DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Informe Técnico / Technical Report

Pages: 65

Title: Review of Requirement Engineering Approaches for Software Product Lines

Author (s): David Blanes Domínguez , Emilio Insfrán

Date: Feb 22, 2011

Key Words: Requirements Engineering, Software Product Lines, Requirements Domain Engineering, Requirements Application Engineering, Scoping

VºBº **Author (s):**

Leader of the Research Group

José Hilario Canós Cerdá David Blanes, Emilio Insfrán

Abstract

The Software Product Lines (SPL) paradigm is one of the most recent topics of interest for the software engineering community. On the one hand, the Software Product Lines is based on a reuse strategy with the aim to reduce the global time-to-market of the software product, to improve the software product quality, and to reduce the cost. On the other hand, traditional Requirement Engineering approaches could not be appropriated to deal with the new challenges that arises the SPL adoption. In the last years, several approaches have been proposed to cover this limitation. This technical report presents an analysis of specific approaches used in the development of SPL to provide solutions to model variability and to deal with the requirements engineering activities. The obtained results show that most of the research in this context is focused on the Domain Engineering, covering mainly the Feature Modeling and the Scenario Modeling. Among the studied approaches, only one of them supported the delta identification; this fact implies that new mechanisms to incorporate new deltas in the Domain specification are needed. Regarding the SPL adoption strategy, most of the approaches support a proactive strategy. However, this strategy is the most expensive and risk-prone. Finally, most of the approaches were based on modeling requirements with feature models giving less support to other important activities in the requirements engineering process such as elicitation, validation, or verification of requirements. The results of this study provide a wide view of the current state of research in requirements engineering for SPL and also highlight possible research gaps that may be of interest for researchers and practitioners.

Keywords

Requirements Engineering, Software Product Lines, Requirements Domain Engineering, Requirements Application Engineering, Scoping.

Table of Contents

List of Figures

1 Introduction

Software development companies continuously look for alternative methods and techniques to improve the process of building software products. The main purpose of these efforts is to reduce costs and to improve the quality of the software product properties such as security, reliability, etc. In this scenario, in the last years, the Software Product Lines (SPL) approach has emerged as a new paradigm to build software based on an intensive reuse strategy. Software Product Line Engineering has proven to be a methodology for developing a diversity of software products and software-intensive systems at lower costs, in shorter time, and with higher quality [39].

Traditional software development methods are inadequate to address challenges of rapid change and growth of requirements. In addition, in the context of SPL, requirements must also capture specific properties such as variability, commonality or evolution. Moreover, the requirements engineering (RE) activity is applied to the different phases of the SPL development: Scoping, Domain Engineering, and Application Engineering. In the last years, several approaches were proposed to deal with these new activities and this new specific needs [28] [31] [17] [34]. For the purpose of our study, we are especially interested in the works that give a total or partial coverage to the Model-Driven Development approach.

Model-Driven Development (MDD) is an approach to software development that proposes the use of models at various levels of abstraction and model transformations as it mains artifacts. These transformations allow converting one source model into another target model. This use of model transformations allows an improvement of the reuse in the process of software development. For this reason, we consider that the SPL process can be significantly improved adopting a MDD strategy.

In this study, we give an overview of the current RE approaches for SPL. The aim of the work is to provide to developers a comparison framework to choose among different approaches that better suit to a particular goal. In addition, we also analyze how these RE approaches give support to the MDD approach. This analysis can help to SPL developers to identify strengths and weaknesses of the approaches regarding the MDD characteristics.

This document is organized as follows. Section 2 introduces the main concepts of RE, SPL and MDD. Section 3 introduces the evaluation criteria used for comparing the different approaches included in this study. Section 4 evaluates the selected RE approaches for SPL against the defined evaluation criteria. Section 5 shows the comparison among the RE approaches and a discussion derived from this comparison. Section 6 presents the related work, and finally, section 7 presents the conclusions and further work.

2 Preliminary concepts

In this section, we introduce the fundamentals of requirements engineering, software product lines, and model-driven development which are the main areas of this study.

2.1 Requirements Engineering

The success of a software system depends on the achievement degree on the understanding of the user and environment needs. In order to understand these needs, the Requirements Engineering plays a central role in the software development. This success relies directly on the quality of the software product developed. According with Zave [47], the RE is "the branch of software engineering concerned with the real-world goals for functions of and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families".

In order to understand what is the RE is necessary to define the requirement concept. There are multiple definitions regarding what a software requirement is. According to the IEEE Std. [14], a requirement is:

(1) A condition or capability needed by a user to solve a problem or achieve an objective.
(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
(3) A documented representation of a condition or capability as in (1) or (2).

The classic way to categorize requirements is according to whether they are functional or non-functional. A f*unctional* requirement is a requirement that specifies a function that a system or system component must be able to perform. A non-functional requirement is a requirement that defines restrictions on a system or system component that it must satisfy, such as reliability, security, usability, etc.

These requirements are documented in the Requirements Specification (RS). The RS is a document that contains a complete description of the behavior and constraints of the system to be developed.

The RE includes several activities. There is no consensus about which activities involve the RE. However a well-know classification is provided by Betty Cheng and Joanne Atlee [9], where the RE can be divided into five main activities: requirements elicitation, requirements modeling, requirement analysis, validation, and verification. Following, we introduce these main activities.

Elicitation

The requirement elicitation is usually the first activity in the RE process. It comprises activities that enable the understanding of the goals, objectives, and motives for building a proposed software system. Elicitation also involves identifying the requirements that the resulting system must satisfy in order to achieve these goals.

Modeling

In requirements modeling, a software requirements specification is expressed in terms of one or more models. These models should be more precise, complete and clearly than the models created during the elicitation activity. The process of creating precise models helps to identify details that were not identified in the elicitation activity.

Analysis

The requirements analysis includes activities to evaluate the quality of the requirement specification. Many of them analyze errors of well-formedness errors in requirements. The errors can be ambiguity, inconsistency or incompleteness. Other analyses look for anomalies, such as unknown interactions among requirements, possible obstacles to requirements satisfaction, or missing assumptions.

Validation and verification

The requirements validation ensures that models and documentation accurately express the stakeholder's needs.

Requirement management

The requirements management is an activity that comprises a number of tasks related to the management of requirements, including the evolution over time and across product families.

2.2 Software Product Lines

There are many reasons to adopt a Product Line Engineering approach. Many of them are related with the economic success. The large-scale reuse allows reducing costs, reducing time to market and improving the quality of resulting products.

The improvement of costs and time to market are strongly correlated in SPLS: the approach supports large-scale reuse during software development. This reuse is based on the principle of having a set of software-intensive systems sharing a common, manages set of features satisfying 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 [10]. A core asset is an artifact used in the production of more than one product in SPL [10].

The SPL development consists of two main processes [39]: Domain Engineering and Application Engineering.

- The *Domain Engineering* aims the development of software assets for reuse and effectively establishes the product line infrastructure. This process includes the commonality and variability definition of the Software Product Line. The Domain Engineering process is composed of five key sub-processes: product management, domain requirements engineering, domain design, domain realization, and domain testing [39]. For our point of view, we are interested in two sub-process of the Domain Engineering: Product Management, and Domain Requirements Engineering.

  Product management deals with the economic aspects of the Software Product Line in particular with the market strategy. The goal product management includes activities like: to decide a market strategy, to define which products will be part of the product line, or activities related with the economic product management. However, the product

management is a process too general, which involves multiple areas like economics. Inside the product management, we can find an activity called: *Scoping*. Scoping can be defined as the process of deciding in which parts of an organization's products, features and domains systematic reuse is economically useful [17]. The main goal in the scoping is to define the products that will be developed in the SPL and the main features of them. Three levels of Scoping can be identified [43]: Product Portfolio, which determines which products should be included in the SPL; Domain Scoping, that aims to identify and bounding the domain; and the Asset Scoping, which determines specific assets to be developed for reuse. These levels are dependent with each other. The main artifact produced is the Product Portfolio. The *Product Portfolio* contains the product types offered by a company in a product line. This artifact serves as input for the Domain Requirements Engineering sub-process.

In *the Domain Requirements Engineering,* a requirements specification is produced, including the high-level commonality and variability modeling. In this sub-process, the requirements are analyzed to identify those that are common to all application and those that can only be in specific applications.

- The *Application Engineering* includes the development all the activities to develop a single product. This sub-process includes the identification of new requirements for the application (called deltas). In this case, a decision must be taken whether include the requirements in the SPL family or just consider the requirement for one product. This decision is based on the effort estimation (cost, time, reuse strategy).

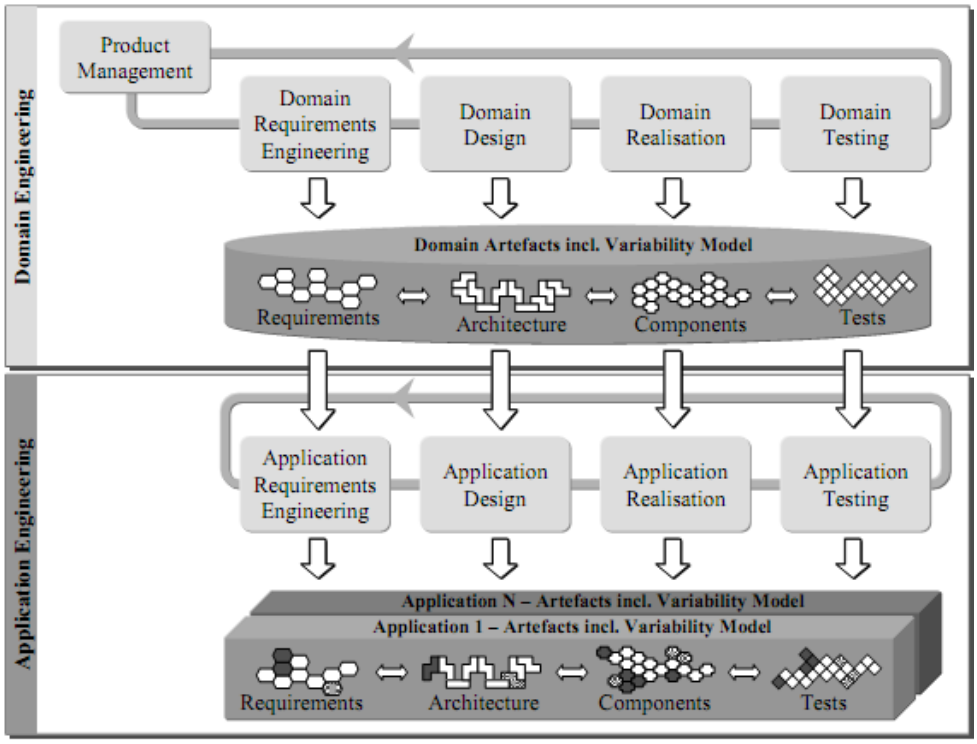

Figure 1 The Software Product Line Engineering process[1]

[1] Figure taken from [39]

2.3 Model-Driven Development

Model-Driven Development (MDD) is an approach for developing software systems that has gained wide acceptance in the last few years. It promotes a new form of building software systems based on the construction and maintenance of models at different levels of abstraction to drive the development process.

Model-Driven Architecture (MDA) [33] is a specific MDD deployment effort, proposed by the Object Management Group (OMG) [36], around industrial standards including MOF, UML, QVT, etc.

One of the bases in the MDA architecture is the Meta-Object Facility standard [38]. This standard proposes a meta-model architecture based on four layers. Considering a model as one abstraction of a real world phenomenon, the meta-model is the abstraction where the model properties are reflected. The MOF architecture includes four layers:

- *Level M3: Meta-metamodeling.* In this layer, the meta-metamodel is used to define other languages.
- *Level M2: Meta-models*. The meta-models are used to describe the models used in the M1 level.
- *Level M1: Model*. The models are defined at this level. For example we could define the UML Class Diagram.
- *Level M0: Instances*. At this level, the real world objects are described.

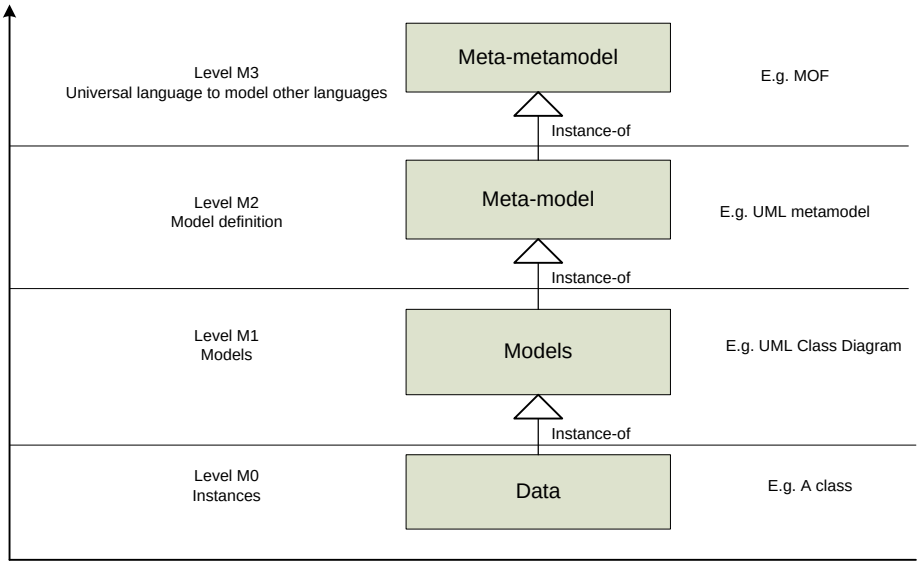

Figure 2 MOF architecture

In a MDD approach, a software system is developed by refining models. A model allows defining the functionality, structure and behavior of a system. The models allow to work in an abstraction level closer to the domain concepts, instead of be centered in platform-oriented concepts as the traditional software development. This refinement is implemented as transformations over models. A *model transformation* is defined by Kleppe et al. [22] as "the automatic generation of a target model from a source model, according to a transformation definition. A *transformation definition* is a set of transformation rules that together describe

how a model in the source language can be transformed into a model in the target language. A *transformation rule* is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language." The transformations are based on rules. This rules links constructions in the source model to constructions in the target model and they are defined at meta-model level (Figure 3).

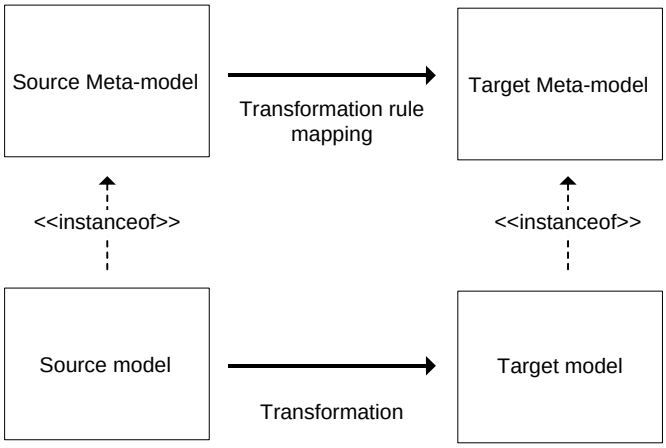

Figure 3 MDA Transformation schema

Several languages are proposed to define model transformations. The OMG proposes the Query/View/Transformation language, called QVT. Several implementation of this standard are proposed: QVT-Operational, QVT-Relations, or QVT-like languages as the ATLAS Transformation Language (ATL) [19].

One related approach is the use of Domain Specific Language (DSL). A DSL is a language proposed to solve a particular problem domain. These kinds of languages are more useful than the generic purpose languages due the fact that they are more closely aligned with the target domain.

3 Evaluation criteria

In this section, we present the evaluation criteria, used in the next chapter, to compare the approaches. The criteria are divided into five main criterions: Software Product Line support, Requirements Engineering treatment, Model-driven coverage, Tool support, and Validation of the proposal. These criteria are refined to consider the specific characteristics of the SPL, RE, MDD areas. The Table 1 shows the criteria refinement. These criteria refinements are discussed in subsections 1 to 5.

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Scoping, Domain Engineering, Application Engineering
	1.2. Which adoption strategy is followed?	Proactive, Reactive, Extractive
	1.3. Which tasks of the Domain Engineering are supported?	Conceptual modeling, Commonality and variability modeling, Feature modeling, Scenario modeling
	1.4. Which tasks of the Application Engineering are supported?	Derivation, Delta identification
	1.5. Which tasks of the Scoping are supported?	Product portfolio scoping, Domain Scoping, Asset scoping
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Elicitation, Modeling, Analysis, Management, Verification and Validation
	2.2. What artifacts are used?	Feature Model, Goals, Use Cases or Scenarios, NFR, Object Models, ER models, Behavioral models, Formal methods, Other.
	2.3. Traceability	Vertical, Horizontal, Both, None
3. Model-driven	3.1. Model representation	Standard model, Non-standard model Template, Structured Natural Language, Natural language, Other
	3.2. Transformation language	ATL, M2M, MOF, QVT, VIATRA…
	3.3. Transformation type	Refactoring, formal refinement, language migration, code generation
	3.4. Degree of automation	Manual, semi-automatic, automatic
4. Tool support		Prototype, tool, industrial tool
5. Validation		Example, Case Study, Academic Study, Industrial Study, None.

Table 1 Evaluation criteria discomposed

3.1 Software Product Lines

The aim of this criterion is to examine the support to the Software Product Line development strategy by the approaches.

3.1.1 SPL activities

In the previous section, we present the two main activities in the SPL development: Domain Engineering, and Application Engineering. For our point of view, we are interesting in two sub-processes from the Domain Engineering activity: Product management, and Domain Requirements Engineering. However, the Product Management sub-process includes aspects from other areas like economics. For this reason, we consider only the part of this sub-process most closer to the RE activity: the Scoping activity. In consequence, we consider three

activities of the SPL process for this study: Scoping, Domain Engineering and Application Engineering. The *Scoping* is the activity concerned with the establishment of the SPL boundaries and the reusability strategy. The *Domain Engineering* aims the development of a requirements specification for the common PL. The *Application Engineering* aims the development of a requirement specification for a single product.

3.1.2 Adoption strategy

This criterion analyses how each technique supports the proactive, extractive, and reactive approaches [21]. In the *Proactive approach*, the organization analyses, designs, and implements a fresh SPL to support the full scope of products needed on the predictable horizon. In the *Reactive approach*, the organization incrementally grows an existing SPL when the demand arises for new products or new requirements on existing products. In the *Extractive approach*, the organization extracts existing products into a single SPL.

3.1.3 Domain Engineering tasks

Conceptual modeling

Activities to identify, define, and organize the concepts that are relevant to the domain and their mutual relationships, in order to facilitate a precise and concise description of the domain.

Commonality and variability modeling

We consider within Commonality and variability modeling the activities to identify similarities and differences between the requirements. This includes the separation of requirements that are valid for the whole domain from those that are only valid in special cases, e.g., for a specific product variant. This activity is strongly related to domain and feature modeling.

Feature modeling

Activities to identify, study, and describe features relevant in a given domain. The aim of feature modeling is to express relations between features, properties of features, and/or superstructures of features. One such essential view is commonality and variability. Others could be feature configuration and interaction. A purpose of feature modeling is to help structure the requirements and define the allowable variants in a product line.

Scenario modeling

This task includes activities to describe and model run-time behavior of members of the system family. This not only includes the functionality of the systems and their interactions with users, but also aspects such as security, safety, reliability, and performance.

3.1.4 Application Engineering tasks

Derivation

The goal of this task is to derive the Application Requirements Model from the Domain Requirements Model. Therefore, we can consider that one approach supports the requirements derivation, if it provides a mechanism to obtain a requirement specification for a single product from the Domain requirements specification.

Delta identification

This task involves the identification of new requirements for one single product. In this case the PL expert can decide to include the new requirement in the PL family or just define it to one single product.

3.1.5 Scoping tasks

According with Schmid [43] there are three levels of scoping: portfolio scoping, domain scoping, and asset scoping. The *Product Portfolio Scoping* aims at identifying the particular products that ought to be developed as well as the features they should provide. The *Domain Scoping* is the task of bounding the domains that are supposed to be relevant to the product line. The *Asset Scoping* aims at identifying the particular (implementation) components that should be developed in a reusable manner.

3.2 Requirement Engineering

The goal of this criterion is to analyze the coverage of the Requirement Engineering activities.

3.2.1 Requirement engineering tasks

We use the classification proposed by Cheng & Atlee [9] categorizing the requirements tasks in: elicitation, modeling, analysis, and management. *Elicitation* refers to the activities performed to be able to understand the goals, objectives, and high-level functions necessary for the proposed software system. The *Analysis* consists of evaluating the quality of requirements. *Modeling* allows requirements to be expressed in terms of one or more models that document the user needs and constraints clearly and precisely. Requirements *verification* is the process of ensuring that the system requirements are complete, correct, consistent, and clear. Requirements *management* is the process of scheduling, coordinating, and documenting the requirements engineering.

3.2.2 Requirement artifacts used

This criterion analyzes the concepts and notations used to identify and model the requirements of the software system to be built. Some works employ *goals, scenarios, or Non-Functional Requirements (NFR)* as a conceptual framework to identify user requirements. The use of *object models, entity-relationship models, or behavioral models* are also alternatives. *Formal methods* are strongly related to models with mathematical foundations.

3.2.3 Traceability

Traceability refers to the ability to follow the life of a requirement either back to its origin or forward to its transformation into a design artifact. We use the classification proposed in Kovačević & Alférez [23] understanding two types of traceability: vertical and horizontal. We consider the vertical traceability as the ability to relate requirements from domain specific requirements to product specific requirements. The horizontal traceability is the ability to relate domain requirements to the domain architecture. It includes the mapping between variation points of these artifacts.

3.3 Model-driven Development

The goal of this criterion is present and compares existing RE approaches that rely on MDD techniques.

3.3.1 Model representation

A requirement can be structured as: model (*standard* model expressed in a language considered standard or *non-standard*). Requirements can also be expressed into natural language or other type of textual or graphical representation.)

3.3.2 Transformation language

The transformations among models are the core of the Model Driven Development. In the last year several specialized languages has been proposed in order to specify model-driven transformations. E.g. The OMG [36] has adopted the QVT specification language, used in the MDA [33] proposal. Other alternative is ATL [19] is a QVT-like model transformation language with an execution environment based on the Eclipse framework.

3.3.3 Transformation type

We use the classification of model transformations provided by Mens et al. [32]. We distinguish two types according the source and target language, *endogenous*, when the source and target model are expressed in the same language and in the same abstraction level; *exogenous*, when different modeling languages and abstractions levels are used to express source and target models. Moreover, we can distinguish depending of the abstraction level between: *horizontal*, where the source and target model reside at the same abstraction level; and *vertical* transformations, where the source and target model reside at different levels.

Table 2 illustrates that the dimensions horizontal versus vertical and endogenous versus exogenous are orthogonal.

Table 2 Orthogonal dimensions of model transformations

	Horizontal	Vertical
Endogenous	Refactoring	Formal refinement
Exogenous	Language migrations	Code generation

3.3.4 Degree of automation

A transformation can be *automatic* if the entire process of obtaining the target model is carried out without the transformation user's participation. The *interactive* needs partial user participation. The *manual* approaches are entirely user dependent.

3.4 Tool support

This criterion analyzes if the approach gives tool support. In many cases, there is an *academic prototype*. Others proposals can give a *mature tool*. Finally, there are many tools validated in an *industrial* environment.

3.5 Validation

We consider five levels of validation for the analyzed studies. From lower to higher: the study is shown through a simple *example*; the study is shown through and *academic case study*; the study is shown with an industrial *case study*; through an empirical *controlled experiment* with a control group; the study has been put into practice in an *industrial case study*.

The case study is a study to examine a phenomenon or unit, collect data, and analyze the results of a single case. An *empirical controlled experiment or study* is a process testing hypotheses against an experiment without the influence of the observer.

4 RE approaches for SPL

In this chapter, we present an analysis of the most important Requirements Engineering approaches that have been proposed to support the development of Software Product Line applications. For each of these methods we present the following information:

- A general description of the method.
- A study of the activities of the SPL proposed by the method. We analyze in which degree are covered Scoping, Domain Engineering or Application Engineering activities.
- A study of the RE techniques and artifacts that are proposed by the method.
- Tool support. We focus on analyze if the approach provides a tool for support the RE specification.
- Evaluation of the approach. We analyze if the approach is validated with case study of controlled experiment. If proceed, we analyze parameters as control group, context (academic, industrial, etc.).

We have not included in this selection studies that does not cover the RE activity explicitly.

4.1 FeatRSEB

Griss et al. '98 [28] propose FeatRSEB. This approach integrates the feature modeling of Feature-Oriented Domain Analysis (FODA) [20] into the process and workproducts of the Reuse-Driven Software Business (RSEB) [15]. The RSEB is a use-case driven systematic reuse process where architecture and reusable subsystems are first described by Use Cases and the transformed into object models that are traceable to these Use Cases. The variability in RSEB is captured by structuring Use Case and Object Models using explicit Variation Point and Variants. The goal of this extension is provide an effective reuse-oriented model as a "catalog" capability to link Uses Cases, Variation Points, reusable components and configured applications.

The process starts building the Use Case for the product family. When this construction starts, then the Feature Model is construed in a concurrent way. The next step is to do a commonality and variability analysis, first in the Use Cases and secondly into the Feature Model. The Feature Model construction is outlined as follows:

The individual exemplar Use Case models are merged into a Domain Use Case model, using Variation Points capture and expressing the differences. A "trace" relation is used to keep trace with the originating exemplars. An initial Feature Model is created with functional features derived from de domain Use Case model. Then a RSEB analysis object model is created to augment the Feature Model with architectural features. These features relate to the system structure and configuration rather than a specific function. Finally the RSEB design model is created to augment the Feature Model with implementation features.

Software Product Line support. FeatRSEB is focused on cover the domain. Two main artifacts are used to describe the requirements for a product line: a Use Case model and a Feature Model. In one hand the Use Case is user oriented (System Engineering). In another hand, the Feature Model is reuser oriented (Domain Engineering). The Use Case model provides the

"what" of the domain (a description of what systems in the domain do); and the Feature Model provides the "which" of the domain (which functionality can be selected when engineering new systems).

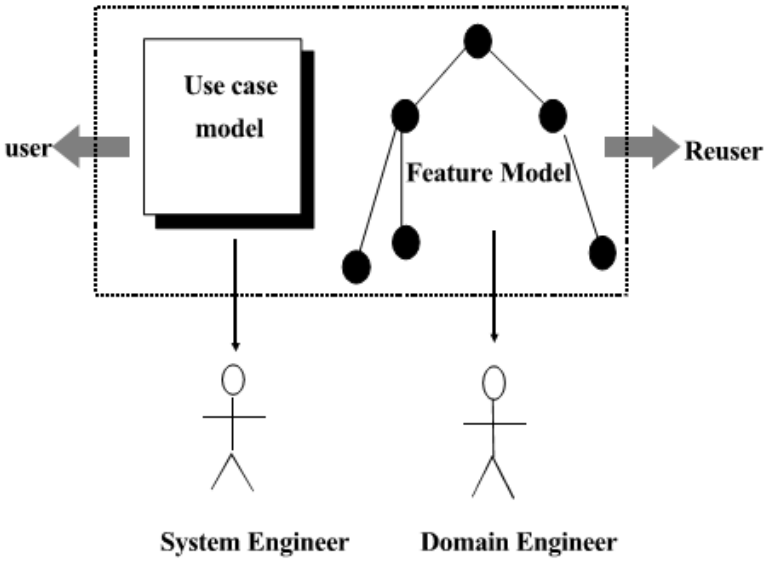

Figure 4 FeatRSEB - The two main models[2]

FeatRSEB supports all develop strategies (proactive, reactive, and extractive). It can be applied to existing SPL or new ones.

The commonality and variability model is supported with the Feature Model. This model contains three types of features:

- *Mandatory*. These features correspond to core capabilities.
- *Optional*. Correspond to capabilities which can be unnecessary in some systems of the domain.
- *Variant*. Correspond to alternative ways to configure a mandatory or an optional feature.

The Scenario Modeling of the SPL is supported with the Domain Use Cases. The model starts with the individual case models. A Domain Actor model is constructed. The exemplar Use Case models are merged, replacing the original actors.

Regarding the scoping, only the A*sset Scoping* is supported with the domain Use Case construction and the extraction of functional features from the domain Use Case model.

Requirements engineering support. FeatRSEB covers the requirement modeling with the Use Cases. The requirement variability is modeled through the Requirement Modeling.

The feature model is represented in UML (see Figure 5). It represents a linked set of feature elements containing data describing attributes of the features. These features are

[2] Image taken from [28]

linked together by a set or relations (UML dependencies or refinements). Some of the feature elements may also have relations (trace) to elements in other models.

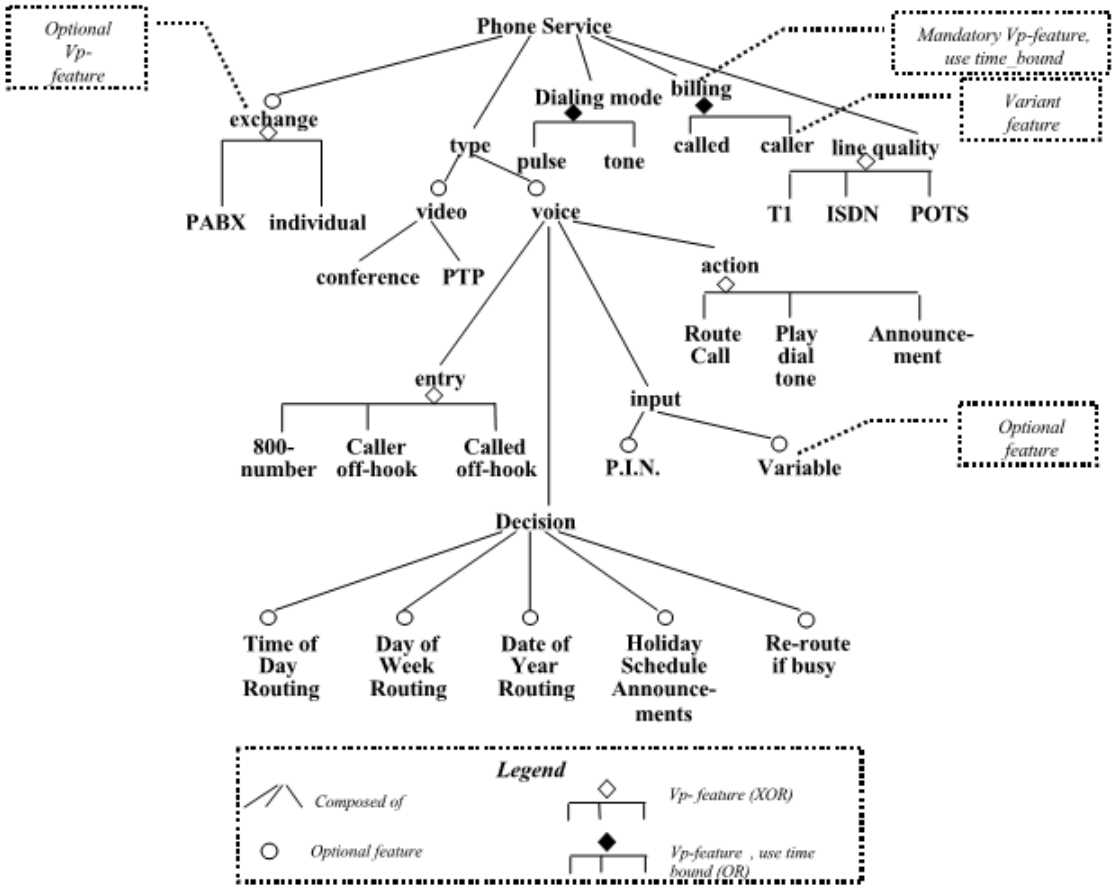

Figure 5 FeatRSEB - Feature Model[3]

The set of features can be specified and structured using the following notation:

- *Conmposed_of relationship*. A feature can be modeled as a composed set of several sub-features.
- *Existence attribute*. A feature can be mandatory or optional.
- Alternative relationship. A feature can act as variation point, where other features are variants (vp-feature).
- *Biding time attribute of vp-features*. At reuse time the vp-featues are a XORed disjunction of their variants. Instead, at use time the vp-feature acts as ORed disjunction of its variants.
- *Requires and Mutual_exclusion constraints*. These rules define semantic constraints on optional and variable features.

Each feature node of the Feature Model is an iconic view of a more complete feature element. FeatRSEB suggest the implementation of a class with a UML stereotype called “feature”·. The Figure 6 shows and expanded view of the Feature Model using a Class Diagram with stereotypes.

[3] Figure taken from [28]

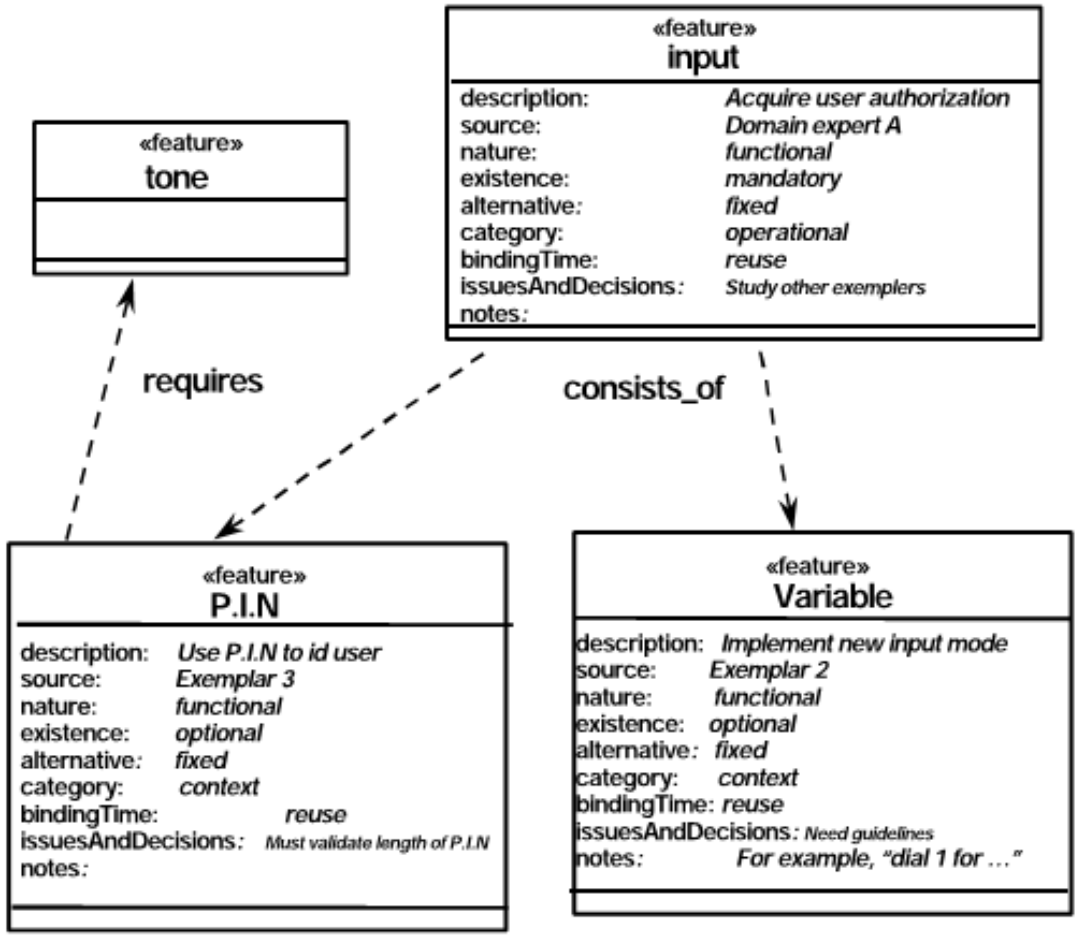

Figure 6 FeatRSEB - Expanded view of the Feature Model[4]

The traceability in FeatRSEB is both horizontal and vertical. The elements from the Domain Use Case have related with the exemplar Domain Use Cases. Moreover the features from the Feature Model has traced to other elements like Use Cases, Variation Points and Objects.

Model-driven coverage. The approach is model-oriented in the sense of that many models are created during the process, however no transformations between models are provided. The Domain Use Case and the Feature Model are based into UML extensions. E.g. a feature is represented in a class diagram with the stereotype "feature".

Tool support. A tool support is not provided. The paper mentions two ways two find and appropriate tool: start from a standard UML tool (e.g. Rational Rose) and add RSEB and FODA extensions using the UML extension mechanism. Another way would be creating or modification existing reuse tool-sets (e.g. ReuseNICE or UML-NICE).

Validation of the approach. The paper uses an example of how was applied the approach into the FODAcom project [21]. However there is any experiment used to validate the approach. For this reason, we consider just a validation by example.

[4] Image taken from [28]

Table 3 Criteria comparison – FeatRSEB

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Scoping, Domain Engineering
	1.2. Which adoption strategy is followed?	All strategies
	1.3. Which tasks of the Domain Engineering are supported?	Commonality and Variability modeling (Feature Model), Feature Modeling, Scenario Modeling (Domain Use Cases).
	1.4. Which tasks of the Application Engineering are supported?	Not supported.
	1.5. Which tasks of the Scoping are supported?	Asset (Use Case and Feature Model).
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Elicitation (partially with Use Cases), Modeling, Analysis, Management
	2.2. What artifacts are used?	Use Case, Feature Model (UML stereotyping), Object Models
	2.3. Traceability	Both.
3. Model-driven	3.1. Model representation	UML Standard (Use Cases), UML extended with stereotypes (Feature Model)
	3.2. Transformation language	Not Supported
	3.3. Transformation type	Not Supported
	3.4. Degree of automation	Not Supported
4. Tool support		Not provided
5. Validation		Example.

4.2 VODRD

Mannion et al. [31] prose VODRD, a method that relies on stakeholder viewpoints to organize user requirements. The VODRD's goal is to improve the requirements reusing. This aim is done with a domain analysis that finds where requirements overlap. This requirements analysis is made through a viewpoint-oriented domain.

VODRD is an iterative method consisting of four steps: scope the domain, characterize the domain, document the viewpoints, and analyze the viewpoints. The outputs from this method are a Domain Dictionary, Domain Viewpoints, and a Catalog of reusable requirements collected within viewpoints. The Figure 7 shows a general overview of the VODRD method with its four steps.

Firstly, the existing user-requirement specification is analyzed to identify the domain stakeholders and establish a Domain Dictionary. The requirements of each domain system are assigned to the appropriate Viewpoint. Next, the requirements are compared to identify which are reusable.

Software Product Line support. VODRD gives support to the Domain Scoping and the Domain Engineering. The Domain Scoping support is given by identifying the stakeholders in the domain. VODRD suggest interviewing staff, and analyzing documents to scope the domain. The output of the process is a set of viewpoints and the reusable requirements. These reusable requirements can be used as input to another SPL process.

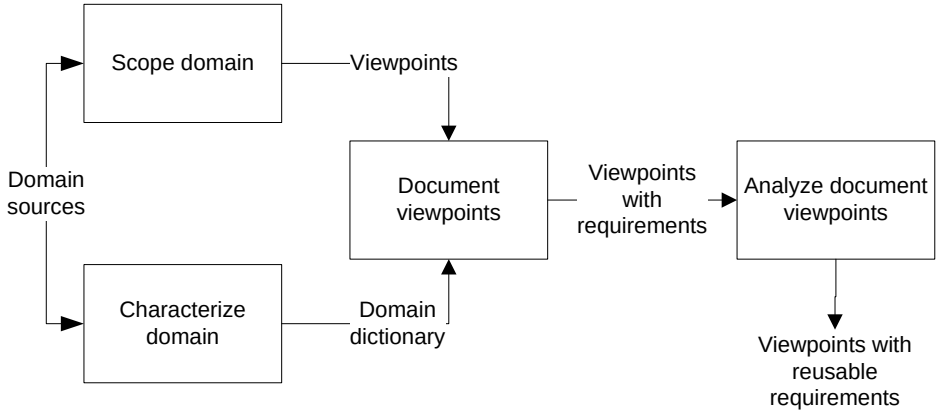

Figure 7 VODRD - The VODRD method process

Requirements engineering support. VODRD supports domain modeling through the analysis of requirements to extract the viewpoints and the reusable requirements. VORD covers conceptual modeling through the domain dictionary and the viewpoint documentation. The work assumes that a textual requirements specification is given, so it based into a reactive or extractive strategy. The Application Engineering is not supported, since it just covers the Domain Engineering. VORD covers partially the Domain scoping, through the identification of the main domain stakeholders and its domain viewpoints. Regarding the RE tasks, the elicitation is supported through staff interviews and previous documentation analysis. The RE modeling is supported by building the textual templates for each requirement. The RE management is covered by defining the associations between requirements. The notations proposed are: domain dictionary, and textual requirement templates. The MDD infrastructure is not supported. The horizontal traceability is supported with the definition of relations between single requirements.

Table 4 VODRD - The Viewpoint template

Viewpoint	Name
Rationale Associations	Justification for inclusion Viewpoint name or set of requirements
Requirement Definition Rationale Associations	Number in document Statement of requirement Justification for inclusion <requirement link>

Model-driven coverage. The approach proposes several models: the Domain Dictionary has textual form, and the Viewpoints are documented with templates (see Table 4). Each viewpoint has its own template.

Regarding the model transformations, there are not mechanisms provided to deal with transformations between the models.

Tool support. The paper claims for a practical tool support to link in an effective way requirements. However, any tool is proposed.

Validation of the approach. The VODRD approach is illustrated through a case study. It models a spacecraft mission control system. The VODRD method was used as part of the European Space Operations Centre's new generation of spacecraft control systems. This case study was a part of the European Space Operations Centre program.

Table 5: Criteria a comparison - VODRD approach

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Scoping, Domain Engineering
	1.2. Which adoption strategy is followed?	Extractive
	1.3. Which tasks of the Domain Engineering are supported?	Conceptual modeling (Domain Dictionary), C&V modeling (Viewpoints analysis)
	1.4. Which tasks of the Application Engineering are supported?	Not supported
	1.5. Which tasks of the Scoping are supported?	Domain scoping (Viewpoints)
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Modeling (viewpoints)
	2.2. What artifacts are used?	Domain Dictionary, Viewpoint, Textual Requirements
	2.3. Traceability	Vertical
3. Model-driven	3.1. Model representation	Textual (Domain Dictionary), Template (Viewpoints)
	3.2. Transformation language	Not supported
	3.3. Transformation type	Not supported
	3.4. Degree of automation	Not supported
4. Tool support		Prototype
5. Validation		Case study

4.3 John and Muthing

John and Muthing [17] describe a method to extend Use Case diagrams and textual Use Cases with explicit commonality and variability. This approach is part of PulSE CDA [6], the domain analysis approach of the PuLSE product line framework.

Software Product Line support. The approach gives support to the Domain Engineering and Application Engineering. The support at the Domain Engineering is given by the Use Case diagrams that model the scenario and the commonality and variability.

In the Use Case diagram, any model element may potentially be a variant in a product line context. An actor is a variant, for example, if a certain user class is not supported by a product. A Use Case is a variant if it is not supported by some products in the family. However, alternative Use Cases are captured outside of the Use Case diagram in a decision model. This is done because such information would overload the use-case diagram, making it less readable, and thus less useful.

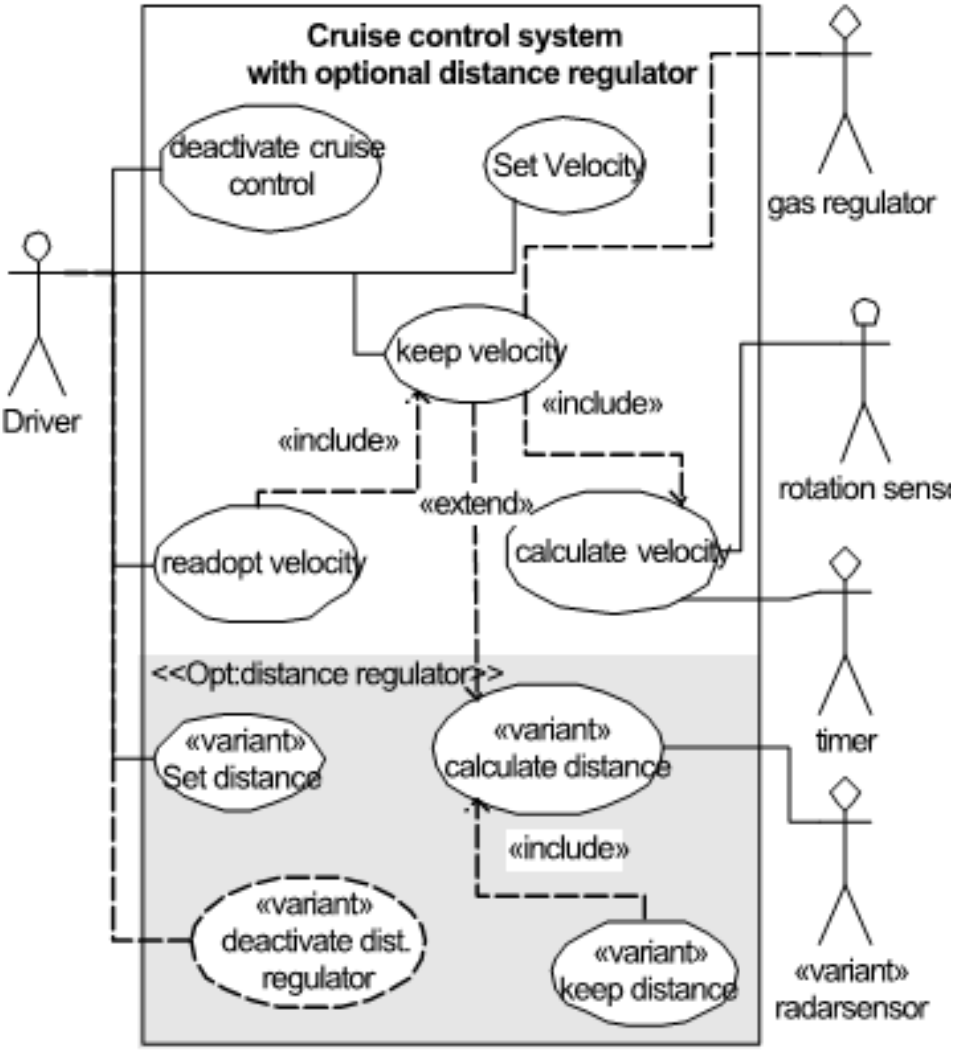

Figure 8 John and Muthing - Use Case Diagram[5]

In the textual case description any text fragment may be variant. A variable text fragment is market with the parts <variant> and </variant>. The Table 6 show and example for the Use Case "keep velocity". The example shows two possible variants: with no regulator – variant ALT1-, and the second with regulator –ALT2-.

The Application engineering is supported with an instantiation process guided by a Decision Model. The Decision Model captures the motivation and interdependencies of variation points. Table 7 shows an excerpt for the case study "cruise control system" [17]).

Requirements engineering support. The requirement modeling is support by the use of the Use Cases technique. The process assumes the Use Cases for the single products and a Use Case for the family is produced. The artifacts used for the modeling are Use Case Diagrams and Textual Use Cases extended to deal with variability, and Decision Models.

The approach gives only vertical traceability. The Decision Model is used to trace Domain Uses Cases to Product Uses Cases. The horizontal traceability is not discussed.

[5] Image taken from [17]

Model-driven coverage. The models build in this approach uses a template-form. There are not automatic transformations provided.

Table 6 John and Muthing - Generic Use Case Description

Use Case Name: keep velocity
Short Description: keep the actual velocity value over gas regulator <variant> by controlling the distance to the cars in front </variant>
Actor: driver, gas regulator
Trigger: actor, driver, <variant> actor distance regulator </variant>
Precondition: --
Input: starting signal, velocity value vtarget
Output: infinit
Postcondition: vactual = vtarget
Success guarantee: vactual = vtarget
Minimal guarantee: The car keeps driving
Main success Scenario:

1. <keep velocity> is selected by actor driver
2. Does a distance regulator exist?
   <variant ALT1: no; only cruise control>
   -compare vactual and vtarget
   If vactual < vtarget: gas regulator increase velocity
   -restart <keep velocity>
   If vactual < vtarget: gas regulator decrease velocity
   -restart <keep velocity>
   else restart <keep velocity>

   </variant>

3. Does a distance regulator exits?
   <variant ALT2: yes; cruise control + distance regulator>
   -- compare vactual and vtarget
   If vactual < vtarget: gas regulator increase velocity
   -restart <keep velocity>
   If vactual < vtarget and atarget & aactual: gas regulator decrease velocity
   -restart <keep velocity>
   If vactual < vtarget and atarget > aactual: gas regulator inrease velocity
   -restart <keep velocity>
   else restart <keep velocity>

   </variant>

Tool support. The paper does not mention tool support.

Validation of the approach. The approach is illustrated with an example called “cruise control system”. The example belongs to the automotive domain. A cruise control system supports the

driver in keeping a constant velocity and does real time monitoring and control of the cars speed.

Table 7 John and Muthing - Partial Decision Model

Variation Point	Decision	Actions
1	The car has no distance regular	Remove Use Case “set distance” from Use Case diagram
		Remove Actor “radar sensor” from Use Case diagram…
		Remove Variant <variant Opt> from Use Case “keep velocity” point 2
		Remove Variant <Alt 2> from Use Case “keep velocity” point 3
	The car has a distance regulator	Remove the variant tag from all uses cases in the Use Case diagram
		Remove the <variant Opt> tag and the </variant> tag from Use Case “keep velocity” point 2…

Table 8 Criteria comparison – John and Muthing

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Scoping, Domain Engineering, Application Engineering
	1.2. Which adoption strategy is followed?	Proactive
	1.3. Which tasks of the Domain Engineering are supported?	C&V modeling (Use Cases with variation points), Scenario Modeling (Use Cases)
	1.4. Which tasks of the Application Engineering are supported?	Derivation
	1.5. Which tasks of the Scoping are supported?	Asset (Identification of core Uses Cases)
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Modeling
	2.2. What artifacts are used?	Textual Use Cases, Decision Model
	2.3. Traceability	Vertical
3. Model-driven	3.1. Model representation	Template (Use Case Diagram, Textual Use Cases, Decision Model)
	3.2. Transformation language	Not supported
	3.3. Transformation type	Not supported
	3.4. Degree of automation	Not supported
4. Tool support		Not supported
5. Validation		Example

4.4 DREAM

Moon ’05 [34] proposes defines a process for developing domain requirements where commonality and variability are explicitly considered. This process is included into a CASE environment called DREAM, which manages the commonality and variability analysis of domain requirements.

DREAM uses a new term called “Primitive Requirement (PR)”. A PR is a transaction that has an effect on an external actor. Its granularity is in between that of a Use Case and an

atomic operation of a Use Case; the purpose of a PR is to make the domain requirements more concrete and to discover the variability and rationale of the domain requirements.

The commonality and variability is defined at the PR level, and that the notion of variation point is explicitly represented in PR-Elements.

The process consists of four major steps:

1. **Scoping domain requirements**. For developers to identify and specify domain requirements in a consistent and precise fashion, the scope of a domain requirement should be first defined. The basic concepts and terms used in the domain are defined in the Domain Terminology. Table 9 shows domain terminology for the news information repository domain.

Table 9 DREAM - Domain Terminology

Term	Description	Related terms
News	• New information about specific and timely events • Text, image, audio, video data service	News information News things
Scrapbook	• A blank book in which miscellaneous items are collected and preserved	
News service	• An organization that gathers news stories and the distributes them to the media or subscribers • Related to NOD system (News On Demand)	New agency
…	…	…

2. Identifying domain requirements using PR. After making an agreement on the scope of the domain requirements, legacy systems are analyzed to extract domain requirements common to them. By considering the common requirements of the legacy systems, a set of similar requirements with variations can be identified. The concept of PR is used as a unit of the identified requirements in this step. A matrix relating PRs and legacy systems is used to identify such similar requirements, by means of context-generalization (grouping legacy systems having the same PRs) and PR- generalization (grouping similar PRs).

   A PR-Context Matrix is introduced to identify all PRs in a domain and to obtain the CV properties for each PR. The PRs are listed in rows and systems build from the domain are arranged in columns. An “O” at the intersection indicates that the PR is found in the System. An “X” indicates that the system does not have the PR.

The initially constructed PR-Context Matrix can be defined by conducting two kinds of generalizations:

- PR Generalization. Two or more PRs with similar functionalities can be generalized into one PR.
- Context Generalization. Two or more legacy systems composed of the same PRs can be generalized into a single named “context”.

The next step is to identify the Common Variability (CV) property for each Pr in the table.

Table 10 DREAM - A PR-Context Matrix for News Information Repository Domain

PR	CV Property / Ratio	MBC	KBS	YTN	ET Times	Chosun Daily
PR1 Login		O	O	O	O	O
PR2 Logout		O	O	O	O	O
PR3 Register		O	O	O	O	O
PR4 Modify member information		O	O	O	O	O
PR5 Add an article to a scrapbook		X	X	X	O	O
PR6 Search a scrapbook		X	X	X	O	X
PR71 Forward an article by e-mail		O	X	X	O	O
PR72 Forward an article by mobile phone		X	O	O	X	X
PR8 Write an opinion		O	O	X	X	O

Refining domain requirements using PRs. Each identified PR is given a detailed description as a form of PR specification. A PR specification is written for each PR with respect to structural and behavioral aspects where variabilities are explicitly specified. In addition, this step identifies and specifies constraints between domain requirements such as: dependency, generalization, and alternative.

Table 11 DREAM - A PR-Context Matrix for News Information Repository Domain – Context Generalization

PR	CV Property / Ratio	BS1(2)	YTN	ET Times	Chosun Daily
PR1 Login	C / 100%	O	O	O	O
PR2 Logout	C / 100%	O	O	O	O
PR3 Register	C / 100%	O	O	O	O
PR4 Modify member information	C / 100%	O	O	O	O
PR5 Add an article to a scrapbook	P / 40%	X	X	O	O
PR6 Search a scrapbook	P / 20%	X	X	O	X
PR71 Forward an article	C / 100%	O	O	O	O
PR71 via e-mail		X	X	O	X
PR72 via mobile phone		X	O	X	X
PR8 Write an opinion	C / 60%	O	X	X	O

3. **Developing a domain Use Case model**. The domain Use Case model is constructed to represent a higher level of abstraction for domain requirements. That is, the domain Use Cases serve as a unit of domain requirements from the viewpoint of application development. Based on the relationship with PRs, expressed in terms of a matrix, the initially identified Use Cases are refactored and categorized.

Table12 DREAM - Excerpt of PR Specification for PR1 Register

PR	PR1. Register			
Description		Variability		
		Type	Cardinality	Variants
Behavior PRelement	PR1a. System verifies customer's real name	External computation	[0..1]	Real name checking service
	PR1b. Customer enters Member Basic Data			
	PR1c. System checks for availability of entered customer's ID and passwords			
	...	...	...	...
Static PRelement	Member Supplementary Data	Data	[1..n]	Occupation Income Education Interest

A Domain Use Case Model is constructed by applying Use Case modeling techniques.

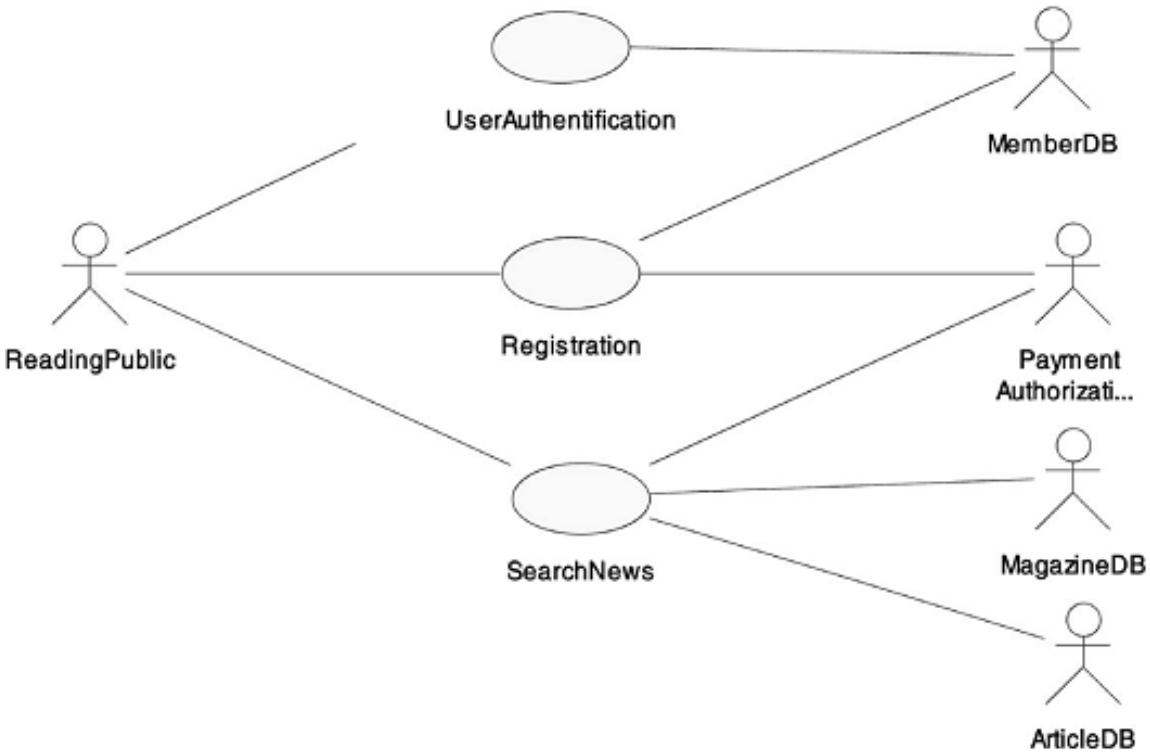

Figure 9 DREAM - Domain Use Case Model initial[6]

Each domain Use Case consists of a smaller unit of functionalities, the PR. The relationship between PRs and domain Use Cases are identified and captured in a PR-Use Case matrix.

[6] Figure taken from [34]

Table 13 DREAM - PR-Usecase matrix

PR	CV Property / Ratio	User Authentification	Registration	Search News
PR1 Login	C / 100%	O		
PR2 Logout	C / 100%	O		
PR3 Register	C / 100%		O	
PR4 Modify member information	C / 100%		O	
PR5 Add an article to a scrapbook	P / 40%			O
PR6 Search a scrapbook	P / 20%			O
PR7 Forward an article	C / 100%			O
PR8 Search a news	C / 100%			O
PR9 Show a news	C/ 100%			O
PR10 Modify an opinion	P/ 42.8%			O
PR12 Delete an opinion	P / 42.8%			O
PR13 Manage payment	C/ 100%		O	O

The PR-Use Case Matrix is used to refactor and categorize the Use Case model. The refinement of the initial Domain Use Case diagram aims to produce more reusable domain requirement by identifying common and variable parts.

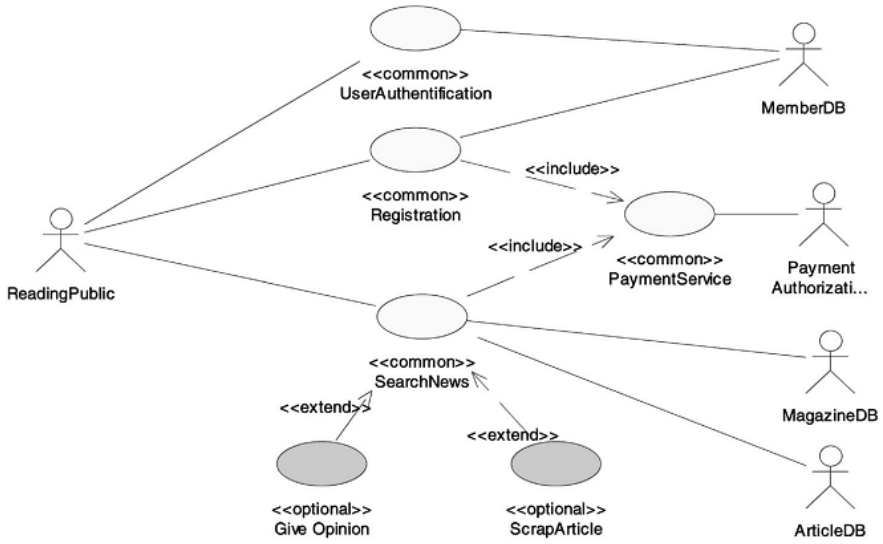

Figure 10 DREAM - Domain Use Case Model refined[7]

Software Product Line support. DREAM is centered in the Domain Engineering. After the application of the process, a Use Case model for the family is produced. The Domain Modeling is included by the production of the Domain Terminology. The commonality and variability is modeled by the PR-context and the PR-Use Case matrixes. The scenario modeling is included with the Domain Use Case diagrams.

[7] Figure taken from [34]

A partial support to the Application Engineering is given by the selection of the application-specific requirements from the domain requirements.

Regarding the scoping, asset scoping is supported by producing core assets for product lines. The domain scoping is supported with the Domain Terminology.

There is not explicit mention to the adoption strategy. We consider that supports partially proactive and extractive support.

Requirements engineering support. The approach covers partially the elicitation with the elaboration of the Domain Terminology and the PR matrix and the modeling of requirements with the Domain Use Case Diagram.

The process uses four different artifacts: Domain Terminology, PR-Context matrix, PR-Use Case matrix, Domain Use Cases. The tree first models are template-based. Only the Use Case diagram uses a standard notation with stereotyping with deal with commonality and variability.

There is not mention to the traceability support.

Model-driven coverage. The approach provides a meta-model for representing domain requirements. Domain requirements are divided into functional and nonfunctional requirements. However, there a no transformations provided for the proposed models.

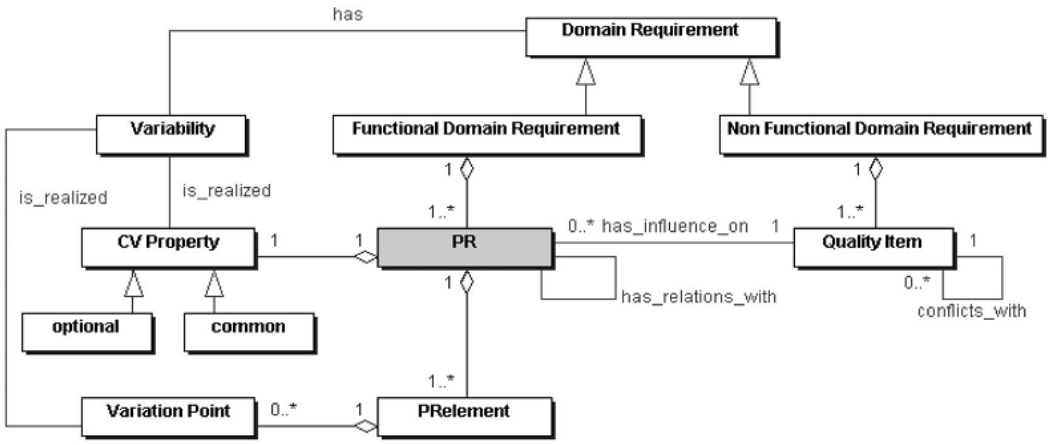

Figure 11 DREAM - Meta-model for domain requirement[8]

Tool support. The paper presents a tool named DREAM (Domain Requirements Asset Manager). DREAM support the management of the commonalities and variability of domain requirements and customizes the requirements of individual systems from these domain requirements. The Domain Use Case modeling relies on external third-party modeling tools, such as such as Rose XDE by IBM [13] or Together Control Center by Borland [7]. DREAM supports the export and import of domain Use Case models to/from XMI files. The use of this format allows the importation of entities from repositories and providing connectivity to other tools.

[8] Figure taken from [34]

Validation of the approach. The process is illustrated with a case study for an e-Travel System domain. The case study was done with the collaboration of the Korean national government and the Daewoo Information Systems Corporation (an IT company in Korea). An e-Travel System is a family of B2B2C travel business applications that provide facilities such e-travel catalogs, online reservations, secure e-payment systems, etc.

Table 14 Criteria comparison – DREAM

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Scoping, Domain Engineering, partially Application Engineering
	1.2. Which adoption strategy is followed?	Proactive and extractive
	1.3. Which tasks of the Domain Engineering are supported?	Domain Modeling (Domain Terminology), C&V (PR-Context and PR-Use Case matrices), Scenario Modeling (PRs and Use Cases)
	1.4. Which tasks of the Application Engineering are supported?	Derivation (selection of Use Cases from Domain Use Case Model)
	1.5. Which tasks of the Scoping are supported?	Domain Scoping (Domain terminology) and Asset Scoping (Identification core Use Cases)
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Elicitation, Modeling
	2.2. What artifacts are used?	Domain Terminology, PR-Context matrix, PR-Use Case matrix, Domain Use Cases.
	2.3. Traceability	Not supported
3. Model-driven	3.1. Model representation	Template (Domain Terminology, PR-Context matrix, PR-Use Case matrix), UML Standard (Use Case Diagram)
	3.2. Transformation language	Not supported
	3.3. Transformation type	Not supported
	3.4. Degree of automation	Not supported
4. Tool support		Tool (DREAM)
5. Validation		Case Study

4.5 PLUS

In Gooma et al. '04 [26] is proposed PLUS. PLUS is a model-driven evolutionary development approach for Software Product Lines based in the Unified Modeling Language (UML) 2.0 notation. The Evolutionary Software Product Line Engineering Process is a highly iterative software process, which consists of two main phases (Figure 12). During Software Product Line Engineering, a product line multiple-view model, product line architecture, and reusable components are developed. During Software Application Engineering, given the features for the individual product line member, the application multiple-view model and architecture are derived.

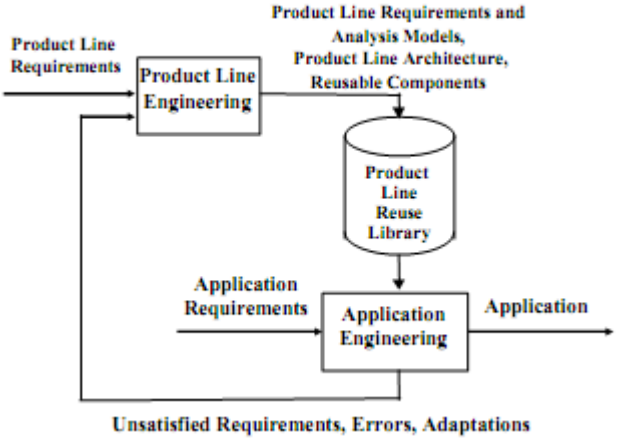

Figure 12 PLUS - Evolutionary Process Model for SPL[9]

This work is focused on the Requirements Engineering activity. It consists of three main activities: *Product Line Scoping, Use Case Modeling, Feature Modeling*.

The process starts with the development of Use Cases during requirements modeling. The traditional Use Cases are extended with “kernel”, “optional” and “alternative” relationship to model commonality and variability. When the Use Cases are written, then the Feature Model is developed to capture the commonality and variability in product line requirements.

Software Product Line support.

There is not support to the scoping activity. The Domain Engineering is supported with the use of Uses Cases and Feature Models. These Use Cases are extended with UML stereotypes to model the commonality and variability. Figure 13 shows an Use Case to model a microwave Software Product Line.

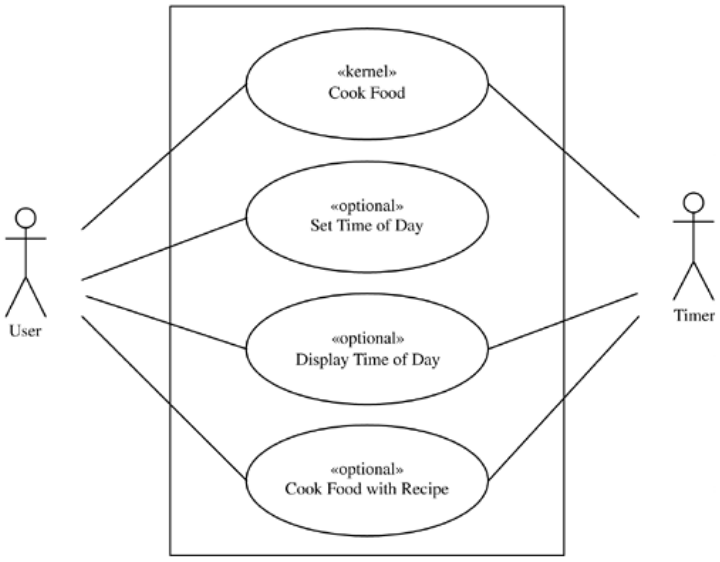

Figure 13 PLUS - Example of Use Case in PLUS[10]

[9] Figure taken from [26]
[10] Figure taken from [26]

The product line commonality is given by the Cook Food kernel Use Case. The Use Case variability is handled with the notion of Variation Point. A Variation Point is a location in a Use Case where a change can take place [15]. The term variation in this context means a situation that is handled differently by different members of the product line. In the example, the Cook Food kernel Use Case models de variability, as well as by the optional Use Cases and the variation points in these optional Use Cases. Thus, part of the Cook Food Use Case description captures product line commonality (the Use Case main sequence and alternatives), and part of it captures product line variability (the description of the variation points). This is different from the optional Use Cases, where the description is entirely of product line variability.

When a Use Case becomes too complex because modeling the alternatives or variation points within the Use Case is very intricate, dependencies between Use Cases can be defined by include and extend relationships. The objective is to maximize the extensibility and reuse of Use Cases. Abstract Use Cases are determined to identify common patterns in several Use Cases, which can then be extracted and reused.

The next step is to capture the common functionality, in this case with the Cook Food kernel Use Case. It is captured through the description of the main sequence and alternatives. The user is the primary actor, and the timer is the secondary actor. The Table 15 shows a textual Use Case for the Use Case Cook Food.

Table 15 PLUS - Use Case Textual Template from PLUS

Use Case name: Cook Food.
Reuse category: Kernel.
Summary: User puts food in oven, and microwave oven cooks food.
Actors: User (primary), Timer (secondary).
Precondition: Microwave oven is idle.
Description:

1. User opens the door, puts food in the oven, and closes the door.
2. User presses the Cooking Time button.
3. System prompts for cooking time.
4. User enters cooking time on the numeric keypad and presses Start.
5. System starts cooking the food.
6. System continually displays the cooking time remaining.
7. Timer elapses and notifies the system.
8. System stops cooking the food and displays the end message.
9. User opens the door, removes the food from the oven, and closes the door.
10. System clears the display.

Alternatives:

Line 1: User presses Start when the door is open. System does not start cooking.
Line 4: User presses Start when the door is closed and the oven is empty. System does not start cooking.
Line 4: User presses Start when the door is closed and the cooking time is equal to zero. System does not start cooking.
Line 6: User opens door during cooking. System stops cooking. User removes food and presses Cancel, or user closes door and presses Start to resume cooking.
Line 6: User presses Cancel. System stops cooking. User may press Start to resume cooking. Alternatively, user may press Cancel again; system then cancels timer and clears display.

Postcondition: Microwave oven has cooked the food.

After the Use Case model, the next step is to address is the feature model and to determine how the Use Cases and Use Case variation points correspond to features. The feature model is developed as a result of a commonality/variability analysis in which the common, optional, and alternative features are determined. The common features identify the common functionality in the product line, as specified by the kernel Use Case; the optional and alternative features represent the variability in the product line as specified by the optional Use Cases and the variation points.

Table 16 shows the relationships between the features and the Use Cases. For example, Microwave Oven Kernel is a common feature determined from the kernel Use Case, Light is an optional feature determined from the Cook Food Use Case; however, it represents a Use Case variation point also called Light.

Table 16 Excerpt of the feature/Use Case dependencies in the microwave oven Software Product Line

Feature	Feature Category	Use Case Name	Name Use Case Category/Variation Point (vp)	Variation Point Name
Microwave Oven Kernel	common	Cook Food	kernel	
Light	optional	Cook Food	vp	Light
Turntable	optional	Cook Food	vp	Turntable
Boolean Weight	default	Cook Food	vp	Weight Sensor
Analog Weight	alternative	Cook Food	vp	Weight Sensor
Power Level	optional	Cook Food	vp	Power Level
12/24 Hour Clock	parameterized	Set Time of Day	vp	12/24 Hour Clock

The Feature Model is used to model variability. However, the Use Cases are used to determine the functionality of the system and the Feature Model is oriented to the reuse. A functional feature can be modeled as a group of Use Cases that are reused together. When a group of Use Cases is always reused together, they can be mapped to a feature and depicted as a Use Case package. These features can be functional features or parameterized features. A feature can correspond to a single Use Case, a group of Use Cases, or a variation point within a Use Case.

The application engineering is supported partially. Firstly, given the product line Feature Model, the features for selected for the application are selected. Based on this selection, the Use Cases related with the features are selected from the Domain Use Case Model. However, identification for Deltas is not considered in this approach.

This works does not consider the scoping activity.

Requirements engineering support. The approach gives a partial support to the requirement elicitation with the Use Case technique. The Use Cases captures the functional requirements of the software product family, including commonality and variability. Moreover this technique is used to model the functional requirements.

The approach proposed build three artifacts: Use Case Diagram, Textual Use Cases and Feature Model. The variability is represented in all of them.

The traceability supported is just vertical. The set of select features for a product are related with the Domain Feature Model. However this model is not related towards the SPL architecture.

Table 17 Criteria comparison – PLUS

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Doman Engineering
	1.2. Which adoption strategy is followed?	All strategies
	1.3. Which tasks of the Domain Engineering are supported?	C&V modeling (Use Cases, Feature Modeling), Feature Modeling, Scenario modeling (Use Cases)
	1.4. Which tasks of the Application Engineering are supported?	Derivation
	1.5. Which tasks of the Scoping are supported?	Not supported
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Elicitation (Use Cases), Modeling (Use Cases, Feature Model)
	2.2. What artifacts are used?	Use Case Diagrams, Textual Use Case Templates, Feature Model
	2.3. Traceability	Vertical
3. Model-driven	3.1. Model representation	UML extended (Use Cases, Textual Use Case Templates) Tabular (Feature Model)
	3.2. Transformation language	Not supported
	3.3. Transformation type	Not supported
	3.4. Degree of automation	Not supported
4. Tool support		Not supported
5. Validation		Case Study

Model-driven coverage. This approaches proposed to build some models, however there is not support to model transformations. Regarding the proposed models, PLUS uses the UML stereotypes to extend the Use Case Model with variability. For Domain Use Cases, the stereotypes «kernel», «optional», and «alternative» are used, respectively, to distinguish among Use Cases that are always required, Use Cases that are sometimes required, and Use Cases in which a choice must be made. The Feature Model is represented

Tool support. The approach does not mention any specific tool.

Validation of the approach. The approach is validated with several academic case studies. The book [26] shows three cases studies: a Microwave Oven SPL, an Electronic Commerce SPL, and a Factory Automation SPL.

4.6 PLUSS

Eriksson et al. ’05 propose PLUSS [11], an approach to manage natural-language requirements specifications in a Software Product Line. PLUSS is based on the work by Griss et al. on FeatuRSEB [28]. PLUSS utilizes a Feature Model to manage variability among the textual requirements at the Domain Engineering level. However, instead of used the Feature Model as

4+1 view, the FM as a tool for structuring Use Cases into reusable packages for a system family. This vision allows use the product instantiation, defining views/filters showing only those requirements that are relevant for a specific product.

Software Product Line support. PLUSS enforces a common and complete requirements specification for an entire product line. The scoping activity is not mentioned in this approach. However the Domain Engineering and the Application Engineering activities are covered. Regarding the adoption strategy, PLUSS supports the pro-active and extractive strategies.

In the Domain Engineering activity, PLUSS proposes model the variability and commonality with a Feature Model. This model moreover implies to cover the Feature Modeling. Finality the Scenario Modeling is supported performing the Use Case technique.

Regarding the variability and commonality, a FODA [20] feature model extension is proposed PLUSS feature models provide an "at-least-one-must-be-selected" relation called ''multiple adaptor features". Furthermore, the FODA's alternative features to are renamed to "single adaptor features". Single and multiple adaptor features are represented by the letters 'S' and 'M', respectively, surrounded by a circle as shown in Figure 14 .

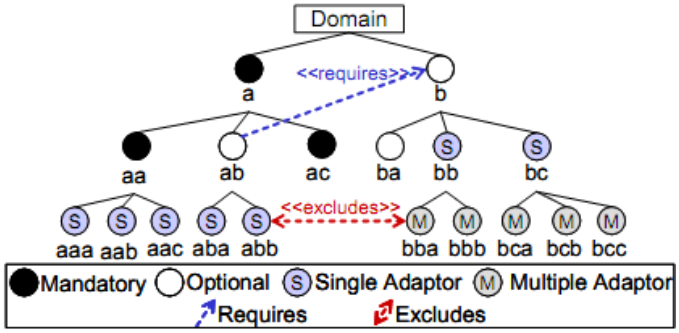

Figure 14 PLUSS - A Feature model example in the PLUSS notation[11]

A set of features in the FM can compose a Use Case package. This fact allows visualizing variants within Use Case specifications using the FM. The Figure 15 shows how uses cases can be mapped to features of any type to capture required variants among the members of a system family.

The Use Cases are written in a RUP-SE "flow of events notation" [40] . This notation is used for tabular descriptions of Use Case scenarios in natural language. Table 18 shows the use of this notation to describe a Use Case Scenario.

Table 18 PLUSS - Example of Use Case Scenario

Step	Actor Action	Blackbox System Response	Blackbox Budgeted Req.
1	The Actor…	The System…	It shall…
2	…	…	…
3	The Use Case end when…	…	…

[11] Figure taken from [11]

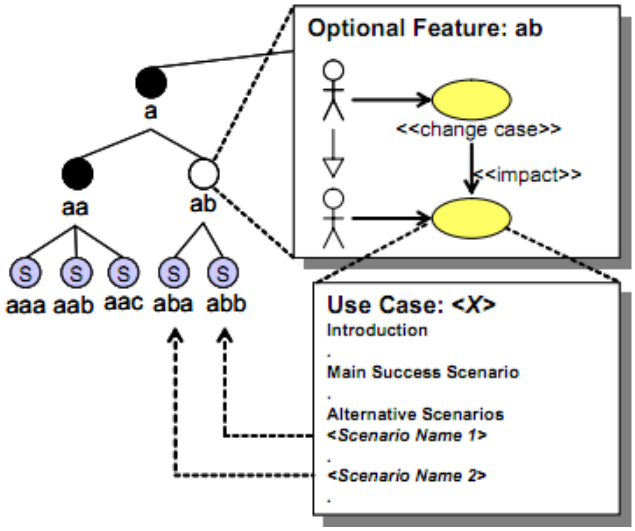

Figure 15 PLUSS - An example of the Relationship between Features,

A Use Case realization describes how a particular Use Case is realized within the system design in terms of collaborating design elements. PLUSS choses describe Use Case realizations in natural language description based on the RUP-SE "White Box Flow of Events" [40].

Whitebox Action	Whitebox Budgeted Req.
DesignElement_1...	It shall...
DesignElement_2...	...
DesignElement_3...	...
...	...
...	...
...	...
...	...
...	...
...	...

Figure 16 PLUSS - Example of Use Case Realization

PLUSS use the concept of Use Case Parameter, introduced in Jacobson et al. [15]. Moreover, Mannion et al. [31] distinguished between local parameters and global parameters in their work on reusable natural language requirements. In PLUSS the scope of a local parameter (denoted by "$") is the Use Case which contains it; and the scope of a global parameter (denoted by "@") is the whole domain model. Figure 17 shows an example of the use of parameters in variation points.

PLUSS already support the Application Engineering. When a new product is going to be added to a product family, initial requirements analysis is performed. The result of this analysis is a set of Change Requests to be added to the Domain Model and regarding new features. Then the Domain Engineering Team is responsible to perform a change impact analysis to decide if the requested set of requirements will be allowed in the product. Since a common Use Case model is maintaining for a whole product family in PLUSS, product instantiation is then basically done by adding any new requirements to the model and then using the feature model to choose among its variants.

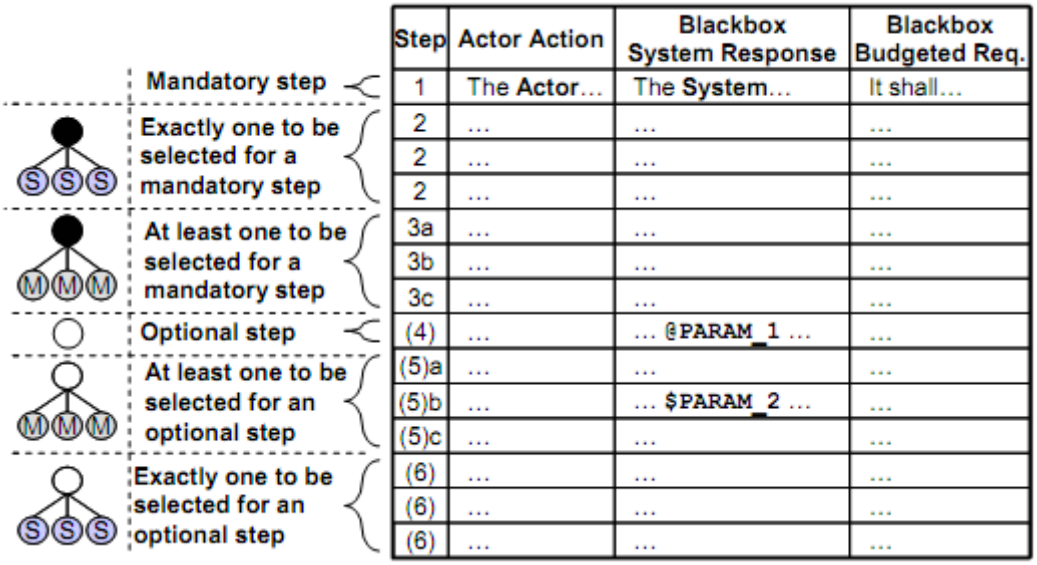

Figure 17 PLUSS - An example of the Relationship between Features

Requirements engineering support. PLUSS support the partially elicitation and modeling of functional requirements with the Use Case technique. According with the authors, non-functional requirements can be related to Use Cases using the "Blackbox Budgeted Requirements" column in the Use Case Scenario description.

PLUSS supports vertical traceability. The general principle for traceability in PLUSS is that traceability links are only maintained for the common model, and never between generated product instances of the model and other specifications. Traceability information for product instances of the product line model is delivered as separate reports together with the requirements documents.

Model-driven coverage. PLUSS proposed to build many models. The Feature Model is an extension of the *de facto* standard FODA notation [20]. The Use Case Scenario and Use Case Realization are written in natural language using a template based in the RUP-SE "flow of events notation" [40] .

There are not provided transformations between models. However, a meta-model for integration of features, Use Cases and Use Case realizations is proposed (Figure 18). It describes how Use Cases, scenarios and scenario steps are included by feature selections.

Tool support. PLUSS proposed an extension of two commercial tools: the requirements management tool Teleogic DOORS and the UML modeling tool IBM-Rational Rose. The Telelogic DOORS is used to manage the system family Use Case models, and the IBM-Rational Rose is used for drawing feature graphs and UML diagrams. Both tools are widely used and accepted in industry.

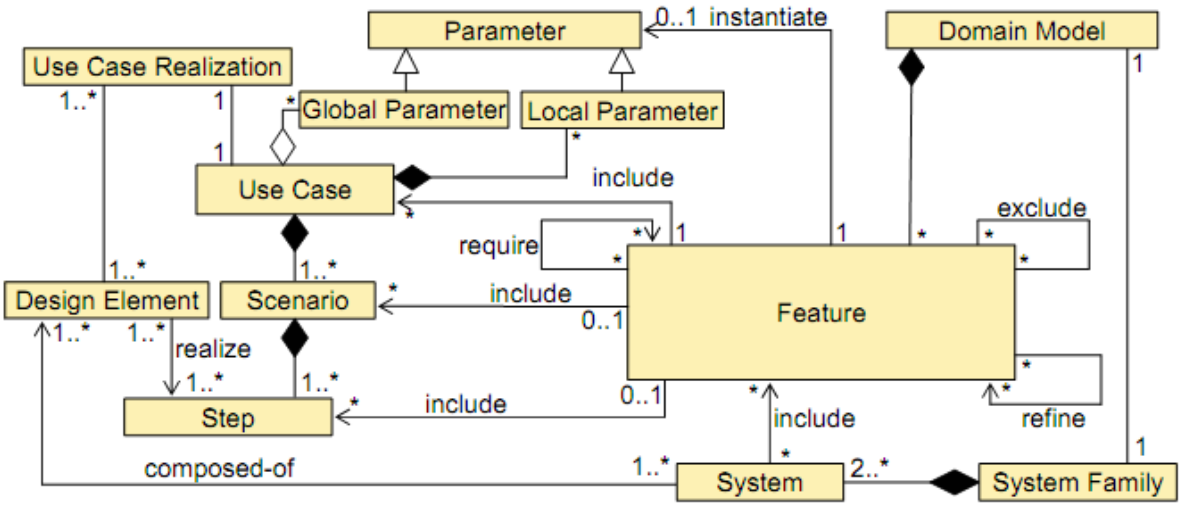

Figure 18 PLUSS - The PLUSS Meta-model[12]

Validation of the approach. According with the authors, PLUSS was applied and evaluated in an industrial case study based on two product lines in the defense system domain. The study states that PLUSS performs better than clone-and-own reuse of requirement specifications in the considered industrial contexts.

Table 19 Criteria comparison – PLUSS

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Doman Engineering, Application Engineering
	1.2. Which adoption strategy is followed?	Proactive and Reactive
	1.3. Which tasks of the Domain Engineering are supported?	C & V modeling (Feature Model), Feature Modeling (Feature Model), Scenario Modeling (Use Cases)
	1.4. Which tasks of the Application Engineering are supported?	Delta analysis (Change Case Impact Analysis)
	1.5. Which tasks of the Scoping are supported?	Not supported
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Elicitation (Use Cases), Modeling (Use Cases)
	2.2. What artifacts are used?	Feature Model, Textual Use Cases, Textual Use Case Realizations
	2.3. Traceability	Vertical
3. Model-driven	3.1. Model representation	Extension De facto Standard (Feature Model), Template (Use Case Scenario, Use Case Realization)
	3.2. Transformation language	Not provided
	3.3. Transformation type	Not provided
	3.4. Degree of automation	Not provided
4. Tool support		Industrial supported
5. Validation		Industrial Case Study

[12] Figure taken from [11]

4.7 Orthogonal Variability Model

Klaus Phol *et al.* '05 [39] propose a method for domain requirement engineering and application engineering. Both processes rely on the Orthogonal Variability Model (OVM), which represents variability apart from requirement artifacts, differentiates between variation points, variants, and constraints among these entities, and explicitly define the variability of the product line.

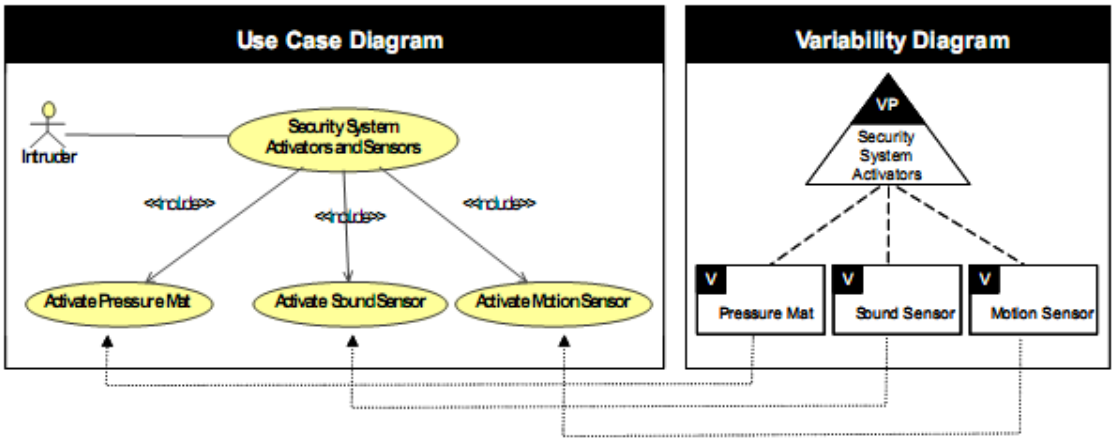

Figure 19 OVM - The Orthogonal variability model[13]

Software Product Line support. The OVM approach covers the Scoping, Domain Engineering an Application Engineering activities. Moreover all adoption strategies are supported.

OVM supports Product Portfolio, Domain Scoping and Asset Scoping. Portfolio Analysis allows a systematic evaluation of the Product Portfolio. During the analysis, each product (or product type) is rated according to two variables and thereby its location in a two-dimensional matrix is determined. On example of the Product Portfolio matrix is the Boston Consulting Group [46]. The asset scoping and domain scoping are accomplished with the commonality and variability analysis.

The method for domain requirement engineering starts by first identifying common requirements. These requirements are identified using an Application-Requirements matrix (see Table 20 for an example). This matrix relates the requirements with the application in which they occur. A requirement in row x of the matrix is common if, and only if, it appears in all columns of the matrix, that is, is present in all applications. In the example, the requirement R1 is mandatory for all applications and is thus a candidate to be defined as a common product line requirement.

Table 20 OVM - Example of the structure of an Application–Requirements Matrix

Application Requirements	App. 1	App. 2	App. 3	App. 4
R1	Mandatory	Mandatory	Mandatory	Mandatory
R2	-	-	Mandatory	Mandatory
R3	-	Mandatory	-	-
…	…	…	…	…

[13] Image taken from [39]

The next step is to perform the Variability Analysis. The variability analysis has the goal to identify requirements variability and to define the variation points and their variants related to these requirements. In a similar way to identify the common requirements, the Application-Requirements matrix is used to identify requirements shared among a subset of applications (examining different columns of the same row) or requirements differing among themselves (examining different rows of the first column of such matrix). This leads to identification of variants and variation points and to relating requirement artifacts to variants. Constraints among these are also defined.

In Requirement Application Engineering, stakeholder requirements are elicited and mapped to common and variable artifacts. If the stakeholder requirements for the application cannot be satisfied by reusing common or binding variable domain requirement artifacts, application-specific requirement artifacts may be introduced. The difference between application-specific and domain requirements, the so-called deltas, are then taken into account to decide whether realization in the application is to be performed or not.

Requirements engineering support. OVM covers the elicitation, modeling and management of requirements. The approach does not provide a particular way to perform requirements modeling due to the orthogonal nature of OVM.

OVM inherently supports both horizontal and vertical traceability.

Model-driven coverage. OVM proposes to build the Application-requirements matrix and the OVM.

There is not support to automatic transformations. However, the approach is inspired in the metamodel proposed by Bachman et al. [5]. The metamodel is represented using UML 2.0.

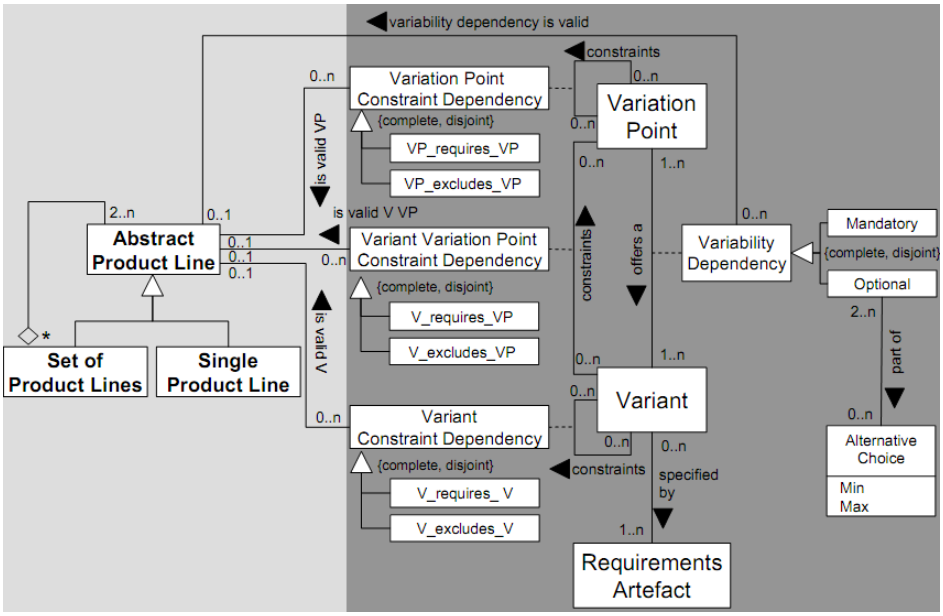

Figure 20 OVM - Orthogonal Variability Metamodel

Tool support. In terms of tool support, prototype support is available as extension to DOORS, providing features such as determining overlaps and differences of the variability of two product lines, retrieve all product lines offering a certain variant, retrieve all variants common for all product lines, retrieve all variants defined for a given variation point.

Validation of the approach. The paper doesn't mention any validation of the approach. However the OVM was applied to several industrial examples. Consequently, we consider that the approach is validated with an example.

Table 21 Criteria comparison – OVM

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Scoping, Domain Engineering, Application Engineering
	1.2. Which adoption strategy is followed?	All strategies
	1.3. Which tasks of the Domain Engineering are supported?	C & V (Application-requirements matrix), Feature Model (OVM), Scenario Modeling (specified by rel.)
	1.4. Which tasks of the Application Engineering are supported?	Delta analysis
	1.5. Which tasks of the Scoping are supported?	Domain Scoping (C&V analysis), Product Portfolio (Portfolio analysis), Asset Scoping (C&V analysis)
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Elicitation, Modeling, Management
	2.2. What artifacts are used?	Application-requirements matrix, OVM
	2.3. Traceability	Both
3. Model-driven	3.1. Model representation	Template (Application-requirements matrix), Non-standard (OVM)
	3.2. Transformation language	Not supported
	3.3. Transformation type	Not supported
	3.4. Degree of automation	Not supported
4. Tool support		Prototype
5. Validation		Example

4.8 NAPLES

NAPLES (Natural language Aspect-based Product Line Engineering of Systems) is proposed by Loughran et al. [30]. NAPLES is a product line engineering approach that uses natural language processing and aspect-oriented techniques to facilitate requirements analysis, commonality and variability analysis, concern identification to derive suitable feature oriented models for implementation. The NAPLES approach addresses product line (PL) engineering throughout the lifecycle by using different techniques, e.g., natural language processing (NLP) and aspect-oriented software development (AOSD), to provide automated support and separation of concerns during the PL lifecycle.

The approach starts with the *Mining Elements* activity which identifies important concepts (e.g., early aspects, viewpoints, commonalities and variabilities) from the requirements documents used as input, and presents them to the user in a format that can be

used to produce a structured model (AORE model and feature model). The EA-Miner [41] tool uses the WMATRIX [42] natural language processor to pre-process the input documents and get relevant information. WMATRIX provides part-of-speech and semantic tagging, frequency analysis and concordances to identify concepts of potential significance in the domain. Part-of-speech analysis automates the extraction of syntactic categories from the text (e.g., nouns and verbs).

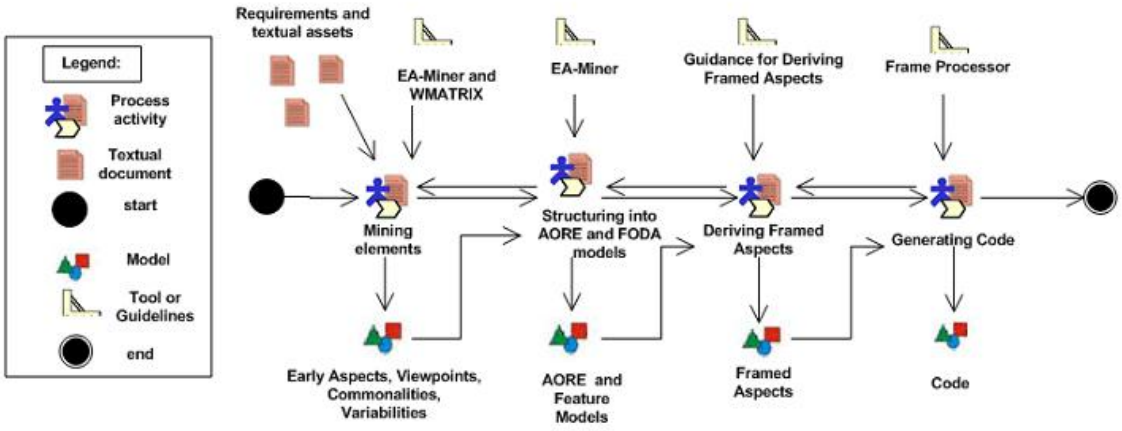

Figure 21 NAPLES approach[14]

The information produced by the NLP processor is then used by EA-Miner to help list possible key domain concept candidates. For example, for the identification of viewpoints, the tool lists the most frequently occurring nouns in the text, and for Early Aspects it lists words whose meaning resembles a broadly scoped concern (e.g., security, performance, parallel, logon, authorize, and so forth). Commonalities and variabilities are also identified in a similar fashion.

After the software developer has identified and selected the concepts of interest in the previous activity, EA-Miner helps to build structured models during *the Structuring into Models* activity. The tool enables the application of screen out functionalities (e.g., add, remove, check synonyms) to discard irrelevant concepts, add new ones and check if the same concepts are identified as different ones. The output is an AORE model showing the viewpoints, early aspects and composition rules as well as a feature model showing features alongside their commonalities and variabilities.

The *Deriving Framed Aspects* activity uses the previous models (AORE and feature model) and provides guidance on how to delineate an aspect-oriented model based on framed aspects. The framed classes and aspects are then used by the frame processor in the Generating code activity to create the code in a specific language.

Software Product Line support. This approach covers the Scoping and the Domain Engineering SPL activities. Regarding the adoption strategy, the proactive is supported. However, the authors discuss that maybe the approach could be effective with other strategies.

[14] Image taken from [30]

The asset scoping is supported with the Mining Element Activity. The original requirement documents, user manuals, and legacy documentation are processed with the EA-Miner tool in order to identify commonalities and variabilities.

The Requirement Domain Engineering is supported. The commonality and variability modeling is supported and is based on a lexicon of relevant domain concepts. The Feature Modeling is supported with the *Structuring into models* activity.

The application engineering is not supported in this approach.

Requirements engineering support. The requirement modeling is supported with the use of Viewpoints.

The approach support horizontal traceability from the requirements to their implementation.

Model-driven coverage. During the Structuring into Models activity two models are used: AORE model that contains the viewpoints, early aspects and composition rules, and the Feature Model, which shows the features alongside their commonalities and variabilities.

Tool support. Tool support is provided with the EA-miner tool. The EA-Miner tool helps the user to identify variabilities by providing the surrounding text in which the word occurs. In Figure 22, after the user selects the "contacts" commonality, the right-hand side shows in which sentences (sentences 7 and 8) of the document the word appears. The rules of thumb for identifying commonalities and variabilities are [34]:

- The tool lists the commonalities on the left-hand side and the user searches for possible variabilities by looking at the surrounding context of a specific commonality (e.g., contacts);
- The user looks at the details on the right-hand side and identifies concepts that modify the commonality in some way (e.g., the size of the list of contacts is variable depending on the model).

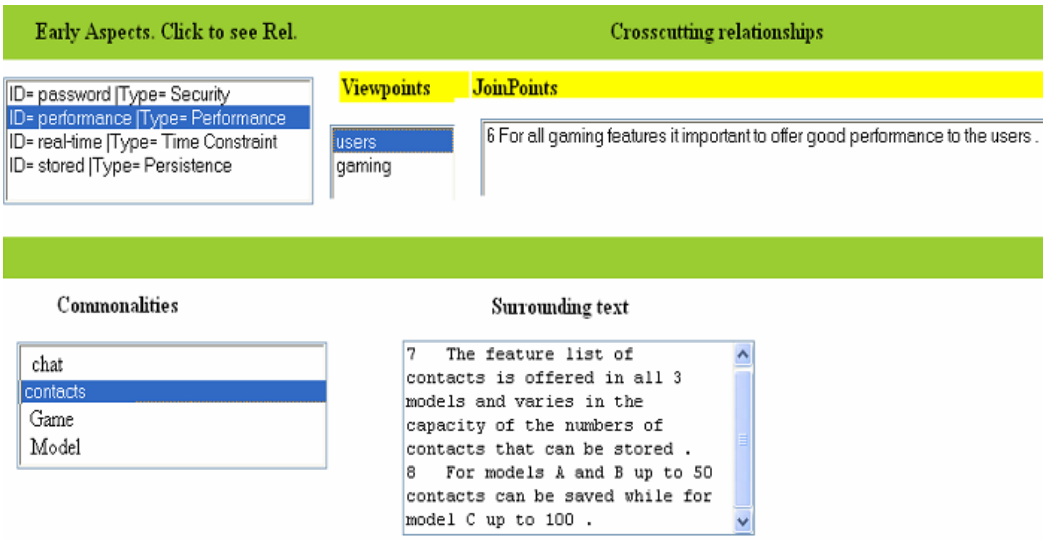

Figure 22 NAPLES - EA-Miner tool

The authors consider that the tool support would be very helpful in order to mine key concepts that will aid the construction of assets for the merged product line.

Validation of the approach. The approach feasibility is show through an example for a product line of mobile phone.

Table 22 Criteria comparison – NAPLES

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Scoping, Domain Engineering
	1.2. Which adoption strategy is followed?	Proactive
	1.3. Which tasks of the Domain Engineering are supported?	C & V (comparison), Feature Modeling(Structuring into models)
	1.4. Which tasks of the Application Engineering are supported?	Not supported
	1.5. Which tasks of the Scoping are supported?	Asset Scoping (Mining Element)
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Modeling (Viewpoints)
	2.2. What artifacts are used?	AOR model, Feature Model
	2.3. Traceability	Horizontal
3. Model-driven	3.1. Model representation	Non-standard (AOR model), Extension of the facto standard (Feature Model)
	3.2. Transformation language	Not supported
	3.3. Transformation type	Not supported
	3.4. Degree of automation	Not supported
4. Tool support		Tool provided
5. Validation		Example

4.9 Mauricio Alférez

Mauricio Alférez et al. ’08 [1] proposes a model-driven approach to model, specify and trace SPL features and requirements. The approach includes domain and application engineering activities.

The approach proposes, at the domain analysis level, a set of activities executed iteratively and incrementally. The actives are:

1. *Identify requirements*. Traditional techniques can be used as inspection of existing documents, interviews or mining techniques.
2. *Group requirements into features*. The SPL requirements are organized into clusters according to the specific SPL features they are related to.
3. *Refactor requirements and features*. The requirements are refactored in order that one requirement points to only one feature.
4. *Model SPL features and Use Cases*. The requirements are structured and represented using Use Case and feature models.

5. *Relate features to uses cases*. The relationships between features and Use Cases are specified visually in a table of trace links.
6. *Generate SPL Use Cases annotated with features*. A model-driven tool [4] uses the relationships between uses cases and features to generate specific Use Case models annotated with features.
7. *Model uses cases as activity diagrams*. A set of Activity Diagrams is built to represent the detailed behavior of each Use Case.
8. *Specify composition rules between uses cases*. Each composition rule defines how a variable Use Case can interfere or modify the normal execution of a mandatory Use Case.

The models produced in the Domain Engineering are used in the Application Engineering to generate Use Case and Activity Models for specific products. Three activities are defined:

1. Define a SPL configuration. A SPL configuration is specified based on the optional and alternative feature selection.
2. Generate a Use Case model from a SPL configuration. The tool generates the Use Case model related to the SPL configuration.
3. Generate activity diagrams from a SPL configuration. The activity diagrams are generated with the tool assistance. The original activity diagrams can be composed using the composition rules defined in the Domain Engineering stage.

Software Product Line support. This approach supports the Domain and Application Engineering activities. It can be used with a proactive strategy with a fresh SPL or; alternatively with an extractive strategy with existing products.

The Domain Engineering covers the commonality and variability modeling with a Feature Model. This model is used to do a feature modeling. Finally, a Scenario modeling is covered with the creation of Use Cases and activity diagrams.

The Application Engineering is supported in this approach. The first activity is to specify a SPL configuration to decide which features will be part of the final application. Based on this configuration, a Use Case model is automatically derived. Finally, the activity diagrams are customized with the composition rules.

Finally, the approach does not mention the scoping activity.

Requirement engineering support. This approach covers the elicitation, modeling and management of requirements. This works suggest the use of traditional technique to elicit requirements. The modeling is covered with Use Cases and Activity Diagrams for the functional requirements and the Feature Model to the commonality and variability. The management is covered by the explicit traceability management.

The approach proposed to build the following artifacts: Feature Model, Use Cases, Activity Diagrams, and a table of Trace links.

The approach gives an explicit support to the requirements traceability. The strategy is based in trace relationships between features and UML elements. The vertical traceability is covered with the derivation of one Domain Specification for a given configuration. This

traceability is supported by a metamodeling strategy. The Figure 23 shows the relations between the metaclasses used to support this strategy.

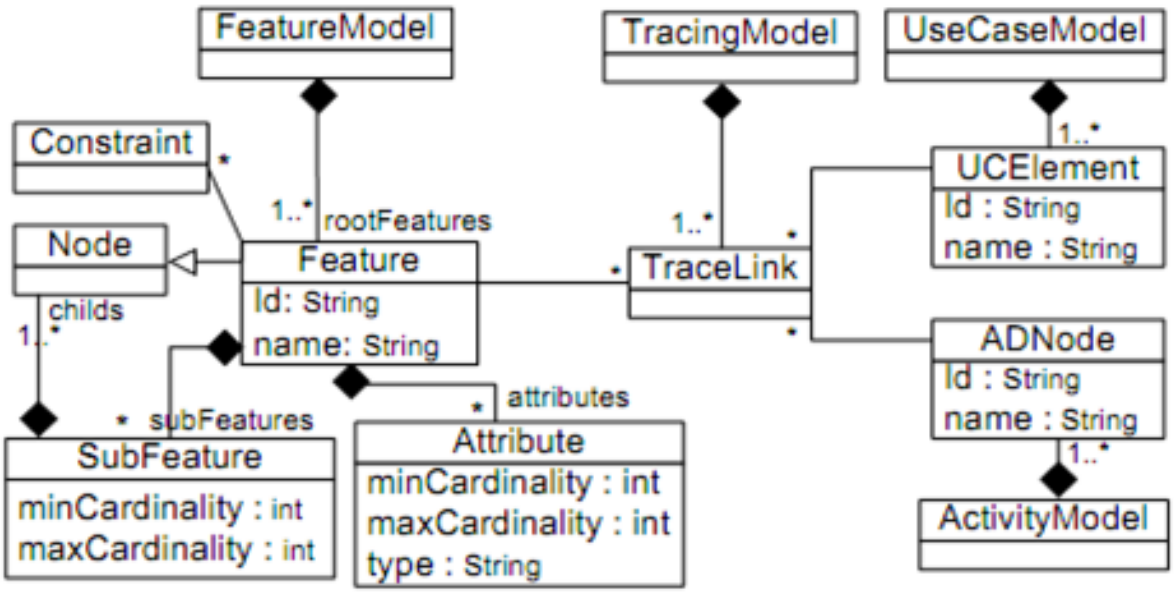

Figure 23 Mauricio Alférez - Traceability support strategy[15]

Model-driven coverage. The approach proposes the elaboration of several models: Feature Model, Use Case Model, Activity Diagram, and one textual model: the Composition Rules. The approach adopts the feature diagram based on [8] as variability model. UML Use Case and activity models specify the SPL requirements. Activity diagrams model the behavior of Use Cases.

The transformations are endogenous and vertical. These transformations are used to obtain the requirements specification for a single product.

Tool support. The paper does not mention a specific tool. However, in the AMPLE website [4], we can find the VML4RE tool [1]. VML4RE is a Domain-Specific Language (DSL) that allows to express the relationships between variability elements and requirements model elements, such as Use Case and activity models. VML4RE runs in the Eclipse Environment and was implemented using technologies provided by open Architecture Ware (oAW) [12], more specifically to implement the different operations or actions to be performed in requirements models, using the XTend transformation language [12]. The goal of VML4RE is to support product derivation of requirements models in SPL by using a domain-specific language. It also supports trace link generation from features to requirements model elements, for further analysis.

Validation of the approach. The approach is illustrated with a Smart Home SPL case study. This system is one of the SPL case studies proposed by the industrial partners of the European project AMPLE [4].

[15] Figure taken from [1]

Table 23 Comparison criteria - Mauricio Alférez et al.

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Domain Engineering, Application Engineering
	1.2. Which adoption strategy is followed?	Proactive, Extractive
	1.3. Which tasks of the Domain Engineering are supported?	Commonality and variability modeling (Feature Model), Feature modeling, Scenario modeling (Use Cases)
	1.4. Which tasks of the Application Engineering are supported?	Derivation
	1.5. Which tasks of the Scoping are supported?	None
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Elicitation, modeling, management
	2.2. What artifacts are used?	Feature Model, Use Cases, Activity Diagrams, table of Trace links
	2.3. Traceability	Both
3. Model-driven	3.1. Model representation	Model (Feature Model, Use Case Model, Activity Diagram), textual(Composition rule)
	3.2. Transformation language	Xtend transformation language
	3.3. Transformation type	Endogenous and vertical
	3.4. Degree of automation	Interactive
4. Tool support		Academic prototype
5. Validation		Case study

4.10 Bragança & Machado

Bragança and Machado '09 [8] propose an evolution of the 4SRS method aimed at Software Product Lines. The four-step rule set (4SRS) is a unified modeling language (UML)-based model-driven method for single system development which provides support to the software architect in this task. The paper describes how to address the transformation of functional requirements (Use Cases) into component-based requirements for the product line architecture. The result is a UML-based model-driven method that can be applied in combination with metamodeling tools such as the eclipse modeling framework (EMF) to derive the architecture of Software Product Lines.

Software Product Line support. This approach covers the Domain Engineering activity. The supported strategy is proactive.

This approach is inspired by the original work of Griss et al.[28], in the sense that variability is modeled in Use Cases. In detail, the variability annotations are based in *extends* and *include* relationships. Figure 24 shows an example of a Use Case Diagram to a Library product line example. It contains visual annotations to model the variability.

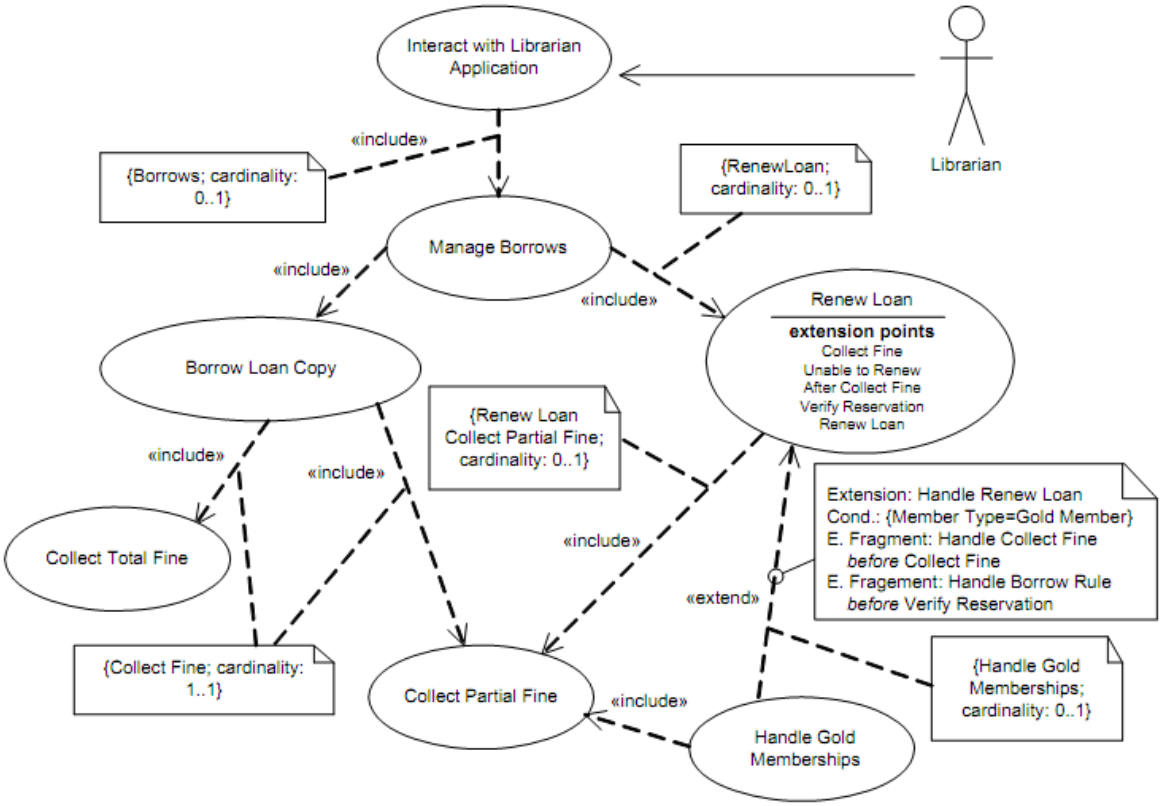

Figure 24 Bragança and Machado - Example of a Use Case diagram for a Library product line[16]

When Use Cases of the domain are identified, their behavior is modeled by activity diagrams. This is not so different from the traditional way of describing Use Case behavior by natural language. Basically, each step in a text description of a Use Case is modeled as an Action node in the activity diagram.

The Application Engineering behavior is show in Figure 25: the main goal of the process is to obtain a Use Case model for a specific application of a domain based on a feature configuration model. For that, the approach maps Use Cases to features. Basically, it consists of three transformations: transform a family Use Case model into a feature model (T1); transform a feature model into a configuration metamodel (Ecore model) (T2); and finally, transform a configuration model and a family Use Case model into an application Use Case model (T3).

Requirements engineering support. The approach supports the Requirements Engineering elicitation and modeling activities with the Use Case technique.

This work proposed three RE artifacts: the Use Case Diagram, Feature Model, and Activity Diagram.

The traceability is both vertical and horizontal. The requirement artifacts are mapped to the Use Case realization at design level. Moreover the derivation process with automatic transformations involves implicitly horizontal traceability.

[16] Figure taken from [8]

Model-driven coverage. The approach uses a UML extended notation. In concrete, it uses an extension of the UML-F profile to deal with new stereotypes to include support for requirements and analysis models.

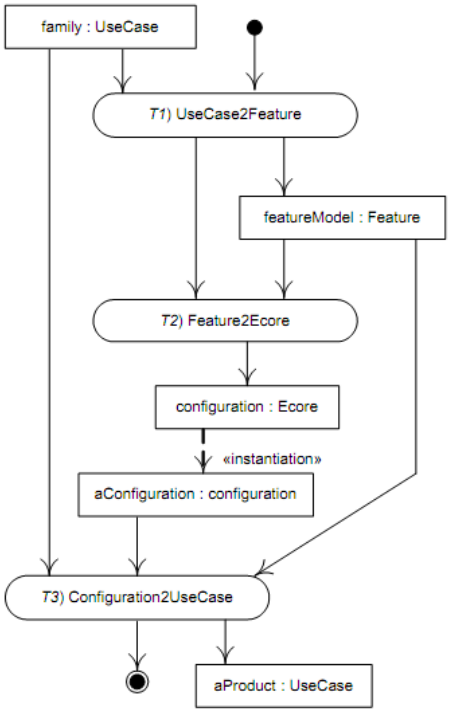

Figure 25 Bragança and Machado - Process for obtaining a product Use Case model[17]

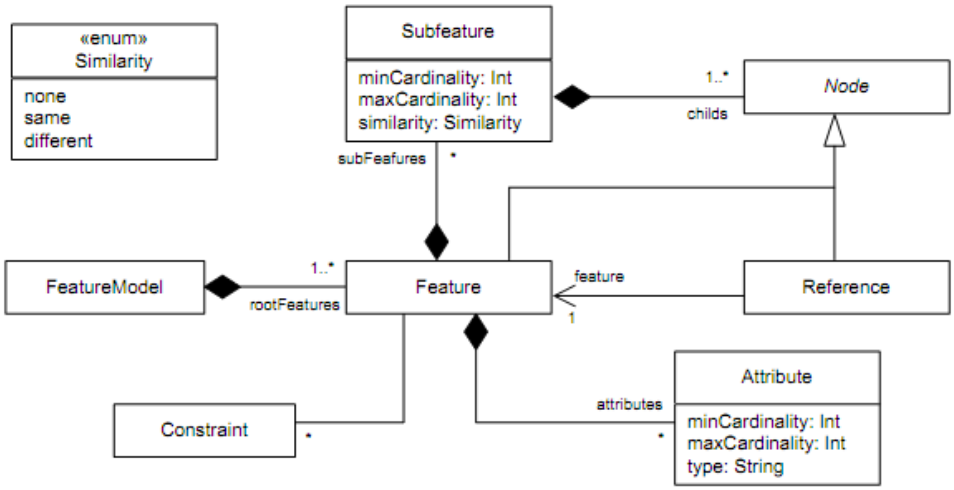

Figure 26 Bragança and Machado - Feature Metamodel

Tool support. The paper presents a prototype based in the Eclipse Modeling Framework (EMF) version 2.2.0 and SmartQVT version 0.1.3. The EMF provides a modeling and code generation framework for Eclipse applications based on Ecore models. These Ecore models support Essential MOF (EMOF) as part of the OMG MOF 2.0 specification [38].

[17] [17] Figure taken from [8]

Validation of the approach. The feasibility of the approach is shown with an example.

Table 24 Comparison criteria - Bragança & Machado

Criterion	Sub-criterion	Option
1. Software Product Line Support	1.1. Which activities of the SPL are covered?	Domain Engineering, Application Engineering
	1.2. Which adoption strategy is followed?	Proactive
	1.3. Which tasks of the Domain Engineering are supported?	C & V modeling (Feature Model), Feature Modeling (Feature Model), Scenario Modeling (Use Case Diagrams, Activity Diagram)
	1.4. Which tasks of the Application Engineering are supported?	Automatic derivation
	1.5. Which tasks of the Scoping are supported?	Not supported
2. Requirements Engineering	2.1. Which tasks of the requirements engineering are used?	Elicitation (Use Case), Modeling (Use Case, Feature Model, Activity Diagram)
	2.2. What artifacts are used?	Use Case Diagram, Feature Model, Activity Diagram
	2.3. Traceability	Both
3. Model-driven	3.1. Model representation	UML extension (Use Case Diagram, Feature Model, Activity Diagram)
	3.2. Transformation language	QVT operational
	3.3. Transformation type	Endogenous and vertical
	3.4. Degree of automation	Interactive
4. Tool support		Prototype
5. Validation		Example

5 Comparison

In the previous section, we analyzed each selected approach against the comparison criteria. In this section, we use this information to compare the approaches, altogether, with the criteria and we perform an analysis of the approaches in order to determine the coverage degree of the initial criteria. Table 25 shows the results for each approach against the evaluation criteria. Moreover, in this section, we analyze separately the results obtained for each evaluation criteria.

Software Product Line support.

Regarding the Scoping activity, seven of ten works support it. Between these works, only OVM [34] supports the Domain (with Commonality and Variability analysis), Portfolio (Portfolio analysis) and Asset Scoping (Commonality and Variability analysis). DREAM [34] supports Domain (Domain terminology) and Asset Scoping (Identification core Use Cases). Other works supports just the Asset Scoping (FeatRSEB [28] with Use Cases in addition to Feature Model, and John & Muthing [17] and DREAM [34] with the identification core Use Cases). Finally, VODRD only supports the Domain Scoping [31] with the use of Viewpoints. We consider that the Asset scoping was based on artifacts like Use Cases. However, consider the models as assets for an organization could be an interesting research trend. The use Models as reusable assets could provide a unique opportunity to mitigate complexity, improve consumability, and reduce time to market [29].

The Domain Engineering activity was supported in all of them approaches. The Domain Engineering covered tasks were: Commonality and Variability modeling (10 of 10 works), Scenario Modeling (6 of 10 [1], [28], [17], [34], [26], [9])) , Feature Modeling (6 of 10 [1], [28], [26], [11],[39], [30]), and Conceptual Modeling ([31],[34]). The Commonality and Variability is provided by Feature Models, Viewpoint Analysis, Use Cases with variations, PR-Context and PR-Use Cases matrixes or Application-Requirement matrix. The Feature Modeling was usually done with Feature Models: adopting the FODA notation (e.g. [28], [11]), or UML-based representations (e.g. [1]). The OVM [39] approach proposed an interesting notation to do the Feature Modeling using the Orthogonal Variability Model, which allows expressing variability independently of the requirement notation. Finally, the Conceptual Modeling was expressed with a dictionary in [31], and with the Domain Terminology in [34]. The approaches give more attention to functional requirement and its variability; however, the inclusion of non-functional requirements in the specifications could improve the system understandability, allowing reflecting new system qualities and restrictions.

The Application Engineering was the less supported with 5 of 10 works [17], [34], [11], [34], [8]. About the Application Engineering, three works give a partial support ([17], [34], [26]). John & Muthing [17] propose a Decision Model. In DREAM [34] the application-specific requirements are selected from domain requirements. In PLUS [26] the applications Use Cases are selected from the Domain Use Cases, depending on the Feature selection. Mauricio Alférez et al. [1] supports the derivation of Domain Requirements based on the SPL configuration. Only PLUSS [11] gives a complete support, including the Delta Identification. In general, the approaches were focused on customizations over the Domain Requirements Specification;

however, it should be interesting giving more flexibility to the developers with the supporting to the identification of new requirements (Deltas).

Regarding the Adoption Strategy, only three approaches support the three strategies: Proactive, Extractive and Reactive ([28], [26], [34]). The Proactive strategy was used in most of the approaches [17], [34], [26], [11], [39], [30], [1], [8]. However, according with Krueger [24] this strategy is the most expensive and risk-prone. One alternative to the Proactive strategy is an Extractive adoption, which was used in six approaches ([28], [26], [34], [31], [34], and [1]). Finally, the Reactive was used only in on approach [11].

Requirements engineering support.

The Modeling activity was supported in all of the approaches. Many approaches cover the Elicitation activity ([28], [34], [26], [11], [34], [8], [1]). The Analysis activity was covered only in [28]. The modeling was covered in [28], [31], [17], [26], [11], [39], [30], [1], [8]. The Management was supported in [28], [34], [1]. It is significant that we do not found any approach that mentions the requirement verification.

The approaches suggest many different artifacts to model requirements: Uses Cases ([28], [17], [34], [26], [11], [8]), Feature Model ([28],[26],[11],[30],[8]), Object Model ([28]), Domain Dictionary ([31]), Viewpoints ([31]), Decision Model ([17]), and Activity Diagram ([8]). Other artifacts are approach depending like PR-Context matrix, PR-Use Case matrix in [34], the Object Variability Model in OVM [39] or the AOR model in NAPALS [30]. In general, the approaches were focused on the modeling of functional requirements and its variability, giving less attention to the non-functional requirements.

Regarding the traceability support, the DREAM [34] approach does not mention a traceability support. Other approaches provide a vertical support [17], [31], [26], [11]. One approach provides just horizontal support [30]. Finally, other approaches provide both vertical and horizontal support [28], [34], [8].

Model-driven coverage.

There model representation was heterogeneous. Many approaches use standard UML [28], [34], or variants [26], [8], [1]. VODRD [31] was the unique that propose a textual model: the Domain Dictionary [31]. In PLUS [26] a tabular model is proposed: the Feature Model. Other use template-based models [17], [31], [34]. Other approaches use non-standard models [30]. Finally, other approaches extend the facto standards [11], [30].

Regarding the transformation language, Bragança & Machado propose the use of QVT operational. This approach proposes a formal refinement. The degree of the automation is a formal refinement. Mauricio Alférez et al. propose the use of the Xtend language. This language is used to do transformations endogenous and vertical in an interactive way with the user.

Tool support.

Many approaches do not mention automatic support [28], [17], [26]. Other approaches provide a prototype [31], [1], [8]. A tool is mentioned in [34], [30]. In this category, DREAM [34] presents a tool with the same name to support the management of the commonalities and variability of domain requirements, and to customize the application requirements. However, the domain Use Case modeling is depending on third party tools. In NAPLES [30], the tool helps to identify the relevant key domain concept candidates. Finally, only one work proposed an industrial tool [11]. The tool proposed by PLUSS is an extension to the commercial requirements management tool Teleogic DOORS [44] using its integrated scripting language DXL. This use takes advantage of the industrial acceptance of DOORS; however, the main disadvantage is that is based on a third party tool. We can conclude that the approaches need mature tools in order to gain acceptance. The lack of (commercial) tools for many activities is a major risk for achieving the intended benefits and final acceptance within the organization [45].

Validation of the approach.

Most of the approaches provide an example to illustrate the proposal feasibility ([28], [17], [30], [8]). Other approaches use the Case Study to give a prove of concept of its approaches ([28], [17], [39], [30], and [8]). VODRD solves a problem that gives support to a spacecraft mission. This case study contains 539 requirements. DREAM describes a case study to the development of e-Travel Systems in collaboration with the Electronics and Telecommunications Research Institute. PLUS applies in [26] three case studies: a Microwave Oven SPL, an Electronic Commerce SPL, and a Factory Automation SPL. Mauricio et al. illustrate the proposed approach with a Smart Home SPL case study, which is documented in [4]. Only PLUSS [11] give an industrial validation with a Case Study. This case study based on two product lines in the Swedish defense contractor Land Systems Hägglunds. This company is a leading manufacturer of combat vehicles, all-terrain vehicles and a supplier of various turret systems. Land Systems Hägglunds process baseline for software development, against which PLUSS was compared, is development according to the IBM-Rational Unified Process (RUP) [25]. In conclusion, most of the approaches use the Case Study as “proof of concept” instead of use it as evaluation method. Furthermore, the approaches should improve its validations with quantities or qualitative evaluations and rigorous design experimentations (e.g. randomization, replication of the studies).

Table 25 Overview of the approaches against the evaluation criteria

Approach / Criteria	SPL support					RE			MDD				Tool support	Validation
	Activities	Adoption strategy	DE tasks	AE tasks	Scoping tasks	RE tasks	Artifacts	Traceability	Model representation	Transformation Language	Transformation Type	Degree of Automation		
FeatRSEB '98[28]	Scoping, Domain Engineering	All	C&V (FM), FM, SM (Use Cases)	None	Asset (Use Cases and FM)	Elicitation, Modeling, Analysis, Management	Use Cases, Feature Model, Object Models	Both	UML Standard (Use Cases), UML extended with stereotypes (Feature Model)	None	None	None	None	Example
VODRD '98 [31]	Scoping, DE	Extractive	CM (Dictionary), C&V (Viewpoint Analysis)	None	Domain (Viewpoints)	Modeling	Domain Dictionary, Viewpoint, Textual Req.	Vertical	Textual (domain dictionary), Template (Viewpoints)	None	None	None	Prototype	Case Study
John and Muthing '02 [17]	Scoping, DE, AE	Proactive	C&V (Use Cases with VP), SM (Use Cases)	Deriv.	Asset (core Use Cases)	Modeling	Textual Use Cases, Decision Model	Vertical	Template (Use Case Diagram, Textual Use Cases, Decision Model)	None	None	None	None	Example
DREAM '05 [34]	Scoping, DE, AE	Proactive, Extractive	CM(Domain Termin.), C&V (PR-Context & PR-Use Case), SM (PRs & UC)	Deriv.	Domain (core Uses Cases), Asset (Domain terminology)	Elicitation, Modeling	Domain Terminology, PR-Context matrix, PR-Use Case matrix, Use Cases	None	Template, UML standard	None	None	None	Tool (DREAM)	Case Study
PLUS '04 [26]	DE	All	C&V (UC, FM), Feature Modeling, SM (UC)	Deriv.	None	Elicitation, Modeling	Use Case Diagrams, Textual Use Cases, FM	Vertical	Extended UML, Tabular	None	None	None	None	Case Study

PLUSS '05 [11]	DE, AE	Proactive, Reactive	C&V (FM), FM, SM (UC)	DA	None	Elicitation, Modeling	Feature Model, Textual Use Cases Textual Use Case Realizations	Vertical	Extension De Facto Standard, Template	None	None	None	Industrial	Industrial Case Study
OVM '05 [39]	Scoping, DE, AE	All	C&V (Application-RE matrix), FM, SM (specified by rel.)	DA	Domain (C&V), Portfolio, Asset (C&V)	Elicitation, Modeling, Management	Aplication-RE matrix, OVM	Both	Template (Application-Requirements Matrix), Non-standard	None	None	None	Prototype	Example
NAPLES '05 [30]	Scoping, DE	Proactive	C & V (comparison), Feature Modeling(Structuring into models)	None	Asset (Mining Element)	Modeling	AOR model, Feature Model	Horizontal	Non-standard, Extension the facto Standard	None	None	None	Tool (EA-minder tool)	Example
Mauricio Alférez et al. '08 [1]	DE, AE	Proactive, Extractive	C&v (Feature Model), FM, SM (Use Cases)	Deriv.	None	Elicitation, modeling, management	Feature Model, Use Cases, Activity Diagrams, table of Trace links	Both	Model (Feature Model, Use Case Model, Activity Diagram), template (Composition rule)	Xtend	Endogenous & vertical	Interactive	Academic prototype	Case study
Bragança & Machado '09 [8]	DE, AE	Proactive	C & V (Feature Model), FM (Feature Model), SM (Use Case Diagrams, Activity Diagram)	None	None	Elicitation, Modeling	Use Case, Feature Model, Activity Diagram	Both	UML extension	QVT operational	Formal refinement	Interactive	Academic prototype	Example

6 Related Work

In this section, we discuss other works that perform evaluations of RE approaches in the area of SPL.

Kovačević et al. 07 [23] makes a survey about the state of art in Requirements Engineering for Software Product Lines and Model-Driven Requirements Engineering. Two separately comparisons were performed. First, the MDD approaches are analyzed, making difference between non-aspect and aspect-oriented approaches. Second, many SPL approaches in SPL are analyzed. The authors define a common criterion for both comparisons, and another specific one for each separate comparison. This works points out facts like: most of the SPL approaches do not define a coherent and clear set of requirements and variation models with the respective relationships between them. The study points that a traceability strategy well defined is necessary. The authors suggest the combined use of the MDD technology and the aspect-oriented development to solve these issues.

Nicolas and Toval '09 [35] [i]perform a Systematic Literature Review (SLR) to study the generation of textual requirements specifications starting from models. The SLR was conducted with three research questions and assessed 30 papers in the last five years. This review reveals that a lot of work exists on generating requirements specifications from models of different kinds but that there is a lack of support for modeling and generating documents of different types in SPL. The work is focused on the use of textual requirements; however, it should be interesting include in the revision other types of requirements representations.

Vander Alver et al. '10 [3] perform a Systematic Literature Review (SLR) about Requirements Engineering for Software Product Lines. The paper is focused on assess research quality, synthesize evidence to suggest important implications for practice, and identify research trends, open problems, and areas for improvement. This SLR was conducted with three research questions and assessed 49 studies dates from 1990 to 2009. This review reveals that most of the approaches have limitations in terms of validity and credibility of their findings. Moreover, the study reveals a lack of tool support and guidance to adopt the proposed methods. For our point of view, this work does not analyze with enough level of details the use of requirements. It should be interesting analyze factors as which techniques were used, or which kind of models were employed.

In this survey, we analyzed a selection of the most relevant RE approaches in the SPL area based on citation impact. Oppositely as Kovačević et al. 07 [23], in our work we include as criterion an analysis of MDD. In the last year several MDD approaches has been proposed. Capture information about the model representations, the transformation language used, the transformation type and the degree of automation could help the developers to decide the most suitable approach to their needs. About the conclusions from Kovačević et al. 07 [23], we are according that the adoption of a MDD strategy could help to follow the relationships between the different requirement models, and with the variation models. Other factor is the requirement representation. In contrast of Nicolas and Toval '09 [35] that are focused on textual requirements, in this survey we consider any type of requirements representation: text, models, standards, etc. Finally, in contrast with Vander Alver et al. '10 [3], our survey

includes in the criterion which RE engineering tasks were covered and which requirements artifacts were used. We consider necessary include this factors in a compassion criteria about SPL.

7 Conclusions

In this work, we analyzed several RE approaches for SPL. We give special emphasis to the approaches that supports the MDD. This comparison provides information about the advantages and disadvantages of the approaches included in the study. The results obtained from this comparison have allowed us to identify several research gaps. Most of the research is focused on the Scoping and Domain Engineering activities, but the Application Engineering is the less supported. Inside the Domain Engineering, the Commonalty and Variability modeling was fully supported. Most of the approaches provide support to the Feature Modeling and Scenario Modeling Domain activities.

The Application Engineering has less support: only PLUSS [11] includes the Delta Identification. The approaches should provide mechanism to incorporate new Deltas to the SPL instead of just produce derivations based on the Domain Requirements specifications.

The Scoping activity has a similar lack: only the OVM [34] approach supported the three scoping activities. Due the fact that the scoping precedes the Domain Requirements activity, more integration with these two activities should be proposed. There is heterogeneity about the analyzed assets: core Uses Cases, Features or Primitive Requirements. One interesting trend of research is to use a MDD approach to deal with this heterogeneity and complexity.

One remarkable result is that the Proactive strategy adoption was the most common suggested by the approaches. However, according with Krueger [24] this strategy is the most expensive and risk-prone. It could be interesting to combine this strategy with the reactive or the extractive strategies to avoid these disadvantages.

With respect with the Requirement Engineering, the approaches were focused on cover the elicitation and modeling activities. The modeling of requirements was focused on functional requirements and its variability, giving less attention to the non-functional requirements. Nevertheless, the inclusion for the treatment of non-functional requirements in the approaches could help improve the quality of the software applications in the product family. Only one approach gives support to the analysis and management. Furthermore, we could not found any approach that supports explicitly the validation and verification of requirements. Including these activities in the SPL development could allow us to check whether or not the used artifacts satisfy the stakeholder needs. Similarly, we found only three approaches giving vertical and horizontal traceability support. A well-defined traceability strategy could improve the quality of the software applications.

Regarding the MDD coverage, most of the approaches used models. There is a wide heterogeneity in the use of these models. We found in the proposals from UML-based models to variants, tabular, or template models. Only VODRD [31] uses a textual model for its Domain Dictionary. With reference to the model transformations, only two approaches define transformations among the models. However, the adoption of a MDD approach could help to solve problems in the current RE proposal for SPL like the model heterogeneity, the lack of well-defined traceability strategies or the automation of the derivation process to obtain one specification for a single product in the SPL.

Regarding the tool support, only PLUSS [11] provides an industrial tool with the extension of Telelogic DOORS (currently IBM DOORS®). Tool support could help in SPL requirement activities like derivation of Domain Requirements specifications or to set traceability relationships among these artifacts. Moreover, providing tool support could to increase the opportunities of an approach to be adopted in practice.

With respect to the validation of the published approaches, only PLUSS [11] gives an industrial validation of its proposal. Furthermore, most of the approaches provide just an example to illustrate the proposal ([28], [17], [30], [8]). There is common use of the Case Study as "proof of concept" instead of use it as a well-defined validation method. The use of experiments to validate the proposals suitability could increase the acceptance of the proposed requirements approaches. Moreover, a validation using industrial data could encourage other companies to adopt these requirements SPL approaches.

All these issues provide a clear motivation for further research on RE for SPL development. Our future work includes the analysis of this current knowledge on applying RE techniques to tailor a specific RE approach for the elicitation, modeling, analysis, verification and management of requirements and its variability. This work is part of the MULTIPLE project (with reference TIN2009-13838), which has the goal to define and implement a technological framework for developing high-quality software product lines.

References

[1] Alférez M., Kulesza U., Weston N., Araujo J., Amaral V., Moreira A., Rashid A., Jaeger M. C., A Metamodel for Aspectual Requirements Modelling and Composition. Technical report. Universidade Nova de Lisboa, Portugal, 2008.

[2] Alférez M., Kulesza U., Sousa A., Santos J., Moreira A., Araújo J., Amaral V., A Model-Driven Approach for Software Product Lines Requirements Engineering. SEKE, pp. 779 -- 784, 2008.

[3] Alves V., Niu N., Alves C., Valença G., Requirements Engineering for Software Product Lines: A Systematic Literature Review. Information and Software Technology Volume 52 (8), pp. 806 -- 820, Elsevier, 2010.

[4] AMPLE Project Research Group. Retrieved on February 2011 at http://ample.di.fct.unl.pt/.

[5] Bachmann F., Goedicke M., Leite J., Nord R., Pohl K., Ramesh B., Vilbig A., A Meta-Model for Representing Variability in Product Family Development. Proceedings of the 5th International Workshop on Product Family Engineering, Siena, Italy, pp. 66-80, 2003.

[6] Bayer J., Muthig D., Widen T., Customizable Domain Analysis. Proceedings of the First International Symposium on Generative and Component-Based Software Engineering, Portland, Oregon, USA, Springer, pp. 178 -- 194, 2000.

[7] Borland Together Control Center. Available from http://www.borland.com/us/products/together/, 2001.

[8] Bragança A., Machado R. J., Automating Mappings between Use Case Diagrams and Feature Models for Software Product Lines. 11th International Software Product Line Conference, pp. 3 -- 12, 2007.

[9] Cheng Betty H. C., Atle J. M.: Research Directions in Requirements Engineering. FOSE 2007, pp. 285-303, 2007.

[10] Clements P., Northrop L., Software Product Lines: Practices and Patterns. Addison Wesley, 2002.

[11] Eriksson M., Börstler J., Borg K., The PLUSS Approach - Domain Modeling With Features, Use Cases and Use Case Realizations. Proceedings of the 9th international conference, SPLC 2005, Rennes, France, 2005.

[12] Efftinge S., Friese P., Haase A., Hübner D., Kadura C., Kolb B., Köhnlein J., Moroff D., Thoms K., Völter M., Schönbach P., Eysholdt M., Hübner D., Reinisch S, openArchitectureWare User Guide. Available from http://www.openarchitectureware.org/pub/documentation/4.3.1/openArchitectureWare-4.3.1-Reference.pdf, 2008.

[13] IBM Requisite pro. Available from http://www-01.ibm.com/software/awdtools/reqpro/, 2011.

[14] IEEE Std. 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society Press, 1990.

[15] Jacobson I., Griss M., Jonsson P., Software Reuse-Architecture. Process and Organization for Business Success. Addison-Wesley, 1997.

[16] Jacobson I., Object-Oriented Software Engineering - A Use Case Driven Approach. ACM Press, Addison-Wesley, 1992.

[17] John I., Eisenbarth, M., A Decade of Scoping – A Survey. 13th International Software Product Line Conference, USA, 2009.

[18]John I., Muthig D., Modeling Variability with Use Cases. Fraunhofer IESE, Technical Report IESE Report No. 063.02/E, 2002.

[19]Jouault F., Kurtev I., Transforming Models with ATL. Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica, Springer, pp. 128-138, 2005.

[20]Kang K., Cohen S., Hess J., Novak W., Peterson S., Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[21]Krueger C. W., Easing the Transition to Software Mass Customization. Proceedings of the 4th International Workshop on Software Product-Family Engineering (PFE 2001), Bilbao, Spain, October 3-5, pp. 282 -- 293, 2001.

[22]Kleppe A., Warmer J., Bast. W., MDA Explained, The Model-Driven Architecture: Practice and Promise. Addison Wesley, 2003.

[23]Kovačević, J., M. Aférez, Kulesza U., Moreira A., Araújo J., Amaral V., Survey of the state-of-the-art in Requirements Engineering for Software Product Line and Model-Driven Requirements Engineering. AMPLE Deliverable D1.1, 2007.

[24]Krueger C. W., Easing the Transition to Software Mass Customization. Proceedings of the 4th International Workshop on Software Product-Family Engineering (PFE 2001), Bilbao, Spain, October 3-5, pp. 282 -- 293, 2001.

[25]Kruchten P., The Rational Unified Process - An Introduction, Second Edition, Addison-Wesley, 2000.

[26]Gomaa H., Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures: Addison-Wesley, 2004.

[27]Gotel, O., Finkelstein, A., An Analysis of the Requirements Traceability Problem. IEEE Int. Conference on Requirements Engineering (ICRE '94), 1994.

[28]Griss M. L., Favaro J., d' Alessandro M., Integrating Feature Modeling with the RSEB. Proceedings of the 5th International Conference on Software Reuse, IEEE Computer Society, pp. 76-85, 1998.

[29]Larsen G., Model-driven development: Assets and reuse. IBM Systems Journal, 45(3): pp. 541 -- 553, 2006.

[30]Loughran N., Sampaio A., Rashid A., From Requirements Documents to Feature Models for Aspect Oriented Product Line Implementation. Workshop on MDD in Product Lines held in conjunction with MODELS' 05, Montego Bay, Jamaica, 2005.

[31]Mannion M., Keepence B., Harper D., Using Viewpoints to Define Domain Requirements. IEEE Software 15(1), pp. 95 -- 102, 1998.

[32]Mens T., Van Gorp P., A Taxonomy of Model Transformation. Electronic Notes in Theoretical Computer Science, vol. 152, pp. 125 -- 142, 2006.

[33]MDA Guide Version 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

[34]Moon M., Yeom K., Chae H. S., An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line. IEEE Transactions on Software Engineering, vol. 31, pp. 551 -- 569, 2005.

[35]Nicolás J., Toval A., On the generation of requirements specifications from software engineering models: A systematic literature review. Information and Software Technology 51, pp. 1291 -- 1307, 2009.

[36]OMG, http://www.omg.org/

[37]OMG, MDA Guide Version 1.0.1. http://www.omg.org/

[38]OMG, Meta Object Facility (MOF) 2.0 Core Specification (formal/06-01-01), OMG, 2006, Available at http://www.omg.org, 2006.

[39]Pohl, K., Böckle, G., Van Der Linden, F., Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, 2005.

[40]Rational Software: The Rational Unified Process for Systems Engineering Whitepaper, Ver. 1.1, Available at: http://www.rational.com/media/whitepapers/TP165.pdf, 2003.

[41]Sampaio A., Chitchyan R., Rashid A., Rayson P., EA-Miner: A tool for automating aspect-oriented requirements identification. In 20th IEEE/ACM International Conference on Automated Software Engineering (ASE) 2005. Long Beach, California, USA, 2005.

[42]Sawyer, P., P. Rayson, Garside R., REVERE: Support for Requirements Synthesis from Documents. Information Systems Frontiers 4(3), pp. 343 -- 353, 2002.

[43]Schmid K., A comprehensive product line scoping approach and its validation. ICSE 2002: 593-603, 2002.

[44]Telelogic, A.B. DXL Reference Manual, DOORS 7.1, 2004.

[45]Van der Linden F., Schmid K., Rommes E., Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering. Springer, Berlin, Heidelberg, Paris, 2007.

[46]Welge M., Al-Laham A., Strategisches Management (in German). 2nd edition, Gabler, Wiesbaden, 1999.

[47]Zave P., Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys 29 (4), pp. 315 -- 321, 1997.