IIT , 1–35 ()

Formalizing Interactive Staged Feature Model Configuration

EBRAHIM BAGHERI, TOMMASO DI NOIA, DRAGAN GASEVIC, AZZURRA RAGONE

ebrahim.bagheri@nrc-cnrc.gc.ca,t.dinoia@poliba.it,dgasevic@athabascau.ca,ragone@disi.unitn.it

Abstract. Feature modeling is an attractive technique for capturing commonality as well as variability within an application domain for generative programming and software product line engineering. Feature models symbolize an overarching representation of the possible application configuration space, and can hence be customized based on specific domain requirements and stakeholder goals. Most interactive or semi-automated feature model customization processes neglect the need to have a holistic approach towards the integration and satisfaction of the stakeholder's soft and hard constraints, and the application-domain integrity constraints. In this paper, we will show how the structure and constraints of a feature model can be modeled uniformly through Propositional Logic extended with concrete domains, called $\mathcal{P}(\mathcal{N})$. Furthermore, we formalize the representation of soft constraints in fuzzy $\mathcal{P}(\mathcal{N})$ and explain how semi-automated feature model customization is performed in this setting. The model configuration derivation process that we propose respects the soundness and completeness properties.

Keywords: Feature models, Variability, Software Product Lines, Soft Constraints, Logic languages

1. Introduction

Software product family engineering is concerned with capturing the commonalities, universal and shared attributes of a set of software-intensive applications for a specific problem domain [1]. It allows for the rapid development of variants of a domain specific application through various configurations of a common set of reusable assets often known as *core assets*, which support the management of commonality as well as variability [2]. On the one hand,*commonality* is supported by providing domain analysts with the ability and the required tools for capturing all of the shared conceptual information with the applications of a given domain. On the other hand, *variability* is addressed by allowing the domain modelers to include application specific attributes and features within their unified model but at the same time restrict their use; this way, commonality and variability are handled simultaneously. In this context, *feature modeling* is an important conceptual technique for modeling the encompassing attributes of a family of systems [3]. It provides for addressing commonality and variability both formally and graphically, allows for the description of interdependencies of the product family attributes (features) and the expression of the permissible variants and configurations of the product family. By reusing domain assets as a part of the feature model configuration process, the application development team will be able to offer a new and more reliable product of that family in a shorter time at a lower cost.

Due to its perceived benefits, feature modeling has received great attention both in industry and academia since its inception in the early 90's by the Software Engineering Institute. The definition of its concepts originated from the Feature Oriented Domain Analysis (FODA) methodology [4]. As a core value for developing a product family, FODA proposed that the abstraction away of all distinguishing factors be performed until no difference between the applications of a given problem domain exists, i.e., a generic all encompassing model, which is able to be specialized without the need for further expansion to represent different forms of a domain product family be created. Other methodologies have been gradually developed on this basis such as Organization Domain Modeling (ODM) [5], FeatuRSEB [6], Feature-Oriented Reuse Method (FORM) [7] and Cardinality-based feature modeling [8].

Various success stories in using product families and associated techniques have been reported. As an example, Clements and Northrop [9] reported that Nokia was able to increase their production capacity for new cellular phone models from 5-10 to around 30 per year. The main challenge that Nokia faced was that as the market demand and customer taste changed rapidly, different products with varying specifications and user interfaces needed to be developed to keep up with the requests. The use of product family engineering made it possible to quickly and efficiently create the required software and hardware variations for the different products and use variability to customize the newly configured cellular phones. More recently, Microsoft has also become interested in software product families. It has incorporated the feature modeling process at the front end of its Software Factory life cycle to provide support for software product family engineering, which has been applied to it's Health Level 7 (HL7) [10] software factory initiative. This initiative employs feature modeling to make the rapid design and development of HL7 conformant applications easier and more accessible to health informatics practitioners.

In light of these large-scale industrial applications of product families, the desire to create an overarching feature model for a product family entails the development of very large feature models that need to be customized before they can be used for a specific application in a given domain. In order to develop an instance of a product family from the relevant feature model, its most desirable features need to be selected from the feasible configuration space of the product family, whose entirety may not be of interest for a specific application at hand. The selection of the best set of features for a product would be based on the strategic goals, requirements and limitations of the stakeholders, as well as the integrity constraints of the feature model. Once the desired features are specified, the feature model can be customized such that it includes the wanted features and excludes the non-relevant ones. A final fully-specific feature model with no points for further customization is called a *configuration* of the feature model based on the selected features. In many cases, a configuration is gradually developed in several stages.

In each stage, a subset of the preferred features are selected and finalized and the unnecessary features are discarded yielding a new feature model whose set of possible configurations are a subset of the original feature model. This feature model is referred to as a *specialization* of the former feature model and the staged refinement process constitutes *staged configuration* [11]. Czarnecki et al. have promoted the idea of staged configuration due to its effectiveness in contexts that are engaged with the notions of 1) *time*, i.e., different phases in the production life cycle; 2) *roles* that represent different perspectives on the product such as security, reliability, etc.; and 3) multiple *target deployment markets* for the product family.

1.1. Problem Challenges

Despite the effectiveness of the staged configuration process, it is still very hard to manually create appropriate configurations from large industrial-scale feature models. The reason for this is multifold:

1. In a large feature model, it is infeasible for a group of experts to keep track of all the mutual interdependencies of the features; therefore, understanding the implications of a feature selection decision becomes very difficult. In other words, selecting the feature that would maximize the satisfaction of the stakeholders and at the same time minimize the unintended consequences is both important and complicated;

2. Understanding the requirements and needs of the stakeholders and attempting to satisfy them simultaneously can be viewed as a complex constraint satisfaction problem. In cases where the stakeholders have multiple requests, ensuring that all of the requests have been satisfied and keeping track of how they have been addressed is a complex process;

3. A feature model may be employed by designers other than its original developers, and since different viewpoints may exist while interpreting a single feature model, ensuring that a correct and shared understanding of the features and their implications exists requires formal specification and semantic annotation of the feature model that would guarantee a unique interpretation. Besides enhancing common understanding, a formal specification of feature models along with semantic annotations would also provide grounds for addressing *soft constraints*. We use the notion of soft constraints to refer to service level agreements, non-functional requirements and quality attributes;

4. Consistency checking and verification of a given configuration for a feature model is very time consuming (with high computational complexity) and error prone on large-scale feature models. This is due to the need for performing many cross-reference integrity and stakeholder constraint checks on the developed configuration. The configuration needs to be checked against the feature model constraints to see whether it respects their enforcement in terms of the inclusion of all mandatory features and the exclusion of undesired features, and should also be verified with regards to the stakeholders stated requirements and restrictions.

Here, we attempt to show how a semi-automated approach to feature model configuration (*interactive configuration*), based on a fuzzy propositional language $\mathcal{P}(\mathcal{N})$ [12], which is employed for modeling the structure of a feature model and its related integrity and stakeholder constraints, is able to address the aforementioned concerns.

1.2. Contributions and Overview

The work in this paper attempts to create an interactive feature model configuration process. Most of the interactive configuration procedures in the literature mainly focus on satisfying and analyzing the stakeholders' hard constraints and also validating the consistency of a developed feature model configuration. Therefore, decision making in tradeoff situations where the choice between competing features[1] needs to be made cannot be formally supported by these approaches. It is important to take a holistic approach towards the integration and satisfaction of the stakeholder's soft and hard constraints, and the application-domain integrity constraints, which would assist the modelers in making the best feature selection decisions based on the stated soft and hard constraints of the stakeholders and be able to conveniently keep track of the feature model specialization process and also verify the validity of its results. Specifically, as the main contributions we show how:

1. $\mathcal{P}(\mathcal{N})$ is a suitable language to express the structure and the integrity constraints of a feature model, as well as the hard constraints of the stakeholders;
2. The soft constraints (preferences) of the stakeholders are represented using a fuzzy extension of $\mathcal{P}(\mathcal{N})$, which would allow for a more relaxed reasoning procedure;
3. Formal and fuzzy reasoning techniques are employed to develop a sound and complete interactive feature model configuration process.

More informally, the work proposed in this paper has the following high level description: Many stakeholders of a software product line have competing and at times inconsistent needs and requirements. Some of these requirements are not in the form of inclusion or exclusion of specific features within the product line but are rather in the form of quality attributes that need to be addressed. For this reason, we propose to annotate feature models with additional information in the form of fuzzy variables that represent the possible and relevant quality attributes. In this way, the fuzzy variables will allow us understand to what extent each feature is able to satisfy a given quality attribute and is therefore suitable to be selected based on the given requirements of the stakeholders. We benefit from fuzzy $\mathcal{P}(\mathcal{N})$ to model and reason over the quality attributes represented using fuzzy variable in our formal interpretation of feature models. Our approach, will consider the stakeholders feature selections (called hard constraints in this paper) along with their preferred quality attributes (referred to as soft constraints) and will interactively develop a final feature model configuration based on these provided information.

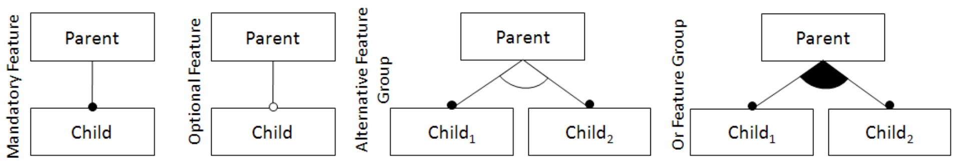

Figure 1. The main graphical notations employed in feature modeling.

The remainder of this paper is organized as follows: Section 2 provides an overview of the main concepts of feature modeling along with an introduction to the extended version of the sample Graph Product Line feature model provided in [13]. Then, $\mathcal{P}(\mathcal{N})$ and its fuzzy extension are introduced. In Section 4, our interactive feature model configuration process is formalized, followed by a case study in Section 5. Related work on techniques for feature model configuration are reviewed in Section 6. Finally, the paper is concluded with some closing remarks.

2. Feature Modeling

Features are important distinguishing aspects, qualities, or characteristics of a family of systems [14]. They are widely used for depicting the shared structure and behavior of a set of similar systems. To form a product family, all the various features of a set of similar/related systems are composed into a feature model. A feature model is a means for representing the possible configuration space of all the products of a system product family in terms of its features. For this reason, it is important that feature models are able to capture variability and commonality between the features of the different applications available in a given domain. As we will see in the following paragraphs in this section, feature models provide suitable means for modeling commonality, by allowing the domain modelers to form a common feature model representation for multiple applications, as well as variability by providing means to capture competing features of different applications under one unified umbrella. An example of capturing commonality is when a similar feature that exists in multiple applications is represented as unique feature in the overall domain representation, while an example of variability is when one notion is viewed differently by separate applications and is therefore modeled using competing features.

Feature models can be represented both formally and graphically; however, the graphical notation depicted through a tree structure is more favored due to its visual appeal and easier understanding. More specifically, graphical feature models are in the form of a tree whose root node represents a domain concept, e.g., a domain application, and the other nodes and leafs illustrate the features. In this context, a feature is a concept property related to a user-visible functional or nonfunctional requirement, e.g., domain application task, modeled in a way to capture commonalities or possibly differentiate among product family variants.

In a feature model, features are hierarchically organized and can be typically classified as:

- *Mandatory*, the feature must be included in the description of its parent feature;
- *Optional*, the feature may or may not be included in its parent description given the situation;
- *Alternative feature group*, one and only one of the features from the feature group can be included in the parent description;
- *Or feature group*, one or more features from the feature group can be included in the description of the parent feature.

Figure 1 depicts the graphical notation of the feature relationships. The tree structure of feature models falls short at fully representing the complete set of mutual interdependencies of features; therefore, additional constraints are often added to feature models and are referred to as *Integrity Constraints (IC)*. The two most widely used integrity constraints are:

- *Includes*, the presence of a given feature (set of features) requires the *existence* of another feature (set of features);
- *Excludes*, the presence of a given feature (set of features) requires the *elimination* of another feature (set of features).

In addition to this information, several feature modeling notations such as the cardinality-based feature models provide means for expressing the plausible cardinality of the features. Some of the possible cardinality values are $< n >, < n_1..n_2 >$[2], which show the acceptable number of feature occurrences in the composition of the parent feature.

2.1. The Graph Product Line Feature Model

Lopez-Herrejon and Batory have proposed the Graph Product Line (GPL) example as a standard problem for evaluating product line methodologies [13]. GPL covers the classical set of applications of graphs in the domain of Computer Science. The intention is to be able to develop configurations of GPL, which are able to address different problem domains. For instance, GPL can be configured to perform several graph search algorithms over a directed or undirected graph structure. Similar to other feature models, GPL can be configured by selecting the set of its features that are relevant to the problem statement on hand.

The graphical representation of GPL is shown in Figure 2. As it can be seen, it consists of three main features: 1) Graph Type: features for defining the structural representation of a graph; 2) Search: traversal algorithms in the form of features that allow for the navigation of a graph; 3) Algorithms: other useful algorithms that manipulate or analyze a given graph. Clearly, not all possible configurations of the features of GPL produce valid graph programs. For instance, a configuration

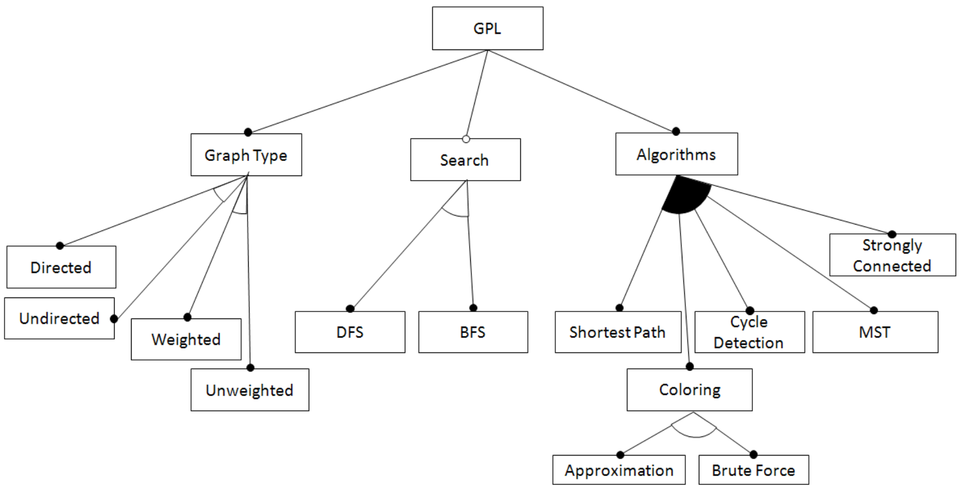

Figure 2. The graph product line feature model.

of GPL that checks if a graph is strongly connected cannot be implemented on an undirected graph structure. Such restrictions are expressed in the form of integrity constraints. Some examples of these constraints are:

$$\texttt{IC}=\begin{cases}\text{Cycle Detection } \textsc{excludes} \text{ BFS;}\\ \text{Cycle Detection } \textsc{includes} \text{ DFS;}\\ \text{Strongly Connected } \textsc{includes} \text{ DFS;}\\ \text{strongly Connected } \textsc{includes} \text{ Directed;}\\ \text{strongly Connected } \textsc{excludes} \text{ Undirected.}\end{cases}$$

Now based on the integrity constraints, a configuration such as {GPL, Graph Type, Directed, Undirected, Cycle Detection, DFS} is an acceptable specialization of GPL, but {GPL, Graph Type, Directed, Undirected, Cycle Detection, DFS, BFS} is not, since the BFS feature cannot occur when the Cycle Detection feature has been selected. Other constraints on GPL can be found in [13].

3. Formalism for Feature Modeling

There have been different approaches towards the formalization of feature models among which the use of pure propositional logic [15], description logic [16], and iterative tree grammars [17] are the most important ones.

Here we use Propositional Logic enriched with concrete domains as originally proposed in [12]. Interested readers can refer to [12] for more details.

Definition 1 (The language $\mathcal{P}(\mathcal{N})$) *Let $\mathcal{A}$ be a set of propositional atoms, and $\mathcal{F}$ a set of pairs $\langle f, D_f\rangle$ each made of a feature name and an associated concrete domain D_f, and let k be a value in D_f. Then the following formulas are in $\mathcal{P}(\mathcal{N})$:*

1. *every atom $A \in \mathcal{A}$ is a formula in $\mathcal{P}(\mathcal{N})$*
2. *if $\langle f, D_f\rangle \in \mathcal{F}$, $k \in D_f$, and $c \in \{\geq, \leq, >, <, =, \neq\}$ then (fck) is a formula in $\mathcal{P}(\mathcal{N})$*
3. *if ψ and φ are formulas in $\mathcal{P}(\mathcal{N})$ then $\neg\psi$, $\psi \wedge \varphi$ are formulas in $\mathcal{P}(\mathcal{N})$. We also use $\psi \vee \varphi$ as an abbreviation for $\neg(\neg\psi \wedge \neg\varphi)$, $\psi \rightarrow \varphi$ as an abbreviation for $\neg\psi \vee \varphi$, and $\psi \leftrightarrow \varphi$ as an abbreviation for $(\psi \rightarrow \varphi) \wedge (\varphi \rightarrow \psi)$.*

We call $\mathcal{L}_{\mathcal{A},\mathcal{F}}$ the set of formulas in $\mathcal{P}(\mathcal{N})$ built using $\mathcal{A}$ and $\mathcal{F}$. Moreover we call facts, all those singleton formulas containing only a single atom $A \in \mathcal{A}$ or a restriction over a concrete feature (fck).

In order to define a formal semantics of $\mathcal{P}(\mathcal{N})$ formulas, we consider interpretation functions $\mathcal{I}$ that map propositional atoms into $\{\mathsf{true}, \mathsf{false}\}$, feature names into values in their domain, and assign propositional values to numerical constraints and composite formulas according to the intended semantics.

Using $\mathcal{P}(\mathcal{N})$ we can easily represent IS-A and equivalence relations using pure Propositional Logic or involving concrete features. For example,

$$GraphType \wedge Algorithms \leftrightarrow GPL$$

is a $\mathcal{P}(\mathcal{N})$ formula that states that a graph product line configuration is equivalent to the configuration of both a graph type and an algorithm feature.

As it can be seen in $\mathcal{P}(\mathcal{N})$, rules are either satisfied and are true or are false otherwise. In order to be able to represent varying degrees of truthfulness over concrete features, a fuzzy extension of $\mathcal{P}(\mathcal{N})$ can be formulated.

Definition 2 (Fuzzy $\mathcal{P}(\mathcal{N})$) *The alphabet of Fuzzy $\mathcal{P}(\mathcal{N})$ is a tuple $\langle \mathcal{A}, \mathcal{F}, \mathcal{C}, \overline{\mu}\rangle$ where:*

- *$\mathcal{A} = \{A_i\}$ and $\mathcal{F} = \{f_j\}$ are defined as for $\mathcal{P}(\mathcal{N})$;*
- *$\mathcal{C} = \{cn_h\}$ is a set of attributes such that $\mathcal{F} \cap \mathcal{C} = \emptyset$;*
- *$\overline{\mu} = \{\mu^{A_i}_{cn_h}\}$ is a set of fuzzy membership functions.*

The following formulas are in fuzzy $\mathcal{P}(\mathcal{N})$:

1. *$cn_h = \mu^{A_i}_{cn_h}$;*
2. *$F \rightarrow \bigwedge_h cn_h = \mu^{F}_{cn_h} \wedge \psi$, with $\psi \in \mathcal{P}(\mathcal{N})$ and $F \in \mathcal{A}$;*

If a fuzzy $\mathcal{P}(\mathcal{N})$ formula is in the form 1 we call it fuzzy fact, if it is in the form 2 we call it fuzzy clause.

Some of the more widely used examples of fuzzy membership functions are the triangular, trapezoidal, and Gaussian functions. For each fuzzy predicate $\mu \in \overline{\mu}$ the

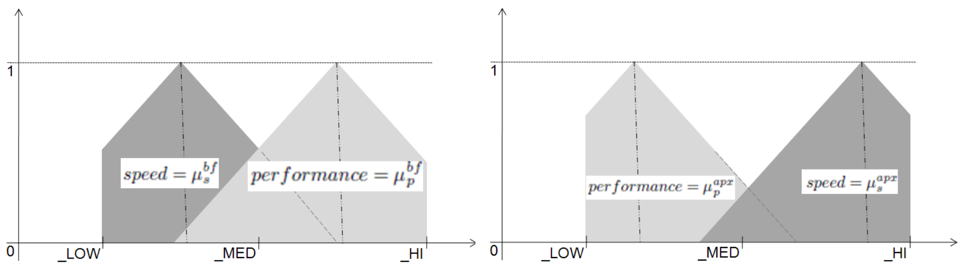

Figure 3. The Semantic annotation of a) *brute force* and b) *approximation* graph coloring features.

only restriction is $\mu \in [0, 1]$. For the sake of illustration and due to the simplicity of its representation, we use the triangular function $tri(d_1, d_2, a, b, c)$, where d_1 and d_2 are the domains of the function, a, b, c are the parameters throughout this paper.

In order to clarify the syntax of fuzzy $\mathcal{P}(\mathcal{N})$ formulas we represent the following two fuzzy clauses related to features of the graph product line:

$$Brute\_Force \rightarrow performance = \mu_p^{bf} \wedge speed = \mu_s^{bf} \wedge Coloring \quad (1)$$

$$Approximation \rightarrow performance = \mu_p^{apx} \wedge speed = \mu_s^{apx} \wedge Coloring \quad (2)$$

These two clauses show that *brute force* and *approximation* are two methods for graph coloring. Each of them has been annotated with information about its performance and speed. As mentioned earlier, the information regarding the annotation of the abstract concepts need to be provided by outside sources of information. Figure 3 depicts the fuzzy values of these two features. The membership functions on the left side of the figure basically show that the brute force technique for graph coloring is rather slow but has high performance in terms of accuracy. On the other side, the approximate technique for graph coloring has been described in terms of its higher speed and weaker performance. The ability to annotate abstract models is very important in feature modeling, due to the fact that feature models are abstract representations of a family of products where domain-specific information that would be possible unification options for the open variables of clauses in a first-order format have been removed from the models; therefore, reasoning is only feasible at the abstractions level represented in propositional form and their fuzzy annotations.

4. Feature Modeling in $\mathcal{P}(\mathcal{N})$

In this section, we will describe how the structure, constraints and semantic fuzzy annotations of feature models can be expressed in the introduced fuzzy $\mathcal{P}(\mathcal{N})$ language. We will explore the conversion process of mandatory and optional features, *or* and *alternative* feature groups and some cardinality expressions. The representation of *includes* and *excludes* integrity constraints will also be discussed. The representation of feature models in fuzzy $\mathcal{P}(\mathcal{N})$ would allow for the development of a formal interactive feature model configuration process.

4.1. Conceptual Modeling

The process of formalizing feature modeling information in our proposed structure entails the development of multiple separate knowledge bases, each of which represents a different aspect of a feature model. These knowledge bases are introduced as follows:

4.1.1. Structural Knowledge

The structural knowledge base ($\mathcal{SKB}$) consists of the description of the hierarchical feature relationships. The fundamental feature relationships introduced earlier in Section 2 are expressed in this knowledge base.

Mandatory a mandatory feature can be viewed as a prerequisite for the existence of its parent feature; therefore, in logical terms, it can be expressed as an equivalence formula:

$$Child \leftrightarrow Parent$$

In case multiple features are mandatory, they can be formulated by

$$Child_1 \wedge Child_2 \ldots \wedge Child_n \leftrightarrow Parent$$

which means that all of the child features should be present in the configuration of the parent feature and vice versa.

Optional optional features are free to appear or not in the formation of the parent feature; hence, both states of being false or true are acceptable:

$$Child \rightarrow Parent$$

however, it should be noted that due to the nature of optional features and the fact that they may not be present in the configuration of their parent feature, they often appear in conjunction with other sibling features.

Or feature group an or feature group allows the feature parent to become configured with the appearance of at least one of its child features:

$$Child_1 \vee Child_2 \vee \ldots \vee Child_n \leftrightarrow Parent$$

In Figure 2, the *Algorithms* feature can be represented as: $ShortestPath \vee Coloring \vee Cycle\_Detection \vee MST \vee Strongly\_Connected \leftrightarrow Algorithms$, which means that the algorithms feature can be achieved by the satisfaction of one or more of its child features.

Alternative feature group in an alternative feature group, features are represented in a mutually exclusive form, where the appearance of one feature eliminates the possibility of the other. Essentially, the logical relationship between the features is similar to the one we used for *or feature groups* with the addition of mutually disjunction axioms between siblings in the tree

$$\begin{aligned} Child_1 \vee Child_2 \vee \ldots \vee Child_n &\leftrightarrow Parent \\ Child_1 &\rightarrow \neg\, Child_2 \\ Child_1 &\rightarrow \neg\, Child_3 \\ &\ldots \\ Child_{n-1} &\rightarrow \neg\, Child_n \end{aligned}$$

This formulation shows that the elements of an alternative feature group cannot co-exist in the configuration of their parent feature. The *Search* feature can be represented as: $(Search \leftrightarrow BFS \vee DFS;\ BFS \rightarrow \neg DFS)$.

Cardinality there is no direct method to represent cardinality information in $\mathcal{P}(\mathcal{N})$ other than expanding the cardinality expressions, i.e., to repeat the literals to suit the cardinality restriction. As an example, assuming that a parent feature should have only exactly 2 of its child features in its configuration, we can express it as:

$$\begin{aligned} (Child_1 \wedge Child_2 \wedge \neg Child_3) &\ \vee \\ (Child_1 \wedge \neg Child_2 \wedge Child_3) &\ \vee \\ (\neg Child_1 \wedge Child_2 \wedge Child_3) &\ \leftrightarrow\ Parent \end{aligned}$$

Other forms of expansions for cardinality restrictions can be developed in the same way.

More complex clauses can be formulated based on these primitive relationships. As an example, the *Graph Type* feature is formalized by two separate mandatory alternative feature groups, which can be formulated as follows:

$$\left.\begin{aligned} &Directed \vee Undirected \leftrightarrow Direction \\ &Directed \rightarrow \neg Undirected \end{aligned}\right\} \text{Alternative feature group}$$

$$\left.\begin{aligned} &Weighted \vee Unweighted \leftrightarrow Weight \\ &Weight \rightarrow \neg Unweighted \end{aligned}\right\} \text{Alternative feature group}$$

$$\left. Weight \wedge Direction \leftrightarrow Graph\_Type \right\} \text{Mandatory}$$

The graph product line feature model can then be expressed as:

$Graph\_Type \wedge Algorithms \leftrightarrow GPL.$
$Search \rightarrow GPL.$

The information represented in this form are referred to as structural knowledge of the feature model and will be stored and referred to as $\mathcal{SKB}$.

4.1.2. Integrity Constraints Feature interdependencies that cannot be captured in structured hierarchical format are represented within the Integrity Constraints base ($\mathcal{IC}$).

Includes the presence of a given feature could entail the automatic inclusion (the need for presence) of one or more other features, which is represented as: ($BaseFeature \rightarrow DerivedFeature$). As an example in the graph product line, the selection of the *Cycle Detection* feature would cause the inclusion of *DFS*: ($CycleDetection \rightarrow DFS$)

Excludes the inclusion of one feature may result in the exclusion of one or more features, which is shown by: ($BaseFeature \rightarrow \neg ExcludedFeature$). Similar to the previous example, the inclusion of the *Cycle Detection* feature will require the removal of *BFS*: ($CycleDetection \rightarrow \neg BFS$).

All integrity constraints will be stored and represented through the integrity constraint base ($\mathcal{IC}$).

4.1.3. Utility Knowledge The selection of the correct features of a feature model in the configuration process of a specific product is based on the efficiency of the features to perform the required functional tasks as well as fulfill some of the non-functional requirements which are aligned with the strategic objectives of the product being developed. In order to be able to understand how each feature relates with the functional and non-functional requirements, we propose the annotation of features. For this reason, we adopt the concept of *concerns* from the Preview framework in the multiple-viewpoint requirement elicitation domain [18]. Concerns relate with the high-level strategic objectives of the specific application domain and the target product audience; therefore, they can be used to ensure consistency and alignment between the vital goals pursued by the design of a product and the product family features. Simply put, concerns are the desired business quality and required software quality attributes, which need to be considered through the staged configuration process.

Concerns are expressed at a high-level of abstraction to avoid overlap with some of the actual features of the product family. Examples of concerns can include but are not limited to cost, time, risk, volatility, customer importance, penalty, etc that can be employed based on the feature model on hand. For instance, *speed* and *performance* are the two concerns that have been used to annotate the features of GPL in Figure 3. They show that speed and performance are important decision making criteria in the graph product line configuration process. Now since concerns are abstract concepts, the degree of ability of a feature to satisfy a given concern can be expressed in a fuzzy form; therefore, the annotation of features with concerns and their corresponding degrees of satisfaction are shown through fuzzy $\mathcal{P}(\mathcal{N})$, and the collection of these information is referred to as the *utility knowledge base* ($\mathcal{UKB}$). Utility knowledge depict how various features of a given feature model relate with and to what extent they are able to satisfy the objectives of the configuration process for a specific product.

In our framework we represent $\mathcal{UKB}$ as a set of fuzzy clauses. Referring back to the GPL example and assuming that the important concerns for product configuration are speed and performance, the fuzzy propositional clauses shown in Equations (1) and (2) are the utility knowledge related to the *Brute Force* and *Approximation* graph coloring features. The information regarding the utility annotation of the feature model should be provided at design time by the domain and/or product family experts by providing statements such as *I believe the brute force graph coloring feature is rather slow (speed* $= \mu_s^{bf}$*) but has an acceptable performance (*$performance = \mu_p^{bf}$*)* or *I think that although the approximate graph coloring feature has a high execution speed (speed* $= \mu_s^{apx}$*), it has less accurate results in terms of performance (*$performance = \mu_p^{apx}$*).* Such statements can be easily converted into corresponding fuzzy $\mathcal{P}(\mathcal{N})$ clauses.

4.1.4. Stakeholder Requests Stakeholders and product developers often specify a set of basic features that they would like to see in the final product. For instance, in the GPL configuration process, they may require the inclusion or exclusion of the graph coloring feature. Such requirements and requests are referred to as *hard constraints.* The satisfaction of hard constraints is either feasible or not, which makes the configuration process based on hard constraints a crisp one. However, besides the hard constraints, the stakeholders may also specify their preferences over the defined concerns such as *high speed is very important*, or *lower performance is tolerable.* These kinds of requests are called the *soft constraints* or *preferences.* Stakeholders' hard and soft constraints are represented by $\mathcal{SR}_h$, and $\mathcal{SR}_s$, respectively. Similar to utility knowledge, soft constraints can also be represented using fuzzy $\mathcal{P}(\mathcal{N})$ facts, e.g., *high speed is very important* can be stated as $\mathcal{SR}_s(speed)$ = tri(_LOW, _HI, _MED, $\frac{\_MED+\_HI}{2}$, _HI), which is a triangular fuzzy membership function whose maximum is located at the $\frac{\_MED+\_HI}{2}$ point; therefore depicting the importance of speed in this case. In our framework we represent hard constraints as $\mathcal{P}(\mathcal{N})$ facts and soft constraints as fuzzy $\mathcal{P}(\mathcal{N})$ facts.

Summing up w.r.t. to the knowledge domain modeling we have the following formalization:

$\mathcal{SKB}$**:** The feature model structural knowledge is represented using $\mathcal{P}(\mathcal{N})$ axioms modeled as described in Section 4.1.1.

$\mathcal{IC}$**:** Integrity constraints are $\mathcal{P}(\mathcal{N})$ formulas as described in Section 4.1.2.

$\mathcal{UKB}$**:** Features utility knowledge, i.e. the degree of satisfaction of the quality attributes by each feature, is a set of fuzzy $\mathcal{P}(\mathcal{N})$ clauses involving concerns.

$\mathcal{SR}_h$**:** Stakeholder's hard constraints, i.e. the desired features for the final product, are $\mathcal{P}(\mathcal{N})$ facts.

$\mathcal{SR}_s$**:** Stakeholder's soft constraints (preferences), i.e. desired quality attributes, are fuzzy $\mathcal{P}(\mathcal{N})$ facts involving concerns.

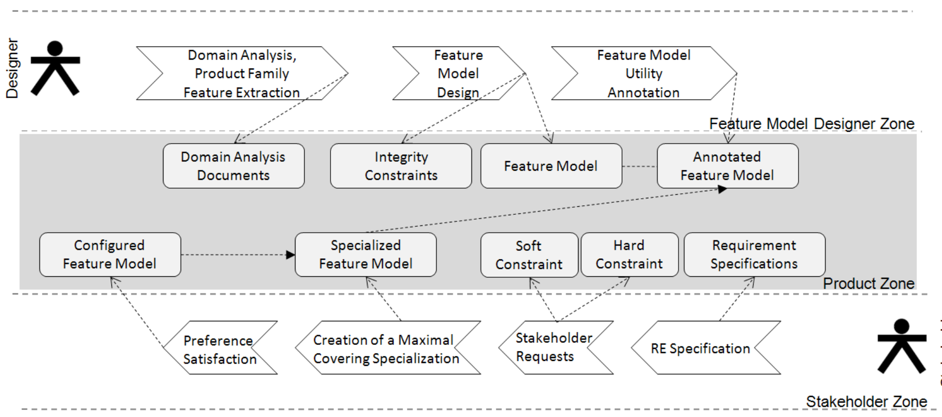

Figure 4. The overview of the interactive feature model configuration process.

4.2. Interactive Configuration

The overview of the interactive feature model configuration process is shown in Figure 4. As it can be seen, the feature model designers need to take three steps:

D1. Perform domain analysis to understand the set of all possible features and their interdependencies in the product family members. Different domain analysis methodologies exist that can be used for this purpose. Hess et al. have critically reviewed some of these techniques [19];

D2. Design a comprehensive feature model based on the result of the domain analysis that properly supports variability. This would include both the feature model and its accompanying integrity constraints;

D3. Annotate the features with appropriate utility knowledge. Such information would show how each feature can contribute to the satisfaction of the high-level abstract objectives of the problem domain. For instance in GPL and with the speed and performance concerns, the designers would need to show how each feature behaves with respect to these two concerns, e.g., *calculating the shortest path is both fast and accurate.*

Once an annotated feature model is provided, the annotation information can be used to reason about the suitability of various features of the feature model for a given purpose. For example, the features that have a slow execution process are not very suitable to be selected for an application that requires realtime performance. The annotation information can go hand in hand with the stakeholders hard and soft constraints to provide the means for an interactive configuration process, where requests are checked against the structure and annotations of the feature model to see which subset of the features are most suitable for the request on hand. In the context of the interactive configuration process, the stakeholders need to perform the following:

S1. Understand their expectations from the final product within the context of the feature model and clearly specify their requirements, most likely with the help of the model designers or requirement engineers;

S2. Differentiate between their hard constraints, which are vital for the target product, and their preferences;

S3. Develop and analyze the maximal covering specialization based on the stakeholders hard constraints and requests. In view of this specialization, the stakeholders might consider revising their requests to reach a more desirable specialization;

S4. Employ the feature recommendations based on the soft constraints to decide on the set of most appropriate remaining open features to fully configure the feature model. The feature recommendation process will guide the stakeholders towards the full configuration of the earlier developed specialized feature model.

As it can be seen in S1 to S4, the interactive configuration procedure benefits from two important steps. In the first important step (S3), the feature model is specialized based on the hard constraints of the stakeholders and a corresponding specialization is provided to the stakeholders, which can be useful for them to decide whether they want to change their selected set of hard constraints or not. If the hard constraints are changed, a new specialization is then developed. In the next step (S4), the remaining open features of the specialization are ordered based on their degree of contribution to the satisfaction of the stakeholders preferences. A rank-ordering is developed on this basis, and features are recommended to the stakeholders, accordingly. The stakeholders can interactively select the features they desire until the feature model is fully configured.

In the following sections, we will show how the hard and soft constraints of the stakeholders are matched with the structure, integrity constraints and annotation information of a given feature model and how they are satisfied.

4.2.1. Step 1: Hard Constraints Satisfaction It is important to satisfy the stakeholders' hard constraints before their soft constraints, since they represent the features that need to be present in the final product. For instance, let's suppose that in the graph product line the stakeholders request that the BFS and Cycle Detection features need to be present in the final product, while at the same time provide some soft constraints (quality attributes), e.g., that they prefer speed over performance. Ideally, the product configuration would need to satisfy all of these constraints, i.e., the hard and soft constraints. However, in reality since there are a lot of interaction and competition between the features, it is not always possible to satisfy all of the constraints simultaneously. In our approach, we have decided to consider and try to satisfy the hard constraints first and then move onto the soft constraints. To put this into perspective, suppose that a bicycle is being built and the buyer of the bicycle has ordered a bicycle with a *comfort saddle* and *V-Brakes* and has also mentioned that it prefers a light bicycle to a heavy fancy one. In this case, our first step would be to configure the bicycle in such a way that it has comfort saddle and V-brakes and then among the options available to us for the comfort saddle and the V-Brakes, we would consider those that are lighter in terms of weight. However, it would not be rational to first choose a configuration for the bicycle that is very light weight but does not have comfort saddle and V-brakes. Therefore, we first try to satisfy the stakeholders' hard constraints and then analyze their soft constraints.

There is also one additional consideration that we take into account and that is the fact that it is not always the case that the requests and requirements put forth by the stakeholders are consistent, i.e., the stakeholders might have requests for different features that cannot be satisfied simultaneously. For instance going back to the bicycle example, the buyer may request a bicycle that has V-brakes but does not have brake pads on it. As it can be seen, the two hard constraints that are given by the buyer are not mutually satisfiable (V-brakes need to be installed with brake pads) and therefore one of the constraints needs to be chosen over the other. Our strategy in dealing with this kind of situation, where conflicting and competing constraints are given, is to select and satisfy the constraint that would allow the satisfaction of the highest number of other hard constraints that are given by the stakeholders.

Let us provide some definitions to build the ground for the hard constraint satisfaction process.

Definition 3 *Let $c \in \mathcal{SR}_h$ be a hard constraint of the stakeholders, $\mathcal{IC}$, and $\mathcal{SKB}$ be the integrity constraints and structural knowledge of the feature model. The enforcement of c onto $\mathcal{SKB} \cup \mathcal{IC}$, will entail a set of facts called consequential facts of c, denoted $Cons(c)$. We define*

$$Cons(c) = \{c' \mid \mathcal{SKB} \cup \mathcal{IC} \cup \{c\} \models c'\}$$

For example, suppose that the *Cycle Detection* feature is a hard constraint of the stakeholders, which means that the stakeholders want to have this feature in their final product. Based on the structural knowledge of GPL and the integrity constraints we have $Cons(CycleDetection) = \{\neg BFS, DFS, CycleDetection\}$. A crucial implication of the entailed facts of the hard constraints is that the entailed facts of one hard constraint may be inconsistent with the other hard constraints provided by the stakeholders. In other words we might have that given two constraints $c_1, c_2 \in \mathcal{SR}_h$ there is a fact $\overline{c}$ such that both $\overline{c} \in Cons(c_1)$ and $\neg\overline{c} \in Cons(c_2)$. For instance, assume $\mathcal{SR}_h = \{CycleDetection, BFS\}$, then $\mathcal{SR}_h \cup \mathcal{SKB} \cup \mathcal{IC} \models \mathsf{false}$, which means that the consequential facts of this set of hard constraints are inconsistent. As a result, some of the hard constraints expressed by the stakeholders might be mutually exclusive making the simultaneous satisfaction of all such requests infeasible; therefore, the aim should be to maximize the number of satisfied hard constraints.

So in the context of Definition 2, with respect to our feature modeling problem, we are interested in computing both a *true/false* assignment to all the propositional variables in $\mathcal{A}$ and a numerical assignment to all the features in $\mathcal{F}$ such that the number of hard constraints (which are atomic propositional variables) assigned to *true* is maximal. In other words, we want to satisfy as much hard constraints as possible in $\mathcal{SR}_h$, given a structured knowledge base $\mathcal{SKB}$ and a set of integrity constraints $\mathcal{IC}$.

Formally, given a set of hard constraints $\mathcal{SR}_h$, the idea is to compute (satisfy) a partition (or all) of $\mathcal{SR}_h$ such that:

1. $\mathcal{SR}_h = \mathfrak{MCS} \cup \mathfrak{UT}$
2. $\mathfrak{MCS} \cap \mathfrak{UT} = \emptyset$
3. $\mathfrak{MCS} \cup \mathcal{SKB} \cup \mathcal{IC} \not\models \mathsf{false}$
4. There is no partition $\mathcal{SR}_h = \mathfrak{MCS}' \cup \mathfrak{UT}'$ such that both satisfies the above conditions and $\mathfrak{MCS} \subset \mathfrak{MCS}'$.

Informally explained, the above conditions state that

1. the goal is to partition the set of expressed hard constraints of the stakeholders into two sets, namely $\mathfrak{MCS}$ and $\mathfrak{UT}$ such that $\mathfrak{MCS}$ contains the maximum number of satisfiable hard constraints, and is called the maximal covering specialization, and $\mathfrak{UT}$ comprises of the set of unsatisfiable hard constraints;
2. the two developed sets should not have any overlapping members;
3. the developed maximal covering specialization needs to be consistent with the structural knowledge of the feature model and also the expressed integrity constraints;
4. the maximal covering specialization should be the largest possible covering set.

In order to solve the above problem we should compute all possible assignments $\mathcal{I}_k$ to variables in $\mathcal{A}$ and $\mathcal{F}$ such that $\mathcal{I}_k \models \mathcal{SKB} \cup \mathcal{IC}$ and find the assignment such that the number of variables in $\mathcal{SR}_h$ assigned to *true* is maximal. Algorithm 4.1 shows the computational procedure. The algorithm calls two functions:

ComputeNewAssignment($\mathcal{SR}_h, \mathcal{IC}, \mathcal{SKB}$)**.** This function returns a pair $\langle more, \mathcal{I} \rangle$ where *more* is a Boolean variable which is *true* if there are still assignments to be computed and *false* otherwise and $\mathcal{I}$ is a new assignment for variables in $\mathcal{A}$ and $\mathcal{F}$. The assignments made by COMPUTENEWASSIGNMENT observe the restrictions enforced by the feature model structural knowledge, integrity constraints and stakeholders hard constraints;

SatisfiedHardPrefernces($\mathcal{SR}_h, \mathcal{I}$)**.** Given a set of hard constraints $\mathcal{SR}_h$ and an assignment $\mathcal{I}$, the function returns the number of variables in $\mathcal{SR}_h$ assigned to *true* with respect to the assignment $\mathcal{I}$.

Algorithm 4.1: MaximalCoveringSpecialization($\mathcal{SR}_h, \mathcal{IC}, \mathcal{SKB}$)

```
$more = true$
$max = 0$
$\mathfrak{MCS} = \emptyset$
while $more = true$
  do { $\langle more, \mathcal{I} \rangle$ = ComputeNewAssignment($\mathcal{SR}_h, \mathcal{IC}, \mathcal{SKB}$)
       if $\mathcal{I} \models \mathcal{SKB} \cup \mathcal{IC}$
         then if SatisfiedHardPrefernces($\mathcal{SR}_h, \mathcal{I}$) $> max$
           then { $\mathcal{I}_{max} = \mathcal{I}$
                  $max$ = SatisfiedHardPrefernces($\mathcal{SR}_h, \mathcal{I}$)
for each $A_i \in \mathcal{A}$
  do { if $\mathcal{I}_{max} \models A_i$
         then $\mathfrak{MCS} = \mathfrak{MCS} \cup \{A_i\}$
         else $\mathfrak{UT} = \mathfrak{UT} \cup \{A_i\}$
return $(\mathcal{I}_{max}, \mathfrak{MCS}, \mathfrak{UT})$
```

Algorithm 4.1 computes all possible feature assignments based on the given feature model structural knowledge, integrity constraints and stakeholders hard constraints, and selects the one that satisfies the most number of stakeholder constraints. The algorithm returns this assignment as the maximal covering specialization, $\mathfrak{MCS}$, and the set of unsatisfied hard constraints in $\mathfrak{UT}$. Ultimately, $\mathfrak{MCS}$ contains the maximal covering specialization of the feature model based on the stakeholders' hard constraints ($\mathcal{SR}_h$), and $\mathfrak{UT}$ will contain the set of unsatisfiable hard constraints. Based on ($\mathfrak{MCS}$) and ($\mathfrak{UT}$), the stakeholders will be able to *interact* with the process by analyzing the resulting specialization of the feature model and deciding whether they would like to change some of their selections. If hard constraints are changed at this stage, the maximal covering specialization will be recalculated accordingly to reflect the new selections of the stakeholders.

It is possible to see from Algorithm 4.1 that the process of finding the maximal covering specialization for a given feature model is sound. A specialization process is sound iff the selected features in the final specialization are consistent with the integrity constraints and the structural knowledge of the feature model.

Theorem 1 (Soundness) *The maximum covering specialization* $\mathfrak{MCS}$ *computed by Algorithm 4.1 is the largest subset of* $\mathcal{SR}_h$ *such that the set of axioms* $\mathfrak{MCS} \cup \mathcal{SKB} \cup \mathcal{IC}$ *is consistent.*

Proof: The algorithm selects an assignment iff the condition $\mathcal{I} \models \mathcal{SKB} \cup \mathcal{IC}$ is satisfied. Among all these assignments it selects the one maximizing the number of hard constraints, *i.e.*, the propositional variables in $\mathcal{SR}_h$, such that their value for the assignment $\mathcal{I}_{max}$ is *true*; therefore, given the strict condition of $\mathcal{I} \models \mathcal{SKB} \cup \mathcal{IC}$, the algorithm is guaranteed to develop sound feature model configurations. ■

We also establish that the algorithm is complete, i.e., it will find a specialization that satisfies all of the stakeholders' hard constraints whenever one exists.

Theorem 2 (Completeness) *If $\mathfrak{MCS}$ is the maximum number of hard constraints that can be* true *at the same time, given $\mathcal{SKB}$ and $\mathcal{IC}$ then it is computed by Algorithm 4.1.*

Proof: The proof is quite straight. In fact, Algorithm 4.1 computes and checks all possible feature assignments. Hence, the algorithm evaluates all of the possible specializations of the feature model given $\mathcal{SBK}$, $\mathcal{IC}$ and $\mathcal{SR}_h$, eliminating the chance for missing a more optimal solution that has not been evaluated by the algorithm. Based on this $\mathcal{I}_{max}$ will be the feature assignment that satisfies the maximum possible number of stakeholder constraints. ■

Although Algorithm 4.1 computes the assignment we are looking for, it has a serious computational drawback. We have to **always** compute all possible interpretations (possible feature assignments). This leads to an exponential blow up. Indeed, given $\mathcal{A}$ and $\mathcal{F}$ we have a number of combinations equal to

$$2^{|\mathcal{A}|} \cdot \prod_{\langle f, D_f \rangle \in \mathcal{F}} |\Delta_c(D)|$$

where the first term represents all possible *true/false* assignments to propositional variables in $\mathcal{A}$ while the second term takes into account all possible values to be assigned to concrete features in $\mathcal{F}$.

We could be more efficient in the computation of $\mathfrak{MCS}$ and $\mathfrak{UI}$ if we consider our problem as an instance of MAX-WEIGHTED-SAT [20].

Definition 4 (max-weighted-sat) *Given a set of atoms $\mathcal{A}$, a propositional formula $\phi \in \mathcal{L}_{\mathcal{A}}$ and a weight function $\omega : \mathcal{A} \longrightarrow \mathbb{N}$, find a truth assignment satisfying ϕ such that the sum of the weights of true variables is maximum.*

We call MAXIMALCOVERINGSPECIALIZATION the instance of MAX-WEIGHTED-SAT problem defined in the following.

Definition 5 (MaximalCoveringSpecialization) *Given a set of hard constraints $\mathcal{SR}_h$, a structural knowledge base $\mathcal{SKB}$ and a set of integrity constraints $\mathcal{IC}$ find a truth assignment $\mathcal{I} \models \mathcal{SKB} \cup \mathcal{IC}$ such that the number of variables $A \in \mathcal{SR}_h$ with $A^{\mathcal{I}} = \mathsf{true}$ is maximum.*

In order to find the maximum number of satisfiable hard constraints in $\mathcal{SR}_h$ we transform our MAX-WEIGHTED-SAT problem into a corresponding Integer Linear Programming (ILP) problem, using the standard transformation of clauses into linear inequalities [21]. For the sake of clarity, here we present the procedure for propositional clauses, as introduced in [21], the interested reader could refer to [12] for the computation of such transformation when clauses are expressed in teh $\mathcal{P}(\mathcal{N})$ language.

In our ILP problem the function to maximize is the one referring to Stakeholder Requests $\mathcal{SR}_h$, as we want to maximize such preferences as much as possible. Hence, the function to be maximized is

$$sr_h = \sum_i x_i$$

where x_i is the corresponding binary variable for each $A_i \in \mathcal{SR}_h$. Therefore, given a solution of the optimization problem, if $x_i = 1$ this mean that the corresponding fact $A_i = true$, while if $x_i = 0$ this mean that the corresponding fact $A_i = false$.

The constraints of the optimization problems are the ones coming both from the structural knowledge base ($\mathcal{SKB}$) and the integrity constraints base ($\mathcal{IC}$). In order to have a set of constraints for our ILP[3] problem we have to encode clauses into linear inequalities, mapping each c_i occurring in a clause ϕ with a binary variable x_i. If c_i occurs negated in ϕ then $\neg c_i$ is substituted with $(1 - x_i)$, otherwise we will need to substitute c_i with x_i. After this rewriting it is easy to see that, considering $\vee$—logical *or*—as classical addition (and $\wedge$ as multiplication), in order to have a clause true the evaluation of the corresponding expression must be a value grater or equal to 1. The following example shows how this conversion is performed.

Example 1 *Consider the following propositional formula*

$$(c_1 \vee \neg c_2) \wedge (c_2 \vee \neg c_3)$$

following the procedure outlined before it will bring us to the two following linear inequalities:

$x_1 + (1 - x_2) \geq 1$

$x_2 + (1 - x_3) \geq 1$

and then we should add the conditions:

$x_j \in \{0, 1\}, j = 1 \ldots 3$

Regarding the complexity of our problem, it is known that MAX-WEIGHTED-SAT is NPO-complete [22], hence not approximable within any constant bound.

Theorem 3 MAXIMALCOVERINGSPECIALIZATION *is an NPO-complete problem.*

Proof:
The proof of this theorem is straightforward, as we explained before, MAXIMALCOVERINGSPECIALIZATION is an instance of MAX-WEIGHTED-SAT problem, and it is known that MAX-WEIGHTED-SAT is NPO-complete [22], hence not approximable within any constant bound. Hence also MAXIMALCOVERINGSPECIALIZATION is a NPO-complete problem. ■

We recall that with NPO we refer to the class of NP-complete Optimization (NPO) problems.

With this formalization, it is now easy to create a feature model specialization process based on a given set of stakeholder hard constraints, feature model structural knowledge and integrity constraints by solving this straightforward Integer Linear Problem. As outlined earlier, the stakeholders can go through a repetitive process of changing their expressed constraints and re-generating the feature model specialization until they are satisfied with the outcome.

4.2.2. Step 2: Soft Constraints Satisfaction In most cases, the maximal covering specialization of a feature model is not a complete model configuration, i.e., many unbound features remain in the specialization that need to be considered. The suitability of such features needs to be evaluated within the context of the stakeholders' preferences, which would allow for the identification of the features that more closely match the objectives of the stakeholders. For instance, if the stakeholders are looking for a fast algorithm, and two features are available with similar corresponding functionality, but one is fast and inaccurate and the other is slow but accurate, the former is selected in the context of such stakeholder preferences.

For this purpose, we have two pieces of valuable information at our disposal for reasoning over stakeholders' preferences, namely utility knowledge of the feature model ($\mathcal{UKB}$), and stakeholders' soft constraints ($\mathcal{SR}_s$). The utility knowledge of the features in the feature model that correspond with the quality attributes for the features are defined in the *domain engineering phase* of the software product line feature model development, i.e., the domain engineers are responsible for defining the quality attributes of the available features of the feature model while they are actually designing the feature model. For instance, they would need to explicitly define the utility knowledge (quality attribute information) for each feature and correctly assign it to that feature. On the other hand, the stakeholders' soft constraints, which correspond to the desired quality attributes by the stakeholders are defined in the *application engineering phase* when a request for a software product line customization is submitted by a group of stakeholders and a new application is being developed. In this context, the domain analysts are responsible for providing $\mathcal{UKB}$ at the domain engineering phase of software product line development and the stakeholders will provide their required $\mathcal{SR}_s$ in the application engineering phase. Given these two pieces of information, we are able to compare what is needed by the stakeholders as expressed in $\mathcal{SR}_s$, and what is provided by the software product line as defined in $\mathcal{UKB}$ to find the best matching set of features.

A feature would be more relevant to the stakeholders' objectives if its utility knowledge closely matches those requested in the stakeholders' soft constraints. For instance, assuming that the soft constraint of the stakeholders is to create a cheap software tool, and if we have two functionally-similar features that can be selected, the feature whose utility knowledge shows that its implementation is not costly will be the preferred feature. In order to be able to identify such features and align the soft constraints of the stakeholders with the utility knowledge of the feature model, we provide the following formalization.

We denote with *mcs* the conjunction of all the facts in $\mathfrak{MCS}$. Formally:

$$mcs = \bigwedge_{A_i^{MAX} \in \mathfrak{MCS}} A_i^{MAX}$$

Definition 6 *Let $f \in \mathcal{SKB}, f \notin \mathfrak{MCS}$ be a feature of the feature model not selected in the maximal covering specialization of the model and $\mathcal{UKB}_f : f \rightarrow \bigwedge_h cn_h = \mu^f_{cn_h} \wedge \psi$ the related fuzzy clause in $\mathcal{UKB}$ such that $\mathcal{UBK} \cup \mathcal{IC} \cup \{mcs\} \models \psi$. We denote with $\mathcal{UKB}^{cn}_f$ the utility annotation of feature f with respect to concern cn, and $\mathcal{SR}^{cn}_s$ the soft constraint of the stakeholders with regards to concern cn. The*

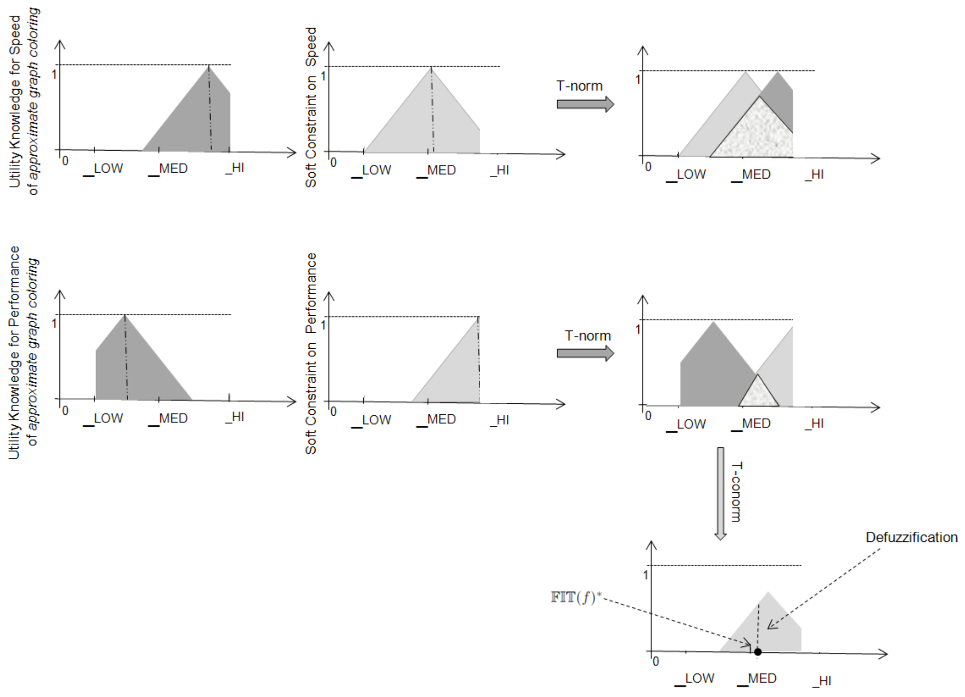
Utility Knowledge for Speed of approximate graph coloring
Soft Constraint on Speed
T-norm
Utility Knowledge for Performance of approximate graph coloring
Soft Constraint on Performance
T-norm
T-conorm
Defuzzification
FIT(f)*
1
0
_LOW
_MED
_HI

Figure 5. Computing the fitness of a feature f within the context of two concerns: speed and performance.

degree of fitness of f in the context of concern cn, denoted $\mathbb{FIT}(f, cn)$, is defined as follows:

$$\mathbb{FIT}(f, cn) = \mathcal{SR}_s^{cn} \otimes \mathcal{UKB}_f^{cn}.$$

where $\otimes$ is a fuzzy T-norm operator[4] *such as* product t-norm *or* minimum t-norm.

Now, since each feature is annotated with multiple concerns, we can interpret the information through Mamdani-type fuzzy inference [23] to calculate the fitness of a feature over all concerns (cn_i), denoted as $\mathbb{FIT}(f)$:

$$\mathbb{FIT}(f) = \bigoplus_{cn_i} \mathcal{SR}_s{}^{cn_i} \otimes \mathcal{UKB}_f{}^{cn_i}.$$

where $\oplus$ is a t-conorm operator. The reason for choosing Mamdani-type fuzzy inference can be explained by its fuzzy representation of both the antecedent and consequence of the rules in contrast with other fuzzy inference methods such as the Takagi-Sugeno-Kang method [24].

The developed fuzzy fitness measure for each feature can serve as an ordering mechanism suitable for feature prioritization. Priorities can be established by defuzzifying the fuzzy fitness measures through some defuzzifier such as the centroid or maximum defuzzifiers [24]. The corresponding priority value for a feature would be represented by $\mathbb{FIT}(f)^*$. The process of calculating $\mathbb{FIT}(f)^*$ for the approximate graph coloring feature is depicted in Figure 5. As it can be seen in this figure, the utility knowledge of this feature and the soft constraints (preferences) of the stakeholders over the two concerns are used to perform the inference. After the application of the Mamdani-type fuzzy inference process, the fitness of the approximate graph coloring feature is shown to be *Medium*.

Definition 7 *Let f_1, f_2 be two features such that $f_1, f_2 \in \mathcal{SKB}, f_1, f_2 \notin \mathfrak{MCS}$. We define $f_1 <_{\mathbb{FIT}} f_2$ as $\mathbb{FIT}(f_1)^* < \mathbb{FIT}(f_2)^*$.*

It is now possible to extend the maximal covering specialization algorithm to support soft constraints. Algorithm 4.2 shows the structure of this process called the MaximalCoveringConfiguration.

Algorithm 4.2: MaximalCoveringConfiguration($\mathcal{SR}_s, \mathcal{SR}_h, \mathcal{IC}, \mathcal{SKB}, \mathcal{UKB}$)

```
$\mathfrak{MCC} \leftarrow$ MaximalCoveringSpecialization($\mathcal{SR}_h, \mathcal{IC}, \mathcal{SKB}$)
$Temp \leftarrow \emptyset$
for each ($f \in \mathcal{SKB}$ and $f \notin \mathfrak{MCC}$)
    do { $\mathbb{FIT}(f)^*[f] \leftarrow$ ComputeFitness($f, \mathcal{SR}_s, \mathcal{UKB}$)
         $Temp \leftarrow Temp \cup f$
$Temp \leftarrow$ OrderDesc($Temp, <_{\mathbb{FIT}}$)
for $i \leftarrow 0$ to Size($Temp$)
    do { $Cons[i] \leftarrow$ ComputeCons($Temp[i], \mathcal{IC}, \mathcal{SKB}$)
         if ($Cons[i] \cup \mathfrak{MCC} \not\models \perp$)&(Stakeholders Approval)
           then
         $\mathfrak{MCC} \leftarrow Cons[i] \cup \mathfrak{MCC}$
return ($\mathfrak{MCC}$)
```

Algorithm 4.2 builds on the MaximalCoveringSpecialization algorithm and extends it by supporting the satisfaction of soft constraints and providing means for interactive configuration. As it can be seen from the algorithm, the features that are not present in the maximal covering specialization are rank-ordered based on their fitness with respect to the soft constraints of the stakeholders and are recommended to the stakeholders in that order, for them to be added to the feature model specialization if their consequential facts do not create inconsistencies with the current specialization. Based on the description provided in Definition 3, the consequential facts for a given feature can be computed using the ComputeCons function; therefore, only features that are not inconsistent with the features in $\mathfrak{MCS}$ will be considered in this process.

The stakeholders are able to interact with this algorithm by accepting or rejecting the recommended features. This process is repeated until all features are processed. The algorithm will end by providing a complete feature model *configuration*, denoted by $\mathfrak{MCC}$.

Corollary 1 MaximalCoveringConfiguration *is both sound and complete.*

Proof: The soundness and completeness of MaximalCoveringConfiguration is reliant on the soundness and completeness of MaximalCoveringSpecialization and the soundness of ComputeCons. We have shown previously in Theorems 1 and 2 that MaximalCoveringSpecialization is sound and complete. It is also straightforward to see that since ComputeCons only permits the evaluation of features with consistent consequential facts with the features present in $\mathfrak{MCS}$ that no inconsistent feature will be able to be present in $\mathfrak{MCC}$. Therefore, MaximalCoveringConfiguration is both sound and complete.

■

In summary, the interactive configuration process consists of the following steps:

1. Hard and soft constraints of the stakeholders are expressed;
2. A maximal covering specialization based on the structural knowledge, integrity constraints and hard constraints of the stakeholders is developed;
3. Stakeholders analyze the suitability of the provided specialization. In light of the provided specialization, they can change some of their initial hard constraints in order to gain a more suitable specialization in case of which a new specialization is developed based on the changed hard constraints;
4. The remaining unbound features of the feature model are ranked based on the stakeholders soft constraints and are recommended to the stakeholders. The stakeholders can choose the most desirable features according to the ranking until a complete feature model configuration is achieved.

In terms of algorithm complexity, our proposed configuration process is quite similar to the work proposed in [25] in that they also consider using SAT solvers for configuring feature models. Our major advantage over their work is that we also consider soft constraints (quality attribute information) of the feature model features in both of the *domain engineering* and *application engineering* phases of the software product line development process. The additional complexity that our process imposes on the configuration task is related to Algorithm 4.2, which has two major steps. In the first step, it benefits from a SAT solver to satisfy the stakeholders' hard constraints which is a step that is also present in the works of other authors such as Benavides et al. [25], and a second step which analyzes the feature model quality attributes. The second step is an extra step in our work in that we perform since we additionally consider quality attributes in the decision making process. As it can be seen in Algorithm 4.2, the second step only has a *linear time complexity*, which does not really significantly impact the implementation and performance of our approach compared to the state of the art methods in feature model configuration with the exception that our work provides means for considering feature model quality attributes. Hence, in terms of scalability and time complexity our work is comparable to the work by others such as [25, 17, 15, 26]. We are currently implementing an Eclipse plugin that uses lp_solve[5] and jFuzzyLogic[6] to perform the proposed approach in this paper.

5. An Illustrative Case Study

In this section, we will show how the proposed approach can be applied to a real world case study. The graph product line is a widely used model for illustrating the details of feature modeling techniques and showing how they work in reality. The work proposed in this paper spans both the *domain engineering* and *application engineering* phases of software product line development. In the following, we will separate the processes in each of these phases and will describe the steps involved in each phase.

5.1. Domain Engineering

1. *Domain modeling:* The modeling process of a software product line, e.g. the development of a feature model for representing various applications of the domain, starts with the task of understanding the main concepts of the domain of discourse, which is followed by their structural description in the form of accepted models such feature models. Similarly, in our approach the first step for the domain engineers is to understand the domain of discourse and model the main concepts through the use of feature models. This step will include the identification of all possible features of the domain be them in the form of application functionality or attributes. All of these information will be categorized and represented in a feature model. The outcome of this step will be a comprehensive feature model that addresses variability and commonality of features between the different applications of the domain being modeled. Lopez-Herrejon and Batory have shown the outcome of this process for the graph manipulation software domain [13], which has been shown in Figure 2.
2. *Utility knowledge (quality attribute) modeling:* In typical feature modeling processes, the outcome of the domain modeling process, which is a standard feature model, is considered to be a sufficient representation of the application domain and is often used within the application engineering phase for creating the target application. However, in our approach, we consider the information gathered in the feature model to be only a subset of the important information that is required for modeling the application domain. The other important knowledge that needs to be captured in order to have a complete understanding of the application domain are what we call the utility knowledge of the features in the feature model. Utility knowledge represent the provided quality attribute or non-functional properties of each feature. In order to gather this information, the domain engineers are required to perform two tasks:
   - identify the important quality attributes or non-functional requirements, referred to as *concerns* in our framework, for the target application domain and completely describe them including information about what the

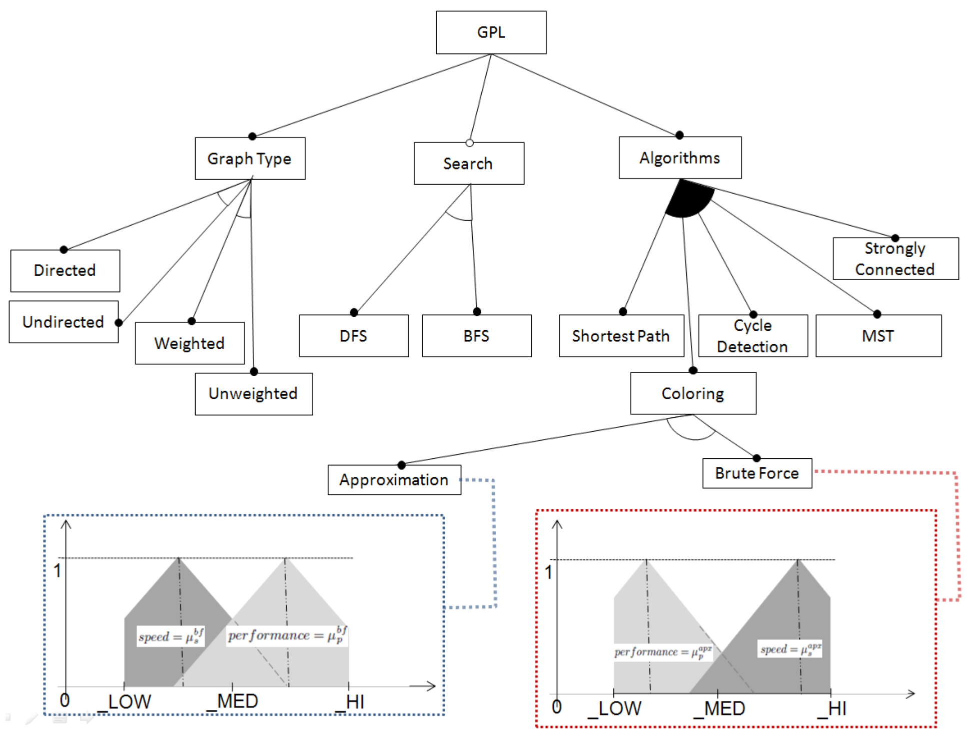

Figure 6. Annotate feature model with information regarding important domain concerns.

possible variations that these concerns might be. For instance, they might identify that speed is an important concern and needs to be considered and that the different possible variations for it are: high speed, medium speed and low speed.

- describe each feature of the feature model with information about how well it is able to satisfy the concerns. For instance, the domain engineer would be able to specify that that a certain feature if included in the final application will have a low speed. This task will provide an additional level of information about the domain in terms of how well the domain features are able to satisfy the important relevant concerns of the domain (the important quality attributes and non-functional properties).

In this paper, we use fuzzy variables to model and represent the utility knowledge related to the features. Such utility knowledge related to two important concerns such as performance and speed are modeled in Figure 3. Given that concerns and their variations (different possible forms of quality attributes and non-functional properties) are represented using fuzzy variables, each feature will be annotated with a fuzzy function for a given concern to show how well that feature is able to contribute to the given concern. Figure 6 shows how a standard feature model is annotated with additional utility knowledge in our proposed work. In this figure, only the annotation of two features, i.e. approximation and brute force graph coloring, has been shown. Other features can be annotated in a similar form.

The outcome of the domain engineering phase in our proposed approach is a comprehensive feature model that constitutes information about the functional and non-functional requirements of the application domain. As we will show in the next step of the case study, given that quality attributes and non-functional requirements are captured in the form of concerns as annotations in our model, it is possible to use these information in the application engineering phase for application development.

5.2. Application Engineering

The application engineering phase is concerned with the development of an appropriate application from the software product line feature model based on some criteria. These criteria are often the constraints and requests that the application stakeholders provide to the application engineers. Hence, application engineers need to perform the following steps in our proposed approach in order to derive the most suitable target application for the stakeholders:

1. identify the set of features that are most important for the stakeholders and need to be present in the target application. These features are usually those that provide the core functionality of the target application. In our approach we refer to these required features as *hard constraints*. In this case study, we will assume that the stakeholders are interested in creating a software graph manipulation package, which is able to provide the suitable functionality for performing graph coloring, and breadth-first search and also checking the strongly-connectedness property of a weighted graph. Therefore, these requirements are the hard constraints that need to be considered in the application engineering process. The hard constraints are represented using $\mathcal{P}(\mathcal{N})$ axioms in our work, so the hard constraints of this case study can be represented as follows:

$$\mathcal{SR}_h = \{GraphColoring, StronglyConnected, Weighted, BFS\},$$

which means that these four mentioned features are strictly required by the stakeholders to be included in the final product configuration.

2. as was mentioned before, we believe that features within a feature model only show a subset of the domain information and hence additional information in the form of important concerns (quality attributes and non-functional properties) need to be considered while creating a target application. In our approach such information are gathered in the domain engineering phase, so it is possible to use such information while performing application engineering. Application engineers would need to find out about the preferences of the stakeholders with regards to the important concerns, referred to as stakeholders' *soft constraints*.

For instance, they would need to know whether the stakeholders prefer speed over performance or vice versa. Such information would allow them to choose the features that satisfy these non-functional requirements the most. In this case study, we assume that the stakholders are able to compromise speed for performance. Therefore, it can be inferred that performance has a higher priority and degree of importance compared with speed. So, the soft constraints can be shown as

$$\mathcal{SR}_s = \{peformance = \mu^p_{\mathcal{SR}_s}, speed = \mu^s_{\mathcal{SR}_s}\},$$

where

$$\mu^p_{\mathcal{SR}_s} = tri(\_LOW, \_HI, \_MED, \frac{\_MED + \_HI}{2}, \_HI),$$
$$\mu^s_{\mathcal{SR}_s} = tri(\_LOW, \_HI, \_LOW, \_LOW, \_MED).$$

3. given the fact that application engineers have extracted information about the required features to be included in the target application and also the desired non-functional requirements and quality attributes of the stakeholders, it is possible to put these information in the context of the information gathered in the domain engineering phase and form an appropriate target application. This process is performed interactively with the involvement of the application engineers and the target application stakeholders using Algorithm 4.2. This process is described as follows:

i) In the first step, the stakeholders' hard constraints, the integrity constraints and the structural knowledge of the feature model are automatically converted into an integer linear program, and the problem of finding the maximal covering specialization is turned into finding a variable assignment that finds an assignment that satisfies the maximum number of stakeholders requests. Such an assignment and its consequential facts are added to $\mathfrak{MCS}$, and the unsatisfiable facts are added to $\mathfrak{UI}$.

The result of this process for this case is:

$$\mathfrak{MCS} = \{Weighted, GraphColoring, DFS, \neg BFS, StronglyConnected, Directed, \neg Undirected\},$$

$$\mathfrak{UI} = \{BFS\}.$$

which shows that all hard constraints other than BFS have been satisfied in the developed specialization of GPL.

The stakeholders are now able to view and analyze the developed specialization and decide whether they want to continue with it or desire to change the hard constraints to gain BFS in tradeoff for some other features.

ii) Assuming that the specialization is accepted, the remaining open features are ranked based on $<_{\mathbb{FIT}}$ and recommended to the stakeholders. Here, the

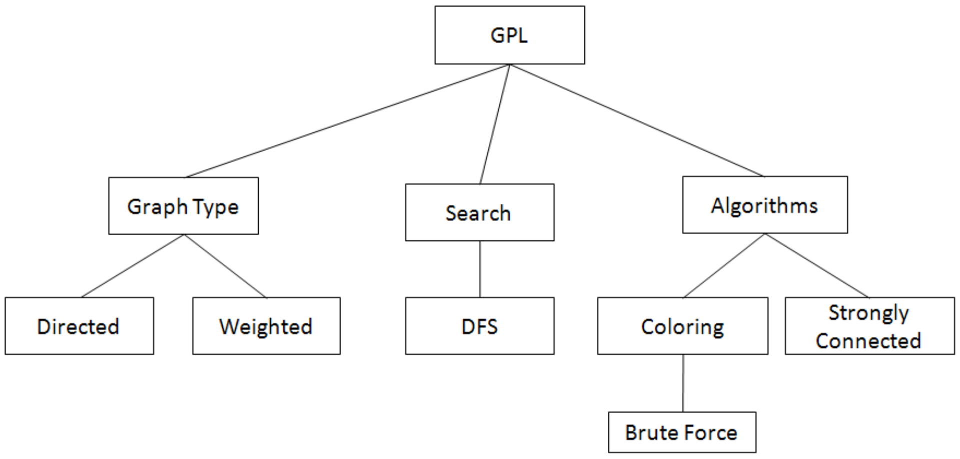

Figure 7. The maximal covering configuration for the case study.

open features that need to be considered are $ColoringApproximation$ and $BruteForceColoring$. In light of the utility knowledge ($\mathcal{UKB}$) provided by the stakeholders, we can infer that $BruteForceColoring <_{\mathbb{FIT}} ColoringApproximation$; therefore, with priority given to $BruteForceColoring$, it will be recommended to the stakeholders first, and if not selected, $ColoringApproximation$ is then suggested. The selection of any of these features will complete the configuration of the GPL feature model based on the hard and soft constraints of the stakeholders. Figure 7 depicts the final maximal covering configuration for this example after the selection of the $BruteForceColoring$ feature.

6. Related Work

Within the realm of software product family engineering, feature models have been viewed as *configurable complex systems* that can be adapted to produce suitable software products under different circumstances and requirements [27]. Therefore, feature model design is most suitable for complex and large-scale problem domains with numerous features and product options. This requires a structured representation for feature models to be able to manipulate and reason over the developed models. To this end, Mannion was the first to propose the adoption of propositional formula for formally representing software product lines [28]. This idea has been extended by creating a connection between feature models, grammars and propositional formula in [17]. Grammars have also been used in other work such as [11] where context-free grammars have been used to represent the semantics of cardinality-based feature models. This semantic interpretation of a feature model relates with an interpretation of the sentences recognized as valid by the context-free grammar. However, such techniques can have limitations in defining the integrity constraints of the feature models. For this reason, Czarnecki and Kim employ the Object Constraint Language (OCL) to express such additional restrictions [29]. In their approach, feature models are converted into UML diagrams where integrity constraints are enforced on them through OCL expressions.

Wang et al. have addressed the issue of the formal representation of feature model from a somewhat different perspective. In their approach, they employ OWL description logic ontologies to model feature model structure and integrity constraints [16]. Their reliant on OWL DL for representing feature models opens up the possibility of employing already existing OWL reasoners to perform feature model configuration validation and verification checks. For instance in their experiments, FaCT++ [30] has been used to see whether a developed configuration is valid or not. Despite its numerous advantages, the use of an ontological representation for feature models poses some open research questions: 1) ontologies are often viewed under the open-world semantics whereas feature models are meaningful in a closed-world setting. There are several proposal to provide closed-world interpretation to ontologies such as the autoepistemic knowledge operator [31] and McCarthy's circumscription [32], but actual reasoner support for these is yet to be provided in reasoners such as Racer [33], Pellet [34] or FaCT++ [30]. 2) Feature models may require the handling of exceptional cases in their formalization, e.g., a parent feature may require the inclusion of some other feature for its configuration, but in some exceptional case if one of its child features is selected this inclusion requirement is lifted. Such exceptional cases are difficult to support under this representation [35]. 3) Although the representation of integrity constraints in description logics entails correct results and advises DL reasoners appropriately, their representation does not match their actual logical meaning, e.g., $StronglyConnected \sqsubseteq \exists hasDFS.DFS$ does not explicitly mean that the existence of the $StronglyConnected$ feature entails the inclusion of the DFS feature, which is due to the descriptive nature of description logics and not a logic programming nature. It is worth noting that description logic based representation is useful for developing explanations of why a feature model configuration is satisfiable or not through the use of justifications [36] or concept abduction in DL [37].

Similar to the description logic representation where reasoning is performed on feature models using standard DL reasoners, feature model configurations can be verified using Logic Truth Maintenance Systems (LTMS) in their representation in the form of propositional formula [38, 39, 40]. Three of the most widely used methods in this area are Constraint Satisfaction Problem (CSP) solvers [25], propositional SATisfiability problem (SAT) solvers [17], and the Binary Decision Diagrams (BDD) [15]. The basic idea in CSP solvers is to find states (value assignments for variables) where all constraints are satisfied. Although being the most flexible proposal for verifying feature model configurations, they fall short in terms of performance time on medium and large size feature models [26]. Somewhat similar to CSP solvers, SAT solvers attempt to decide whether a given propositional formula is satisfiable or not, that is, a set of logical values can be assigned to its variables in such a way that makes the formula true. SAT solvers are a good option for manipulating feature models since they are becoming more and more efficient despite the NP-completeness of the problem itself [40]. Closely related is the employment of Alloy [41] for analyzing the properties of feature models that is based on satisfiability of first-order logic specifications converted into boolean expressions [42]. Also, BDD is a data structure for representing the possible configuration space of a boolean function, which can be useful for mapping a feature model configuration space. The weakness of BDD is that the data structure size can even grow exponentially in certain cases; however, the low time performance results of BDD solvers usually compensates for their space complexity.

More closely related to the theme of this paper, Czarnecki et al. have developed probabilistic feature models where a set of joint probability distributions over the features provide the possibility for defining hard and soft constraints [43]. The joint probability distributions are mined from existing software products in the form of Conditional Probability Tables (CPT); therefore, such tables reveal the tendency of the features to be seen together in a software product rather than desirability, i.e., two features may have been seen together in many configurations of a feature model in the past, but they are not desirable for the given product description on hand. Hence, probabilistic feature models are ideal for representing configuration likelihood but not desirability, which is the core concept of the proposal of our paper. Our paper focuses on the strategic objectives of the stakeholders denoted as concerns and tries to align the best possible feature matches to those concerns [44]; therefore, it addresses desirability rather than likelihood. The concepts of the current paper is more closely related to weighted feature models introduced in [45].

In summary, we believe that the proposed approach in this paper advances the current state in feature modeling by 1) providing a holistic view towards capturing and analyzing both of the stakeholders' hard and soft constraints; 2) creating an interactive feature model configuration process, which is both sound and complete; 3) offering methods for explaining and justifying why a specific set of soft and hard constraints can or cannot be satisfied for a given feature model. However, the interactive configuration process needs to be improved in some aspects in the future: 1) after a maximal covering specialization is created based on the hard constraints, the stakeholders are given the chance to view the developed specialization and decide whether they want to make any changes in their hard constraints to reach a more suiting specialization. In such a case, if the stakeholders decide to change some of the constraints, the process will take all of the revised hard constraints into account and will try to recalculate a new maximal covering specialization, which may be a time consuming process if repeated multiple times; therefore, it would be interesting to look at possible optimizations of the recalculation of the maximal covering specialization; 2) the configuration process relies on the utility knowledge of the feature models to reason about the suitability of a feature for a given soft constraint. This can become an issue for existing feature models that do not have utility knowledge annotations attached to them. For such cases, we will need to look at methods for inferring the possible utility knowledge of the features or other techniques such as bundle annotation of features [46] to ease and speed up the feature model annotation process.

7. Concluding Remarks

The research community has put much emphasis on developing methods for the syntactical validity checking of model configurations. These methods mainly focus on forming grammatical correspondences for the graphical representation of feature models and perform automated syntactical analysis based on the model dependency rules and constraints. However, considering the strategic objectives and goals of the stakeholders and the specific domain requirements in the feature model configuration process can create a *semantic validation process* that will ensure that the requisites of the target audience of the product are met besides having a valid feature model configuration. Such semantic validation process can complement syntactical consistency checking methods in order to create a valid and at the same time useful final configuration of a feature model.

In this paper, we have proposed the use of the fuzzy extension of $\mathcal{P}(\mathcal{N})$ in order to capture both hard and soft constraints of the stakeholders for being able to effectively reason over the needs of the stakeholders. On this basis, we have developed a maximal covering specialization algorithm that creates a sound and complete specialization of a feature model based on stakeholders hard constraints, which is complemented by the maximal covering configuration algorithm that orders and creates a sound and complete configuration given the soft constraints of the stakeholders. The focus of the techniques developed in this paper is to achieve maximum desirability for the developed feature model configuration for the stakeholders. For this reason, stakeholders' objectives take center place in the proposed methods where they are matched against the utility knowledge of the features. The proposed interactive feature configuration process guarantees the complete satisfaction of the hard constraints of the stakeholders by providing a consistent final configuration for cases where the stakeholders requests are consistent, and provides complete justification for why the hard constraints happened to be unsatisfiable if the requests are inconsistent themselves. This has been formally been shown as the soundness and completeness of the process.

Early in this paper, we introduced a set of challenges that need to be addressed. We believe that our proposed approach is able to tackle those challenges appropriately in the following ways: 1) this work is able to maximize the satisfaction of the stakeholders by using MAX-WEIGHTED-SAT which tries to satisfy the most number of stakeholders requests and hence minimize the presence of undesired features; 2) as it can be seen in Algorithm 4.2, the work attempts to satisfy all of the stakeholders' requests but in case where all of these requests are not possible to be satisfied simultaneously, the approach will provide an explanation of how and why it was not possible to satisfy all of the requests using $\mathcal{UT}$, which is a very useful means for the stakeholders to understand why the feature model has been configured in this way and the rationality behind it; 3) the use of fuzzy $\mathcal{P}(\mathcal{N})$ allows us to annotate feature model elements with information in the form of non-functional properties, quality attributes and quality of service, which are not possible to capture in traditional feature modeling formalisms. These information are also considered in the configuration process of our proposed approach; and finally 4) we are able to perform configuration validity verification through the employment of the satisfiability checker algorithms that we have employed in Algorithms 4.1 and 4.2. Hence, in our opinion the proposed approach has been able to address the challenges that it intended to tackle.

As future work, we are interested in the actualization of a feature model configuration process by finding appropriate components that match the specifications of the selected features. We envision that this process is enhanced in our developed framework since the formal structural and utility knowledge of the feature models can be used to perform component matchmaking and discovery.

Notes

1. Competing features refer to the set of features that are not consistent and at the same time are not defined in the stakeholders hard constraints.
2. $< n >$ *or* $< n_1..n_2 >$ show respectively that exactly n or at least n_1 and at most n_2 child features are required to configure a parent feature.
3. To be more precise our problem is a subclass of ILP problem, as it is a binary linear program.
4. A t-norm (t-conorm) operator generalizes conjunction (disjunction) in logic.
5. http://lpsolve.sourceforge.net/
6. http://jfuzzylogic.sourceforge.net/

References

1. Pohl, K., Böckle, G., Van Der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer (2005)
2. Czarnecki, K., Eisenecker, U.: Generative programming. Springer (2000)
3. Lee, K., Kang, K., Lee, J.: Concepts and guidelines of feature modeling for product line software engineering. Lecture Notes in Computer Science **2319** (2002) 62–77
4. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A., INST, C.M.U.P.P.S.E.: Feature-oriented domain analysis (FODA) feasibility study. Carnegie Mellon University, Software Engineering Institute (1990)
5. Simos, M.: Organization domain modeling (ODM): Formalizing the core domain modeling life cycle. ACM SIGSOFT Software Engineering Notes **20** (1995) 196–205
6. Griss, M., Favaro, J., dAlessandro, M.: Integrating feature modeling with the RSEB. In: Proceedings of the Fifth International Conference on Software Reuse, Citeseer (1998) 76–85
7. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A feature-; oriented reuse method with domain-; specific reference architectures. Annals of Software Engineering **5** (1998) 143–168
8. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice **10** (2005)
9. Clements, P., Northrop, L.M.: Software product lines, visited june 2009, http://www.sei.cmu.edu/programs/pls/sw-product-lines_05_03.pdf (2003)
10. Regio, M., Greenfield, J.: Designing and Implementing an HL7 Software Factory. In: OOPSLA-Companion. (2005)
11. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration using feature models. Lecture notes in computer science (2004) 266–283
12. Ragone, A., Noia, T.D., Sciascio, E.D., Donini, F.M.: Logic-based automated multi-issue bilateral negotiation in peer-to-peer e-marketplaces. Autonomous Agents and Multi-Agent Systems **16** (2008) 249–270
13. Lopez-Herrejon, R., Batory, D.: A standard problem for evaluating product-line methodologies. Lecture Notes in Computer Science (2001) 10–24

14. Kang, K., Lee, J., Donohoe, P.: Feature-oriented product line engineering. IEEE software **19** (2002) 58–65
15. Mendonca, M., Wasowski, A., Czarnecki, K., Cowan, D.: Efficient compilation techniques for large scale feature models. In: Proceedings of the 7th international conference on Generative programming and component engineering, ACM New York, NY, USA (2008) 13–22
16. Wang, H., Li, Y., Sun, J., Zhang, H., Pan, J.: Verifying feature models using OWL. Web Semantics: Science, Services and Agents on the World Wide Web **5** (2007) 117–129
17. Batory, D.: Feature models, grammars, and propositional formulas. Lecture notes in computer science **3714** (2005) 7
18. Sommerville, I., Sawyer, P.: Viewpoints: principles, problems and a practical approach to requirements engineering. Annals of Software Engineering **3** (1997) 101–130
19. Hess, J., Novak, W., Carroll, P., Cohen, S.: A Domain Analysis Bibliography. SEI-90-SR-3 Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (1990)
20. Ausiello, G., Crescenzi, P., Kann, V., Marchetti-Sp, Gambosi, G., Spaccamela, A.M.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer (2003)
21. Papadimitriou, C., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Prentice-Hall, Inc. (1982)
22. Ausiello, G., Crescenzi, P., Kann, V., Gambosi, G., Spaccamela, A.M.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer (2003)
23. Mamdani, E.: Application of fuzzy logic to approximate reasoning using linguistic synthesis. In: Proceedings of the sixth international symposium on Multiple-valued logic, IEEE Computer Society Press Los Alamitos, CA, USA (1976) 196–202
24. Yager, R., Filev, D.: Essentials of fuzzy modeling and control. New York (1994)
25. Benavides, D., Trinidad, P., Ruiz-Cortes, A.: Automated reasoning on feature models. In: LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005. Volume 3520., Springer (2005) 491–503
26. Benavides, D., Segura, S., Trinidad, P., Ruiz-Cortés, A.: A first step towards a framework for the automated analysis of feature models. Technical report (2006)
27. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Design Automation Conference, 2001. Proceedings. (2001) 530–535
28. Mannion, M.: Using first-order logic for product line model validation. Lecture notes in computer science (2002) 176–187
29. Czarnecki, K., Kim, C.: Cardinality-based feature modeling and constraints: A progress report. In: International Workshop on Software Factories. (2005)
30. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. Lecture Notes in Computer Science **4130** (2006) 292
31. Donini, F., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Transactions on Computational Logic (TOCL) **3** (2002) 177–225
32. McCarthy, J., Laboratory, S.A.I.: Circumscription-A Form of Non-Monotonic Reasoning. Department of Computer Science, Artificial Intelligence Laboratory, Stanford University (1980)
33. Haarslev, V., Moller, R.: RACER system description. Lecture Notes in Computer Science (2001) 701–706
34. Parsia, B., Sirin, E.: Pellet: An owl dl reasoner. In: Proceedings of the International Workshop on Description Logics. Volume 104. (2004)
35. Motik, B., Horrocks, I., Rosati, R., Sattler, U.: Can owl and logic programming live together happily ever after? In: International Semantic Web Conference. (2006) 501–514
36. Horridge, M., Parsia, B., Sattler, U.: Laconic and precise justifications in owl. In: Proceedings of the 7th International Conference on The Semantic Web, Springer (2008) 323–338
37. Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F., Mongiello, M.: Concept abduction and contraction in description logics. In: Proceedings of the 16th International Workshop on Description Logics (DL03. Volume 81. (2003)
38. Schobbens, P., Heymans, P., Trigaux, J.: Feature diagrams: A survey and a formal semantics. In: 14th IEEE International Conference Requirements Engineering. (2006) 139–148

39. Janota, M., Kiniry, J.: Reasoning about feature models in higher-order logic. In: Software Product Line Conference, 2007. SPLC 2007. 11th International. (2007) 13–22
40. Benavides, D., Segura, S., Trinidad, P., Ruiz-Cortes, A.: FAMA: Tooling a framework for the automated analysis of feature models. In: Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS). (2007) 129–134
41. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. **11** (2002) 256–290
42. Gheyi, R., Massoni, T., Borba, P.: A theory for feature models in alloy. In: First Alloy Workshop. (2006) 71–80
43. Czarnecki, K., She, S., Wasowski, A.: Sample spaces and feature models: There and back again. In: Proceedings of the 2008 12th International Software Product Line Conference-Volume 00, IEEE Computer Society Washington, DC, USA (2008) 22–31
44. Bagheri, E., Gasevic, D.: Good prose is the selection of the best software features. submitted to IEEE Software (2009)
45. Robak, S., Pieczynski, A.: Employing fuzzy logic in feature diagrams to model variability in software product-lines. In: 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2003. Proceedings. (2003) 305–311
46. Bagheri, E., Ghorbani, A.A.: The analysis and management of non-canonical requirement specifications through a belief integration game. In: Knowledge and Information Systems, Springer (2008)