Embedded software product lines: domain and application engineering model-based analysis processes

Lorea Belategi1,*†, Goiuria Sagardui1, Leire Etxeberria1 and Maider Azanza2

1Mondragon Unibertsitatea Goiru 2, 20500 Arrasate, Spain
2University of the Basque Country UPV/EHU, San Sebastián, Spain

SUMMARY

Nowadays, embedded systems are gaining importance. At the same time, the development of their software is increasing its complexity, having to deal with quality, cost, and time-to-market issues among others. With stringent quality requirements such as performance, early verification and validation become critical in these systems. In this regard, advanced development paradigms such as model-driven engineering and software product line engineering bring considerable benefits to the development and validation of embedded system software. However, these benefits come at the cost of increasing process complexity. This work presents a process based on UML and MARTE for the analysis of embedded model-driven product lines. It specifies the tasks, the involved roles, and the workproducts that form the process and how it is integrated in the more general development process. Existing tools that support the tasks to be performed in the process are also described. A classification of such tools and a study of traceability among them are provided, allowing engineering teams to choose the most adequate chain of tools to support the process. Copyright © 2012 John Wiley & Sons, Ltd.

Received 15 March 2012; Revised 20 June 2012; Accepted 13 July 2012

KEY WORDS: software product line; model-based analysis process; model-driven development; quality attributes; performance

1. INTRODUCTION

Embedded systems are becoming ubiquitous both in industry and in our everyday lives, and the software that runs on them is fundamental for them to function. As a result, embedded system engineering teams need to face the challenges these systems pose to the development of software.

Although cost, quality, and time-to-market have always been main concerns in software engineering, these concerns are even more pivotal in embedded system software. Development time must be met and changing requirements managed as in other domains, but embedded software architectures are usually complex and fragile, technological platforms evolve and change constantly, and required quality attributes such as reliability or safety add even more complexity to development. Specifically, embedded systems distinguish themselves by the following characteristics: heterogeneity (hardware/software), distribution (on potential multiple and heterogeneous hardware resources), ability to react (supervision and user interfaces modes), criticality, and real-time and consumption constraints [1], and this impacts on how their software is developed. Considering the context these systems normally operate in, where malfunction can even result in casualties (e.g., healthcare systems), we need to be able to ensure the quality of their software.

*Correspondence to: Lorea Belategi, Mondragon Unibertsitatea, Goiru 2, 20500 Arrasate, Spain.
†E-mail: lbelategui@mondragon.edu

Copyright © 2012 John Wiley & Sons, Ltd.
Model-driven engineering (MDE) is a paradigm that helps to address software development by reducing the gap between the problem domain and the software implementation domain using abstractions (i.e., models), which are systematically transformed to concrete software implementations [2]. Models become the central artifact in the software development process, hence increasing productivity and shortening development time. The advantages of MDE even exceed these time issues. In the particular case of embedded systems, software verification and validation (V&V) from early development stages is crucial to ensure software quality and is one of the most spread means to ensure such quality is analysis [3]. Nevertheless, performing embedded software V&V is not trivial. Among other difficulties, in most cases, embedded software is hardware dependent (the hardware imposes requirements on the software that the software has to cope with) and has to run under different configurations (communicating with different number and kind of devices). MDE supports this early V&V, as models can be annotated with information related to quality attributes (using UML and profiles or domain-specific metamodels) [1, 4, 5].

Model-driven engineering offers multiple advantages both for the development and V&V of embedded system software. However, it does not deal with another of the challenges that embedded software poses: variability. Families of related embedded systems exist, which vary from each other in terms of their behavior, quality attributes, platform, network, physical configuration, middleware, scale factors, and in a multitude of other ways. In combination with MDE, software product line engineering (SPLE) can be an adequate alternative to traditional embedded software development to explicitly manage this variability.

Hence, MDE and SPLE are paradigms that bring benefits when dealing with the complexity of embedded software development. Nevertheless, these benefits come at the cost of increasing the complexity of the development process [6], and this complexity is even exacerbated when the software needs to comply with real-time constraints such as performance. As a result, the process to engineer these systems needs to cater for a diverse set of stakeholders (e.g., software, electronic, or real-time engineers), an increased number of workproducts (e.g., feature models, system models, code, and documentation), and new tasks (e.g., domain analysis or product derivation). Moreover, if we add V&V, this entails that an already complex process needs to deal with the analysis-related tasks and that the analysis process itself needs to consider the more diverse group workproducts required by MDE and SPLE (e.g., metamodels, transformations, and feature models).

In order to tackle the aforementioned complexity, this work focuses on the analysis process required for the V&V of an embedded model-driven software product line (SPL), based on UML and MARTE (UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems). It specifies the tasks, the involved roles, and the workproducts that form the process and how it is integrated in the more general development process. Existing tools that support the tasks to be performed are also described. Specifically, on the basis of [7], the contributions of this paper are the following:

- A more general analysis process where the different roles, tasks, and workproducts are identified. Existing tools that can support the process are also included.
- A classification of existing tools and a study of traceability among them, allowing engineering teams to choose the most adequate chain of tools to support the analysis process.
- An enriched metamodel of feature models that includes the means to model software allocation and analysis variability.
- A metamodel for relationship models, the workproduct that relates the different pieces of the analysis process, managing variability and relationships, and suggesting relevant AnalysisContexts.

This paper is organized as follows. Section 2 presents how model analysis is performed in model-driven embedded SPLs. Section 3 analyzes related works. A proposal process for such goal and tools to support the process are described in Section 4, and Section 5 presents the conclusions and future work.

2. MODEL ANALYSIS FOR MODEL-DRIVEN EMBEDDED SOFTWARE PRODUCT LINES

Quality is an important aspect that needs to be taken into account from the beginning and during the whole software life cycle. Software quality is the degree to which software possesses a desired

---

Footnote:

combination of attributes [8], and a quality attribute is a property of workproducts or goods by which its quality will be judged by stakeholders. However, performing software V&V is not trivial, and even less performing embedded software V&V due to critical quality attributes and limited resources involved in the system development.

There are many works that propose the use of models annotated with specific profiles in general with the MARTE profile in particular to help in performing model-based analysis following MDE [9–12].

2.1. MARTE profile and its elements for model analysis: AnalysisContext

MARTE profile [1], standardized by Object Management Group, facilitates validating and verifying temporal aspects through model analysis. MARTE analysis is intended to support accurate and trustworthy evaluations by using formal quantitative analyses based on mathematical models, which may supplement designer intuition profile [1]. Quantitative analysis techniques determine the output values such as response times, deadline failures, and resource utilizations on the basis of data provided as input, for example, execution demands or deadlines. To perform model-based analysis using MARTE, rather than requiring a special version of design models to be created only for the analysis purposes [1], extra annotations required for analysis are attached to existing design models (application, platform, and deployment) by the use of stereotypes that map model elements into the semantics of an analysis domain and tagged values. AnalysisContext is the main concern of MARTE to perform model analysis. It identifies models that gather information about system behavior and workload, execution platform, and allocation (Y approach [13]) for the analysis and specifies global parameters (properties that describe different cases being considered for analysis). Therefore, stereotypes related to the AnalysisContext term are classified in two concepts:

- **WorkloadBehavior**: It is a container of a set of end-to-end system operations used for analysis and defined by a set of workload events triggered over time. These stereotypes are used in design models where constraints, scenarios, and software design, including functional and quality requirements, are specified.

- **ResourcesPlatform**: It is a container for the resources used by the system behavior represented by the design model. These stereotypes are used in platform and allocation models where resources and their properties are described and platform design is specified.

To summarize, AnalysisContext allows analyzing what could be a real-time situation of the system by describing a specific scenario and the execution platform through analysis models with annotations that refer to the quality attributes that need to be analyzed.

2.2. Variability in AnalysisContext for model analysis in embedded software product lines

MARTE profile was defined for single system modeling and analysis. So when model-based analysis is performed for an embedded SPL, variability is a key aspect that must be considered: not all products of the SPL have the same functionalities; often, some of the hardware devices and other performance-affecting factors can vary from one product to another and so on. Therefore, analysis can vary from one product to another one. As a result, the AnalysisContext term and the analysis process itself must be extended to address SPL model analysis.

The wide variability issues involved in this SPL context make V&V even more complicated and require explicit management. SPLE separates development in two processes [14]: domain engineering (DE), the process in which the commonality and the variability of the product line are defined and realized; and application engineering (AE), the process in which the applications of product line are built by reusing domain artifacts and exploiting the product line variability [15]. Thus, besides taking into account the development, realization, and management changes that SPLE introduces (e.g., more workproducts, new tasks, and multidisciplinary teams), there are two architecture abstractions that must also be considered when carrying out model-based analysis: (i) line architecture and (ii) instance architecture.

---

6See footnote 4.
In this regard, two approaches can be applied when evaluating products depending on the desired goal:

- When the goal of the evaluation is to ensure that the required quality levels for all the products are achieved in the SPL, in order to reduce the evaluation effort, a subset of representative products can be selected and evaluated from all the products of the line, and, in this way, data can be extracted to estimate the results of all the products [16].
- If the goal is to evaluate a specific product, that product is derived and then evaluated, which is performed in AE.

Any of the two evaluation approaches can be followed by applying the process described on this paper. This process enables to obtain specific product analysis models either from a subset of the product line, in order to estimate analysis results for the whole line, or from specific products of the line.

As a case in point, some fragments of an E-Commerce system SPL are shown in Figure 1, where new branches have been defined to support model analysis. This system has to run under different platforms and devices (Home Customers access the system through PCs, whereas Business Customers may also access by mobile), and functionalities may change from one to another (e.g., Home Customers have restricted information compared with Business Customers). Even quality attributes or their degrees are not the same for all products derived from the product line. On the first branch in Figure 1(a), mandatory, optional, and alternative functional features and quality attributes, which describe the variability behavior of the AnalysisContext’s first concept (WorkloadBehavior), are specified. Each of the quality attributes has some critical scenarios to analyze, for example, the Performance quality attribute has critical scenarios such as Browse Catalog Scenario or Status Follower Scenario. And features may require or exclude other features (by defined constraints). Figure 1(b) facilitates information about platform and allocation variability that helps in configuring a specific product model of the product line and describing diverse ways to allocate system threads, and Figure 1(c) visualizes different analysis types for the critical scenarios and input/output analysis variables.

This section intended to motivate the need of an explicit process to manage model-based analysis in embedded SPLs. The next section provides an overview of the related proposals in this area.

3. RELATED WORK

The capability to be able to perform model analysis to validate and verify quality attributes at early phases facilitates in obtaining a product with the same functionality but different quality levels. This

Figure 1. (a) Feature model based on Etxeberria [16]; (b) allocation variability branch; and (c) analysis variability branch.
quality attribute variability must be taken into account during embedded software development. In the same way, embedded system families not only offer different functionalities but also need to cope with variable quality requirements (e.g., the same system may have different performance requirements depending on its location). Broadly speaking, this implies to cover two areas: SPLE for variability modeling and management and model-based analysis for early V&V.

Regarding SPLE, variability modeling is key to manage variability in software product families [17]. Different approaches related to variability modeling and management have been proposed, but those techniques rely on different technical background, and most variability modeling techniques lack a description of a process. Over the last years, several variability modeling techniques that are aimed to support variability management have been developed. Although these mechanisms are suitable for managing variability and feature models are widely used in embedded systems domains [18–20], no standard way to model variability has been defined yet (Common Variability Language is a request proposal [21]), and few works cover other phases such as software V&V.

Model-based analysis for early V&V in embedded systems has raised a considerable interest. To cite a few, research works such as the one carried out in MeMVaTEx or [22] did not consider variability, although temporal aspects are annotated with MARTE. MeMVaTEx methodology presented in [20] proposes the decomposition of the design process in different abstract levels of the EAST-ADL2 framework. For each level, requirements and solution models are created in a separate way, and the interrelations among the elements of these models are specified through the traceability mechanism of the SysML profile, whereas real-time issues are specified through the MARTE profile. The proposed methodology focuses on requirement traceability from analysis to the implementation phase, taking into account temporal issues and regardless of the variability and V&V of them. Espinoza [22] proposed a methodology that describes a set of steps to perform complex model analysis. In this methodology, different computation blocks must be defined, adequate nonfunctional properties specified, and so on before reusing model elements. It has been defined for a single product model analysis. Thus, some modifications are needed before applying it in embedded SPL. In [23], a model-based methodology oriented to distributed embedded and real-time applications development that focuses on the requirements traceability management is proposed, while the variability and verification phase is left for future work. And Anssi et al. made a deep study about MARTE and AUTOSAR for enabling timing analysis for the automotive domain in [24].

With both areas combined, the works presented by Tawhid and Petriu [9, 12] are the closest ones to our work as SPL and MARTE are considered. In [9], they proposed an SPL modeling approach that considers functional variability and annotations with the MARTE profile for performance in a general way (i.e., using variables). In order to validate quality aspects, concrete values are assigned to general annotations through Atlas Transformation Language (ATL) transformations that are also used to obtain a specific product model. But variability management is slightly defined. Although the research has been improved in [12] specifying each task, not all variability issues (e.g., allocation variability) have been taken into account when carrying out model analysis.

Founded on previous work in both areas, the following section presents a model-based analysis process required for the V&V of an embedded model-driven SPL, with a special emphasis on explicit management of all the involved variability dimensions (i.e., functional, quality, allocation, and analysis).

4. MODEL ANALYSIS PROCESS FOR EMBEDDED SOFTWARE PRODUCT LINES

Model-driven engineering and SPLE are paradigms that have established their benefits for the development of software in general and embedded system software in particular [25, 14, 26]. At the same time, the use of such paradigms together with the requirement to comply with quality attributes such as performance increases process complexity, as the process to engineer these systems needs to cater for a diverse set of stakeholders (e.g., software, electronic, or real-time
engineers), an increased number of workproducts (e.g., feature models, system models, code, and documentation), and new tasks (e.g., domain analysis or product derivation).

This entails that an already complex process needs to deal with the analysis-related tasks. Moreover, the analysis process itself needs to deal with the more diverse group of workproducts required by MDE and SPLE.

As a result, and in order to manage and perform model analysis for ensuring the quality of the products of the SPL, it is necessary to establish a process with the aim of tackling the aforementioned complexity. Following established product line development practices [14], two separate processes were defined for model-based analysis.

- **Model-based analysis process in DE:** This process sets the infrastructure to perform the analysis of the products of the product line. That is, it defines the analysis core assets.
- **Model-based analysis process in AE:** On the basis of the aforementioned analysis core assets, this process specifies how they are applied to perform model analysis.

### 4.1. Model-based analysis process in domain engineering

If the goal is to develop and analyze an embedded SPL, the core assets developed in DE (e.g., models, as MDE is applied) must be prepared to perform analysis later on. This is the aim of the model-based analysis process in DE.

This process takes the feature model and the design models as input and outputs core assets that will serve as the building blocks for model analysis in AE. When SPL development must begin from scratch, these models will be developed at the required stage. Figure 2 depicts the process using *Software & Systems Process Engineering Meta-Model Specification* [27]. The following subsections describe its tasks in detail, the workproducts that are developed in them, and the main roles involved in the model-based analysis process in DE. The different participating roles are central, as embedded

![Figure 2. Model-based analysis process in domain engineering: (a) domain engineering general process and (b) each tasks specification.](image-url)
systems are developed by multidisciplinary groups of stakeholders who contribute to the team with specific knowledge.

4.1.1. Feature model elaboration. In the first task, a feature-oriented domain analysis-based feature model is created [28]. It provides a global vision of the variability of the line. Traditionally, this feature model contains information about features that different products derived from the SPL must give response to. In the case of embedded systems, this model also needs to be extended with the quality feature tree, emphasizing the importance of the quality attributes in these systems [16]. This branch gathers information about quality attributes and the desired quality that needs to be assured, as not all products of the line may require the same quality attributes or degree of them; thus, these attributes should be considered from the beginning of the process. It also contains constraints among functional and platform features that facilitate in detecting impossible configurations. These latest features refer to the possible platform resources and design patterns for the embedded software derived from the SPL. An example of a feature model with these characteristics can be seen in Figure 1. An excerpt of the E-Commerce SPL is modeled, where some functional features and a quality feature tree are described in Figure 1(a). For example, as it is specified, Performance and Schedulability attributes are optional in the quality feature tree, and the functional mandatory feature called Business Type can be the Home Customer or Business Customer.

On the other hand, when model-based analysis is performed, other variability issues apart such as allocation and analysis variability must be considered since the beginning. As this information is not of the same abstraction level, it is worth defining and developing different branches in the feature model gathering the different required information. ‘The process of specifying a family member may also be performed in stages, where each stage eliminates some configuration choices’ [29]. These branches will be explained more thoroughly in the following tasks.

There are many tools that support feature modeling for SPLs such as FeatureModel Plugin (FMP) [30], RequisitePro [31], Feature Model Analyzer (FAMA) [32], and Quality Feature Model Plugin (QFMP) [16], but not all of them are suitable for time-related model analysis purposes. The tool chosen to aid in this task must be able to handle the following:

- Temporal quality attributes: quality attributes such as performance or schedulability.
- Constraints among features: require and exclude type relationships among features.
- Allocation and analysis variability required in later steps: possible software allocations for each platform and different analysis types with diverse variables to perform model analysis to contemplate all the wide variability of products that can be derived from the SPL.

None of the aforementioned tools satisfy all requirements completely. Hence, on the basis of feature-oriented domain analysis and [16], a metamodel for feature models has been defined and described in the following subsection.

During feature model definition, domain engineers and domain experts are the ones responsible for the definition of the feature model, where functional and quality attributes are described. Domain engineers together with software architects identify real-time and embedded systems’ critical scenarios to be analyzed. As a result of this task, a feature model that gathers quality attributes, functional, and platform features besides the constraints among them is obtained, which will serve as input for the following tasks.

4.1.2. Allocation variability elaboration. Allocation variability specification (Figure 1(b)) starts once the functional features and quality attributes have been defined. Therefore, the allocation variability branch is described on the feature model by domain engineers with the help of domain experts (the role with the domain knowledge but often no software development expertise) and hardware experts, identifying and describing constraints among features. To be able to do that, the aforementioned metamodel has been defined. A metamodel is an abstraction of the model that defines the language for expressing a model.**

The metamodel described in Figure 3 gathers information related to all variability information required for model analysis (i.e., functional, platform, allocation, and analysis variability) to be contained in the feature model. It considers functional, platform, and implementation features

---

**This is defined by http://www.omg.org/mof/
besides constraints among features as defined by Kang et al. [28]; critical quality attributes and impacts among features previously extended by [16]; and allocation and analysis-related features defined in this paper. On the basis of the metamodel proposed in [16], this metamodel introduces some concepts to facilitate the use of feature models with the aim of analysis. Two new concepts, that is, AllocationNode and AnalysisNode, have been introduced. The AllocationNode concept is related to allocation variability (Figure 1(b)). It allows to identify a feature as being of allocation type, making it possible to define a feature model branch with allocation variability features, where AllocationNode inheritates FeatureNode and PlatformNode’s properties. Thus, threads identified as AllocationNode can be bound to platform resources due to PlatformNode. On the other hand, AnalysisNode helps to specify diverse analysis types for critical scenarios by the relationship with the Scenario metamodel and also input/output variables. These concepts are used when developing a feature model where analysis-related variability is described (Figure 1(c)).

To summarize, the input feature model is modified, adding an allocation variability branch (based on the metamodel specified earlier), which is the input model for the next task.

4.1.3. Analysis variability elaboration. In the same way the allocation variability branch has been described, the analysis variability branch must be developed. It gathers information related to critical scenarios to be analyzed and analysis types for each quality attribute defined before. Later in this branch, only those possible scenarios related to the selection made in the quality feature tree are shown, somehow suggesting the analyses that can be performed for the configured product model. Taking as basis the metamodel (Figure 3), different types of analysis and input/output variables are specified, and constraints are also defined in the feature model. Type node of the metamodel gathers information about the type of analysis to be carried out on the models (e.g., sensitivity analysis), whereas input/output variables (Variable, Input, and Output nodes of the metamodel) make it possible to perform diverse analysis cases of a specific analysis type, and with both concepts combined, a wide variety of analysis can be performed.

Some of these features have constraints that select or deselect other features of the feature model (e.g., when Browse Catalog Scenario is selected in the quality feature tree for a configuration in Figure 1(a), Browse Catalog Scenario will be selected in the analysis variability branch (Figure 1(c)) as a critical scenario to perform model analysis). This feature model is the output workproduct of this task that will be taken into account when annotating SPL design models.

4.1.4. Real-time specification. Software product line design models (the set of models that represent the SPL: system’s structure, behavior, platform, and deployment) must be modeled considering
variability issues and then annotated with extra information that will later be used for analysis. Models must be developed in a generic way, thus making them capable of representing common and variant aspects of the products that make up the product line. For this purpose, Gomaa [33] and MARTE profiles [1], for variability and temporal information respectively, have been used, where these annotations add temporal aspects information to the models using stereotypes and tagged values (e.g., hostDemand or speedFactor). When modeling and annotating SPL design models, software engineers and domain experts take into account the feature model defined before.

4.1.5. Transformation definition. To be able to generate the analysis models automatically, the mechanisms developed earlier must be connected among them, assuring traceability of the models. Therefore, an analysis environment that gives response to this need has been developed. Having as inputs the feature model and the annotated SPL design models, transformation specialists must define the transformation rules for the relationship model. Note that variability in functionality, quality attributes, platform devices and allocation, and relationships among all those variability issues make analysis variable from one product of the product line to another one [34]. Considering the existence of multiple concepts that are related to each other, a model to systematically manage them all is required, and even more in an SPL setting, where these artifacts contain commonalities and variabilities that must be managed among all products of the product line. The relationship model specifies these relationships and allows to perform different AnalysisContexts.

The metamodel in Figure 4 was defined for this aim. It relates all the variability described in the feature models to themselves through constraints among features from three branches. A set of desired features defines a specific configuration, and an instance of AnalysisContext is the desired result of this analysis environment (an analysis configuration or set of desired features for model analysis). Specifically, it is an analysis type of a critical scenario representing a specific behavior, a concrete platform model and how software is allocated on it (deployment), and the specific value instances for that analysis (analysis case).

The transformation specialists are the ones that take care about the analysis environment, which makes possible the derivation to a specific product model analysis.

4.2. Model-based analysis process in application engineering

Previously developed core assets in DE are used in AE process to perform model analyses, prioritizing the critical scenarios of each specific product model of the SPL.

Desired features must be selected from the different branches of the feature model. In this way, the specific product model configuration is defined considering the specified constraints. Meanwhile, the
relationship model will be modified with each configuration decision taken in the branches, deselecting all those possible choices that were not selected in the configuration feature set (i.e., in the features selected by the user for a specific product). This is possible through the analysis environment developed in the aforementioned DE process. Thus, this model will give specific instances of AnalysisContexts by using the transformation tool to obtain the specific product model derived from the SPL design models.

To achieve the aim of model-based analysis, the following tasks must be realized where analysis core assets developed in DE are used. Through the analysis environment developed in DE, all mechanisms (feature model, annotated SPL design models, and the relationship model) are connected to each other, making possible to configure the specific analysis product model. Thus, once the configuration is performed, transformations would be applied automatically, and the specific AnalysisContexts would be obtained (Figure 5).

4.2.1. Feature model configuration. The main objective of this task is to select desired features related to functionality, platform, and quality attributes. In this way, application engineers start the configuration of a specific product and the derivation of a specific product model analysis by using the reusable assets built in DE that are managed by the model analysis environment. The set of selected features restricts possible choices from the allocation and analysis variability branches and limits the possible analyses to perform, which are gathered together in the relationship model. Those restrictions and limits are defined and handled by the constraints and transformations rules specified before, hence capitalizing on the previous work and increasing reuse.

4.2.2. Allocation variability configuration. The allocation variability input branch is the result of the selection made in the feature model functional and quality branches. It allows application engineers to select just the possible features for that particular feature model instance (i.e., only the allocation variability available once the functional and quality variability have been bound are presented to the engineer). At this step, features related to software deployment into the platform are selected, and the relationship model deselects those AnalysisContexts that do not satisfy the selected set of features (configuration), minimizing the wide range of AnalysisContexts to be carried out.

4.2.3. Analysis variability configuration. And finally, analysis variability branch configuration must be carried out. Features related to the variability that can be found in the analysis are selected in this task, resulting in an analysis variability instance. The relationship model considers this instance to choose the specific AnalysisContexts to be performed. Therefore, the transformations are applied to annotated design models to obtain the specific product analysis models where specific values are bound to input analysis variables that are suggested by the relationship model. Later, depending on

Figure 5. Model-based analysis process in application engineering: (a) transformations and (b) application engineering.
the used analysis tool, MARTE analysis models must be transformed to its input model by using an existing bridge tool or, if such tool does not exist, by defining a new one (Table I). Analysis experts can study the results obtained after performing model analysis and use them as feedback for the SPL model, considering if the desired quality attributes are assured.

4.3. Summary

A model-based analysis process for embedded SPLs has been described in this section, where MDE and SPL mechanisms have been applied to assure quality attributes such as performance. There are a wide variety of tools that facilitate software modeling, but not all of them support MARTE profile annotations. Therefore, the available tools, how they can give support and where (on what tasks) has been studied. The main issue for this study (Table I) has been to analyze how these tools can be combined to assure the traceability of MARTE analysis models, as different tools for different purposes are required during the process.

Although some tools that appeared earlier do not support MARTE, they are worth mentioning for their close relation with the profile, such as ArgoUML or Rational Rose that support SPT (UML Profile for Schedulability, Performance and Time, MARTE predecessor) [35] or Modeling and Analysis Suite for Real-Time (MAST) [36] profiles, respectively. Some tools such as Papyrus come from academia, whereas others such as Rational Software Architect or MagicDraw are commercial tools from industry.

Other research that must be mentioned is the one carried out by Anssi et al. in [24] where they mentioned that the tool support for AUTOSAR†† and MARTE still need to be improved further for an efficient use in an industrial development. This work intends to be a step on such direction.

The concrete points that have been considered for the study are the following:

1. Requirements: if there is a tool for feature modeling or requirement specification.
2. Modeling: a modeling tool and the annotation profiles that are supported (including MARTE).

Table I. Tools traceability during embedded software product line development process.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Modeling</th>
<th>Transformation</th>
<th>Analysis/simulation</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>System</td>
<td>Profile</td>
<td>Derivation/configuration</td>
<td>Bridge to analysis</td>
</tr>
<tr>
<td>FMP</td>
<td>Papyrus</td>
<td>MARTE, SysML, user-developed profiles</td>
<td>ATL, Xpand, Acceleo, JET templates</td>
<td>Yes</td>
</tr>
<tr>
<td>FMP, Doors, RSA</td>
<td>Rhapsody + Eclipse Modeling Tool</td>
<td>MARTE, SysML, user-developed profiles</td>
<td>ATL, JET, IBM TF</td>
<td>Yes</td>
</tr>
<tr>
<td>pure::variants</td>
<td>ArgoUML</td>
<td>SPT</td>
<td>ArgroMDA, Acceleo, ATL, Xpand</td>
<td>ArgoSPE</td>
</tr>
<tr>
<td>Sam Editor</td>
<td>Papyrus (TOPCASED)</td>
<td>MARTE</td>
<td>Yes</td>
<td>PEPA (perf.)</td>
</tr>
<tr>
<td>Cameo Requirements</td>
<td>Rational Rose</td>
<td>MAST</td>
<td>MAST (sched.)</td>
<td>[44]</td>
</tr>
<tr>
<td></td>
<td>MagicDraw</td>
<td>SysML, MARTE, TimeSquare</td>
<td>CoFluent Studio</td>
<td>[45]</td>
</tr>
<tr>
<td></td>
<td>Enterprise Architect</td>
<td>MARTE</td>
<td>XML</td>
<td>EQN Simulator</td>
</tr>
</tbody>
</table>

††Another environment used for developing software: http://www.autosar.org/
Transformation: the model-to-model transformation mechanisms that are supported for product derivation and if there is a transformation bridge available for MARTE analysis model to analysis tool input model.

Analysis/simulation tool: performance and schedulability model analysis tools.

References (Ref.): references for tools, research works, and case studies.

To sum up what has been described on Table I, not all modeling tools that support MARTE profile guarantee traceability of the design models with tools that exist nowadays. Some improvements are required to facilitate model-based analysis with existing analysis tools. Although transformations can be applied from MARTE analysis models into the input model of analysis tools (e.g., Cheddar or MAST), both metamodels need to be understood to specify the transformation rules and thus to develop bridge tools that make possible the automatic transformation. An open source tool chain solution to build an analysis environment could be composed by FMP and Papyrus for modeling the SPL, MARTE profile to add temporal information, ATL transformations to derive to a specific product model and the analysis bridge for the MAST scheduling analysis tool developed by the University of Cantabria.

5. CONCLUSION AND FUTURE WORK

In this paper, a model-based analysis process for embedded SPLs, focusing on quality attributes, has been proposed. Throughout the process, variability issues such as allocation or analysis have been taken into account. This process is defined to supply the lack of research that exists in embedded SPL model-based analysis. In order to ensure all the requirements, even the ones regarding quality attributes, for each specific product of the product line are met, model-based analysis techniques can be applied.

An analysis environment, which includes the process and binds the required elements to carry out the analyses, has been developed. Profiles such as Gomaa and MARTE for variability modeling and quality attributes annotation, respectively, have been used in models. Moreover, as analysis configuration might be performed in different stages, with different knowledge and so on, a staged configuration mechanism composed by allocation and analysis variability branches at the feature model besides the traditional ones has been included in the process, thus enriching the feature model metamodel. As a consequence, features related to diverse abstraction levels can be managed, facilitating the analysis configuration that specify more details in each stage with each decision. A relationship model has been also used (and its metamodel defined). It is a mechanism that gathers and manages relationships among all variability issues (functional, quality attributes, allocation, platform devices, and analysis) that take part in embedded SPL development and model analysis. This model helps in obtaining the specific AnalysisContexts for each specific product configuration of the embedded SPL. It facilitates in prioritizing critical scenarios for specific configurations, helps in binding concrete values to the input/output variables, and gives feature traceability.

A study of the existing tools and how they can help in different stages of the product configuration and derivation and how they can be combined and/or extended to assure model’s traceability has been made. With that aim, tools that come from diverse domains but support MARTE profile have been taken into account. We also complement our proposal, studying whether existing proposals or mechanisms could help in allocation variability and can be introduced in the proposed process. Moreover, we are working on developing an analysis environment tool that facilitates automatic model-based analysis by integrating development and analysis tools.

The presented approach enables to situate the decision making regarding critical requirements at design time, as consequences can be evaluated by studying the results provided by the model analysis tool. Model analysis allows software engineers to predict the quality attribute consequences of alternative design decisions and to select the optimal architecture without costly prototyping. This is exacerbated in a product line setting, where the building of every product is often unfeasible. Moreover, when facing evolution of some features of the line, systematic management of analysis variability at model level allows to evaluate the impact on quality attributes.
Aspect-oriented modeling as a means to develop model-driven product lines has raised considerable attention [46]. This positive way to address variability (i.e., by adding the required features to an initial base core) complements the approach presented in this paper (where nonrequired features are eliminated from the product). The first steps are presented in [47].

The future work to be carried out includes the realization of another real case study where aforementioned variabilities are considered and properly managed and the developed analysis environment is applied in order to perform model analysis. In the same way, this industrial case will help in checking the scalability of the proposal and in identifying possible conflicts.

ACKNOWLEDGEMENTS

This work is co-supported by the Spanish Ministry of Education and the European Social Fund under contract TIN2011-23839 (Scriptongue) and Basque Government under grant UE2011-4 (COMODE Project). The project has been developed by the embedded system group supported by the Department of Education, Universities and Research of the Basque Government.

REFERENCES

AUTHORS’ BIOGRAPHIES

Lorea Belategi was born in Mondragon, Spain, in 1981. She received her MSc degree in Computer Science Engineering from the University of Mondragon, Spain, in 2006. She is currently with the Department of Electronics and Computer Sciences at the University of Mondragon, working towards her PhD degree in the area of software engineering in topics such as MARTE model-based analysis and embedded software product lines.
Goiuria Sagardui obtained her PhD degree in Computer Sciences from the University of the Basque Country (Spain) in 1999. Previously, she finished her BSc in Software Engineering at Deusto University (Spain) in 1997. Since 2000, she has been working as a lecturer/researcher in the Department of Computing and Electronics in Mondragon University, and she has been the coordinator of the software engineering group of the University of Mondragon. Her main research area is software engineering, including topics such as software product lines and model-driven engineering.

Leire Etxeberria obtained her PhD degree in Computer Science from Mondragon University (Spain) in 2008. Previously, she finished her BSc in Computer Science Engineering at Mondragon University in 2004. Presently, she is working as a lecturer/researcher in the Department of Computing and Electronics in Mondragon University.

Maider Azanza obtained her PhD degree in Computer Sciences from the University of the Basque Country in 2011. She is currently a lecturer at the Computer Languages and Systems of the same university, where she teaches software engineering courses. She is part of the Onekin Research Group, and her research interest includes model driven engineering, software product line engineering, and the challenges their combination poses.