature model. This leads to the fact that only one feature model is not sufficient to define a software product line, but different views are needed.

Our approach is based on the existence of several models or system views (e.g., functionality, features, quality) with relationships among them. This approach implies the parameterization of the software production process by means of a multimodel

² The whole specification of the example is available at <http://users.dsic.upv.es/~jagonzalez/CarCarSPL/links.html>.

which is able to explicitly represent the different views of the products within the product line and the relationships among them.

A multimodel can be defined as a set of interrelated models that represent the different viewpoints of a particular system. A viewpoint is an abstraction that yields the specification of the whole system restricted to a particular set of concerns, and it is created with a specific purpose in mind. In any given viewpoint it is possible to produce a model of the system that contains only the objects that are visible from that viewpoint [43]. Such a model is known as a viewpoint model, or view of the system from that viewpoint. The multimodel permits the definition of relationships among model elements in those viewpoints, capturing the missing information that the separation of concerns could lead to [42].

The multimodel used to specify SPLs in order to support the derivation, evaluation and improvement of product architectures is composed of (at least) four interrelated viewpoints:

- **Variability Viewpoint**, which represents the SPL external variability expressing the commonalities and variability within the product line. Its main element is the feature, which is a user-visible aspect or characteristic of a system [44]. It is expressed in the multimodel by means of a variant [45] of the cardinality-based feature model (see Fig. 1).
- **Architectural Viewpoint**, which represents the architectural variability of the product line architecture that realizes the external variability of the SPL expressed in the variability viewpoint. It is expressed in the multimodel by means of the Common Variability Language (CVL) and its main element is the Variability Specification (VSpec). We only represent in the multimodel the architectural variability of the product line architecture. The PLA itself is represented in a base model, which is referenced by the CVL specification. A base model, under the CVL terminology, is a model on which variability is defined using CVL [46]. The base model is not part of CVL and can be an instance of any metamodel defined via MOF [46].
- **Quality Viewpoint**, which represents the hierarchical decomposition of quality into sub-characteristics, quality attributes, metrics and the impacts and constraints among quality attributes. It is expressed in the multimodel by means of a quality model for software product lines [47], which extends the ISO/IEC 25010 (SQuaRE) [48] and allows the definition of NFRs as constraints affecting characteristics, sub-characteristics and quality attributes.
- **Transformation Viewpoint** [49] that contains the explicit representation of the design decisions realized by the different model transformation processes that integrate the production plan for the model-driven development of SPLs. Alternatives may appear in a model transformation process when a set of constructs in the source model admits different representations in the target model. The application of each alternative transformation could generate alternative target models that may have the same functionality but might differ in their quality attributes. In this work, we focus on architectural patterns [50, 51]. Architectural patterns specify solutions to recurrent problems that occur in specific contexts [52]. They also specify how the system will deal with one aspect of its functionality, impacting directly on the product quality

attributes. Architectural patterns can be represented as architectural transformations, as a means to ensure the quality attributes attained by the product architectures. Figure 2 shows an excerpt of the transformation viewpoint of the multimodel, containing one design decision in which we have three alternative architectural patterns that can be applied by means of their own transformation rules.

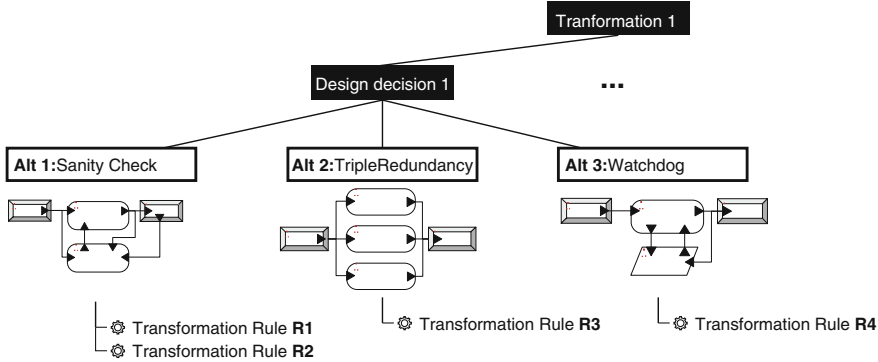


Fig. 2. Transformation viewpoint excerpt.

The multimodel also represents the relationships among elements of each viewpoint with different semantics as *is_realized_by* [53] or *impact* relationships [42]. An excerpt of these relationships is shown in Fig. 3. Through these relationships we can describe in the multimodel:

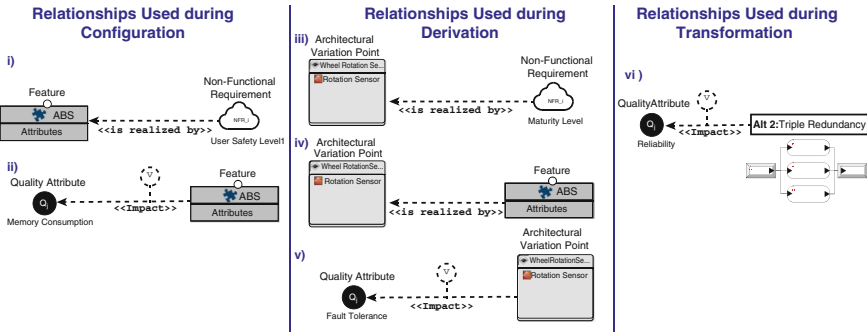


Fig. 3. Multimodel relationships.

- (i) How the UserSafetyLevel1 NFR *is_realized_by* a set of features (e.g., the ABS or the Stability Control).
- (ii) How the selection of a given feature *impacts* positive or negatively on a quality attribute.

- (iii) How the MaturityLevel NFR *is_realized_by* a set of VSpecs (e.g., the WheelRotationSensor).
- (iv) How the ABS feature *is_realized_by* a set of VSpecs (e.g., the WheelRotationSensor).
- (v) How the positive resolution of a given VSpec (e.g. WheelRotationSensor) *impacts* positive or negatively on a quality attribute (e.g., FaultTolerance).
- (vi) How the selection of a given architectural transformation *impacts* positive or negatively on a quality attribute.

These relationships are used to check the consistency of the product configuration in order to decide which variation points should be resolved positively in the CVL resolution model driving the product architecture derivation. The relationships are also used to select and apply the architectural transformations that best fit the prioritized quality attributes driving the transformation activity. All these activities are further described in the following subsections.

3.3 Overview of the QuaDAI Process

The process consists of a set of activities conducted by model transformations that take as input the multimodel viewpoints and the relationships defined among their elements. Figure 4 shows a summary of this process that comprises four main activities:

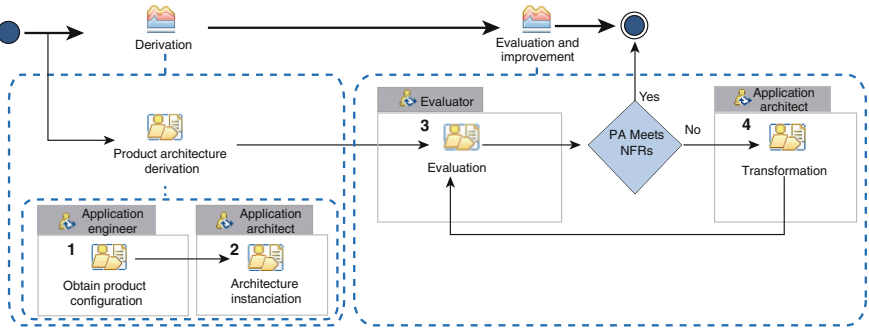


Fig. 4. QuaDAI main activities.

- **Obtain Product Configuration** in which the application engineer defines the configuration of the product under development by selecting the features, NFRs and by establishing the priority of each quality attribute³.

³ NFRs are defined in the quality viewpoint as constraints affecting the quality attributes whereas the prioritization of the quality attributes allows expressing the relative importance of each quality attribute.

- **Product Architecture Instantiation** in which the application architect obtains the first version of the product architecture based on the product configuration by resolving the architectural variability of the PLA.
- **Evaluation** in which the evaluator measures the derived product architecture in order to assess the degree of fulfillment of the NFRs.
- **Transformation** in which the architect applies architectural transformations so as to improve certain quality attributes when the architectural variability is not sufficient to achieve the required NFRs for the product.

3.4 Product Architecture Derivation

The derivation process for obtaining a first version of the product architecture comprises two main activities: the *Product configuration* and the *Architecture instantiation*. Figure 5 shows an excerpt of this process with its main inputs and outputs. In the *product configuration* activity, the application engineer configures the product by selecting the features and the NFRs that the product must fulfill and establishes the quality attributes priorities in the *Obtain product configuration* task. These quality attributes priorities will be used during the derivation phase to choose from a set of architectural variants that having the same functionality differ in their quality attribute levels, and in the evaluation and improvement phases to select the architectural transformations to be applied to the architecture.

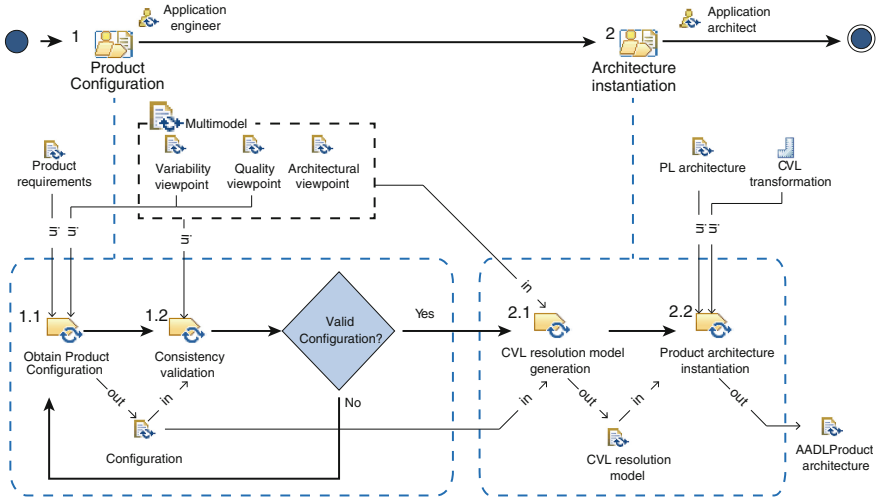


Fig. 5. Excerpt of the derivation process.

Once the product has been configured, we can check the product consistency (*Consistency validation* task). Our approach supports the intra (e.g., consistence of the feature model, consistence of the quality model) and inter-model consistency (e.g., relationships between the feature and the quality models).

Figure 6 shows the flow of steps to obtain a valid product configuration using QuaDAI. In the *Variability viewpoint validation*, we check whether the selection of features fulfills the constraints defined in the feature model. In the *Quality viewpoint validation* we check whether the priorities of the quality attributes defined in the configuration satisfy the *impact* relationships and constraints among them defined in the quality viewpoint. In the *Features-NFRs validation* we check whether the configuration satisfies the *is_realized_by* relationships defined among features and NFRs defined in the multimodel. Finally, in the *Features-attributes validation* we check whether the features selected and the prioritized quality attributes do not violate the *impact* relationships among features and quality attributes defined in the multimodel. The variability viewpoint consistency validation has been operationalized by using the FAMA [54] validator. We transform the cardinality-based feature model into the FAMA metamodel through a QVT model transformation and we project the selection of features by using a model to text transformation. The quality viewpoint and the inter-viewpoint consistency checking are carried out through OCL constraints checked at runtime by the OCLTools validator [55].

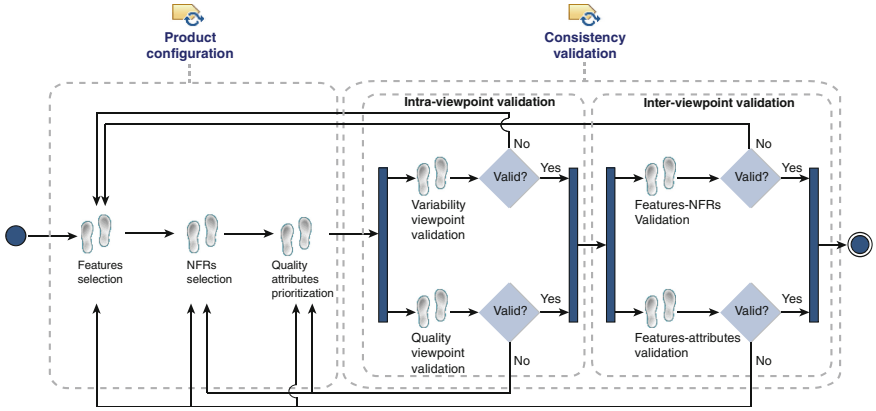


Fig. 6. Configuration and consistency checking process.

In the *architecture instantiation* activity, the application architect generates the product architecture by means of two model transformation activities. The first transformation, *CVL resolution model generation* task, takes as input a valid product configuration and the multimodel (i.e., the relationships between the architectural viewpoint with the variability and the quality viewpoints) and, through a QVT model transformation, generates a CVL resolution model. With the multimodel relationships, the QVT transformation decides which architectural variants have to be positively resolved in each variation point.

Finally, the *product architecture instantiation* task, through a CVL transformation, takes as input the CVL resolution model and generates the product architecture. This product architecture represents the resolution of the PLA architectural variability taking

into account not only the functional requirements but also the NFRs and the quality attributes priorities defined in the configuration.

Figure 7 shows the outline of the Product architecture instantiation in which the product architecture is generated after the resolution of the PLA architectural variability through the CVL resolution model generation. The product architecture shown in Fig. 7 has been generated by the *product architecture instantiation* for the automotive example when the application engineer selects only the *ABS* feature (see Fig. 1) and introduces the product specific NFRs, which come from the system's requirements, demanding a fault tolerance of the ABS greater than 99.5 % and restricting the ABS latency time to 5 ms.

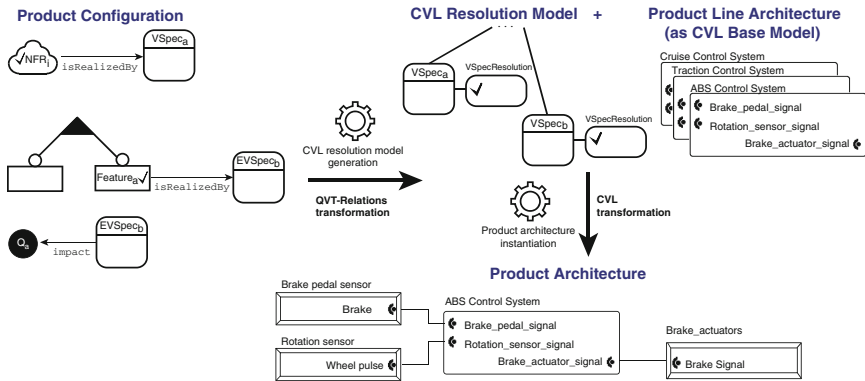


Fig. 7. Product architecture instantiation.

3.5 Product Architecture Evaluation and Improvement

After obtaining the product architecture during the product architecture instantiation, it should be evaluated to assess the degree of fulfillment of the product's NFRs and, in those cases in which the NFRs cannot be achieved by exercising the architectural variability mechanisms of the product line architecture pattern-based architectural transformations can be applied to the product architecture in order to improve its quality. This process comprises two main activities: *Evaluation* and *Transformation*. Figure 8 shows an excerpt of this process with its main inputs and outputs.

In the *Product architecture evaluation task* the evaluator applies the software measures from the quality viewpoint of the multimodel to the product architecture in order to evaluate whether or not it satisfies the desired NFRs. This can be done by means of various measurement methods:

- Measurement through model transformation processes: metrics that require more complex processing can be implemented as model transformations.
- Measurement through architectural modeling tools: in those cases in which some architectural modeling tools have mechanisms to perform the measurement, this will be delegated to such architectural modeling tools.

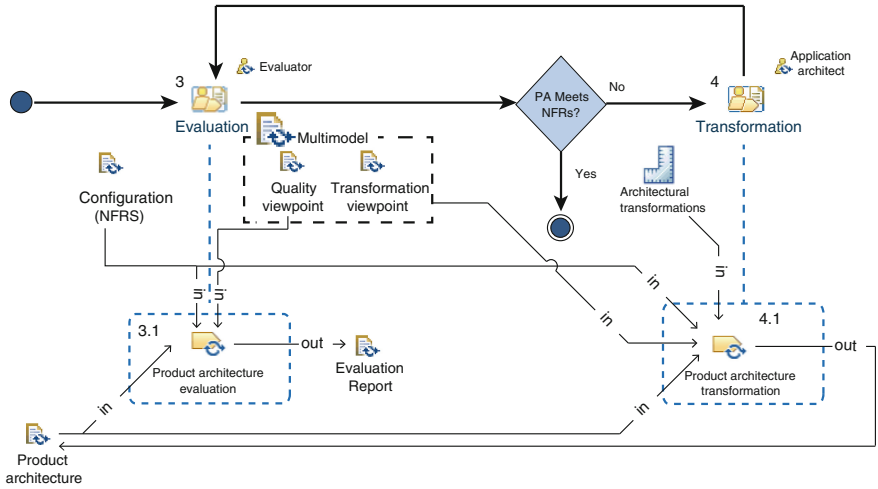



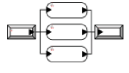

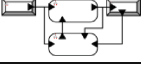
Fig. 8. Excerpt of the evaluation and improvement process.

- **Measurement through OCL restrictions:** in those cases where the metrics can be operationalized in this language, architectural models can be measured by using OCL constraints defined directly on the models at M1 level by means of the OCL Tools. These constraints can be used as a consistency validation for the obtained architectural models (e.g., to validate the memory consumption of all the components that combine the architecture using derived attributes).

Once the measurement process has been carried out by applying the measurement method selected in each case, the multimodel allows us to validate the degree in which the NFRs are fulfilled, using the measurement results. We have developed a tool that gives support to the creation and use of multimodels and which allows, on one hand, to specify the NFRs for both the SPL and the product under development and, on the other hand, to enter the measurement result in the multimodel and to perform the validation of the OCL constraints at runtime by using the OCLTools validator.

The evaluation for the example architecture shown in Fig. 7 may conclude that the architecture meets the latency NFR but that the fault tolerance NFR is not achieved, and architectural transformations may thus be required. In those cases in which the NFRs cannot be achieved by exercising the architectural variation mechanisms, in the second activity, *Product Architecture Transformation*, we can apply pattern-based architectural transformations to the product architecture. This transformation uses the impact relationships among architectural transformations and quality attributes to determine which architectural transformation must be applied to the product line architecture in order to achieve the desired quality attribute levels. In particular, the architectural patterns that we used for the automotive example are: homogeneous redundancy pattern [50] and triple modular redundancy pattern [50] whose details are briefly introduced in Table 5.

Table 5. Architectural transformations description.

Name	Rationale	Left-hand side	Right-hand side
T1: Triple Modular Redundancy (TMR)	Only detects random faults. Since the channels are homogeneous, any systematic fault in one channel must be present in both of the others.		
T2: Sanity Check Pattern (SC)	Detects gross deviations from the controlled value to the actuator value. Provides minimal coverage against faults.		

To define the corresponding impact relationships among architectural patterns and quality attributes, the domain architect must rank each architectural transformation with regard to the Q quality attributes in a trade-off analysis using the AHP technique. For each quality attribute Q_a , s/he compares the N potential architectural transformation in a pairwise comparison. To determine how an architectural transformation Ax supports the quality attribute Q_a , in comparison to the pattern Ay , a weight is assigned (1 for equally important, 3 for moderately more important, 5 for strongly more important, 7 for very strongly more important, and 9 for extremely more important). For example, the domain expert defines that TMR is strongly more important (a weight of 5) than HR with regard to fault tolerance, and that HR is moderately more important (a weight of 3) than TMR with regard to latency.

The result of this comparison is an $N \times Q$ matrix that shows the relative support of the different architectural patterns to the quality attributes as shown in Table 6(a). Then, these values are normalized by applying the formula (1) to (a) to produce Table 6(b), and finally, the impact that an architectural pattern has on a quality attribute Q_a is calculated by applying the formula (2) to produce Table 6(c). This result is stored in the multimodel in the *impact* relationships among architectural transformations and quality attributes (see Fig. 3 in Sect. 3.2) and is valid for all the products in the product line.

$$NormQ_a[i,j] = \frac{Q_a[i,j]}{\sum_{k=1}^n Q[k,j]} \quad (1)$$

$$I[i] = \frac{\sum_{k=1}^n NormQ_a[i,k]}{n} \quad (2)$$

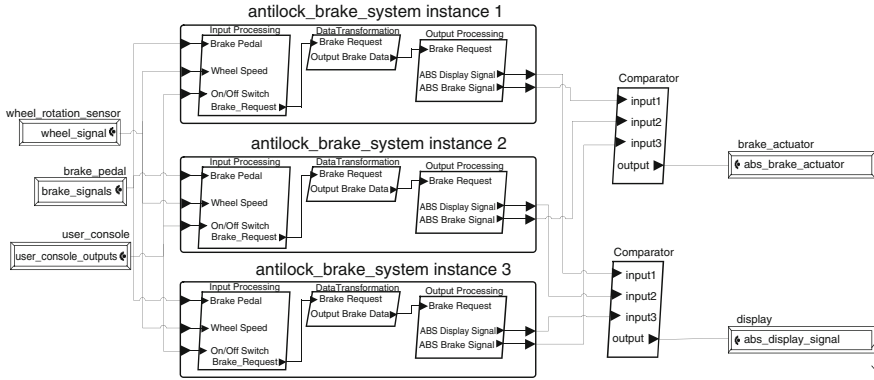
$$R_j = \sum_{i=0}^{k-1} Q_i * I_{ij} \quad (3)$$

In the automotive example, if the architect selects both the latency and the fault tolerance as being of equal importance (i.e., with a weight of 0.5 for each one) the transformation process will select the TMR pattern by applying the formula (3) to the values shown in Table 6(c) (TMR: $0.5 * 0.83 + 0.5 * 0.24 > HR: 0.5 * 0.17 + 0.5 * 0.76$). Figure 9 shows the resulting product architecture after the application of the TMR pattern to the product architecture shown in Fig. 7.

The process iterates until the NFRs are achieved or when the architect detects that it is not possible to build the product with the set of NFRs selected in the configuration.

Table 6. Architectural patterns and quality attributes trade-off analysis.

(a)	Fault Tolerance		Latency		(b)	Fault Tolerance		Latency		(c)	Impacts	
	TMR	HR	TMR	HR		TMR	HR	TMR	HR		Fault Tolerance	Latency
TMR	1	5	1	1/3	TMR	1 / 1.2	5 / 6	1 / 4	1/3 / 1.3	TMR	0.83	0.24
HR	1/5	1	3	1	HR	1/5 / 1.2	1 / 6	3 / 4	1 / 1.3	HR	0.17	0.76
Sum	1.2	6	4	1.3								

**Fig. 9.** Product architecture after applying the TMR pattern.

The evaluation process may result also in a renegotiation of the NFRs with the customer. In this case, the product architecture should be re-evaluated to check the conformance with the new NFRs. Finally, in some cases the architect should vary some architectural variation points to modify the candidate product architecture. For instance, in some cases the first candidate architecture may imply the positive resolution of a set of architectural variation points that may lead to quality attribute levels that are far above of a given NFR. Considering another combination of architectural variation points may also imply the fulfillment of that specific NFR but also other that were previously unfulfilled.

4 Conclusions and Final Remarks

Although in the last few years model-driven software development have been applied to address the problem of software architecture derivation and evaluation in SPL development, it still presents some drawbacks and opportunities. In general, quality assurance has not received proper coverage in existing approaches for product architecture derivation and there is a lack of generic methods that support the derivation and evaluation of product architectures regardless the quality attributes to be evaluated, the architectural description languages used to specify the architecture or the architectural viewpoints of interest. We believe that the use of model-driven principles would

provide a richer semantic representation of a SPL and may be used to relate the different activities that should be performed during the derivation and evaluation of product architectures (e.g., the impact that SPL external variability has on the non-functional requirements or how the architectural design decisions impact on the quality attributes).

We have also introduced QuaDAI as an integrated, generic method for supporting the derivation, evaluation and improvement of product architectures in MD-SPLE. In this method, the product derivation and the architectural transformations are guided by the relationships and constraints established in a multimodel. The multimodel provides a sufficiently formal interrelated model that can be supported by tools capable of automating portions of the product line production planning. The approach explores model-driven concepts and techniques to make explicit the knowledge and rationale used for architectural design by capturing and representing architectural design decisions during the architecting process necessary for reducing architectural knowledge evaporation.

The multimodel is a solution for documenting design decisions and their impact on the product quality attributes. The multimodel can also be used to analyze the cost/benefit of having core assets with certain qualities (impact on quality and cost). As further work, we plan to improve the configuration and consistency validation mechanisms, to provide recommendation mechanisms based on previous selections, and to implement consistency validation for individual entities of the multimodel. In addition, we want to improve the impact specification mechanism and to analyze other multi-objective optimization methods. We also plan to perform replications of the experiments conducted to evaluate the effectiveness of QuADAI with practitioners.

References

1. Rabiser, R., Grünbacher, P., Dhungana, D.: Requirements for product derivation support: results from a systematic literature review and an expert survey. *Inf. Softw. Technol.* **52**, 324–346 (2010)
2. Clements, P., Northrop, L.: *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, Boston (2001)
3. Atkinson, C., Bayer, J., Muthig, D.: Component-based product line development: the Kobra approach. In: *1st International Conference on Software Product Lines*, pp. 289–309, Denver, Colorado (2000)
4. Asikainen, T., Soininen, T., Männistö, T.: A Koala-based approach for modelling and deploying configurable software product families. In: *5th International Workshop on Product-Family Engineering*, pp. 225–249, Sienna, Italy (2003)
5. Botterweck, G., Lee, K., Thiel, S.: Automating product derivation in software product line engineering. In: *Software Engineering Conference*, pp. 177–182, Kaiserslautern, Germany (2009)
6. Cabello, M.E.: *Baseline-Oriented Modeling: una Aproximación Mda Basada en Líneas de Productos Software para el Desarrollo de Aplicaciones*. PhD thesis, Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València (2008)

7. Rossel, P.O., Perovich, D., Bastarrica, M.C.: Reuse of architectural knowledge in SPL development. In: Edwards, S.H., Kulczycki, G. (eds.) 11th International Conference on Software Reuse, pp. 191–200, Falls Church, VA, USA (2009)
8. Duran-Limon, H.A., Castillo-Barrera, F.E., Lopez-Herrejon, R.E.: Towards an ontology-based approach for deriving product architectures. In: 15th International Software Product Line Conference, vol. 2, Munich, Germany (2011)
9. Czarnecki, K., Antkiewicz, M.: Mapping features to models: a template approach based on superimposed variants. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 422–437. Springer, Heidelberg (2005)
10. Ziadi, T., Jézéquel, J.: Software product line engineering with the UML: deriving products. In: 10th Software Product Lines Conference, pp. 557–588, Baltimore, Maryland, USA (2006)
11. Gomaa, H., Shin, M.E.: Automated software product line engineering. In: 40th Annual Hawaii International Conference on System Science, pp. 1–10, Hawaii, USA (2007)
12. Perrouin, G., Klein, J., Guelfi, N., Jézéquel, J.M.: Reconciling automation and flexibility in product derivation. In: 12th Software Product Line Conference, pp. 339–348, Limerick, Ireland (2008)
13. Schaefer, I., Worret, A., Poetzsch-Heffter, A.: A model-based framework for automated product derivation. In: 1st International Workshop on Model-Driven Approaches in Software Product Line Engineering, pp. 14–21, San Francisco, California, USA (2009)
14. Tawhid, R., Petriu, D.C.: Product model derivation by model transformation in software product lines. In: 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, pp. 72–79, Newport Beach, Canada (2011)
15. Sánchez, P., Loughran, N., Fuentes, L., Garcia, A.: Engineering languages for specifying product-derivation processes in software product lines. In: Software Language Engineering, pp. 188–208, Toulouse, France (2008)
16. Heidenreich, F., Kopcsek, J., Wende, C.: FeatureMapper: mapping features to models. In: Companion of the 30th International Conference on Software Engineering, pp. 943–944, Vancouver, Canada (2008)
17. Haugen, Ø., Moller-Pedersen, B., Olsen, G.K., Svendsen, A., Fleurey, F., Zhang, X.: Consolidated CVL language and tool. MoSiS Project, D.2.1.4., SINTEF, Univeristy of Oslo (2010)
18. Montagud, S., Abrahão, S.: Gathering current knowledge about quality evaluation in software product lines. In: 13th Software Product Line Conference, pp. 91–100, San Francisco, USA (2009)
19. Kazman, R., Klein, M., Clements, P.: ATAM: method for architecture evaluation. CMU/SEI-2000-TR-004, ESC-TR-2000-004, Software Engineering Institute, Carnegie Mellon University (2000)
20. Kazman, R., Bass, L., Abowd, G., Webb, M.: SAAM: a method for analyzing the properties of software architectures. In: 16th International Conference on Software Engineering, pp. 81–90, Sorrento, Italy (1994)
21. González-Huerta, J., Insfrán, E., Abrahão, S.: Defining and validating a multimodel approach for product architecture derivation and improvement. In: 16th International Conference on Model-Driven Engineering Languages and Systems, pp. 388–404, Miami, USA (2013)
22. Gonzalez-Huerta, J., Insfran, E., Abrahão, S., Scanniello, G.: Validating a model-driven software architecture evaluation and improvement method: a family of experiments. *Inf. Softw. Technol.* **57**, 405–429 (2015)

23. Guana, V., Correal, D.: Improving software product line configuration: a quality attribute-driven approach. *Inf. Softw. Technol.* **55**, 541–562 (2013)
24. Fleurey, F., Haugen, Ø., Møller-Pedersen, B.: A Generic Language and Tool for Variability Modeling. SINTEF, Oslo (2009)
25. Dolan, T.J.: Architecture Assessment of Information-System Families: a Practical Perspective. PhD thesis, Technische Universiteit Eindhoven (2001)
26. Graaf, B., van Dijk, H., van Deursen, A.: Evaluating an embedded software reference architecture—industrial experience report—. In: 9th European Conference on Software Maintenance and Reengineering, Manchester, United Kingdom (2005)
27. Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H.: Architecture-level modifiability analysis (ALMA). *J. Syst. Softw.* **69**, 129–147 (2004)
28. Matinlassi, M., Niemelä, E., Dobrica, L.: Quality-driven architecture design and quality analysis method. VTT Publications 456, VTT Technical Research Centre of Finland, Oulu, Finland (2002)
29. Alves, E., Junior, D.O., Gimenes, I.M.S.: A metric suite to support software product line architecture evaluation. In: XXIV Conferencia Latinoamericana de Informática, pp. 489–498, Santa Fé, Argentina (2008)
30. Gannod, G., Lutz, R.: An approach to architectural analysis of product lines. In: 22nd International Conference on Software Engineering, pp. 548–557, Limerick, Ireland (2000)
31. Maccari, A.: Experiences in assessing product family software architecture for evolution. In: 24th International Conference on Software Engineering, pp. 585–592, Orlando, Florida (2002)
32. Riva, C., Rosso, C.D.: Experiences with software product family evolution. In: 6th International Workshop on Principles of Software Evolution, Helsinki, Finland (2003)
33. Tawhid, R., Petriu, D.C.: Automatic derivation of a product performance model from a software product line model. In: 15th International Software Product Line Conference, pp. 80–89. IEEE, Munich, Germany (2011)
34. Alonso, A., García-Valls, M., Puente, J.: Assessment of timing properties of family products. In: ESPRIT-ATES Workshop, pp. 161–169, Las palmas de Gran Canaria, Spain (1998)
35. Kim, T., Ko, I.Y., Kang, S.W., Lee, D.H.: Extending ATAM to assess product line architecture. In: 2008 8th IEEE International Conference on Computer and Information Technology, pp. 790–797, Khulna, Bangladesh (2008)
36. Olumofin, F.G., Misis, V.B.: A holistic architecture assessment method for software product lines. *Inf. Softw. Technol.* **49**, 309–323 (2007)
37. Etxeberria, L.: Evaluación de atributos de calidad en líneas de productos software de forma efectiva en costes. PhD thesis, Departamento de Electronica e Informatica, Modragon Unibersitatea (2008)
38. Roos-Frantz, F., Benavides, D., Ruiz-Cortés, A., Heuer, A., Lauenroth, K.: Quality-aware analysis in product line engineering with the orthogonal variability model. *Softw. Qual. J.* **20**, 519–565 (2011)
39. Soltani, S., Asadi, M., Gašević, D., Simon, M.H., Bagheri, E.: Automated planning for feature model configuration based on functional and non-functional requirements. In: Proceedings of the 16th International Software Product Line Conference, vol. 1, pp. 56–65, New York, NY, USA (2012)
40. Ghezzi, C., Sharifloo, A.M.: Verifying non-functional properties of software product lines: towards an efficient approach using parametric model checking. In: 15th International Software Product Line Conference, pp. 170–174. IEEE, Munich, Germany (2011)
41. Crnkovic, I., Larsson, M., Preiss, O.: Concerning predictability in dependable component-based systems: classification of quality attributes. In: ICSE 2004 Workshops on Software Architectures for Dependable Systems, pp. 257–278, Edinburgh, Scotland, UK (2004)

42. González-Huerta, J., Insfran, E., Abrahão, S.: A multimodel for integrating quality assessment in model-driven engineering. In: 8th International Conference on the Quality of Information and Communications Technology, pp. 251–254, Lisbon, Portugal (2012)
43. Barkmeyer, E.J., Feeney, A.B., Denno, P., Flater, D.W., Libes, D.E., Steves, M.P., Wallace, E.K.: Concepts for Automating Systems Integration. NISTIR 6928, U.S. Department of Commerce (2003)
44. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. CMU/SEI-90-TR-21 ESD-90-TR-222, Software Engineering Institute, Carnegie Mellon University (1990)
45. Gómez, A., Ramos, I.: Cardinality-based feature modeling and model-driven engineering: fitting them together. In: International Workshop on Variability Modelling of Software-Intensive Systems, pp. 61–68. Linz, Austria (2010)
46. Object Management Group: Common Variability Language (CVL) OMG Revised Submission (2012)
47. González-Huerta, J., Insfran, E., Abrahão, S., McGregor, J.D.: Non-functional requirements in model-driven software product line engineering. In: Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages, pp. 1–6. Innsbruck, Austria (2012)
48. ISO/IEC: ISO/IEC 25000:2005 Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE (2005)
49. González-huerta, J., Insfran, E., Abrahão, S., Mcgregor, J.D.: Architecture derivation in product line development through model transformations. In: 22nd International Conference on Information Systems Development, Seville, Spain (2013)
50. Douglass, B.P.: Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Addison Wesley, Boston (2002)
51. Kruchten, P.: The Rational Unified Process, an Introduction. Addison Wesley, Boston (1999)
52. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: a System of Patterns, vol. 1. Wiley, Chichester (1996)
53. Janota, M., Botterweck, G.: Formal approach to integrating feature and architecture models. In: 11th Conference on Fundamental Approaches to Software Engineering, pp. 31–45, Budapest, Hungary (2008)
54. ISA Research Group: Fama Tool Suite. <http://www.isa.us.es/fama/>
55. Eclipse: Eclipse OCL. <http://projects.eclipse.org/projects/modeling.mdt.ocl>

Model-Driven Engineering and Software Development
Second International Conference, MODELSWARD 2014,
Lisbon, Portugal, January 7-9, 2014, Revised Selected
Papers

Hammoudi, S.; Pires, L.F.; Filipe, J.; das Neves, R.C.
(Eds.)

2015, XIV, 197 p. 77 illus. in color., Softcover

ISBN: 978-3-319-25155-4