Journal of Theoretical and Applied Information Technology
15 August 2012. Vol. 42 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REVIEW AND FUTURE DIRECTIONS OF THE AUTOMATED VALIDATION IN SOFTWARE PRODUCT LINE ENGINEERING

ABDELRAHMAN OSMAN ELFAKI, OMAR AMER ABOUABDALLA, SIM LIEW FONG, MD GAPAR MD JOHAR, KEVIN LOO TEOW AIK, RUZI BACHOK

Faculty of Information Sciences and Engineering

Management and Science University, Malaysia

E-mail: abdelrahmanelfaki@gmail.com ,{ omar_amer, lfsim ,gapar, Kevin, & ruzi}@msu.edu

ABSTRACT

Context: Software Product Line Engineering (SPLE) has emerged as a thriving approach for software products constructions. In SPLE, a triumphant software product is highly reliant on the validity of an SPLE. Hence, validation is a significant process within SPLE.

Objective: In this paper, we reviewed the related works in the area of automated validation of SPLE to bring to light the pros and cons of the related works and suggest the future directions in this research area.

Method: We started by defining the validation operations followed by classification of the related works in eight groups based on the technique or method that is used. The general attributes of each class are highlighted and the main strengths and weaknesses of each class related to the validation of SPLE are thrashed out. Subsequently, we analyzed each work to find out which validation operations are achieved and how they are satisfied. Finally, we abridged the current situation and recommended how the validation of SPLE can be enhanced in each operation.

Results: The research gap in the area of validation of SPLE has been clarified by recommending the future directions.

Conclusion: It is concluded that some works cannot gratify all the validation operations because the technique or method used has its weaknesses that prevented the completion of the validation process. Moreover, we conclude that this area of research has room for improvement by validating the domain-engineering directly instead of validating software products during the configuration process.

Keywords: *Software Product Line, Domain-Engineering, Configuration, Automated Analysis.*

1. INTRODUCTION

Software Product Line Engineering (SPLE) consists of two processes known as, domain-engineering and application-engineering. Collecting software-assets regarding a specific business area is a domain-engineering consciousness. The process of presenting the software assets (in domain-engineering) is called variability modelling. The principal objective of application-engineering is to configure a successful specific software product from the domain-engineering process by managing SPLE assets using variability modelling technique. Configuration is the task of selecting a valid and suitable set of features for a single system. From this definition, it is clear that configuration is part of the application-engineering process.

Now, what is the meaning of the validation of SPLE? Mannion [1] defines validation in SPLE as a mechanism that is used to ensure that an SPLE can produce at least one product that can satisfy the constraint dependency rules. Lan et al. [2] define validation (in variability) as a mechanism to check if the configuration output satisfies corresponding variability constraints (in a specific domain) or not. As a conclusion, validation of SPLE means ensuring that domain engineering contains no errors and that the configuration process is error-free.

In this paragraph, we clarify the significance of the validation of SPLE by providing two reasons. The first is based on the size of SPLE. Usually, a medium-sized SPL contains thousands of features with constraint dependency rules among them. The second is based on the nature of domain-engineering. Developing domain-

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

engineering is a continuous process; when there are new assets, these are added to the existing assets. Cumulative aggregation (for the software assets) may produce some errors. The grouping of assets may be made at different times and by different groups of people. In some cases, there is a parallel development process, i.e., several people add assets (to develop domain-engineering) at the same time. Concluding from the above two reasons, the validation of SPLE is a vital process. The first reason justifies validation in application-engineering process and the second justifies the validation in domain-engineering process. In addition, configuring a successful software product is highly dependent on the validity of an SPLE. Hence, validation is a significant process within SPLE.

Automated validation of SPLE is considered as a part of the automated analysis of SPLE. The automated analysis of SPL is a software program that extracts useful information from the SPL for SPL engineers, experts, or users [3]. In other words, the automated analysis of SPLE is a research field answering the questions of how to get constructive information and how to ensure the correctness of the software products. Two steps are formulated for the lifecycle of the automated analysis of SPLE [4]: 1) Formalization. In formalization, SPL is translated into a specific representation that allows auto-reasoning; and 2) Reasoning. By using standard tools or ad hoc software programs, the SPL representation (formalization) can be reasoned [5].

Although the Feature Model (FM) [6] and the Orthogonal Variability Model (OVM) [7] are the most popular techniques for modelling variability in Software Product Line (SPL), both lack a formal mechanism to reason SPL [8]. This has encouraged the development of other techniques that can be used for modelling and validating variability at the same time. All of these methods are supported by a specific software tool. On the other hand, some methods have been developed for validating SPL within the existing variability modelling technique.

There are numerous papers in the literature which have surveyed methods of modelling the domain-engineering process [9-14]. These studies focus only on methods of variability modelling. Although automated analysis of the SPL is a relatively new issue, there are a great many works in this research area. However, to the best of our knowledge, there are only two papers [15, 16] and one technical report [17] which have surveyed the automated analysis of SPL. These three survey papers are limited only to those works which have used the FM as a variability modelling technique and ignore the other methods of variability modelling.

There are many works used to validate FM, because FM was the first and remains the more popular method to model variability. The automated validation of FM has already been identified as a critical task in [18-21]. Although the FM is a successful variability modelling technique for SPL, some other techniques have also been used to model variability in SPL.

Although the automated validation of SPL is a relatively new area of research, a great many proposals have been put forward in both the academic and industrial fields. We analysed the literature in two steps by: 1) classifying the previous studies based on the technique used to automate the validation operations, and the strengths and weaknesses of each class are presented; and 2) analysing each study and highlighting the validation operations that were implemented.

This paper is structured as follows: in section 2, the validation operations are defined and classified under two groups: domain-engineering, and configuration. Methods for the automated validation of SPLE are discussed and section 3. In section 4, the current situation is highlighted and some directions to improve the validation operations are suggested.

2. OPERATIONS FOR THE VALIDATION OF SOFTWARE PRODUCT LINE ENGINEERING

In the literature, the works in the automated analysis of SPL is divided into several operations. These operations have been identified and discussed in [16, 17]. These operations can be divided into two groups: 1) operations for validating the SPL and 2) operations for analysing the SPL. The first group (operations for validating the SPL) is responsible for detecting, removing, or overcoming errors in SPL; whereas the second group (operations for analysing the SPL) is responsible for providing

Journal of Theoretical and Applied Information Technology
15 August 2012. Vol. 42 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

more information about the SPL. In this paper, we focus only on the operations that relate to the validation of SPL.
In the introduction of this paper, we defined the meaning of the validation method in SPL as a method used to ensure the correctness of assets in domain engineering and to produce error-free products, including the possibility of providing explanations to the modeller so that errors can be detected and eliminated in both the domain-engineering and configuration process.

In the following, we will discuss each operation. The diagrams used to describe these operations are based on a FM. Some figures and definitions are borrowed from [16].

2.1. Determine the validity of SPL

This operation examines SPL validity. An SPL is valid if it can produce at least one product. Due to the incorrect usage of constraint dependency rules, SPL may fail to produce any product. An SPL without any product is called a void SPL. In Figure 1, both features B and F are common features, which means they both must be included in any product. The exclude relation R1 between feature B and feature F means that both features cannot be included together in any product. This condition cannot be implemented because of the nature of features B and F (both are common features). Thus, the SPL in Figure 1 cannot produce any product.

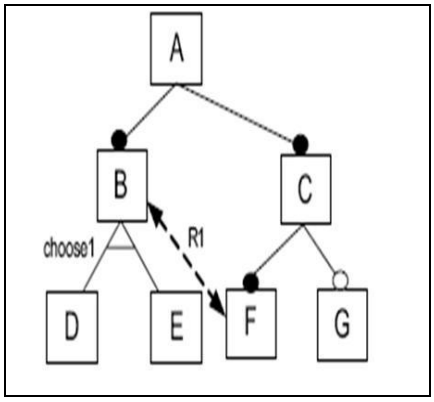

Figure 1: Example of Void SPL.

2.2. Dead Feature
A dead feature is a feature that never appears in any valid product. Dead features occur as a result of the incorrect usage of constraint dependency rules. In Figure 2, common feature B excludes feature C. Feature B must be included in any product (common feature). According to the exclude relation, feature B and feature C cannot be included together in any product. This means feature C is excluded from all products. Thus, feature C is defined as a dead feature.

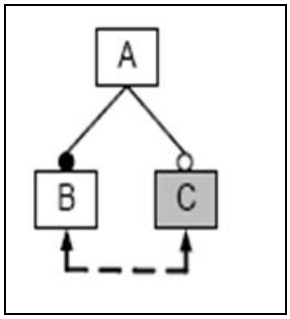

Figure 2: Example of a Dead Feature

2.3. Redundancy Detection

This operation deals with redundancy in an SPL. In an SPL, it is possible that the same set of products can be modelled more than once [19]. This operation is also known as normalization or simplification. Although redundancy is a lightweight error, a huge number of redundancies increase the complexity of an SPL. In addition, removing the redundancy enhances the readability and comprehensibility of an SPL. Figure 3 shows an example of redundancies in an SPL. In Figure 3, feature C denotes the redundancy. Feature B is a common feature which means that it must be included in any product. Common feature B requires feature C, which means that feature C must follow feature B. Therefore, feature C should be included in any product. Feature D also requires feature C, which is repeated information. Thus, the require relation between features D and C is redundant.

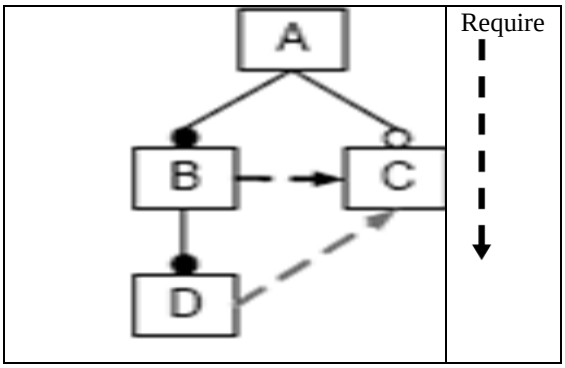

Figure 3: Example of Redundancy

Journal of Theoretical and Applied Information Technology
15 August 2012. Vol. 42 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2.4. False Option Feature Detection

A false optional feature is a feature included in any product but not assigned as a common feature, i.e. a common feature without a common label [20]. Figure 4 illustrates an example of a false option. Feature B is a common feature, which means B must be included in any product. Feature B requires feature D, which means that feature D must be included in all products. This property formulates feature D as a common feature. Thus, feature D has the same behaviour as a common feature but is not labelled as a common feature, which means feature D is a false option.

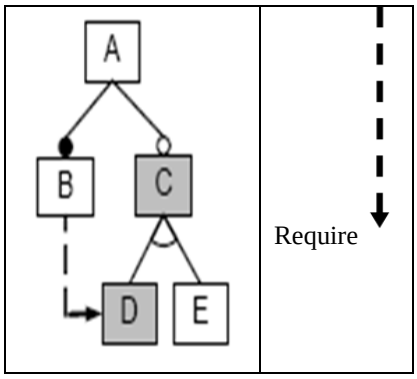

Figure 4: Example of False-Option Feature

2.5. Inconsistency Detection

Inconsistency is identified in [18] as a particular research challenge. Inconsistency occurs as a result of contradictions in constraint dependency rules. This type of error is very complicated because it can take different forms and can occur between groups of features or between individual features. Inconsistency in a FM describes relations between features that cannot be true at the same time, e.g. (A requires B) and (B excludes A), which means selection of A must be followed by selection of B, but selection of B prevents selection of A. Therefore, these relations cannot be true at the same time. An SPL can contain some other complicated forms of inconsistency. For instance, (A requires B) and (B requires C) and (C requires D) and (D excludes A), or ((A and B and C) require (E and F)) and (F excludes B). Another example is: ((A and B and C) requires (D and E)) and (A excludes E). This example describes the existence of features A, B, and C together which requires the existence of features D and E. At the same time, feature A excludes feature E. Thus, these relations could not be implemented at the same time.

Inconsistency is a critical error; it can prevent the production of any software product that has an inconsistency relation between its features. Inconsistency is also known as a conditional dead feature [22].

2.6. Wrong Cardinality Detection

Cardinality is wrong if the maximum or minimum number of variants those allowed to be selected from a variation point cannot be implemented [20]. Figure 5 illustrates an example of wrong cardinality. The maximum number allowed to be selected from this group is 3. Feature B excludes feature D, which means both features B and D cannot be included in one product. Therefore, only two features can be selected for the exclude relation. Thus, describing the maximum number as 3 is wrong.

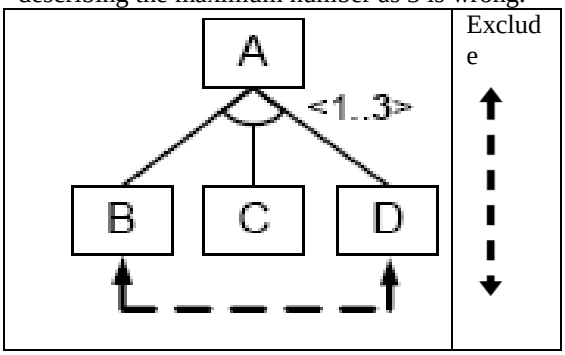

Figure 5: Example of Wrong Cardinality

2.7. Explanation

In general, the source of error is defined in the explanation operation. Debugging a huge SPL manually is almost impossible. Although a software solver can be used to detect whether or not an SPL has errors, defining the sources of errors is still a challenge. For example, the source of error (dead feature) in Figure 2 is the exclude relation between feature B and feature C. Generally, incorrect usage of constraint dependency rules (require, exclude) is the main reason for errors in the configuration process. In this research, the explanation operation is limited to defining the source of error within the configuration process. The other types of error are detected using other operations.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2.8. Corrective Explanation

Corrective suggestions (to overcome the error) are provided in a corrective explanation operation. These operations are vital in the stage-configuration process because they allow users to correct their errors. For instance, in Figure 2, the suggestions to overcome the error (dead feature) could be either: 1) remove the exclude relation or 2) remodel feature B as an option. Determining the best suggestion from the list of provided suggestions represents a challenge in this area of research.

2.9. Decision propagation

The selected feature represents the input for this operation. This operation describes auto-select and auto-deselect features during the configuration process. Based on constraint dependency rules (require and exclude) some features are auto-selected or auto-deselected. For instance, if feature A requires feature B and A is selected then B must be auto-selected. Conversely, if A is removed from the configuration then B must be auto-deselected. This operation is also recognized as dependency analysis or satisfying constraint dependency rules.

2.10. Deadlock Detection

Deadlock is always a serious issue in concurrent systems. In an SPL, deadlock is presented as a challenge that needs to be overcome [23, 24]. Deadlock occurs when two or more configuration actions are blocked and are waiting for each other‟s decisions before they can continue [24]. In an SPL, deadlock can occur in parallel configuration where different users configure solutions simultaneously and the quantity of features is limited. There are four conditions for deadlock: mutual exclusion, hold and wait, no pre-emption, and circular wait [25]. The first three conditions exist in parallel configuration by default. Therefore, circular wait is the most important condition to detect deadlock in parallel configuration. Deadlock detection is a basic operation designed to handle the deadlock problem [25]. Czarnecki [26] illustrates examples using diagrams to describe deadlock situations.

3. METHODS FOR THE AUTOMATED VALIDATION OF SOFTWARE PRODUCT LINE

The methods that are used for validating SPL are classified into eight approaches. This classification is done based on the general properties of each method. These approaches are: unified modelling language, propositional and first order logic, description logic, constraint programming, domain specific language, extensible markup language, higher order logic, and ad-hoc algorithms. In this section, these methods are described in brief and the main characteristics of each method are highlighted. The works in automated validation of SPL also are highlighted and grouped under these eight approaches. Each work is discussed to show how the validation operations (that are included in the work) were solved. Generally, there are advantages and disadvantages to the use of each.

3.1. Unified Modelling Language (UML)

Unified Modelling Language (UML) is a standard modelling language that is used to define, model, and share requirements. It contains different diagrams that allow developers and users to share a common standard language. These UML diagrams can be used to describe the system at different levels of abstraction. Standardization is the main benefit that can be gained from using UML as a variability modelling technique. Mainly, UML was developed for modelling single systems; however, the ability of UML to have standard extensions also makes it suitable for the SPL [27]. Although variability modelling needs many notations, UML can easily provide extensions using UML comments [28]. In the literature, UML is used mainly for modelling variability including the description of how the software product can be derived [29]. Standardization of UML notations has encouraged developers to move from using FMs to using UML [27, 30, 31]. In UML, Object Constraint Language (OCL) is used as a validation tool [32]. Various researchers have adopted UML in different ways to provide solutions for modelling variability in the SPL [33-40]. These methods also implemented OCL to satisfy dependency constraint rules.

The advantages of using UML as a variability modelling technique are:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

standardization, usability and provision of multiple views for SPL [40]. On the other hand, using only OCL as a validation tool represents the main drawback of UML as a variability modelling technique because OCL is usually implemented over a specific object or an object's structure [26]. This property limits the implementation of OCL.

Figure 6 shows how use case diagrams (the use case diagram is a popular UML diagram) can be used to represent variability. In Figure 6(a), "Authentication" is illustrated as a variation point. Both "Give fingerprint" and "Insert chip card" represent variants of the "Authentication" variation point. In Figure 6(b), both "Choose bank transfer" and "Choose standing order" illustrate variants of the "Select order" variation point. Figure 6 is borrowed from [20].

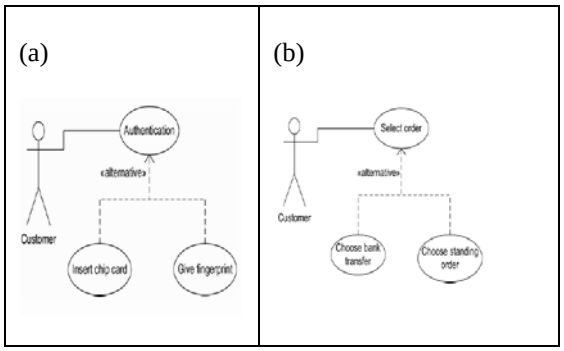

Figure 6: Use case diagrams representing variability Source

Clauss[27] suggests two stereotypes for modelling variability "<<variation point>>", which indicates the variability of an element and „<<variant>>", which indicates the extension part. Clauss [30] suggests the use of OCL to satisfy constraint dependency rules. Korherr and List [41] proposes a UML 2 profile to model variability. Korherr and List's [41] model uses OCL to satisfy the three levels constraint dependency rules (variant-variant, variant-variation point, and variation point-variation point). Ziadi et al. [42] and Ziadi and Jézéquel [43] use OCL in the form of a meta-model level to satisfy constraint dependency. Sturm and Berge [44] present an approach to model domain engineering. This approach enables the validation of domain-specific application models against their domain models and uses OCL to satisfy constraint dependency rules. Various other works have adopted UML in different ways to provide solutions for modelling variability in the SPL [33, 34, 35, 37, 38 , 40, 45,46, 47]. These methods implemented OCL to satisfy dependency constraint rules. Sinnema et al. [48] introduced a framework (base on UML) to model dependency constraints based on formalized, documented and tacit knowledge. Czarnecki and Kim [49] introduce a method to satisfy constraint dependency rules and filtering using OCL. According to [32], OCL is a common tool used to satisfy the dependency constraint rule when variability is modelled using UML.

Table 1 demonstrates an example of OCL code for the variant-variation point constraint. In this example, each variant has only one representation in each variation point.

Table 1: OCL code for variant-variation point constraint.

```
context <<variant>>
inv: self.supertype → select(oclIsKindOf(Variatio
→size()=1
```

3.2. Propositional and First Order Logic (PL and FOL)

In general, truth and provability are the main concepts considered by logic. In logic, a model theory denotes how to study the abstraction and properties of a problem structure [60]. This fact has encouraged researchers to use logic as a solution for SPL validation. Propositional logic attempts to formalize reasoning by using a set of symbols and a set of logical connectives, e.g., not, and, or, and if..., then [51].

Propositional logic as a validation technique for the SPL is completed in two steps. Step one consists of representing an SPL formally using propositional formulas. Step two consists of reasoning an SPL using off-the-shelf tools such as Logic Truth Maintenance System (LTMS), Satisfiability Solvers (SAT), and Binary Decision Diagram (BDD).

Propositional logic is chosen for the benefit provided by its supporting tools. However, unfortunately, propositional logic is not powerful enough to represent most real-life cases [52]. This drawback has encouraged some researchers to introduce FOL as a validation technique for the SPL. First Order Logic adds two quantifiers: for all and there is. These two quantifiers make FOL more expressive and more able to handle most real-life problems [50]. In the literature,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

there are some contributions which validate the SPL based on FOL e.g.,[53].

The first proposals which connected propositional formulas to the FM were those presented by [54,55]. Mannion in [54] explains how an SPL can be represented as a logical expression. However, Mannion and Camaras's [55] model did not investigate dependency constraint rules (require and exclude constraints). Zhang et al. [56-57] propose a propositional logic-based method for the validation of a FM. In [56,57] model, constraints are formalized into logic sentences and general-purpose model checkers are suggested in order to automate the validation process. Two validation operations are satisfied by [56, 57]: 1) determine the validity of SPL and 2) constraint dependency checking. Constraint dependency checking is satisfied at the basic level (feature-to-feature). Zhang et al. [56, 57] model classifies features into three groups: bound, remove, and undecided. After opting for all products, the undecided group represents dead features. By defining the pre-condition and post-condition for each feature, explanation and dead features operations are satisfied.

Batory[58] proposes a coherent connection between the FM, grammar and propositional formulas. Batory's study represents the basic FM using context-free grammars plus propositional logic. This connection allows arbitrary propositional constraints to be defined among features and enables off-the-shelf SATs and Logic-Truth Maintenance Systems to debug the FM. The use of SAT and LTMS solvers satisfies the constraint dependency rules and the explanation operations. Sun et al. [53] propose a formal semantics for the FM using first-order logic. Sun et al. [53] use Alloy Analyser (a tool for analysing models written in alloy) to automate the constraint dependency checking and the explanation operations in the configuration process. Alloy Analyser is a declarative specification language for describing the constraints and structures of complex systems. Gheyi et al. [59,v60] also validate a FM using Alloy Analyser. Using Gheyi et al.'s theory, the constraint dependency checking operation is satisfied.

Storm [61, 62] suggests a method for mapping the FM to a propositional logic formula. This mapping provides a mechanism for validating both 1) determine the validity of SPL and 2) constraint consistency checking operations. Zhang et al. [63] proposed a Binary Decision Diagram (BBD) data structure to handle the dependency constraint checking operation.

A knowledge-based product derivation process [64, 65] is a configuration model that includes three entities of the Knowledge Base (concept model, procedural knowledge and task specification). The second entity (procedural knowledge) is used to satisfy the constraint dependency rules.
Hemakumar [66] uses a connection between context-free grammar and propositional logic to detect inconsistency in the configuration process. However, Hemakuma¨s work detects only direct inconsistency in the software product. Yan et al. [67] propose an optimization method for validating the FM. Yan et al's method removes validation-irrelevant constraints from the FM in order to reduce the size of the problem. Yan et al's method handles dead features and determines the validity of SPL operations. Constraint dependency checking and propagation are defined and supported in Mendonca et al. [68-71]. Salinesi et al. [72] developed a tool to support automatic validation of SPL. Salinesi's tool detects dead features and satisfies constraint dependency checking. Elfaki et al. [73, 74] detect dead features by searching only for predefined cases, i.e. defined dead features in the domain-engineering process. Elfaki et al. [75] propose an interactive configuration. Elfaki et al.[76] introduce first order logic rules to detect inconsistencies in domain-engineering.

As stated earlier, two properties characterize constraint dependency rules in SPL. These properties are: require and exclude. Table 2 shows an example of how propositional logic can be used to represent the constraint dependency rules in SPL. Table 3 illustrates an example of FOL representation for constraint dependency rules.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2: Representation of SPL constraint dependency rules using propositional logic.

Properties	Example	Propositional logic representation
Require	In Figure 4, the feature B requires the feature D.	$B \longrightarrow D$
Exclude	In Figure 5, B excludes D.	$\neg (B \wedge D)$

Table 3: Representation of SPL constraint dependency rules using FOL.

Property	FOL representation	Explanation
Require	$\forall x, y: require(x, y) \wedge select(x) \Longrightarrow select(y)$	If *x* requires *y* and *x* is selected then *y* must be selected.
Exclude	$\forall x, y: exclude(x, y) \wedge select(x) \Longrightarrow \neg select(y)$	If *x* excludes *y* and *x* is selected then y must not be selected.

In addition to the tools availability, the ability to describe the problem in an abstract way is the main advantage of using PL and FOL. PL and FOL are limited to work with certain environment in which all elements and variables are well defined. In SPL, if all constraints dependency rules are well defined then this is a certain environment.

3.3. Description Logic (DL)

Description logic (DL) is defined as a knowledge representation technique. In description logic, formal knowledge regarding a specific domain is described in a well-structured and well-understood way [77]. Description logic is characterized by its ability to build complex classes and relation from simple ones. It is the formal logic used to develop ontologies and is supported by a wide range of well-established solvers and reasoners.
Various works (78-82) have proposed different approaches for representing FMs using ontologies. These works satisfy two operations: 1) determine the validity of SPL and 2) constraint dependency rules. Wang et al. [83, 8] propose that Ontology Web Language (OWL) be used to validate a FM. Wang et al. [8] use OWL-DL to capture the inter-relationships between the features in a FM. Asikainen et al. [84] satisfy the constraint dependency rules and the explanation operations by translating the model into Weigh Constraint Rule Language (WCRL), which is a general-purpose knowledge representation language. Wang et al [8] support the constraint dependency rules and the explanation operations by using Fast Classification of Terminologies (FaCT++) and Renamed ABox and Concept Expression Reasoner (RACER) as tools for reasoning. Dedeban [85] use OWL-DL and a rule-based system to support the constraint dependency rules and the explanation operations. Kaviani et al. [86] map a FM to ontology in order to deal with the non-functional requirements and also satisfy the constraint dependency rules. AboZaid et al. [87] use semantic web technology for validating a FM. AboZaid et al's [87] proposal detects dead features and provide explanations. Table 4 shows the DL representation for require and exclude constraint dependency rules.

Table 4: Representation of SPL constraint dependency rules using DL.

Property	Description logic representation	Explanation
Require	GRule ⊑ ∃has f1.f2	f1 requires f2
Exclude	GRule ⊑ ∃ ¬(has f1.f2)	f1 excludes f2

The main advantage of DL is the ability of transmission from simple relations to very complicated relations. DL is limited to work with certain environment only.

3.4. Constraint Programming (CP)

In constraint programming, a problem is structured as a finite set of variables, finite set of domain values for these variables, and finite set of constraints between these variables. The problems that are solved by constraint programming are recognized as constraint satisfaction problems. The responsibility of a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

constraint program is to find the solutions that satisfy these constraints. A solution of constraint programming is described as assigning a value (from the domain) to each variable in such a way as to satisfy all constraints simultaneously. The basic algorithm for solving constraint programming is based on finding all possible combinations of values (assigning values to variables). Afterwards, the algorithm checks each combination for satisfaction of the constraint. A successful combination satisfies all constraints simultaneously. This algorithm is completely inefficient. This inefficiency has motivated the researcher to develop different search algorithms for solving constraint programming problems.

The use of constraint programming to deal with the analysis of the FM is suggested in [88, 89] where the FM is translated into a Constraint Satisfaction Problem (CSP). In the automated analyses of SPL, CSP proposals use traditional constraint solvers as an implementation tool. These studies satisfy two validation operations (constraint dependency checking, determine the validity of SPL, and explanation) in non-interactive mode. Trinidad et al. [90, 91] define a method for detecting dead features. Trinidad et al's method is based on finding all products and then searching for unused features. Trinidad et al. [92, 93] detect false optional features based on finding all products and then searching for common features among those which are not assigned as common. White et al. [94, 95] propose a method for automated analysis of SPL configuration errors in the FM. White et al's method starts by transferring the current invalid configuration and the FM constraints into a CSP solver. Then, the solver derives a classification of the investigative CSP. Finally, this classification is transformed into a series of suggestions to select or deselect features. These recommendations aim to convert the invalid configuration into a valid configuration. White et al's [95] method solves the configuration problems without interactivity with the users. Djebbi et al. [96] use Integer Linear Programming (ILP) notations to satisfy both the filtering and dependency constraint checking operations. Table 5 shows the constraint programming representation for require and exclude constraint dependency rules.

Table 5: Representation of SPL constraint dependency rules using constraint programming.

Property	**Constraint programming representation**	**Explanation**
Require	if (f1 > 0) f2>0	f1 requires f2
Exclude	if (f1 > 0) f2=0	f1 excludes f2

Availability of open-source tools is the main motivation to work with CSP in SPL. On the other hand, the aiming of CSP searching is to find all solutions that satisfy the constraints which mean CSP works in application engineering. CSP almost fails with the huge size of data.

3.5. Domain Specific Language (DSL)

Domain specific language (DSL) is a special type of programming language which is oriented to a specific domain. Consequently, DSL is not able to solve general problems [97]. The advantages and disadvantages of DSL are summarized below [97]:

Examples of advantages are:

- More expressive than normal programming languages;
- Explains the domain in high level of abstraction which provides a clear picture of the domain. This property could be used as a learning tool;
- Easy for domain experts to be involved in the developing, maintaining, testing and updating processes.

Examples of disadvantages are:

- Cost of learning is comparatively high considering its limited applicability;
- A supporting tool needs to be developed;
- Standardization is difficult: occasionally, there is different vocabulary within the same domain.

Cao et al. [98] developed an algorithm to transfer FMs into data structures. This algorithm generates complete feature instances from a feature diagram under constraints. Cao et al. [98] use the Generic Modelling Environment (GME) to develop the algorithm; however, their algorithm satisfies only the constraint dependency checking and explanation operations. Deursern and Klint [99] propose a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

feature description language to describe the FM. From this language, FM algebra is described based on rules over the ASF+SDF Meta–Environment [100]. Using the system of Deursern and Klint [99], two validation operations (constraint dependency checking, and explanation operations) are satisfied, but in non-interactive mode. Pohjalainen [101] describes a subset of regular expressions that can be used to express a FM. Pohjalainen [101] presents a compiler for translating a FODA model to a deterministic finite state machine with support for implementing model constraints via post-augmentation of the compiled state machines. This model satisfies three validation operations (constraint dependency checking, determine the validity of SPL, and explanation). Groher and Voelter [102] propose an approach for managing variability on the model level. The Groher and Voelter [102] approach uses techniques of DSL to develop a supporting tool. This approach satisfies the constraint dependency checking operation and their approach is validated using a home automation system. Table 7 shows an example of how the FM could be represented as a Java property file. The DSL presented in Table 6 was suggested in Deursern and Klint[99].

Table 6: DSL representation of FM

car.transmission = automatic, manual
car.engine = electric, gasoline
car.body= true
car.cruise = false

3.6. Extensible Markup Language (XML)

Extensible Markup Language (XML) is defined as a set of rules used for encoding data and documents electronically [103]. Extensible Markup Language is considered to be a generic format which can ensure maximum flexibility in providing data, information and generating documents in different structural formats. The strengths of XML are: simplicity, usability, and generality. Although XML was initially developed to deal with documents, due to these strengths, XML has a wide range of applications and has proved useful for representing different data structures [103]. Moreover, the great strength of XML is the availability of different XML specifications that satisfy different applications.

Cechticky et al. [104] propose a feature meta-model and use a XML for expressing the complex composition rules that can be found in features. Cechticky et al. [104] describe a compiler that can translate the constraint model designed as a FM into an XML structure and which can check compliance with the constraints in the configuration process.

The XML-based Variant Configuration Language (XVCL) [105-109] is a configuration language. In this configuration language, domain models are analysed and variation points, variant and constraint dependency rules are recorded. The implementation of XVCL is based on (XML Metadata Interchange) XMI and XML technologies, and four validation operations (constraint dependency checking, determine the validity of SPL, propagation, and explanation) are satisfied.

3.7. Higher Order Logic (HOL)

In Higher Order Logic (HOL), a predicate can handle more predicates as arguments [110] To the best of our knowledge there is only one work to date which has used HOL to reason about SPL variability model. Janota and Kiniry [111] formalize a FM using HOL. This formalization satisfies the constraint dependency checking, determine the validity of SPL, and explanation operations.

3.8. Ad hoc algorithms

The underpinnings of some the proposals in the literature are not clearly expressed. We categorize these types of proposals as ad hoc algorithms. Lengyel et al. [112] propose an algorithm to handle constraints in the FM which is based on graph rewriting-based topological model transformation. The implementation of the Lengyel et al. [112] method is done based on the semantics of OCL and constraint dependency checking is satisfied based on the feature-to-feature level. Broek et al. [113] present an algorithm to eliminate constraints from the FM. This algorithm only eliminates the require and exclude constraints. Broek and Galvão [114] design an algorithm to validate the FM based on transforming the FM into a generalized feature tree. In a generalized feature tree, multiple occurrences could be true for one feature. Using the algorithm of Broek and Galvão [114], dead features, and constraint dependency checking are implemented.

Weyns et al. [115] suggest an SPL for automated transportation systems. Deadlock avoidance is done manually through an I/O client. La Rosa et

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

al. [116] define precedence and order constraints to avoid deadlock. Although these constraints prevent contradictory constraints that lead to deadlocks during configuration, there is no description of the deadlock detection method. Aalst et al. [117] discuss a deadlock scenario in the configuration process. Their method checks the variation points that are attached to parallel splits, decision points, and synchronization points. All these points are represented in a configurable process model. Aalst et al. [117] represent a process model using workflow nets (a special type of Petri net) to ensure the configuration is deadlock-free. Yang et al. [118,119] propose an SPL design and implementation method based on the feature-oriented adaptive component model. In order to ensure deadlock-free configuration, Yang et al. [118] define and compose behavioural protocols using CSP (Communicating Sequential Processes) composition operators, whereas Yang et al. [119] integrate a Labelled Transition System Analyser (LTSA)[1] tool in their model.

Cordy et al. [120] developed an algorithm to verify SPL based on model checking technique. First, Cordy's proposed featured transition systems to formally represent SPL then later Cordy used abstraction-based model checking for the verification of SPL. Cordy's work satisfying the constraint dependency rules. Bagheri et al. [121] developed an algorithm to validate the configuration process. Bagheri's algorithm is developed on based of propositional logic and concrete domains. Bagheri's algorithm is validating only the configuration process.

4. SUMMARY AND DISCUSSION

Various proposals that deal with validation operations have been discussed in this paper. First, the main validation operations are discussed. Then, the proposals were classified in eight groups based on the main attributes of each proposal. Later, each class was defined and discussed, and the main pros and cons were highlighted. Finally, each proposal was analysed and its contribution regarding SPL validation was described.

[1] *LTSA* is a tool for verifying the concurrent systems. LTSA checks the specifications against the required properties. *www.doc.ic.ac.uk/**ltsa**/*

In the following, the current situation is summarized and the future directions are suggested.

The most basic and most important operation of all SPL validation operations is the constraint dependency checking. There is no guarantee that an error-free software product can be produced without satisfying the constraint dependency rules. In this paper, many of the works discussed in section 3 satisfied constraint dependency rules. These works implement the constraints checking operation based on the one layer (also called feature-to-feature or variant-to-variant) basis. In the configuration process, each selected feature (selected feature is defined as a feature selected to be part of the software product) is checked regarding the constraint dependency rules. This type of constraints verification is based on feature-to-feature, i.e. one layer. The constraint relations between parent features must be reflected in their child features. The works that have been discussed in this chapter do not show how the constraint relations between parent features are reflected in their child features.

Explanation (discovery of errors) is mentioned and implemented in many of the works discussed in this paper. In these works, explanation is implemented at the end of the configuration process, which generally contains several steps. Discovery of the errors which need correction at the end of the configuration process is considered time consuming. Interactive explanation in which a user can correct the configuration errors immediately (i.e. guide the user step by step) is the best way to reduce time consumption.

Corrective explanation, which provides suggestions for a user to resolve errors, is mentioned in [94] in which a constraint solver is used to derive the minimal set of features which should be selected or deselected to bring the configuration to a valid state. Their method is also implemented at the end of the configuration process. The addition of an interactive mechanism would enhance corrective explanation.

Dead feature detection is implemented in [90, 91] based on finding all products and searching for unused features. In the automated analysis of SPL, finding all products is the toughest

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

operation to perform and unfeasible, even with a medium-size SPL.

The inconsistency detection operation is discussed in [22] in which inconsistency is detected in the configuration process; however, it is limited in that it only detects inconsistency between the configuration features. For example, if there is an inconsistency between two features and only one of them is included in the configuration process then this inconsistency cannot be detected.

Some proposals attempt to deal with deadlock detection. In the literature, methods dealing with deadlock use extra constraints, tools or methods. Detecting deadlock without extra constraints represents a challenge.

False optional feature detection is discussed in Trinidad et al. [90,91] based on finding all products and then searching for common features (included in all products) among those are not assigned as common. This method is a very high-cost solution.

The (FeAture Model Analyser) FAMA framework [122] defines a deductive operation for wrong cardinality in general. Wrong cardinality is described in the literature as a general problem. However, this description is not complete. The error could be in the minimum number, the maximum number or both. Moreover, regarding the specific variation point, in some cases, there is no wrong cardinality and in some cases there is. Wrong cardinality therefore needs to be divided into more sub-problems. Table 7 shows the current and future directions in the validating of SPLE.

Table 7: The current and future directions in the validating of SPLE.

Operation	Current Situation	Future Directions
Determine Validity of SPL	Configuring at least one software product to prove the validity of the SPL	Providing auto-solution for invalid SPL.
Inconsistency Detection	Check the inconsistency during configuring a software product.	Remove the inconsistency from the domain-engineering.
Dead Features Detection	Finding all products first, then search for unused features.	Finding dead features in the domain-engineering.
Redundancy Detection	Check the redundancy during configuring a software product.	Detect and remove the redundancy in the domain-engineering.
False-option features detection	Finding all products first, then search for common features (those features included in all products) and signed as not common features.	Finding and fixing the false-option features in the domain-engineering.
Wrong cardinality detection	Detecting wrong cardinality during the configuration process.	Provide auto-solution for wrong cardinality problem
Constraint consistency check	Check the constraint consistency during the configuration process.	Provide an interactive configuration which can guides user step by step.
Explanation and corrective explanation	Implementing the explanation and corrective explanation after the configuration process.	Provide an interactive explanation and corrective explanation in each user choice.
Deadlock	Using extra constraints	Detect the deadlock without additional cost.

5. CONCLUSION

Generally, the problem of the current research is that the checking of the software product's correction only happen after it has been

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

developed as in the process of application engineering. This is not feasible to ensure the correctness of the SPL because the medium-size SPL can contain huge number of software products. Validating domain-engineering itself represent the challenges due to the huge-size of data. The second problem is that the current configuration tools are lack of interactivity. Due to the type of stage-configuration, the interactivity is a must.

The current works are limited to work only in a certain environment, i.e., where constraint dependency rules are well known in all cases. In some SPL, constraint dependency rules are different from product to product. These types of SPL are known as uncertain SPL environments. The working with uncertain SPL is a good area for future work.

REFERENCES

[1] Mannion, M. (2002). *Using first-order logic for product line model validation*, the Second Software Product Line Conference SPLC2, San Diego CA,USA.

[2] Lan, Q., Liu, S., Li, B., Chen,Y., Pang, S., Yin, J. (2006). Research on Variability Metamodeling Method, The First International Symposium on Pervasive Computing and Applications (SPCA06), Urumchi, Xinjiang, P.R. China.

[3] Benavides, D., Ruiz-Cort´es, A., Batory, D., Heymans, P., (2008). *First International Workshop on Analyses of Software Product Lines* (ASPL'08)', Limerick, Ireland.

[4] Benavides, D. (2007). On the automated analysis of software product line using fearure models. A framework for developing automated tool support, PhD. Thesis, University of Sevilla, Spain.

[5] Janota, M., Kiniry,J., Botterweck, G.(2008). *Formal Methods in Software Product Line: Concepts, Survey, and Guidelines*, Lero Technical Report Lero-TR-SPL-2008-02.

[6] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S. (1990). *Feature oriented domain analysis (FODA) feasibility study*, Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, USA.

[7] Pohl, K., Böckle, G., Linden, F. van der.(2005). Software Product Line Engineering Foundations Principles and Techniques, Springer, Verlag Heidelberg Germany.

[8] Wang, H., Li, Y.F., Sun, J., Zhang, H., Pan, J.(2007).*Verifying feature models using OWL*, Journal of web semantics , 5 (2) Elsevier, pp. 117–129.

[9] Harsu, M., *A survey on Domain Engineering*.(2002). Report 31, Institute of Software Systems, Tampere University of Technology, http://practise2.cs.tut.fi/pub/

[10] Trigaux, J.C., Heymans, P. (2003). *Modeling variability requirements in Software Product Lines: a comparative survey*, Technical report (EPH3310300R0462 / 215315), Computer Science Institute, University of Namur, Belgique.

[11] Bayer, J., Eisenbarth, M., Lehner, T., Puhlmann, F., Richter, E., Schnieders, A., Weiland, J.(2004). *Domain Engineering Techniques and Process Modeling*, PESOA-Report No. 09/2004.

[12] Alana, E., Rodriguez, A. I.(2007). Domain Engineering Methodologies Survey. Technical Report, GMV, www.gmv.com.

[13] Sinnema, M., Deelstra, S. (2007). *Classifying variability modeling techniques*, Journal on Information and Software Technology, Vol.49, Elsevier, pp.717–739.

[14] Chen, L.,Babar, A. M., Ali, N.(2009). Variability Management in Software Product Lines: A Systematic Review, 13th International Software Product Line Conference, San Francisco, CA, USA.

[15] Benavides, D., Ruiz–Cort´es, A., Trinidad, P., Segura, S. (2006). *A Survey On The Automated Analyses Of Feature Models*, Jornadas de Ingenier´ıa del Software y Bases de Datos (JISBD'06).

[16] Benavides, D., Segura, S., Ruiz-Cort´es, A. (2010). *Automated Analysis of Feature Models 20 Years Later: A Literature Review,*, Information Systems journal, Volume 35 , Issue 6, Elsevier, PP. 615-636.

[17] Benavides, D., Segura,S., Ruiz-Cort´es, A.(2009b), Automated analysis of feature models: A detailed literature review. Technical Report ISA-09-TR-04, ISA research group, 2009. Available at

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

http://www.isa.us.es/, University of Sevilla, Spain.

[18] Batory, D. , Benavides, D., Ruiz-Cortés, A. (2006). *Automated analysis of feature models: Challenges ahead*, Communications of the ACM, 49(12), pp.45–47.

[19] Massen, T. von der., Litcher, H. (2004). *Deficiencies in Feature Models*, Workshop on Software Variability Management for Product Derivation- Towards Tool Support, collocated with SPLC 2004: Boston, MA, USA.

[20] Massen, T. von der., Litcher, H. (2005). *Determining the variation degree of feature models*. In Software Product Lines Conference, LNCS 3714, pp. 82–88, Rennes, France.

[21] Czarnecki, K., Eisenecker, U.(2002). *Generative Programming: Methds, Tools, and Applications*, Addison-Wesley, Boston MA, USA.

[22] Hemakumar, A. (2008). *Finding Contradictions in Feature Models*, First International Workshop on Analyses of Software Product Lines (ASPL'08)', collocated with SPLC08. Limerick, Ireland.

[23] Kurakawa, K., 2004. *Feature modeling from holistic viewpoints in product line engineering*, in the 11th Asia-Pacific Software Engineering Conference (APSEC'04),IEEE, Busan, Korea.

[24] Mendonca, M., Oliveira,T., Cowan, D. (2006). *Collaborative and Coordinated Product Configuration*, In international Software Product Line Conference (SPLC06) , Doctoral Symposium, Baltimore, Maryland, USA.

[25]] Kifer, M., Smolka, S.A, 2007. Introduction to Operating System Design and Implementation: The OSP 2 Approach. Springer.

[26] Czarnecki, K., 1998. *Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models*. PhD Thesis Technical University of Ilmenau, Germany.

[27] Clauss, M. (2001).*Generic Modeling using UML extensions for variability*. In Workshop on Domain Specific Visual Languages at OOPSLA01, Tampa Bay, FL, USA.

[28] Saval, G., Puissant, J.P., Heymans, P., Mens, T. (2009), *Some Challenges of Feature-based Merging of Class Diagrams*, Proc. of the third International Workshop on variability modeling of Software-intensive Systems (VaMoS09), Sevilla Spain.

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

[30] Clauss, M. (2001). *Modeling variability with UML, Generative and Component-Based Software Engineering*, Third International Conference GCSE 2001-Young Researchers Workshop, Erfurt, Germany

[31] Speck, A., Clauss, M., Franczyk, B. (2002), *Concerns of Variability in "bottom-up" Product-Lines*, Proceedings of Second Workshop on Aspect-Oriented Software Development, Bonn, University Bonn, Germany.

[32] Trigaux, J.C., Heymans, P. (2003). *Modeling variability requirements in Software Product Lines: a comparative survey*, Technical report (EPH3310300R0462 / 215315), Computer Science Institute, University of Namur, Belgique.

[33] Robak, S., Franczyk, B., Politowicz, K. (2002), *Extending The UML For Modelling Variability For System Families*, international journal of applied mathematics and computerscience,vol.12, no.2, pp.285–298.

[34] Riebisch, M., B¨ollert, K., Streitferdt, D., Philippow, I. (2002). *Extending feature diagrams with UML multiplicities*, 6th World Conference on Integrated Design & Process Technology (IDPT2002).

[35] Streitferdt, D., Riebisch, M., Philippow, I.(2003). *Details of formalized relations in feature models using OCL*, 10th IEEE International Conference on Engineering of Computer–Based Systems (ECBS 2003), IEEE Computer Society, pp. 45–54, Huntsville, USA.

[36] Massen, T. von der., Litcher, H. (2003), *Requiline: A requirements engineering tool for software product lines*. In F. van der Linden, editor, Proceedings of the Fifth International Workshop on Product Family Engineering (PFE), volume 3014 of Lecture Notes in Computer Sciences, Siena, Italy,Springer–Verlag.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[37] Philippow, I., Riebisch, M., Boell, K.(2003). The *Hyper/UML Approach for Feature Based Software Design*, the 4th AOSD Modeling With UML Workshop. Colocated with 6th international conference on the Unified Modeling Language UML, San Francisco, California, USA.

[38] Oliveira, J., Gimenes, I., Huzita, E., Maldonado**,** J. (2005). *A Variability Management Process for Software Product Lines*, the 2005 conference of the Centre for Advanced Studies on Collaborative research, IBM Centre for Advanced Studies Conference, Toranto, pp. 225 – 241, Ontario Canada.

[39] Schnieders, A. (2006). *Modeling and Implementing Variability in State Machine Based Process Family Architectures for Automotive Systems*, the 3rd International workshop on Software Engineering for Automotive Systems ICSE06, Shanghai, China.

[40] Gomaa, H., Shin, M. (2007). *Automated Software Product Line Engineering and Product Derivation*, 40th Annual Hawaii International Conference on System Sciences, Hawaii, USA.

[41] Korherr, B., List, B. (2007). *A UML 2 Profile for Variability Models and their Dependency to Business Processes*, 18th International Workshop on Database and Expert Systems Applications, IEEE, Regensburg, Germany.

[42] Ziadi, T., Jezequel, J.M., Fondement, F. (2003). *Product Line Derivation with UML*, Software Variability Management Workshop, pp. 94–102, Groningen, The Netherlands. In J. Davies, editor, *ICFEM 2004*, volume 3308 of *Lecture Notes in Computer Sciences*, pages 115–130. Springer–Verlag.

[43] Ziadi, T., Jézéquel, J.M.(2006), *Product Line Engineering with the UML: Deriving Products*. Chapter in Software Product Lines, pp. 557--586, Springer, Berlin Heidelberg.

[44] Sturm, A., Reinhartz-Berger, I. (2004). *Applying the Application-based Domain Modeling Approach to UML Structural Views*, book chapter in Conceptual Modeling – ER 2004,Springer, Berlin Heidelberg, Germany.

[45] Halmans, G.,Pohl, K. (2003), *Communicating the variability of a software-product family to customers*, journal of Software and Systems Modeling, Volume 2, Number 1, PP. 15-36.

[46] Alférez, M, Santos, J., Moreira, A., Garcia,A., Kulesza,U, Araújo, J., Amaral, V. (2010), *Multi-view Composition Language for Software Product Line Requirements*, book chapter in Software Language Engineering, ISSN 0302-9743, Springer Berlin / Heidelberg, pp. 103-122

[47] John,I., Muthig D. (2002), *Tailoring Use Cases for Product Line Modeling*, proc. International Workshop on Requirements Engineering for Product Lines, Co-located with the IEEE Joint International Requirements Engineering Conference (RE02), Essen, Germany.

[48] Sinnema, M., Deelstra, S., Nijhuis, J., Bos, J. (2006). *Modeling Dependencies in Product Families with COVAMOF*, 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06), IEEE.

[49] Czarnecki, K., Kim, C.(2005).*Cardinality-based feature modeling and constraints: A Progress Report*, Proc. of the International Workshop on Software Factories at OOPSLA05, San Diego California, USA.

[50] Gallier, J.H. (2003). Logic for Computer Science Foundations Of Automatic Theorem Proving, University of Pennsylvania, Philadelphia, Pa, USA.

[51] Bilaniuk, S. (2003), *A Problem Course in Mathematical Logic*, a text for a problem-oriented course on mathematical logic and computability, 1991 Mathematics Subject Classification. 03, Department of Mathematics Trent University, Peterborough Ontario Canada.

[52] Copi, I.M., Cohen., C.(2009), *Introduction to Logic*, Pearson Prentice Hall, New Jersey, USA.

[53] Sun, J., Zhang, H., Li, Y.F., Wang, H. (2005), *Formal Semantics and Verification for Feature Modeling*, the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS05), Shanghai, China.

[54] Mannion, M. (2002). *Using first-order logic for product line model validation*, the Second Software Product Line Conference SPLC2, San Diego CA,USA.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[55] Mannion, M., Camara. J.(2003), *Theorem proving for product line model verification*, In Software Product-Family Engineering(PFE), volume 3014 of Lecture Notes in ComputerScience, pages 211–224. Springer–Verlag, Germany.

[56]Zhang, W., Zhao, H., Mei, H. (2004). *A propositional logic-based method for verification of feature models*, the 6th International Conference on Formal Engineering Methods ICFEM04, LNCS 3308, pp.115-130.

[57] Zhang, W., Mei, H., Zhao, H.(2006), Feature-driven requirement dependency analysis and high-level software design. Requirements Engineering, 11(3), Springer London, PP.205–220.

[58] Batory, D (2005). *Feature Models, Grammars, and Propositional Formulas*, the 9th International Software Product Lines Conference SPLC05, Rennes France.

[59] Gheyi, R., Massoni, T., Borba, P. (2006), *A Theory for Feature Models in Alloy*, the ACM SIGSOFY First Alloy Workshop, Portland, United States, PP. 71-80.

[60] Gheyi, R., Massoni, T., Borba, P. (2008), *Algebraic laws for feature models*. Journal of Universal Computer Science, 14(21):3573–3591.

[61] Storm, van der. T. (2004), Variability and component composition, In Software Reuse: Methods, Techniques and Tools: 8th International Conference, ICSR 2004. Proceedings, volume 3107 of Lecutre Notes in Computer Sciences, Springer, Berlin Heidelberg, Germany, pp. 157–166.

[62] Storm, van der. T. (2007), Generic feature-based software composition, In Software Composition, volume 4829 of Lecture Notes in Computer Sciences, Springer–Verlag, pp. 66–80.

[63] Zhang, W., Yan, H., Zhao, H., Jin, Z. (2008), *A bdd-based approach to verifying clone-enabled feature models'constraints and customization*, In High Confidence Software Reuse in Large Systems, 10th International Conference on Software Reuse, ICSR, Proceedings, volume 5030 of Lecture Notes in Computer Sciences, Springer–Verlag, pages 186–199.

[64] Hotez, L., Krebs, T. (2003). Supporting the Product Derivation Process with A Knowledge Base Approach, the 25th International Conference on Software Engineering ICSE2003, Portland Oregon USA.

[65] Hotez, L., Krebs, T. (2003). A Knowledge Based Product Derivation Process and Some Idea How to Integrate Product Development, the Software Variability Management Workshop, Groningen The Netherlands.

[66] Hemakumar, A. (2008). *Finding Contradictions in Feature Models*, First International Workshop on Analyses of Software Product Lines (ASPL'08)', collocated with SPLC08. Limerick, Ireland.

[67] Yan, H., Zhang, W., Zhao, H., Mei. H. (2009), *An optimization strategy to feature models' verification by eliminating verification-irrelevant features and constraints*. In 11th International Conference on Software Reuse(ICSR 2009), Falls Church, VA, USA , PP. 65–75.

[68]Mendonca, M., Oliveira,T., Cowan, D. (2006). Collaborative and Coordinated Product Configuration, In international Software Product Line Conference (SPLC06) , Doctoral Symposium, Baltimore, Maryland, USA.

[69] Mendonca, M., Cowan, D.D.(2007). Support for Collaborative Feature-Based Product Configuration in Software Product Lines, Technical Report CS-2007-030, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

[70] Mendonca, M, Bartolomei, T.T., Cowan, D. (2008a). Decision-Making Coordination in Collaborative

Product Configuration, 23rd Annual ACM Symposium on Applied Computing, Fortaleza, Ceará, Brazil.

[71] Mendonc, M., Cowan, D., Malyk, W., Oliveira, T.(2008b), Collaborative product configuration: Formalization and efficient algorithms for dependency analysis. Journal of Software, 3(2), PP.69–82.

[72] Salinesi, C., Rolland, C., Mazo, R.(2009), *Vmware: Tool support for automatic verification of structural and semantic correctness in product line models*. In Third International Workshop on Variability Modelling of Software-intensive Systems (VaMoS09), Sevilla Spain, PP.173–176.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[73] Elfaki, A., Phon-Amnuaisuk, S., Ho, C.K.(2008). *Knowledge Based Method to Validate Feature Models*, first International Workshop on Analyses of Software Product Lines (ASPL'08)', collocated with SPLC08, Limerick, Ireland.

[74] Elfaki, A., Phon-Amnuaisuk, S., Ho, C.K. (2009). *Using First Order Logic to Validate Feature Model*, Proc. of the third International Workshop on variability modeling of Software-intensive Systems, Sevilla Spain.

[75] Elfaki, A., Phon-Amnuaisuk, S., Ho, C.K. (2010). *An Interactive Method for Validating Stage configuration***,** journal of software engineering and application, volume 3, number 6,pp. 614-627.

[76] Elfaki, A., Phon-Amnuaisuk, S., Ho, C.K. (2009). *Investigating Inconsistency Detection as a Validation Operation in Software Product Line*, Book chapter in studies in computational intelligence, pp. 159-168, Berlin / Heidelberg Springer.

[77] Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., (2003), *The description logic handbook:theory, implementation, and applications*. Cambridge University Press, New York, NY, USA.

[78] Falbo, de Almeida., Guizzardi, G., Duarte, K.C., (2002),*An Ontological Approach to Domain Engineering*, 4th international conference on Software engineering and knowledge engineering, Ischia, Italy. ACM, PP. 15-19.

[79]] Peng, X., Zhao, W., Xue, Y., Wu, Y.(2006), *Ontology-Based Feature Modeling and Application-Oriented Tailoring*, 9th International Conference on Software Reuse (ICSR 2006), Turin, Italy, LNCS 4039, Springer-Verlag Berlin Heidelberg, pp. 87 – 100.

[80] Czarnecki, K., Kim, C., Kalleberg, K.,(2006), *Feature Models are Views on Ontologies*, 10th International Software Product Line Conference (SPLC'06), Baltimore, Maryland, USA.

[81] Fan, S., Zhang, N. (2006), *Feature model based on description logics*. In Knowledge-Based Intelligent Information and Engineering Systems, 10th International Conference, KES, Part II, volume 4252 of Lecture Notes in Computer Sciences. Springer–Verlag.

[82] Asikainen, T., Mnnistand, T., Soininen, T. (2007), *Kumbang: A domain ontology for modelling variability in software product families*, Advanced Engineering Informatics, Volume 21, Issue 1, Elsevier, PP. 23-40.

[83] Wang, H., Li, Y.F., Sun, J., Zhang, H., Pan, J. (2005). *A Semantic Web Approach to Feature Modeling and Verification*, workshop on Semantic Web Enabled Software Engineering (SWESE'05), Galway, Ireland.

[84] Asikainen, T., Männistö, T., Soininen, T. (2004). Using *a Configurator for Modelling and Configuring Software Product Lines Based on Feature Models*, In the Workshop on Software Variability Management for Product Derivation, software Product Line Conference (SPLC3), Boston, USA.

[85] Dedeban,V. (2007). Ontology-*driven and Rules-based System for Management and Pricing of Family of Product.* Master Thesis, Norwegian University of Science and Technology Department of Computer and Information Science, Norway.

[86]] Kaviani, N., Mohabbati, B., Gasevic, D., Finke, M. (2008), *Semantic Annotations of Feature Models for Dynamic Product Configuration in Ubiquitous Environments*, the 4th International Workshop on Semantic Web Enabled Software Engineering at 7th International Semantic Web Conference, Karlsruhe, Germany.

[87] AboZaid, L., Kleinermann, F., De Troyer. O.(2009), *Applying semantic web technology to feature modeling*. In *SAC '09*, Proceedings of the 2009 ACM symposium on Applied Computing, ACM, , New York, NY, USA, PP. 1252–1256.

[88] Benavides, D., Ruiz-Cort´es, A., Trinidad, P.(2004) *Coping with automatic reasoning on software product lines.* In Proceedings of the 2nd Groningen Workshop on Software Variability Management, Groningen, The Netherlands.

[89] Karataş, A. et al. *Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains*. SPLC 2010, LNCS 6287, pp. 286–299, 2010.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[90]Trinidad, P., Benavides, D., Ruiz-Cort´es, A. (2006).*Isolated Features Detection in Feature Models*, in Advanced Information Systems Engineering, 18th International Conference, CAiSE2006,short Paper Proceedings, Luxembourg, Luxembourg.

[91] Trinidad, P., Benavides, D., Dur´an, A.,Ruiz-Cort´es, A., Toro, M. (2008), *Automated error analysis for the agilization of feature modeling*. Journal of Systems and Software, 81(6), PP.883–896.

[92] Trinidad, P., Benavides, D., Ruiz-Cort´es, A.(2006), *A first step detecting inconsistencies in feature models*. In CAiSE Short Paper Proceedings, Advanced Information Systems Engineering, 18th International Conference, CAiSE 2006, Luxembourg, Luxembourg.

[93] Trinidad, P., Benavides, D., Dur´an, A.,Ruiz-Cort´es, A., Toro, M. (2008a), *Automated error analysis for the agilization of feature modeling*. Journal of Systems and Software, 81(6), PP.883–896.

[94] White, J., Schmidt, D., Benvides, D., Trinidad, P., Ruiz-Cortes, A. (2008). *Automated Diagnosis of Product Line Configuration Errors on Feature Models*, the 12th international conference of software product line, Limerick Ireland.

[95] White, J. , Dougherty, B., Schmidt, D., Benavides, D. (2009), *Automated Reasoning for Multi-step Feature Model Configuration Problems*, 13th International Software Product Line Conference, San Francisco, California, USA, pp. 11-20.

[96] Djebbi, O., Salinesi, C., Diaz, D. (2007), *Deriving product line requirements: the red-pl guidance approach*. In 14th Asia-Pacific Software Engineering Conference (APSEC), Los Alamitos, CA, USA, IEEE Computer Society., pp. 494-501.

[97] Mernik, M., Heering, J., Sloane, A. M. (2005), *When and how to develop domain-specific languages*. ACM Computing Surveys, 37(4), PP.316–344.

[98]Cao, F., Bryant, R., Carol, B. (2003). *Automating Feature-Oriented Domain Analysis*, the International Conference on Software Engineering Research and Practice (SERP'03), Las Vegas, Nevada, USA,pp. 944–949.

[99]Deursen, A., Klint, P. (2002). Domain–Specific Language Design Requires Feature Descriptions, Journal of Computing and Information Technology, 10(1):1–17.

[100]Klint, P. (1993). *A meta-environment for Generating Programming Environments*, ACM Trans, Softw. Eng. Methodol, 2(2):pp.176–201.

[101]Pohjalainen, P. (2008). *Feature Oriented Domain Analysis Expressions*, in Nordic Workshop on Model Driven Software Engineering(NW-MoDE'08), Reykjavik, Iceland.

[102] Groher, I., Voelter, M. (2007), *Expressing Feature-Based Variability in Structural Models*, Workshop on Managing Variability for Software Product Lines, Co-located with the 7th software product line conference SPLC, Kyoto, Japan.

[103] Cunningham, A.L.(2006), *Language, Deals and Standards: The Future of XML Contracts*, Journal of Washington University Law Review., VOL 84, No 2, Washington University School of Law, USA, PP. 313-374.

[104]Cechticky, V., Pasetti, A., Rohlik, O., Schaufelberger, W. (2004). XML-Based Feature Modelling, in the 8th International Conference on Software Reuse (ICSR-8), Madrid, Spain.

[105] Wong, T., Jarzabek, S., Swe , S.M., Shen, R., Zhang, H. (2001), *XML implementation of frame processor*, *Symposium on Software Reusability*, *SSR'01*, Toronto, Canada, pp. 164-172.

[106] Jarzabek, S., Zhang, H. (2001). XML-based Method and Tool for Handling Variant Requirements in Domain Models, 5th IEEE International Symposium on Requirements Engineering RE01, pp. 116-173, Toronto, Canada.

[107]Swe S.M., Zhang, H., Jarzabek, S. (2002), *XVCL: a tutorial*, Proc. of 14th Int. Conf. on Software Engineeringand Knowledge Engineering, SEKE'02, Italy, ACM Press, pp. 341-349.

[108] Zhang, H., Jarzabek, S.(2003), An XVCL-based Approach to Software Product Line Development, Int. Conf. on Software Engineering and Knowledge Engineering, SEKE'03, San Francisco Bay, USA.

[109] Zhang, H., Jarzabek, S.(2004), XVCL: a mechanism for handling variants in software product lines, Science of Computer Programming, Volume 53 , Issue 3, Special issue: Software variability management PP. 381 – 407.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[110]Peter, A. (2002). An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof, 2nd ed, Kluwer Academic Publishers.

[111] Janota, M. ,Kiniry, J. (2007). Reasoning about Feature Models in Higher-Order Logic, the 11th International Software Product Line Conference (SPLC07), Heidelberg, Germany.

[112] Lengyel, L., Levendovszky, T., Charaf, H. (2004). Constraint Handling in Feature Models, 5th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest.

[113] Broek, P., Galvão, I., Noppen, J. (2008).Elimination of Constraints from Feature Trees, first international Workshop on Analyses of Software Product Lines (ASPL'08)', collocated with SPLC08. Limerick, Ireland.

[114] Broek, P., Galvao, I. (2009), Analysis of feature models using generalized feature trees. In Third International Workshop on Variability Modelling of Software-intensive Systems (VaMoS09) Spain, number 29 in ICB-Research Report, pages 29–35, Essen, Germany, January 2009. Universit¨at Duisburg-Essen.

[115] Weyns, D., Helleboogh, A., Holvoet, T., Schelfthout, K., Betsbrugge, W., 2008. Towards a Software Product Line for Automated Transportation Systems. In the 2nd International Workshop on Dynamic Software Product Lines (DSPL 2008) collocated with SPLC08, pp.45-52,Limerick, Ireland.

[116] La Rosa, M., Aalst, W., Dumas, M., Hofstede, A., 2009. Questionnaire-based variability modeling for system configuration. J. Software and Systems Modeling, 8(2), Springer, pp. 251-274.

[117] Aalst, W.M.P., Dumas, M., Gottschalk ,F., Hofstede, A.H.M., La Rosa, M., Mendling, J., (2008), *Correctness-Preserving Configuration of Business Process Models*, Book chapter in Fundamental Approaches to Software Engineering,pp. 46-61,Springer Berlin / Heidelberg,Volume 4961/2008.

[118] Yang,Y., Peng, X., Zhao,W. (2007), A Feature-Oriented Adaptive Component Model for Dynamic Evolution, In the 11th European Conference on Software Maintenance and Reengineering, Amsterdam, the Netherlands, pp.49-57.

[119] Yang Y., Peng, X., Zhao, W. (2008), Feature-Oriented Software Product Line Design and Implementation Based on Adaptive Component Model, The First Workshop on Domain Specific Analysis and Design for Reuse, held with the 10th International Conference on Software Reuse(ICSR2008), Beijing China.

[120] Maxime Cordy, Andreas Classen, Patrick Heymans, Axel Legay, Simulation-Based Abstractions for Software Product-Line Model Checking, In 34th International Conference on Software Engineering, ICSE 2012,, Zurich, Switzerland, 2012.

[121] Ebrahim Bagheri, Tommaso Di Noia, Dragan Gasevic, Azzurra Ragone, Formalizing Interactive Stage Feature Model Configuration, Journal of Software Maintenance and Evolution: Research and Practice, 2010.

[122] Trinidad, P., David Benavides, D., Ruiz-Cort, A., Segura, S., Jimenez, A. (2008b). *FAMA Framework*, 2008 12th International Software Product Line Conference, Limerick Ireland.