Configuring Software Product Line Feature Models based on Stakeholders' Soft and Hard Requirements

Ebrahim Bagheri, Tommaso Di Noia, Azzurra Ragone, and Dragan Gasevic

NRC-IIT, Politecnico di Bari, University of Trento, Athabasca University

Abstract. Feature modeling is a technique for capturing commonality and variability. Feature models symbolize a representation of the possible application configuration space, and can be customized based on specific domain requirements and stakeholder goals. Most feature model configuration 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 configuration is performed. The model configuration derivation process that we propose respects the soundness and completeness properties.

1 Introduction

Software product line engineering (SPLE) is concerned with capturing the commonalities, universal and shared attributes of a set of applications for a specific 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*. In SPLE, *feature modeling* is an important technique for modeling the attributes of a family of systems [2]. 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, a new product can be developed in a shorter time at a lower cost. Large-scale industrial applications of software product families entail the development of very large feature models that need to be customized before they can be used for a specific application. 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. 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*. 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. This feature model is referred to as a *specialization* of the former feature model and the staged refinement process constitutes *staged configuration* [2]. Despite the effectiveness of staged configuration, it is still hard to manually create configurations for industrial-scale feature models. The reason 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 is a complex task;
3. 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 $\mathcal{P}(\mathcal{N})$ is able to address the aforementioned concerns. 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 needs to be made cannot be formally supported by these approaches. As we will discuss, in cases where a choice needs to be made between several competing features, stakeholders' soft constraints can help in making the right decision. This is the fact that has been neglected in almost all other available approaches in the literature. It is important to take a holistic approach towards the 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. 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.

The spotlight of this paper is that it is a novel work which considers the stakeholders' soft constraints, i.e. desired quality attributes, while configuring a feature model. It uses a variant of propositional logics along with fuzzy logics to represent feature models and their quality attributes and to be able to bring the stakeholders' soft and hard constraints under one umbrella. Capturing both soft and hard constraints allows our proposed approach to be the first of its kind to simultaneously consider integrity constraints, stakeholder requests and quality attributes during the feature model configuration process, a process which has been formally shown to be *sound* and *complete*.

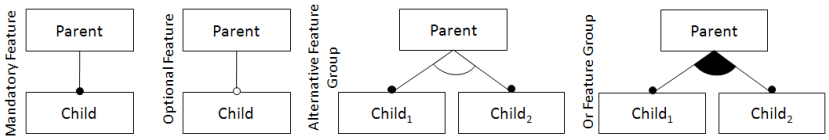

Fig. 1. The main graphical notations employed in feature modeling.

2 Feature Modeling

Features are important distinguishing aspects, qualities, or characteristics of a family of systems [3]. 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. Graphical feature models are in the form of a tree whose root node represents a domain concept, and the other nodes and leafs illustrate the features. In a feature model, features are hierarchically organized and can be typically classified as: *Mandatory*; *Optional*; *Alternative feature group*; *Or feature group*. Figure 1 depicts the graphical notation of the feature relationships. This tree structure 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 requires the *existence* of another feature; *Excludes*: the presence of a given feature requires the *elimination* of another feature. Lopez-Herrejon and Batory have proposed the Graph Product Line (GPL) as a

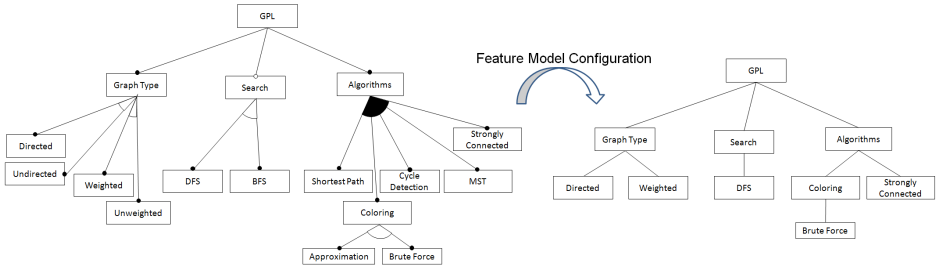

Fig. 2. The graph product line feature model.

standard problem for evaluating product line methodologies [4]. The intention is to develop configurations of GPL for different problem domains. For instance, GPL can be configured to perform several search algorithms over a directed or undirected graph structure. The graphical representation of GPL is shown in Figure 2. Clearly, not all possible configurations of GPL produce valid graph programs. For instance, a configuration 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: Cycle Detection EXCLUDES BFS; Cycle Detection INCLUDES DFS; Strongly Connected INCLUDES DFS; Strongly Connected INCLUDES Directed; Strongly Connected EXCLUDES Undirected.

3 Formalism for Feature Modeling

There have been different approaches towards the formalization of feature models among which the use of pure propositional logic [5], description logic [6], and iterative tree grammars [7] are the most important ones. Here we introduce Fuzzy Propositional Logic enriched with concrete domains, called fuzzy $\mathcal{P}(\mathcal{N})$. For the sake of completeness, we report basic definitions [8].

Definition 1 (Concrete Domains) *A* concrete domain D *consists of a finite set* $\Delta_c(D)$ *of numerical values, and a set of predicates* $C(D)$ *expressing numerical constraints on* D*. We denote with* $|\Delta_c(D)|$ *the cardinality of the set* $\Delta_c(D)$*.*

We define $\mathcal{P}(\mathcal{N})$ as a propositional logic with concrete domains which is most suitable for the feature modeling domain. Indeed, we have to model both qualitative and quantitative features.

Definition 2 (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 an attribute 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* $(\psi \rightarrow \varphi) \wedge (\varphi \rightarrow \psi)$*.*

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.

Definition 3 (Interpretation and models) *An interpretation* $\mathcal{I}$ *for* $\mathcal{P}(\mathcal{N})$ *is a function (denoted as a superscript* $\cdot^{\mathcal{I}}$ *on its argument) that maps each atom in* $\mathcal{A}$ *into a truth value* $A^{\mathcal{I}} \in \{\mathsf{true}, \mathsf{false}\}$*, each attribute name* f *into a value* $f^{\mathcal{I}} \in D_f$*, and assigns truth values to formulas as follows:*

- $(fck)^{\mathcal{I}} = \mathsf{true}$ *iff* $f^{\mathcal{I}} ck$ *is true in* D_f*,* $(fck)^{\mathcal{I}} = \mathsf{false}$ *otherwise*
- $(\neg\psi)^{\mathcal{I}} = \mathsf{true}$ *iff* $\psi^{\mathcal{I}} = \mathsf{false}$*,* $(\psi \wedge \varphi)^{\mathcal{I}} = \mathsf{true}$ *iff both* $\psi^{\mathcal{I}} = \mathsf{true}$ *and* $\varphi^{\mathcal{I}} = \mathsf{true}$*, according to truth tables for propositional connectives.*

Given a formula φ *in* $\mathcal{P}(\mathcal{N})$*, we denote with* $\mathcal{I} \models \varphi$ *the fact that* $\mathcal{I}$ *assigns* true *to* φ*. If* $\mathcal{I} \models \varphi$ *we say* $\mathcal{I}$ *is a* model *for* φ*, and* $\mathcal{I}$ *is a model for a set of formulas when it is a model for each formula.*

An interpretation $\mathcal{I}$ is completely defined by the values it assigns to propositional atoms and numerical features.

Example 1 *Consider the following example where a fast Graph Coloring Algorithm (CGA) is represented as an algorithm whose computational complexity is of order* $O(n \log n)$

$$FastCGA \rightarrow GCA \wedge (complexity \geq O(n \log n))$$

Possible models for the formula are:

$$\mathcal{I} = \{FCGA = \mathsf{false}, CGA = \mathsf{true}, complexity = O(2^n)\}$$
$$\mathcal{I} = \{FCGA = \mathsf{false}, CGA = \mathsf{true}, complexity = O(n^2 \log n)\}$$
$$\mathcal{I} = \{FCGA = \mathsf{true}, CGA = \mathsf{true}, complexity = O(n)\}$$

Given a set of formulas $\mathcal{C}$ in $\mathcal{P}(\mathcal{N})$ (representing a set of constraints), we denote by $\mathcal{I} \models \mathcal{C}$ the fact that $\mathcal{I}$ is a *model* for $\mathcal{C}$. a set of constraints is *satisfiable* if it has a model. $\mathcal{C}$ logically implies a formula φ, denoted by $\mathcal{C} \models \varphi$ iff φ is true in all models of $\mathcal{C}$. We denote by $\mathcal{M}_{\mathcal{C}} = \{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$, the set of all models for $\mathcal{C}$, and omit the subscript when no confusion arises. 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 singleton formulas containing only a single atom $A \in \mathcal{A}$ or a restriction over a concrete attribute (fck). As it can be seen in this formulation, rules are either satisfied and are true or are false otherwise. In order to represent varying degrees of truthfulness over concrete attributes, a fuzzy extension of $\mathcal{P}(\mathcal{N})$ is formulated.

Definition 4 (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 $\mathcal{F} = \langle f, D_f \rangle$ are defined as for $\mathcal{P}(\mathcal{N})$;*
- *$\mathcal{C} = \{cn_h\}$ is a set of attributes such that $\mathcal{F} \cap \mathcal{C} = \emptyset$;*
- *$\overline{\mu} = \{\mu_{cn_h}^{A_i}\}$ is a set of fuzzy membership functions.*

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

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

*If a fuzzy $\mathcal{P}(\mathcal{N})$ formula is in form 1 we refer to it as fuzzy fact, otherwise as fuzzy clause. Given a fuzzy clause $A_i \rightarrow \bigwedge_h cn_h = \mu_{cn_h}^{A_i} \wedge \psi$, we call $\bigwedge_h cn_h = \mu_{cn_h}^{A_i}$ the **fuzzy part of the clause** and ψ the **deterministic part of the clause**.*

Some of the widely used examples of fuzzy membership functions used in fuzzy $\mathcal{P}(\mathcal{N})$ are the triangular, trapezoidal, and Gaussian functions. For each fuzzy predicate $\mu \in \overline{\mu}$ the 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 in this paper. In order to clarify the syntax of fuzzy $\mathcal{P}(\mathcal{N})$ formulas, we represent

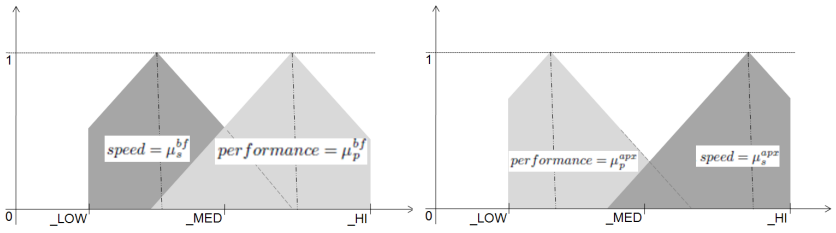

Fig. 3. The Semantic annotation of a) *brute force* and b) *approximation* graph coloring features.

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. Here

$performance = \mu_p^{bf} \wedge speed = \mu_s^{bf}$ and $performance = \mu_p^{apx} \wedge speed = \mu_s^{apx}$ are the *fuzzy part* of clause (1) and clause (2) respectively while $Coloring$ is the *deterministic part* for both of them. 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 is rather slow but has high performance in terms of accuracy. On the other side, the approximation technique has higher speed and weaker performance. Our proposed fuzzy $\mathcal{P}(\mathcal{N})$ is a language that not only fully supports the representation of feature modeling structure and constraints in propositional form, but also provides the means for expressing additional information about the non-functional properties of the features using fuzzy variables. Besides the adequate expressiveness of fuzzy $\mathcal{P}(\mathcal{N})$ for representing feature models, its support for defining fuzzy information and hence the support for non-functional quality information is what distinguishes our work from those proposed previously in the literature (e.g., [5–7]) that do not have support for quality attributes and non-functional requirements. With the additional value of fuzzy $\mathcal{P}(\mathcal{N})$, feature models can carry the kind of information that are not expressible in typical feature modeling notation. As an example, if the brute force and approximate features were expressed in pure propositional logic as proposed by [5, 7], the decision about which one to chose would be difficult, although for instance we know that we prefer performance over speed. This is because such information is not captured in pure propositional form. However in fuzzy $\mathcal{P}(\mathcal{N})$, it is easy to see that the brute force feature has a better performance and worse speed than the approximation feature; therefore, given our preference it can be chosen. Hence, fuzzy $\mathcal{P}(\mathcal{N})$ allows us to benefit from the soft constraints of the stakeholders (e.g., preferring performance over speed) to make the right choices between competing features.

4 Conceptual Modeling

The formalization of feature modeling information in our proposed language entails the development of multiple separate knowledge bases. Besides the structural information of the feature model (i.e., feature hierarchies represented as $\mathcal{SKB}$) and integrity constraints between the features ($\mathcal{IC}$), the rest of the knowledge bases are as follows: The selection of the correct features of a feature model in the configuration process 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 stakeholders. In order to be able to understand how each feature relates with the functional and non-functional requirements, we propose to annotate the features. For this reason, we adopt the concept of *concerns* from the Preview framework [9]. 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 attributes, which need to be considered through the staged configuration process. Examples of concerns can include to cost, time, risk, volatility, customer importance, etc. For instance, *speed* and *performance* are the two concerns that have been used to annotate the features of GPL in Figure 3. This is because speed and performance are important decision making criteria in the GPL 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 $\mathcal{P}(\mathcal{N})$ clauses, and the collection of these information is referred to as the *utility knowledge base* ($\mathcal{UKB}$). Utility knowledge depicts how various features of a given feature model

relate with and to what extent they are able to satisfy the objectives of the configuration. 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 accuracy in terms of performance* ($performance = \mu_p^{apx}$). Such statements can be easily represented in fuzzy $\mathcal{P}(\mathcal{N})$, shown in (1) and (2).

Stakeholders and product developers often specify a set of basic features that they want to see in the final product. For instance, in the GPL configuration process, they may require the inclusion of the graph coloring feature. Such requirements 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*. In this paper, Stakeholders' hard and soft constraints are represented by $\mathcal{SR}_h$, and $\mathcal{SR}_s$, respectively. Similar to utility knowledge, soft constraints can 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 (See Figure 3). Summing up w.r.t.

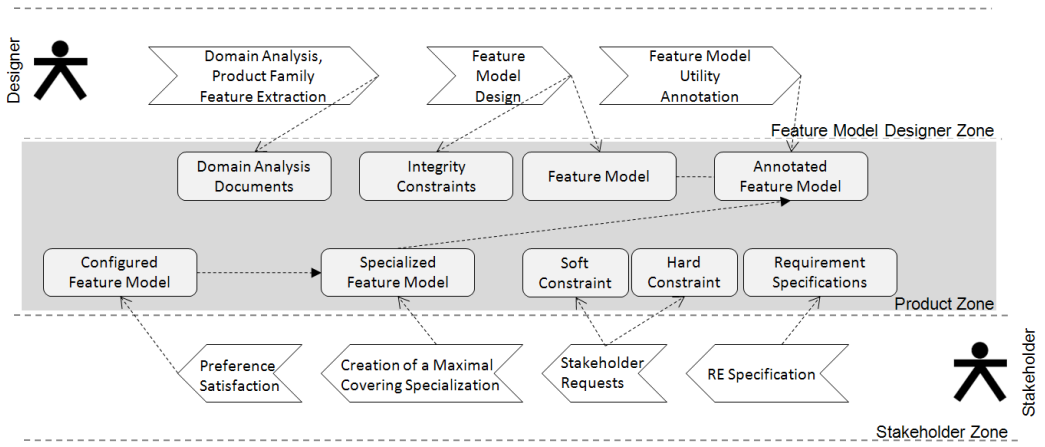

Fig. 4. The overview of the interactive feature model configuration process.

to the feature modeling knowledge, we have the following formalization: $\mathcal{SKB}$: the feature model structural knowledge represented using $\mathcal{P}(\mathcal{N})$ axioms; $\mathcal{IC}$: integrity constraints defind through

$\mathcal{P}(\mathcal{N})$ formulas; $\mathcal{UKB}$: features utility knowledge as a set of fuzzy $\mathcal{P}(\mathcal{N})$ clauses involving concerns; $\mathcal{SR}_h$: stakeholders' hard constraints as $\mathcal{P}(\mathcal{N})$ facts; $\mathcal{SR}_s$: stakeholders' soft constraints (preferences) as fuzzy $\mathcal{P}(\mathcal{N})$ facts involving concerns.

5 Interactive Feature Model Configuration

The overview of our proposed 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. Available domain analysis methodologies exist that can be used for this purpose;
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., *Finding MST is both fast and accurate*.

Once an annotated feature model is developed, the annotation information can be used to reason about the suitability of a feature for a given purpose. For example, the features that have slow execution speed 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. 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 soft constraints that can be tolerated if not satisfied;
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 guides the stakeholders towards the configuration of the feature model.

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 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 (developed in S3) are rank-ordered based on their degree of contribution to the

satisfaction of the stakeholders' soft constraints. The features are recommended to the stakeholders according to the rank-order. The stakeholders can interactively select the features they desire until the feature model is fully configured.

Hard Constraints Satisfaction: It is important to satisfy the stakeholders' hard constraints before their soft constraints, since they represent the *required* features of the product. Let us provide ground definitions for the hard constraint satisfaction process.

Definition 5 *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. In other words, 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 $\mathcal{SR}_h \cup \mathcal{SKB} \cup \mathcal{IC} \models$ 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. Formally, given a set of hard constraints $\mathcal{SR}_h$, the idea is to compute a partition of $\mathcal{SR}_h$ such that: 1) $\mathcal{SR}_h = \mathcal{MCS} \cup \mathcal{UT}$; 2) $\mathcal{MCS} \cap \mathcal{UT} = \emptyset$; 3) $\mathcal{MCS} \cup \mathcal{SKB} \cup \mathcal{IC} \not\models$ false; 4) There is no partition $\mathcal{SR}_h = \mathcal{MCS}' \cup \mathcal{UT}'$ such that $\mathcal{MCS} \subset \mathcal{MCS}'$. 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 5.1 shows the computational procedure. The algorithm calls two functions: COMPUTENEWASSIGNMENT: 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 $\mathcal{F}$. SATISFIEDHARDPREFERNCES: Given $\mathcal{SR}_h$ 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 5.1: MAXIMALCOVERINGSPECIALIZATION($\mathcal{SR}_h, \mathcal{IC}, \mathcal{SKB}$)

more $=$ *true*
max $= 0$
$\mathcal{MCS} = \emptyset$
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** $\mathcal{MCS} = \mathcal{MCS} \cup \{A_i\}$
  - **else** $\mathcal{UT} = \mathcal{UT} \cup \{A_i\}$

return $(\mathcal{I}_{max}, \mathcal{MCS}, \mathcal{UT})$

Algorithm 5.1 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 ($\mathcal{MCS}$) and the set of unsatisfied hard constraints ($\mathcal{UT}$). Ultimately, $\mathcal{MCS}$ contains the maximal covering specialization of the feature model based on the stakeholders' hard constraints ($\mathcal{SR}_h$), and $\mathcal{UT}$ will contain the set of unsatisfiable hard constraints. Based on ($\mathcal{MCS}$) and ($\mathcal{UT}$), 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. It is possible to see from Algorithm 5.1 that the process of finding the maximal covering specialization for a given feature model is sound and complete. 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. It is also complete iff it will find a specialization that satisfies all of the stakeholders' hard constraints whenever one exists.

Theorem 1 (SOUNDNESS) *The maximum covering specialization $\mathcal{MCS}$ computed by Algorithm 5.1 is the largest subset of $\mathcal{SR}_h$ such that the set of axioms $\mathcal{MCS} \cup \mathcal{SKB} \cup \mathcal{IC}$ 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.*

Theorem 2 (COMPLETENESS) *If $\mathcal{MCS}$ 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 5.1.*
Proof. In fact, Algorithm 5.1 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.

Although Algorithm 5.1 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 $\mathcal{F}$ all possible interpretations are 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 $\mathcal{MCS}$ and $\mathcal{UT}$ if we consider our problem as an instance of MAX-WEIGHTED-SAT [10].

Definition 6 (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:

Definition 7 (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 $A^{\mathcal{I}} =$ 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 [11]. For the sake of clarity, here we present the procedure for propositional clauses, as introduced in [11]. 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 means that $A_i = true$, while if $x_i = 0$ then $A_i = false$.

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 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.

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. The ILP-based specialization algorithm is now computationally efficient as well as sound and complete [11, p.314].

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' soft constraints. 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. For this purpose, we have two pieces of valuable information at our disposal, namely utility knowledge of the feature model ($\mathcal{UKB}$), and stakeholders' soft constraints ($\mathcal{SR}_s$). 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. Also note that, by definition, $\mathcal{UBK} \cup \mathcal{IC} \cup \{mcs\}$ is never inconsistent. There always exists an interpretation $\mathcal{I}$ such that both $\mathcal{I} \models mcs$ and $\mathcal{I} \models \mathcal{UBK} \cup \mathcal{IC}$.

Definition 8 *Let $f \in \mathcal{SKB}$, $f \notin \mathcal{MCS}$ 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 $\mathcal{UBK} \cup \mathcal{IC} \cup \mathcal{MCS} \models \psi$. 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 degree of fitness of f in the context of concern cn, denoted $\mathbb{FIT}(f, cn)$, where $\otimes$ is a fuzzy T-norm operator defined as follows:*

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

In other words, given the interpretations of $\mathcal{I}^{mcs} \models \mathcal{UBK} \cup \mathcal{IC} \cup \mathcal{MCS}$, we consider all those fuzzy clauses such that for their deterministic part ψ the relation $\mathcal{I}^{mcs} \models \psi$ holds too. Since each feature is annotated with multiple concerns, we can interpret the information through Mamdani-type fuzzy inference [12] to calculate the fitness of a feature over all concerns (cn_i), denoted as

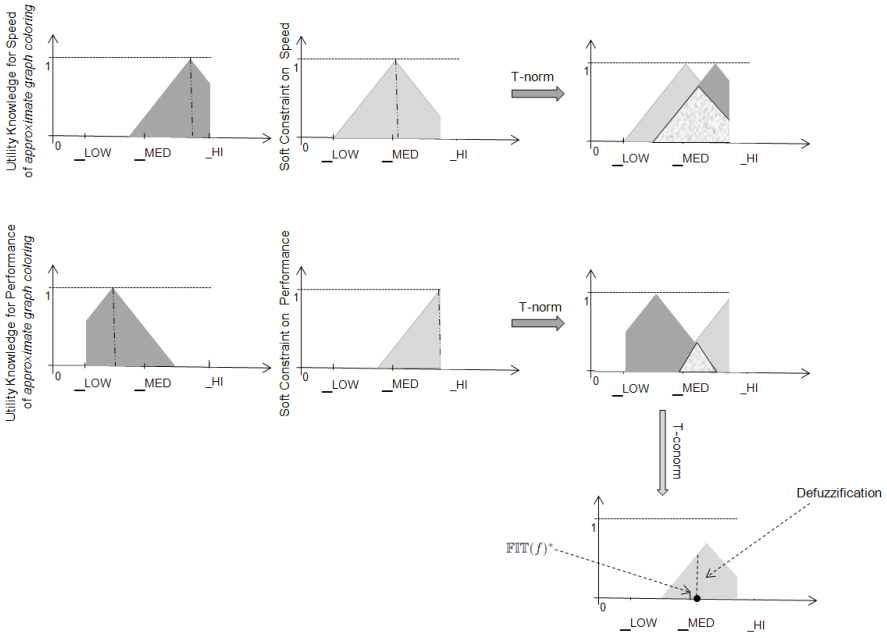

Fig. 5. Computing the fitness of a feature f in context of two concerns: speed and performance.

$\mathbb{FIT}(f)$:

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

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 TSK method [13]. The developed fuzzy fitness measure for each feature can serve as an ordering mechanism for feature prioritization. Priorities can be established by defuzzifying the fuzzy fitness measures through some defuzzifier such as the centroid or maximum defuzzifiers [13]. The corresponding priority value for a feature would be represented by $\mathbb{FIT}(f)^*$. The process of calculating $\mathbb{FIT}(f)^*$ for the approximate feature is depicted in Figure 5 (feature utility knowledge on the left, stakeholder soft constraints in middle, and reasoning results on right). As it can be seen in this figure, the utility knowledge of this feature and the soft constraints of the stakeholders over the two concerns are used for performing the inference. After the Mamdani-type fuzzy inference process, the fitness of the approximate feature for that specific stakeholder is shown to be *Medium*.

Definition 9 *Let* f_1, f_2 *be two features such that* $f_1, f_2 \in \mathcal{SKB}, f_1, f_2 \notin \mathcal{MCS}$. *We define* $f_1 <_{\mathbb{FIT}} f_2$ *as* $\mathbb{FIT}(f_1)^* < \mathbb{FIT}(f_2)^*$.

With the above ordering, it is now possible to extend the maximal covering specialization algorithm to support soft constraints. Algorithm 5.2 shows the structure of this process called the MAXIMALCOVERINGCONFIGURATION. Algorithm 5.2 builds on MAXIMALCOVERINGSPECIALIZATION

by supporting the satisfaction of soft constraints and providing means for interactive configuration. 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. They can be added to the feature model specialization if and only if they are not conflicting with the features in $\mathcal{MCS}$ computed via MAXIMALCOVERINGSPECIALIZATION.

Algorithm 5.2: MAXIMALCOVERINGCONFIGURATION($\mathcal{SR}_s, \mathcal{SR}_h, \mathcal{IC}, \mathcal{SKB}, \mathcal{UKB}$)

$\mathcal{MCS} \leftarrow$ MAXIMALCOVERINGSPECIALIZATION($\mathcal{SR}_h, \mathcal{IC}, \mathcal{SKB}$)
$Temp \leftarrow \emptyset$
for each ($A_i \rightarrow \bigwedge_h cn_h = \mu_{cn_h}^{A_i} \wedge \psi \in \mathcal{UKB}$ such that both $\mathcal{IC} \cup \mathcal{SKB} \cup \mathcal{MCS} \models \psi$ and $\mathcal{IC} \cup \mathcal{SKB} \cup \mathcal{MCS} \cup \{A_1\} \not\models \bot$)
do { $Feat \leftarrow \langle A_1,$ COMPUTEFITNESS$(A_1, \mathcal{SR}_s, \mathcal{UKB})\rangle$
$Temp \leftarrow Temp \cup Feat$
$Feat \leftarrow$ ORDERDESC$(Feat, <_{\text{FIT}})$
$\mathcal{MCC} \leftarrow \mathcal{MCS}$
for $i \leftarrow 0$ **to** SIZE$(Feat)$
do { $\langle A, fit\rangle \leftarrow Feat[i]$
if ($\mathcal{IC} \cup \mathcal{SKB} \cup \{A\} \cup \mathcal{MCC} \not\models \bot$)&(Stakeholders Approval)
then
$\mathcal{MCC} \leftarrow A \cup \mathcal{MCC}$
return ($\mathcal{MCC}$)

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

In summary, the *interactive configuration process* consists of the following steps: 1) Hard and soft constraints of the stakeholders are defined; 2) A maximal covering specialization based on the structural knowledge, integrity constraints and hard constraints of the stakeholders is developed (by MAXIMALCOVERINGSPECIALIZATION); 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 new 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 (using MAXIMALCOVERINGCONFIGURATION). The stakeholders can choose the most desirable features based on the ranking until a complete configuration is achieved.

5.1 An Illustrative Example

Lets suppose that a group of stakeholders are interested in creating a software graph manipulation package, which is able perform graph coloring, and breadth-first search and also checking the strongly-connectedness property of a weighted graph. Further, suppose that they are able to compromise speed for performance. We would need to elicit the hard and soft constraints of the stakeholders along with the utility knowledge of the features of GPL. As will be seen, we only need the utility knowledge of the two child features of the *graph coloring* feature in this example, namely $ColoringApproximation$ and $BruteForceColoring$; therefore, the information given in Figure 3 will be used. The stakeholder hard constraints can be represented as:

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

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

Since it has been expressed that the stakeholders prefer performance to speed, it can be inferred that performance has a higher priority compared to speed. This soft constraints can be shown as

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

where

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

Given these information, we are now able to compute the maximal covering configuration of GPL: 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 is added to $\mathcal{MCS}$, and the unsatisfiable facts are added to $\mathcal{UT}$. The result is:

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

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 trade for some other features.

In the second step, assuming that the specialization is accepted, the remaining open features are ranked based on $<_{\mathbb{FIT}}$ and recommended to the stakeholders. Here, the 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. The right-side of Figure 2 depicts the final configuration of GPL after the selection of the $BruteForceColoring$ feature.

6 Related Work

Mannion was the first to propose the adoption of propositional formula for formally representing software product lines [14]. This idea has been extended by creating a connection between feature models, grammars and propositional formula in [7]. Grammars have also been used in other work such as [2] 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 [15]. In their approach, feature models are converted into UML diagrams where integrity constraints are enforced on them through OCL expressions.

Feature model configurations can be verified using Logic Truth Maintenance Systems (LTMS) in their representation in the form of propositional formula [16–18]. Three of the most widely used methods in this area are Constraint Satisfaction Problem (CSP) solvers [19], propositional SATisfiability problem (SAT) solvers [7], and the Binary Decision Diagrams (BDD) [5]. 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 [20]. 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 [18]. Closely related is the employment of Alloy [21] for analyzing the properties of feature models that is based on satisfiability of first-order logic specifications converted into boolean expressions [22]. 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 [23]. 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; therefore, it addresses desirability rather than likelihood. The concepts of the current paper is more closely related to weighted feature models introduced in [24].

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 integrity constraints. However, considering the strategic objectives and goals of the stakeholders and the specific domain requirements in the feature model configuration process can ensure that the preferences of the target audience of the product are met as well. In this paper, we have introduced and proposed the use of the fuzzy extension of $\mathcal{P}(\mathcal{N})$ in order to capture both hard and soft constraints of the stakeholders. On this basis, we have developed a maximal covering specialization algorithm that creates a 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 possibility to represent and consider stakeholders' soft constraints in the feature model configuration process is one of the novel aspects of our work.

References

1. Pohl, K., Böckle, G., Van Der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer (2005)
2. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration using feature models. SPLC'04 (2004) 266–283
3. Kang, K., Lee, J., Donohoe, P.: Feature-oriented product line engineering. IEEE software **19** (2002) 58–65
4. Lopez-Herrejon, R., Batory, D.: A standard problem for evaluating product-line methodologies. GCSE'01 (2001) 10–24
5. Mendonca, M., Wasowski, A., Czarnecki, K., Cowan, D.: Efficient compilation techniques for large scale feature models. In: International Conference on GPCE. (2008) 13–22
6. 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
7. Batory, D.: Feature models, grammars, and propositional formulas. SPLC'05 (2005)
8. Ragone, A., Noia, T.D., Sciascio, E.D., Donini, F.M.: Logic-based automated multi-issue bilateral negotiation in peer-to-peer e-marketplaces. JAAMAS **16** (2008) 249–270
9. Sommerville, I., Sawyer, P.: Viewpoints: principles, problems and a practical approach to requirements engineering. Annals of Software Engineering **3** (1997) 101–130
10. Ausiello, G., Crescenzi, P., Kann, V., Marchetti-Sp, Gambosi, G., Spaccamela, A.M.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. (2003)
11. Papadimitriou, C., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Prentice-Hall, Inc. (1982)
12. Mamdani, E.: Application of fuzzy logic to approximate reasoning using linguistic synthesis. In: sixth international symposium on Multiple-valued logic. (1976) 196–202
13. Yager, R., Filev, D.: Essentials of fuzzy modeling and control. John Wiley (1994)
14. Mannion, M.: Using first-order logic for product line model validation. SPLC'02 (2002) 176–187
15. Czarnecki, K., Kim, C.: Cardinality-based feature modeling and constraints: A progress report. In: Workshop on Software Factories. (2005)
16. Schobbens, P., Heymans, P., Trigaux, J.: Feature diagrams: A survey and a formal semantics. In: 14th IEEE International Conference Requirements Engineering. (2006) 139–148
17. Janota, M., Kiniry, J.: Reasoning about feature models in higher-order logic. In: Software Product Line Conference, 2007. (2007) 13–22
18. Benavides, D., Segura, S., Trinidad, P., Ruiz-Cortes, A.: FAMA: Tooling a framework for the automated analysis of feature models. In: VAMOS Workshop. (2007) 129–134
19. Benavides, D., Trinidad, P., Ruiz-Cortes, A.: Automated reasoning on feature models. In: CAiSE'05. Volume 3520. (2005) 491–503
20. 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)
21. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. **11** (2002) 256–290
22. Gheyi, R., Massoni, T., Borba, P.: A theory for feature models in alloy. In: First Alloy Workshop. (2006) 71–80
23. Czarnecki, K., She, S., Wasowski, A.: Sample spaces and feature models: There and back again. In: SPLC'08, IEEE Computer Society Washington, DC, USA (2008) 22–31
24. Robak, S., Pieczynski, A.: Employing fuzzy logic in feature diagrams to model variability in software product-lines. In: ECBS'03. (2003) 305–311