Automated Planning for Feature Model Configuration based on Functional and Non-Functional Requirements

Samaneh Soltani¹, Mohsen Asadi¹, Dragan Gašević², Marek Hatala¹, Ebrahim Bagheri²
¹Simon Fraser University, Canada
²Athabasca University, Canada
{ssoltani, masadi, dgasevic, mhatala}@sfu.ca, ebagheri@athabascau.ca

ABSTRACT
Feature modeling is one of the main techniques used in Software Product Line Engineering to manage the variability within the products of a family. Concrete products of the family can be generated through a configuration process. The configuration process selects and/or removes features from the feature model according to the stakeholders’ requirements. Selecting the right set of features for one product from amongst all of the available features in the feature model is a complex task because: 1) the multiplicity of stakeholders’ functional requirements; 2) the positive or negative impact of features on non-functional properties; and 3) the stakeholders’ preferences w.r.t. the desirable non-functional properties of the final product. Many configurations techniques have already been proposed to facilitate automated product derivation. However, most of the current proposals are not designed to consider stakeholders’ preferences and constraints especially with regard to non-functional properties. We address the software product line configuration problem and propose a framework, which employs an artificial intelligence planning technique to automatically select suitable features that satisfy both the stakeholders’ functional and non-functional preferences and constraints. We also provide tooling support to facilitate the use of our framework. Our experiments show that despite the complexity involved with the simultaneous consideration of both functional and non-functional properties our configuration technique is scalable.

Keywords
Software Product Line Engineering, Feature Model, Configuration, Artificial Intelligence.

1. Introduction
Software Product Line Engineering (SPLE) aims at developing a set of software systems that share common features and satisfy the requirements of a specific domain [1]. SPLÉ decreases development costs and time to market, and improves software quality through strategic reuse of assets within a domain of interest. A technique adopted in SPLÉ for managing reusability is commonality and variability modeling through which common assets and their variability are formalized. A software product line lifecycle encompasses a domain engineering process and an application engineering process. In the domain engineering process, a comprehensive formal representation of the products of the domain is developed. This includes a variability model and core assets of the product family. Feature models are among the prevalent variability modeling techniques in SPLÉ and represent variability in terms of differences between the features of the products that belong to the software family. A feature is “a logical unit of behavior specified by a set of functional and non-functional requirements” [2].

On the other hand, the application engineering process is responsible for capturing target application requirements, deriving a concrete product from the variability model through a configuration process, and deploying the product into the users’ environment [3]. Using feature models as variability modeling tools, the configuration process selects a suitable set of features to satisfy the stakeholders’ requirements. Selecting the best set of features based on the stakeholders’ needs is a complicated process because:

1. Features may have either positive or negative impact on the different business concerns of a product, and hence expose different quality attributes. We refer to business concerns of a product (e.g., security and customer satisfaction) and its quality attributes (e.g., performance and cost) as non-functional properties (NFPs), where a non-functional property is defined as: “A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property” [9]. For example, a feature may have a negative impact on security, but a positive impact on customer satisfaction or it could have high performance but low reliability.

2. In addition to functional requirements, stakeholders may have several constraints and preferences over non-functional properties in the product derivation. For example, one stakeholder may ask for a product with high security, high customer satisfaction, and cost less than $1000; and can mention that the customer satisfaction is more important than security.

The need for considering these additional requirements regarding non-functional properties and preferences over non-functional properties (point 2 above) leads to the increased complexity of the configuration process. Therefore, selecting a set of desirable features, which satisfy the stakeholders’ requests and expectations and optimize their preferences, is a very hard task. Various configuration techniques and tools have been developed to help reduce the complexity of the configuration task by automating some steps of it. A few existing techniques allow for the configuration of feature models based on both functional and non-functional requirements [20][25]. Some techniques have addressed this problem by transforming the feature model configuration problem into a Constraints Satisfaction Problem (CSP), and have used CSP-solvers to build optimal configurations [20][26]. The main problem with these techniques is time inefficiency. Other techniques solve this problem by applying approximation algorithms, but their final configurations are only partially optimal [25]. To our knowledge, almost all these works except [18] only support limited types of NFPs (i.e., quantitative NFPs such as footprint and cost) and do not consider qualitative NFPs (e.g., security). Moreover, no work has considered the preferences of stakeholders in terms of the relative importance between non-functional properties in the process of feature model configuration; and relative importance varies depending on the stakeholders’ standpoint and application domain [6]. Relative importance of non-functional properties is especially important for the stakeholders and software designers who are able to define the relative importance among the available functional and non-functional options but have difficulty in deterministically picking their choice from those options [6]. Thus, a product line configuration technique should not only be able to operate over deterministic functional choices, but should also be able to operate given the relative importance between both functional and non-functional properties.

The above issues motivated us to address the following research question: How can a product be automatically derived from a feature model in such a way that it satisfies stakeholders’ requested
functionality and at the same time optimizes their preferences and requirements over the non-functional properties?

To address this research question, we look into preference-based planning techniques. Various preference-based planning techniques exist that produce a plan by optimizing a set of given preferences [4][28]. Hierarchical Task Network (HTN) planning is a popular planning technique, which is suited for domains with hierarchical task decomposition [28][15]. The HTN Planning technique generates plans from a developed hierarchal network of domain tasks and actions [4]. That is, having modeled tasks, actions, and their constraints in the HTN formalism, HTN planners produce a sequence of actions that perform some given tasks. In comparison to other planning techniques, the network structure of HTN can significantly reduce the search space for a plan [4]. Recently, requirements engineering research has employed planning techniques for preference-driven goal-oriented requirements engineering. Their reported results have shown the suitability of planning techniques for similar structures [29]. By way of analogy between the feature model configuration process and the HTN planning problem, we are motivated to investigate the applicability of HTN planning for the product line configuration problem. Hence, we hypothesize that HTN planning can form the basis for a configuration technique that can answer our research question.

We propose and develop a framework, similar to [18], which extends feature models with annotations reflecting different non-functional properties. We then use HTN planning [4][5] to select a set of features which: 1) satisfy the stakeholders’ functional requirements; and 2) optimize the non-functional requirements and preferences of the stakeholders. In comparison to existing configuration approaches, our approach has the benefit of not only satisfying the structural and syntactic constraints of feature models during the configuration process, but also taking both qualitative and quantitative NFPs as well as the relative importance over NFPs into account. In our proposed approach, the configuration problem is converted into an HTN planning problem and planning techniques are utilized to solve the problem [5]. SHOP2 [5], an HTN-based planning system widely used for planning problems, is employed to identify an optimal plan. To configure a product line, we select the features chosen by the SHOP2 planner.

In the context of feature model configuration, the main contributions of this paper are as follows:

- We introduce an easy-to-understand formalism for capturing the stakeholders’ preferences over non-functional properties represented in terms of relative importance;
- We produce an optimal feature model configuration, by transforming a feature model and stakeholder preferences into a planning problem by considering both functional and non-functional requirements.

This paper is organized as follows: Section 2 introduces feature modeling, non-functional properties, and the HTN formalism. Next, in Section 3, the configuration problem is formally defined and a utility function is developed based on non-functional requirements and preferences. Section 4 introduces the transformation rules for converting a feature model annotated with non-functional properties into the HTN formalism. Implementation aspects and tooling are explained in Section 5, which is followed with a comprehensive evaluation and analysis given in Section 6. Section 7 systematically compares our approach with related works and Section 8 concludes the paper and outlines the future work.

2. Foundation

2.1 Feature Models

In SPL/E, a feature model is used mainly for representing variability between products. A feature model provides a formal and graphical representation of the variability relations, constraints, and dependencies of the product lines’ features. It has a tree-like structure [8] in which features are typically classified as: mandatory—a feature must be included in the description of its parent feature in each configuration of the feature model; optional—a feature may or may not be included in its parent description in a configuration of the feature model; alternative feature group—one or more features from a feature group can be included in the description of the parent feature in a configuration of the feature model. Additionally, a number of relations are defined to represent mutual interdependencies (also referred to as integrity constraints) between features. The two most widely used integrity constraints are [8]: requires—the presence of a given (set of) feature(s) requires the inclusion of another (set of) feature(s); and excludes—the presence of a given (set of) feature(s) requires the exclusion of another (set of) feature(s).

Feature models address commonality through core features—mandatory features whose parents are mandatory as well. A feature model can be formally defined as follows:

Definition 1 (Feature model). A feature model is a sextuple FM = (ℱ,M,ℱXO,ℱXC,ℱO,FIC) where 1) ℱ is a set of features; 2) ℱO ⊆ ℱ × ℱ is a set of parent and optional child feature pairs; 3) ℱX ⊆ ℱ × ℱ is a set of parent and mandatory child feature pairs; 4) ℱXO ⊆ ℱ × ℱ and ℱXC ⊆ ℱ × ℱ are sets of pairs of child features and their common parent feature grouping the child features into optional and alternative groups, respectively; 5) ℱIC ⊆ ℱ × ℱ is a set of integrity constraint s (i.e., requires or excludes).

Figure 1 depicts a Check out feature model—a part of the online shopping feature model [7].

2.2 Non-Functional Requirements and Properties

According to a commonly referenced definition [12], non-functional requirements describe the properties, characteristics or constraints that a software product must exhibit. Non-functional properties of software can then encompass aspects like development constraints, business concerns, or external interfaces.

We assume that non-functional properties can be specified in either a qualitative form or a quantitative form. The qualitative non-functional properties such as customer satisfaction or user friendliness can be described using an ordinal scale consisting of a set of predefined qualitative values, which we are referred to as qualifier tags. For example, High negative, Medium negative, Low negative, Low positive, Medium positive, and High positive form the qualifier tags defined for the customer satisfaction. A qualifier tag represents a possible impact of functionality on a qualitative non-functional property. For example, the Credit card feature (i.e., functionality) has high positive impact on international sale; hence it can be annotated with the High-positive qualifier tag for international sale.

On the other hand, metric based values are defined for the quantitative non-functional properties and can be measured for a product. For example, performance can be measured for and assigned to the Credit card feature. After measuring a non-functional value for features, the non-functional property of a product is computed by aggregating the non-functional values of features involved in the product. Based on the nature of NFPs, different aggregation functions can be applied [10][11]. To compute the non-functional values of a product, for some NFPs such as cost and response time, the values are summed; while for others like availability and reliability values are multiplied.
Formally, we define stakeholders’ preferences and constraints as:

must have at least medium security).

We assume that a product has some level of qualitative non-functional system should not have any functionality (i.e., feature) that provides positive impact on security less than the medium level.

In addition to preferences, stakeholders may define constraints (Definition 2); and 3) $CO \subseteq NFP \times V$ is a set of constraints over the values of non-functional properties (the values in $V$ can be either numeric values or qualifier tags based on the type of property).

2.4 Hierarchy Task Network (HTN) Planning

HTN planning fits well with domains consisting of low level actions and high level tasks. High level tasks are hierarchically refined into lower level tasks and finally into actions. HTN planning consists of a planning domain, planning problem, and an output plan [4]. Definition 4 formalizes an HTN domain [4].

Definition 4 (HTN Planning Domain). An HTN planning domain is a quadruple $D = (O, T, M, V)$ where 1) $O$ is a set of operators; 2) $T$ is a set of tasks; 3) $M$ is a set of methods; and 4) $V$ is a set of domain predicates.

An operator (denoted as $o$) represents a low level action, which can be executed in the domain and is formally defined as a quintuple $o = (name(o), pre(o), eff(o), del(o), value(o))$. $pre(o)$ defines a required condition for performing the operator. The effect of performing the operator can also be represented by using a post condition $eff(o)$, $del(o)$ or operator’s delete list shows what becomes false after performing the operator. For every operator, an optional value $value(o)$ can be defined, which shows a required cost for the execution of that operator. The total value of an output plan is the sum of the values of the operators in the plan.

An operator construct (denoted as $t$) represents higher level activities in HTN and can recursively be decomposed into lower level tasks, and finally operators. In HTN, only operators can be executed and tasks can only be reduced into sub-tasks and operators [15]. Refinement of a task into sub-tasks is done using one or more methods (denoted as $m$) corresponding to the task. So, every method describes how a task is decomposed into lower level tasks or operators. A method is a quadruple $M = (name(m), task(m), pre(m), dec(m))$ where $task(m)$ is a parent task, $pre(m)$ is a pre-condition, and $dec(m)$ is a list of subtasks into which the parent is decomposed. A precondition $pre(m)$ defines a required condition for decomposing the parent task. A method is applied only when its precondition is satisfied [4].

The planning problem describes characteristics of a required plan — the objective, initial state, and constraints. The HTN planning problem is formally defined as follows [4]:

Definition 5 (HTN planning problem). An HTN planning problem is a triple $PP = (S, T, D)$ where 1) $S$ is a set of logical atoms (initial state); 2) $T$ is a set of initial tasks; and 3) $D$ is a planning
domain description defined in Definition 4.

As a result of applying a planning technique, a plan containing a sequence of actions that satisfies the objective and the constraints defined in the planning problem is produced. HTN planning formulates the plan by recursively decomposing the tasks into sub-tasks until it reaches the primitive tasks, which can be performed [15]. Similar to [4], we define an HTN plan.

**Definition 6 (HTN Plan).** Let \( PP = (S, T, D) \) be a planning problem defined according to Definition 5. A plan for planning problem \( PP \) is a double \( \pi = (O', c) \) where 1) \( O' \subseteq O \) is a set of operators that will achieve tasks \( T \) from initial state \( S \) in domain \( D \); and 2) \( c \) is the total value of the plan.

3. Problem Statement and Infrastructure

In this section, we highlight the challenges of modeling non-functional properties in product lines and optimizing them for every product derived based on stakeholders’ preferences.

### 3.1 Modeling Non-functional Properties

In the standard feature modeling notation [1], features mainly represent the functional aspects of a product line and non-functional aspects are often neglected. For example, in the online shopping, the *Credit card* feature refers to the functionality provided by the feature and no information regarding its quality such as performance is provided. Some researchers have extended the feature modeling notation with NFPs [20][18][38]. Similarly, we extended feature models with the notion of NFPs, which can be either qualitative or quantitative. In our approach, we assume that *atomic features* (i.e., leave features) in a feature model have concrete implementations. Non-atomic (i.e., non-leaf) features are used for variability and composition relationships of the atomic features. Hence, NFPs are defined for leaf features. If an intermediate feature contains implementations and non-functional properties, we create a mandatory child feature for the intermediate feature and assign the non-functional properties to the child feature.

After identifying domain features, developing a feature model and implementing its atomic features, we can then analyze impact of features on NFPs. For qualitative NFPs, based on existing domain knowledge, the impact of each feature on non-functional properties can be identified and proper qualifier tags can be assigned to each feature’s non-functional properties. On the other hand, quantitative NFPs for the features can be measured using a suitable metric and assigned to the features. We assume that some techniques, like those proposed in [21], can be employed to measure NFPs for each feature. For example, as shown in Figure 1, feature Credit Card is annotated with low negative security, high positive international sale, and medium positive customer satisfaction and its estimated cost, response time, and availability are $600, 50ms, and 90\%$, respectively. Extended feature models are formally defined as follows:

**Definition 7 (Extended Feature Model).** An extended feature model is a nonuple EFM = \( \{ F, F_O, F_M, F_D, F_X, F_Y, F_c, F_A, NFP, A_Q \} \) where 1) \( F, F_O, F_M, F_D, F_X, F_Y, F_c \) stand for a feature model (FM) according to Definition 1; 2) \( F_A \subseteq F \) is a set of atomic features; 3) \( A_Q \subseteq F_A \times (NFP \times V) \) is a set of pairs of atomic features and their annotation pairs of non-functional properties and their value.

### 3.2 Optimizing Stakeholders’ Preferences

In the application engineering process, a concrete product is generated by configuring a feature model based on the target application requirements and by instantiating the reference architecture based on the configured feature model. Stakeholders of each application may have different preferences over NFPs that must be considered in the configuration process. Configuring a feature model based on the stakeholders’ requirements and preferences usually means selecting features such that a feature model configuration satisfies the stakeholders’ functional requirements and constraints and optimizes their preferences. To optimize the configuration with respect to preferences, feature ranks must be computed based on their impact on the NFPs which may be of different importance for the target stakeholders. Additionally, both qualitative and quantitative NFPs must be considered in the computation of feature ranks.

As mentioned in Sec. 2, the stakeholders’ preferences are in the form of relative importance over non-functional properties. To calculate the absolute ranks of non-functional properties based on the preferences, we applied the *Stratified Analytical Hierarchy Process* (S-AHP) algorithm proposed in our previous work [14]. S-AHP is based on the Analytic Hierarchy Process (AHP) [22], which is a well-known pair-wise comparison method used to calculate the relative ranking of different options based on stakeholders’ judgments. We used S-AHP because it enables ranking non-functional properties based on the defined relative importance between them; according to the study in [14], S-AHP is easy to use and does not need too much effort from stakeholders; and finally, it significantly reduces the number of needed pairwise comparisons. A complete description of S-AHP is outside of the scope of this paper and is available in [14].

To consider qualitative non-functional properties in feature ranks, the stakeholders or application engineers should provide a mapping function from qualifier tags onto real values. For example, for customer satisfaction, one can define the following mapping function.

\[
M_{\text{customer sat.}}(QT) = \begin{cases} 
-1 & \text{High negative} \\
-0.5 & \text{Medium negative} \\
-0.25 & \text{Low negative} \\
0.25 & \text{Low positive} \\
0.50 & \text{Medium positive} \\
1 & \text{High positive} 
\end{cases}
\]

The other way for calculating the corresponding real-numbers for the qualifier tags of a qualitative non-functional property is to use S-AHP. In this way, stakeholders specify the relative importance between the qualifier tags of each non-functional property and S-AHP calculates the rank of each qualifier tag. For example, the stakeholder specifies that for the international sale property *high positive* >> *medium positive*, which means a feature with high positive impact on international sale is slightly more important than the feature with medium positive on international sale. In both the methods (i.e., both the mapping function and S-AHP), we assume the values are normalized into the [-1, +1] range.

After defining the ranks of non-functional properties and the mapping function for the qualitative non-functional properties, we describe the following utility function to calculate the ranks of each feature based on an extension to [23]. The rank of features is calculated based on their impact on non-functional properties by considering preferences of the stakeholders formulated in terms of the weight of non-functional properties.

**Definition 8 (Utility function).** Let us assume there are \( \alpha \) quantitative NFPs to be maximized, \( \beta \) quantitative non-functional properties to be minimized, and \( \theta \) qualitative non-functional properties whose impact needs to be maximized. The utility function for feature \( f \) is defined as:

\[
R(f) = \sum_{i=1}^{\alpha} w_i \times \frac{q_i(f) - \mu_i}{\sigma_i} + \sum_{j=1}^{\beta} w_j \times (1 - \frac{q_j(f) - \mu_j}{\sigma_j}) + \sum_{k=1}^{\theta} w_k \times M_k(QT(f))
\]

where \( w \) is the weight of each non-functional property calculated by S-AHP such that \( 0 \leq w_i, w_j, w_k \leq 1 \) and \( \sum_{i=1}^{\alpha} w_i + \sum_{j=1}^{\beta} w_j + \sum_{k=1}^{\theta} w_k + \)
The overall rank of a product is calculated by aggregating the ranks of features selected for the product. The aggregation function used for calculating the product rank depends on the aggregation functions, which exist over non-functional properties of a feature. As discussed in Sec. 2, some quantitative non-functional properties such as response-time are additive and the quality of a composition is calculated by multiplying the quality of features involved in the composition. The multiplication type can be converted into an additive type by computing the logarithm of non-functional values. For qualitative non-functional properties, a qualifier tag assigned to a feature represents a qualitative impact of the feature on the non-functional property. Considering the mapping function that maps the qualifier tags into real numbers, we can calculate the overall impact of a composition of features by adding the impacts of the features involved in the composition. Hence, the aggregation function over the utility functions of features is an additive type by computing the logarithm values of non-functional values. In addition to preferences, the stakeholders’ functional requirements and constraints over non-functional properties must be considered. Hence, the configuration problem is concerned with selecting features that satisfy the functional requirements (i.e., the requested functionality) and constraints and optimize the preferences. Formally, the configuration problem and the configuration are defined as follows.

**Definition 9 (Configuration Problem).** Let \( SPCM = (NFR, RI, CO) \) and \( EFM = (F, F_D, F_M, F_{3O}, F_{XO}, F_{4O}, F_R, NFP, AQ) \) be a: i) stakeholder’s preference and constraint model; and ii) extended feature model as per Definitions 3 and 7. A configuration problem is a triple \( CP = (EFM, SPCM, F_D) \), where EFM is an extended feature model, SPCM is the set of preferences and constraints over NFPs defined by the stakeholders, and \( F_D \) is a set of required atomic features.

**Definition 10 (Configuration).** Let \( CP = (EFM, SPM, F_D) \) and \( EFM = (F, F_D, F_M, F_{3O}, F_{XO}, F_{4O}, F_R, NFP, AQ) \) be a configuration problem and extended feature model, respectively. A configuration is a double COF = \( (F', \mathbb{R}(F')) \) where 1) \( F' \subset F \subset F \) is a set of selected features; and 2) \( \mathbb{R}(F') \) is the total rank of the configuration.

Therefore, in the configuration process, an application engineer deals with the selection of a set of features containing the stakeholders’ required functionality. The total rank of the selected features is the maximum rank based on the stakeholders’ preferences. To help application engineers in the configuration process, we propose applying an AI planning technique.

### 4. Automatic Feature Model Configuration

Having annotated a feature model with NFPs, the process for deriving a new product starts by selecting and deselecting features based on the stakeholders’ requirements reflected through desired features and preferences expressed in terms of relative importance between NFPs. To automate the configuration process, we define transformation rules to convert a configuration problem to a planning problem. To do so, we develop transformation rules to represent extended feature models in the HTN formalism. The transformation is done in two steps: generating an HTN domain model from a feature model, and generating a planning problem from a configuration problem.

When transforming a feature model and the corresponding configuration problem into the HTN domain and HTN problem, we need to convert the maximization problem into a minimization problem, because the SHOP2 planner, used in our implementation, works on minimization only (Negating the ranks of features can do this).

#### 4.1 Generating the HTN Domain

Considering the analogy between tasks in HTN and features in a feature model, there is a need to define task decompositions to reflect feature relations. The HTN method element is used to define decomposition of tasks into sub-tasks or operators. As mentioned in Definition 4, a method contains the parent task, a list of children, and a pre-condition. Here, we have two options: i) we can define different methods with a common parent task; but, an HTN planner (i.e., SHOP2) for decomposing a task into sub-tasks selects just one method for each plan. Hence, this option is suitable for constructing a set of predefined methods. ii) we can define one method for a decomposition in which all children can be considered a list of sub-tasks in the method. In this case, the HTN planner performs the method if and only if the preconditions of all sub-tasks or operators are satisfied. We can use this option for defining decomposition with mandatory child features. Also, before transforming an extended feature model into an HTN domain, a pre-processing step is done to replace optional and OR features. First, every OR group in the feature model is converted into a set of optional features with AND relations between them (Figure 2a). Next, similar to [29] for optional goals, every optional feature \( f^o \) is replaced with a new feature \( f^\phi \) which is decomposed into two alternative features \( f^\phi = f^d \) and \( f^d \) stands for a “dummy” feature (Figure 2b).
ional formula (called attainment formula [29]) is created according to the features that exist in its sub-tree (Figure 3). For instance, the attainment formula for feature Payment Gateway from Figure 1 is $\phi_{PG} = \phi_{VA} \land \phi_{CS} \land \phi_{IP} \land \phi_{P}$. These formulae are later used for pre-conditions of the other features in the feature model.

**Generating Operators.** HTN operators are generated from atomic features in the feature model. Each atomic feature $f$ is translated into an operator $o_f$ and the rank of the feature calculated by the utility function is translated into value($o_f$). A precondition is defined for each operator based on the non-functional properties and integrity constraints defined over its corresponding feature. The preconditions are defined as logical AND expressions of: 1) domain predicates corresponding to qualifier tags of quantitative non-functional properties with which the feature is annotated; 2) an evaluation expression to check whether the feature is allowable to be selected or not based on quantitative non-functional properties constraints; and 3) attainment formula of features having requires and excludes relations with the feature (see Figure 4b). Next, the domain predicate, which corresponds to feature $f$ (i.e., $v_f$) is added as an effect of the introduced operator (i.e., eff($o_f$) = $v_f$). Whenever operator $o_f$ is performed, its corresponding domain predicate becomes true (i.e., $v_f = \text{true}$). For handling quantitative NFP constraints, a logical atom is created showing the maximum available value of corresponding NFPs during the planning process. At the first, this value is set by the requested value of stakeholders and added to the initial state of the planning problem. Then, in the effect of each operator, this value is updated based on the assigned NFP value to the feature. For example, for the Fraud Detection feature in Figure 1, its corresponding operator has $\text{value}(o_{FD}) = 0.5$ (calculated by the utility function) and precondition $\text{pre}(o_{FD}) = v_{\text{Sec} \land \text{AS} \land \text{CS} \land \text{CO}}$, (MaxCost - Fraud-detection > 0) $\land$ (MaxResponseTime - Response time Fraud-detection > 0). The operator corresponding to the Fraud Detection feature is performed only when its precondition is satisfied.

If a feature $f$ is AND-decomposed into features $f_i$ and $f_j$, we define one method which connects corresponding task $t_j$ to tasks or operators corresponding to $f_1$ and $f_2$ (Figure 5a). For alternative feature groups with $n$ sub-features (i.e., $f = \text{XOR}(f_1, f_2, ..., f_n)$), $n$ methods are defined with a common parent task $t_j$ corresponding to feature $f$. Each method connects task $t_j$ to one operator $o_{f_i}$ or task $t_j$ corresponding to the sub-feature $f_i$ of the parent feature $f$ (see Figure 5b). The precondition for every method is specified based on the attainment formula of features required by that feature and features excluded by that feature.

Table 1 gives the mapping rules between extended feature models (after eliminating OR and optional features) and the HTN domain.

**Table 1: Mapping between constructs in extended feature models and HTN Domain models – OR and Optional feature Groups are replaced in the pre-processing step.**

<table>
<thead>
<tr>
<th>Extended Feature Model</th>
<th>HTN Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal</td>
<td>Semantic</td>
</tr>
<tr>
<td>(NFP, qual tags) $\in$ NFP $\times QT$</td>
<td>Qualitative NFP and Qualifier tags</td>
</tr>
<tr>
<td>$f_i$ $\in F_a$</td>
<td>Atomic feature</td>
</tr>
<tr>
<td>$R(f_i)$</td>
<td>Rank of atomic feature</td>
</tr>
<tr>
<td>excluded by that feature.</td>
<td></td>
</tr>
</tbody>
</table>

**Generating the HTN Planning Problem**

After defining the planning domain, the feature model configuration problem is transformed into the HTN planning problem (Table 2 summarizes the mappings). The transformation is done by considering the constraints over the NFPs (CO) and setting their corresponding domain predicates as true to form initial state ($\delta$). For example, if a stakeholder asks for at least medium security, and a specific cost (e.g., $1000), the domain predicates $v_{\text{sec} \text{h}} = \text{true}$ and $v_{\text{cost}} = \text{true}$ and the rest of qualifier tags are set to false; the logical atom (“cost 1000”) is also added as an initial state. Next, the set of required atomic features ($F_a$) and the root of the feature model are translated into initial tasks of the planning problem ($T$).

**5. Tooling and Methodology Support**

As already indicated, SPL methodologies include two processes [1][3][8], namely, domain engineering and application engineering. Assuming that a feature model has been annotated with non-functional properties, a target application can be developed by configuring the feature model reflecting the stakeholders’ requirements and preferences.
In the configuration process, the application engineer captures the stakeholders’ functional requirements including the required feature set \( \mathcal{F}_d \), the preferences (i.e., the relative importance of non-functional properties – \( R_1 \)), and the constraints over NFPs (\( \mathcal{C}_O \)). By applying S-AHP and the utility function (Definition 8), which are implemented in our developed tool Visual feature model plug-in (called Vis-fmp) \(^1\) – an extension of feature model plug-in (fmp) \([16]\) – the ranks of NFPs \( (w_f) \) and features \( \mathcal{R}(f_j) \) are computed, respectively. We employ an efficient planner (i.e. SHOP2), which uses a search-control strategy called ordered task decomposition to perform reasoning on the HTN planning domain \([15]\).

### 6. Evaluation

To assess our technique and the corresponding tool Vis-fmp, we formulated the following research questions:

- **RQ1 (Scalability)**: Can the approach configure feature models, in a reasonable time, based on functional and non-functional requirements and preferences?
- **RQ2 (Effectiveness)**: How effective is the approach in producing a feature model configuration? 
  - **RQ2-1**: Does the approach generate reliable results for application engineers?
  - **RQ2-2**: What is the automation level of the approach?

### 6.1 RQ1 (Scalability)

The purpose of this research question is to evaluate whether a feature model configuration can be performed in a reasonable amount of time. Hence, we conducted several experiments to investigate the research question.

#### 6.1.1 Objects of Study

To evaluate the configuration approach, we adapt Betty FM Generator \([34]\), which enables the random generation of highly-customized feature models, \([34]\) to generate feature models with different characteristics (e.g., number of features, probability of mandatory and optional features, probability of OR and XOR groups, and percentage of integrity constraints). We set the characteristics of generated feature models as: 50%, 25%, and 25% for probability of being features in AND, OR, and XOR groups, respectively. Moreover, 50% of features in AND groups are optional features. The branching factor is also set to 10. These characteristics are backed up by most of surveyed feature models \([32][33]\) to reflect the characteristics of real feature models.

Generating optimal configurations based on the stakeholders’ preferences and constraints is NP-hard. Hence, HTN planners similar to CSP solvers have problems in finding optimal solution for large-scale problems. Although the SHOP2 planner applies some heuristics for improving search time for finding an optimal plan (See ref. \([5]\)), due to explicit representations of states in the memory, SHOP2 runs into memory problems for large domains. However, our experiments showed for feature models with size of 200, SHOP2 returns an optimal plan in feasible time.

According to the results of investigation on non-functional properties done by \([12]\), the number of relevant non-functional properties for different application domains is at most 11. Moreover, Sommerville and Sawyer \([13]\) highlighted that the effective number of non-functional properties is around six. Considering these two studies, we defined 10 non-functional properties; six quantitative; and four qualitative non-functional properties. Five qualifier tags were defined for each non-functional property.

#### 6.1.2 Experimental Setup

For each feature model, we applied our tool to configure the feature model based on preferences and constraints. The evaluation was performed on a computer with an Intel Core DUO 2.2 GHZ CPU, 4GB of RAM, Windows Vista, Java Runtime Environment v5.0, SHOP2 v2.8, and SBCL (Steel Bank Common Lisp) v1.0.55.

To configure the feature models, we considered three independent variables including number of features, number of constraints, and integrity constraints; and time as a dependent variable. For each feature model in the study, features were annotated with quantitative non-functional properties and their values were produced by a random function with normal distribution. From a practical point of view, only some of features (not all) may have impact on a qualitative NFP like security. Hence, to reflect this point, features were randomly annotated with 0 to 4 qualitative non-functional properties with normal distribution, that is, most of the features had one or two non-functional properties. Based on our analysis on SPLOT repository \([7]\), which shows average of 18% of integrity constraints for the real feature models \([31]\), we considered two distributions of integrity constraints: 10% and 20%. Finally, for the constraints over NFPs, four cases were considered: no constraint and constraints over 2, 4, and 6 NFPs to reflect the effect of various

---

\(^1\) https://files.semtech.athabascau.ca/public/projects/VIS-FMP/
6.1.3 Experimental Results

Figure 7 illustrates the average time for configuring features models with different numbers of features and percentage of integrity constraints. Our experiments revealed that, for feature models with less than 200 features, the planner returns results in a feasible time (around 16 seconds). We also investigated the effect of constraints over NFPs on the running time of configuration technique proposed in the paper. To this end, we run the tool with feature models containing 100 features and 20% integrity constraints. The results are shown in Figure 8. In all of these situations, the process of generating optimal configurations was successful.

According to the experiment results, all three independent variables (i.e., number of features, number of constraints, and integrity constraints) have an impact on running time. As shown in Figure 7, for the fixed size of integrity constraints, increasing the number of features raises the time for finding optimal configuration, as increasing the number of features expands the search space for finding optimal plan. Also, within the same number of features, the increase of integrity constraints from 10% to 20% causes increase the time for finding a plan, as the planner needs to check more pair combinations of tasks and operators for optimizing a final plan.

\[
\text{Time(Second)} = a + b \times \text{Number of Features} + c \times \text{Percentage of Constraints}
\]

where \(a\), \(b\), and \(c\) are constants obtained from the experiment.

One way to facilitate the application engineers’ tasks during the configuration process is through automating their task. With respect to the automation level, our approach requires only very few manual interventions. The main tasks of the application engineers are: 1) specifying the relative importance of non-functional properties; 2) creating the mapping function for qualitative non-functional properties; and 3) specifying the atomic features that must be included based on the functional requirements. Computing ranks of non-functional properties based on preferences and finding optimal solution are fully automated in our approach.

7. Related work

To systematically compare the approach, we devise a number of criteria that need to be supported by configuration techniques in order to be effective for application engineering. To define the criteria for systematic comparisons, similar to [37], we applied bottom-up and top-down approaches. Following the bottom-up approach, we identified various important aspects of feature model configuration in description of existing related works and added them to the criteria set. Following the top-down approach, we used an existing survey on configuration of software product lines. We do not claim that this criteria set is complete, but it provides appropriate aspects to compare our work with related works. These criteria include: 1) Managing functional and non-functional requirements; 2) Modeling stakeholders’ preferences; 3) Optimization; 4) Considering stakeholder constraints; 5) Providing tooling support; 6) Automating configuration process; 7) Ensuring the feature model constraints; 8) Effective representation of results to stakeholders; 9) Time efficiency.

In the following subsections, we review existing configuration techniques and compare them w.r.t. the above criteria.

7.1 Feature Model Configuration Approaches

The first attempt, important for our research, is by Czarnecki et al. [17] who introduced a stage configuration process. In that work, a number of configuration steps are introduced to remove variability from feature models. A final product is developed by a consecutive specialization of the feature model through different specialization steps. In each specialization step, some part of the feature model variability is resolved.

Benavides et al. [20] developed an automated reasoning technique over extended feature models (i.e., feature models with extra-functional features). Using their extension, they were able to assign extra-functionality such as price range or time range to features. The purpose of their technique is to find a product of a model based on given constraints. Their technique is based on mapping feature models to CSPs and use of CSP solvers [20][35]. Siegmund et al. [18] have developed a technique called SPL conqueror which...
extends feature models with non-functional properties and applies CSP to find optimal configuration based on user defined objective functions. In their technique, a number of preprocessing steps are taken to reduce the search space for optimal configuration.

White et al. [25] used a Filtered Cartesian Flattening (FCF) method to select optimal feature sets according to resource constraints. In their method, they map the feature selection problem to a multi-dimensional multi-choice knapsack problem (MMKP) [25]. By applying existing MMKP approximation algorithms, they provide partially optimal feature configurations in polynomial time. White et al. [26] also formalize stage configuration and introduced a Multi-step Software Configuration problEm solver (MUSCLE) in which they provide a formal model for multi-step configuration and map it to CSPs. Hence, CSP solvers were used to determine the path from the start of the configuration to the desired final configuration. They consider non-functional properties such as cost constraints between two configurations and formalize them as CSP constraints. Their approach is only applicable for multi-stage configuration and focuses on creating new configurations from an already derived product configuration.

Mendonca et al. [24] proposed a translation of basic feature models into propositional logics and used Binary Decision Diagrams (BDD) as the reasoning system. Their approach concentrates on validating feature models and does not offer a facility for automated configuration. It can be used in a multi-stage configuration process to validate the results of every specialization of a feature model (called interactive configuration). An interactive configuration only checks the structural constraints of feature models and does not consider preferences and non-functional requirements. A tool is implemented to support software developers in validation.

Guo et al. also addressed the challenge of optimizing feature model configuration in their work [36]. They proposed an approach in which Genetic Algorithms are employed to optimize Feature Selection (GAFES).

### 7.2 Comparing the approaches

Table 3 summarizes the results of the comparison of the approaches based on the criteria identified earlier.

#### Managing functional and non-functional properties

Stage configuration [17] and the work in [24] provide no guideline for configuration of non-functional properties. FCF [25], MUSCLE [26], and the technique from [20] support selection of features only based on quantitative non-functional requirements. Our approach and SPL conqueror [18] guarantee the selection of features based on functional and non-functional properties. Furthermore, only SPL conqueror [18] and our approach consider both qualitative and quantitative non-functional properties.

#### Modeling stakeholders’ preferences

CSP based approaches [18][26], FCF [25], and GAFES [36] model stakeholders’ preferences in terms of user defined objective functions. Considering the diversity of non-functional properties, it is not easy for stakeholders to define an objective function, which reflects their preferences. However, our approach provides a systematic and easy technique to capture stakeholders’ preference in terms of relative importance and defines the objective function using these inputs. Stage configuration does not provide any support for stakeholders’ preferences.

#### Considering stakeholders’ constraints

Stakeholders may define constraints based on the resources that are available to them and level of non-functionality. These constraints need to be considered and only a configuration which satisfies the stakeholders’ constraints and optimizes the preferences must be produced. FCF [25], GAFES [36], and CSP based approaches [20][18][26] support constraints on the stakeholders’ resources. On the other hand, our approach handles constraints over both qualitative and quantitative non-functional properties.

### Optimization and time efficiency

Generating optimal configurations based on the stakeholders’ preferences and constraints is NP-hard. All CSP approaches [20][18][26] and our approach ensure optimality of the solution, but they require high computation time. To compare the running time of our approach with CSP approaches, we had limitations. The tool developed by Benavides et al [20], called FAMA, does not support optimization which makes it hard to compare with. The SPL conqueror tool is not publicly available and MUSCLE does not have tool support. FCF [25] and GAFES [36] provides partially optimal solutions in a polynomial time. Stage configuration and the work in [24] do not support optimization of the stakeholders’ requirements.

#### Tooling support and automation

All the approaches, except FCF, provide tooling support. Stage configuration provides tooling support, but little automation for feature model configuration is provided. With respect to the usability, our approach and SPL conqueror [18] apply visualization techniques to present the configuration results to the stakeholders. Our tool shows the quality level of selected features along with them in the same view (Figure 6). Other tools provide basics views for representing configurations to the stakeholders.

#### Feature model integrity constraints

All approaches, except [20] ensure the satisfaction of integrity constraints (i.e., requires and excludes relations) during configuration.

### Table 3: Comparative analysis of related works (++; criterion met, − criterion not met, (+/−) criterion partially met)

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Approach</th>
<th>FR/N</th>
<th>Preference</th>
<th>Optimization</th>
<th>Constraints</th>
<th>Automation</th>
<th>Integrity constraints</th>
<th>Tooling support</th>
<th>Time efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage Configuration</td>
<td>Czarnecki et al. [17]</td>
<td>++/−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>+/−</td>
</tr>
<tr>
<td>CSP - Benavides et al. [20][35]</td>
<td>+/−</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>+/−</td>
</tr>
<tr>
<td>FCF White et al. [25]</td>
<td>+/−</td>
<td>+/−</td>
<td>+/−</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+/−</td>
</tr>
<tr>
<td>SPL Conqueror</td>
<td>Siegmund et al. [18]</td>
<td>+/−</td>
<td>+/−</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+/−</td>
</tr>
<tr>
<td>SPL Conqueror</td>
<td>White et al. [26]</td>
<td>+/−</td>
<td>+/−</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+/−</td>
</tr>
<tr>
<td>Mendonca et al. [24]</td>
<td>+/−</td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+/−</td>
</tr>
<tr>
<td>GAFES Guo et al. [36]</td>
<td>+/−</td>
<td>+/−</td>
<td>+/−</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+/−</td>
</tr>
<tr>
<td>Our approach</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+/−</td>
</tr>
</tbody>
</table>

### 8. Conclusion

In this paper, we targeted an open research question in software product lines: how to select a suitable set of features from a feature model based on both the stakeholders’ functional and non-functional requirements and preferences. We formalized the configuration problem as an HTN planning process and employed an existing HTN planner to generate an optimal configuration based on the stakeholder preferences and constraints over non-functional properties. The experimental results revealed that our approach can be very useful as: 1) it provides an optimal configuration based on stakeholders preferences and constraints over non-functional properties; and 2) it has a good performance on feature models with sizes less than 200 features (Sec. 6). For larger feature models, the approach is computational demanding, similar to other related approaches in the literature (Sect. 7). Another HTN planner (called HTNPLAN-P [4]) applies new heuristics and can support larger planning space problems, so that the practical performance time
can be much improved as shown in [4]. Since HTNPLAN-P is not yet publicly available, we could not test our work on larger feature models. In the future work, we will try to implement such a heuristic and perform additional experimentations.

9. REFERENCES


