Soft Constraints in Feature Models: An Experimental Assessment

Jorge Barreiros[1,2]
[1]Instituto Superior de Engenharia de Coimbra
Instituto Politécnico de Coimbra,
Coimbra, Portugal
jmsousa@isec.pt

Ana Moreira[2]
[2]Dept. de Engenharia Informática
Universidade Nova de Lisboa
Caparica, Portugal
amm@fct.unl.pt

***Abstract*—Feature Models specify admissible configurations of products in Software Product Lines. Constraints are used to represent domain specific knowledge, such as requiring or excluding a feature in the presence of another. Configurations failing to conform to these constraints are deemed invalid. However, in many cases useful domain information cannot be expressed comfortably with such forceful, hard constraints. To overcome this problem, softer constraints, of less forcing nature, can be used. We describe a framework for including soft constraints in feature models based on propositional logic. Analysis procedures for detecting inconsistencies and conflicts in this framework are also described. Test sets are built by injecting soft constraints into publicly available feature models, recreating typical patterns of use. These features are then subjected to automated analysis to assess the prevalence of soft constraint related conflicts and interactions.**

Keywords - Feature Models; Software Product Lines; Soft Constraints; Feature Consistency; Feature Interaction; Semantic Validation

I. INTRODUCTION

Soft constraints, described in [1], can be used to represent uncertain configuration information into feature models [2]. Feature models are frequently used in Software Product Line (SPL) development [3] for specifying valid product configurations, that is, configurations corresponding to a variant that can be created by an application engineer using the SPL. Product variants belonging to the same family are created by specifying a feature configuration, which is then realized by the composition of corresponding artifacts from a common pool of assets (such as requirements documents, design models, code, etc.).

Feature models identify valid configurations by using a feature tree annotated with additional domain constraints. These can be represented graphically (e.g., linking dependent features with a dependency arrow) or textually, by means of arbitrary cross-tree expressions (Boolean expressions depending on the configuration variables). Over-constraining may result in an inconsistent feature model, that is, one where no configuration exists, where all the constraints can be satisfied simultaneously.

Feature models can be represented using logic expressions according to well-known transformations described in [4], [5]. A feature model expression is obtained by conjoining the feature tree expression with the domain constraints.

An example of a feature model can be found in Fig. 1, where *Sound*, *Keyboard*, and *Screen* are mandatory subfeatures of the root feature node *Phone*, while *MP3Player* and *Camera* are optional subfeatures. *Polyphonic* and *Monophonic* are mandatory and alternative subfeatures of the *Sound* feature, and *Monochromatic* and *Color* are alternative subfeatures of the Screen feature. One domain constraint is represented: the *requires* arrow describes that selection of the *Camera* feature implies the selection of the *Color* feature.

Links such as the one connecting *Camera* and *Color* in Fig. 1 describe *hard constraints*. Any configuration that does not respect this constraint is invalid. It can be the case, however, that domain information is not comfortably representable using such strict constructs. For example, a situation can be considered where the overwhelming majority of configurations do indeed respect a certain restriction, but a few exceptions may exist. In this case, restrictions on admissible configurations cannot be as strict. A simple example will be the case of a default selection for a group of alternative selections: if the parent feature of such group is selected, then the preferred alternative configurations may be suggested.

In [1], the use of soft constraints is proposed, similar to hard constraints but of less forcing nature, in these situations. The concept of soft constraint has been described earlier in the context of probabilistic feature models [6]. Probabilistic feature models extend standard feature models by the addition of "soft" constraints that are associated with a degree of probability. These are often obtained as the result of a feature mining processes. We consider the use of a similar concept in standard, deterministic feature models, avoiding the need to resort to mechanisms such as data mining or Baysesian networks to fully specify the required feature joint-probability distributions. The use of soft constraints allows richer semantics to be represented in feature models, with advantages such as enhanced analysis and improved configuration support. An example of such a constraint in Fig. 1 would be "*Sound suggests Polyphonic*", expressing domain knowledge that indicates the more common sound configuration option. Naturally, soft constraints do not need to be restricted to parent-child features as described:

other relations such as "*Monophonic suggests Monochromatic*" can be represented. This type of constraints can be useful for efficiently capturing useful domain information that might be lost otherwise, as it is usually absent in standard feature models. It can be used to good effect for multiple purposes, depending on the specific semantics that are adopted as described later, such as allowing interactive configuration tools to suggest configuration choices to the user.

Using soft constraints also allows some semantic consistency analysis that would otherwise be impossible, e.g., if a suggested dependency can never be realized in a feature model, then probably something is not right. Suggestions may also be unsatisfiable for a certain valid partial configuration (e.g., suggestions cannot be satisfied simultaneously if one feature is selected), highlighting that a trade-off analysis may be in order.

We extend the work presented in [1] by:

- Describing structural patterns of application of soft constraints.
- Describing a process for injecting soft constraints into a feature model for the purpose of automated testing.
- Presenting an enhanced discussion of the impact of soft constraints in feature configuration and corresponding analysis technique.
- Using a prototype tool to collect and analyze experimental results using data from publicly available feature model repositories.

In Section II, we present motivating examples for our work. In Section III, we provide a detailed discussion of soft constraints, discussing benefits of their use, proposing a categorization of the different types of soft constraints, and discussing automated analysis procedures. Section IV presents some typical patterns of use of soft constraints, while in Section V the soft constraint injection algorithm is presented. The tool and experimental results are presented in Section VI, followed by a presentation of related work in Section VII and conclusions in Section VIII.

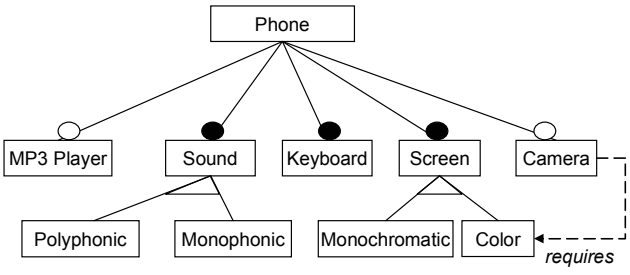

Figure 1. Mobile phone feature model.

II. MOTIVATION

Consider the example in Fig. 2, adapted from [6], where a feature model is used to describe configuration variability for an automobile vehicle. In this case, hard domain restrictions are used to enforce the selection of manual transmission in sports vehicles and to make sure that emission control techniques are always used in products destined for markets with stricter environmental legislations. While observance of such constraints is always found in valid products, soft constraints are used to represent relevant relations between features that, while not as critical or universally applicable as the hard constraints, are also important. In this case, it is well known that the USA market tends to favor vehicles with automatic transmission over those with manual transmission, while the converse is true for the European market. Using soft constraints, such information can be readily represented in the feature diagram, bringing in additional semantics that can be used to good effect.

Another example of the use of soft constraints can be found in Fig. 3. In this case, the feature model is used to represent dynamic variability of the runtime behavior of a real-time system. The system should adapt its behavior to conform to variations in its environment. The state of the operation environment is assessed by appropriate sensors and the corresponding features are (de)selected accordingly, with corresponding impact on the runtime behavior as dictated by the constraints. A base control task is to be active at all times, while fan control is only suggested if the temperature is medium, but mandatory if it reaches a high level. A filtering task is suggested if electric noise is detected.

The need to use soft constraints to describe the variability in this scenario is supported by the fact that the suggested (non mandatory) features may not always be selected because of limited resources (e.g., available CPU load). This means that a feature such as *Fan Control* may in fact remain unselected in the presence of its suggestor (i.e., the *Noisy* feature), which cannot be comfortably expressed using only hard constraints.

These examples suggest that soft constraints can be used to good effect in feature models, by allowing the inclusion of important domain information of non-forcing nature.

III. SOFT CONSTRAINTS

In this section, we discuss the benefits gained by using soft constraints in feature models and present a categorization of alternative semantics. We then discuss automated analysis procedures for identifying inconsistencies and other relevant information, such as features that cannot be selected if satisfaction of a soft constraint is sought.

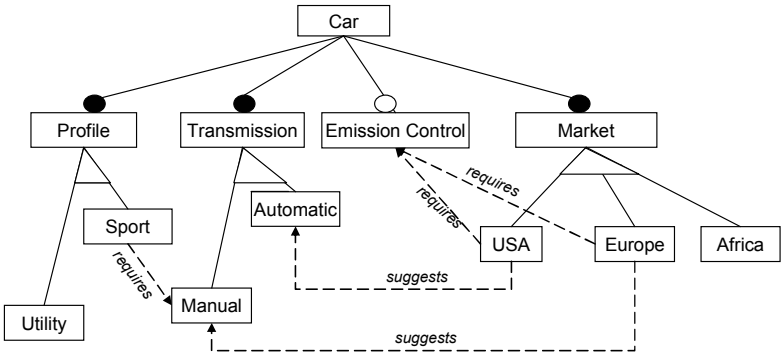

Figure 2. Feature model for car configuration

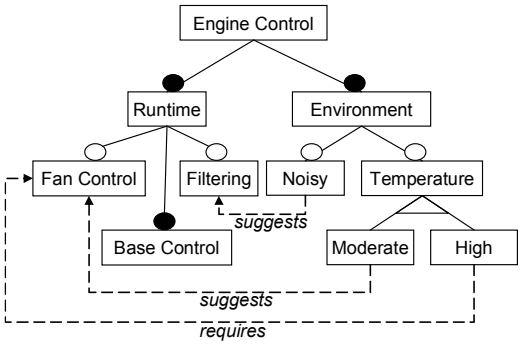

Figure 3. Engine control system

A. Benefits of soft constraints

Benefits of using soft constraints in feature models include:

- **Improved configuration support**: Interactive configuration and completion techniques can assist the configuration of feature models by assessing the liveliness of features after each configuration step. Starting from an empty configuration where all features are considered to be unspecified (neither selected or deselected), after a feature is selected or deselected by the user, the liveliness of all features is re-evaluated with respect to the partial configuration already defined. Features that are found to be dead (always unselected) in that partial configuration can be safely deselected automatically. Conversely, features that are common to all configurations that include the partial configuration so far specified can be automatically selected. For example, if the developer specifies feature *C* in Fig. 4 to be selected, then features *D* and *E* can be automatically deselected by the configuration tool, as no valid configuration including feature *C* will contain either (i.e., both are dead in all configurations where *C* is selected). Similarly, *A* and *root* are common to all such configurations, so they can be selected automatically, leaving only feature B unspecified. Interactive configuration and completion tools can use soft constraint information to make configuration suggestions to the user. For example, if "A suggests B", the configuration tool can propose the selection of *B* by default whenever *A* is selected and B is unspecified. Also, if a valid configuration fails to conform to a large percentage of soft constraints, it can be flagged to the developer as suspicious. Feature configuration support for feature models with soft constraints is described in [7].
- **Improved semantic-oriented consistency checks**: Standard consistency analysis of feature models is concerned with ensuring that valid configurations do exist. If soft constraints are present, it is possible to make sure that configurations are available that verify the suggested dependencies. If that is not the case, this may be a sign that an analysis or modeling error has occurred. For example, if it were actually impossible to configure a car for the European market with manual transmission despite such association being suggested (e.g., because of the unintended collateral side effect of some hard constraints), this would be highly suspicious and should be reported to the developer for additional consideration. This could be the case if hard domain restrictions would make it impossible to select a configuration where both such features are selected.
- **Controlled generalization of feature models**: A generalization of a feature model is a transformation that increases the number of admissible configurations, making sure that previously valid configurations remain valid. In some cases, soft constraints can be used as a mechanism for controlled generalization of feature models. For example, if it was found, after creating the feature model in Fig. 2, that it should actually be possible, under certain circumstances, to produce vehicles without emission control for the USA market, the hard restriction that forbids such products from being created could be transformed into an equivalent soft constraint. This would have the benefit of preserving important domain information while accommodating the need to allow for spurious "rogue" configurations.

B. Semantics and Categorization

Soft constraints can be interpreted according to different semantics, from configuration suggestions (e.g., describing a predominant configuration options such as described in [6]) to stricter impositions that must be enforced if possible (i.e., a feature must be selected if possible). According to the adopted interpretation, different types of analysis and interpretations may be possible. Therefore, we must consider the possible semantics. These can be broadly categorized in two different categories:

- **Annotational**: A soft constraint with an annotational semantics does not impose any additional restriction when added to a feature model. Its main purpose is to embed domain information in the feature model to assist the configuration automation and semantic consistency checking. The validity of any specific product configuration is never influenced by the presence of an annotational soft constraint.

- **Normative**: A normative soft constraint must be considered when assessing the validity of a product configuration. These constraints represent configuration information that may potentially condition the validity of some configurations. A normative soft constraint must be satisfied if possible, but can be ignored otherwise. The concept of "possible satisfaction" is, generally, always dependent on the characteristics of the feature model and is also potentially dependent on domain-specific information (external to what is represented on the feature model: see below). A normative soft constraint may change the validity of a configuration (with respect to the unconstrained feature model), but it may never cause a feature model to become inconsistent. Normative constraints can be interpreted informally as meaning "requires-if-possible", "may-require", "require-if-does-not-make-configuration-invalid" or some other similar designation. Applying normative constraints entails the need to assess the "possibility" of selecting a specific feature. The topology of the feature model and cross-tree-constraints is always a decisive factor in making that assessment (i.e., it cannot be reasonably considered "possible" to select a feature when doing so would generate an invalid configuration). However, it may be the case that the feature model information is not sufficient to assess the possibility of selecting a feature: in this case, external factors, not represented in the feature model would come into play. This suggests the following additional characterization of normative constraints:

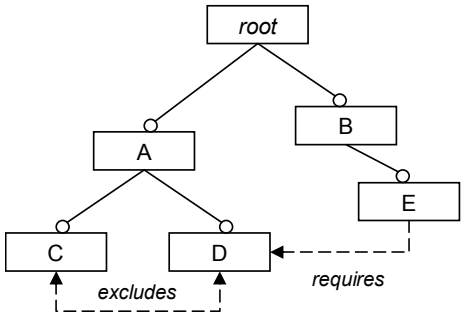

Figure 4. Iterative configuration example

- **Internal:** The feature model holds all the information required to assess selection possibility.

- **External:** The information in the feature model alone is not sufficient for assessing possibility of selection. External factors come into play.

In the example of Fig. 2, if the soft constraints are interpreted under annotational semantics, then any configuration that upholds the hard constraints is considered valid, regardless of complying or not with the soft constraints. On the other hand, if an (internal) normative semantic is considered, the following interpretation holds: "If the *USA* feature is selected, then the *Automatic* feature must be selected, unless doing so would generate an invalid configuration". That is, a normative soft constraint should be interpreted as a hard constraint, unless doing so would turn an otherwise valid configuration into invalid. In Fig. 3, a potential example of external normative soft constraints is represented: in this case, the *Fan Control* feature should always be selected if the *Moderate* heat feature is selected, unless that is not possible, according to domain information that is not necessarily integrated in the feature model. For example, knowing that the implementations of the *Base Control*, *Fan Control*, and *Filtering* features compete for a limited resource (CPU load), assessing of the possibility of including the *Fan Control* feature must be conducted with respect to external information. It is out of the scope of this work to discuss how such external information would be obtained or retrieved – as examples, an oracle could be used to provide the required information or a domain specific ontology could be queried. External normative constraints require considering information beyond the one available on the feature model and will not be discussed further in this work. Therefore, in the remainder, when referring to normative soft constraints, internal semantics are assumed.

Table I presents a description of soft constraints and their intended meaning. Table II presents a summary overview of hard and soft constraint categorization and their effects on feature model consistency and configuration validity.

TABLE I. SOFT CONSTRAINT DESCRIPTION

Soft Constraint	Interpretation
A suggests *B*	A⇒ B
A discourages *B*	A ⇒ ¬B
A absence-suggests *B*	¬A ⇒ B
A absence-discourages *B*	¬A ⇒ ¬B

TABLE II. CONSTRAINT CATEGORIZATION SUMMARY

Nature	Subtype	Affects FM consistency?	Affects config Validity?
Hard		Yes	Yes
Soft	Normative	No	Yes
	Annotational	No	No

C. Normative Soft Constraint Analysis

Normative soft constraints may change the assessment of the validity of configurations with respect to the unconstrained feature model. This results in a change of the model expression when a new soft constraint is introduced in an existing feature model. The effect of inserting an internal normative soft constraint (A suggests B) into a feature model with feature expression *F(A,B,…)* can be obtained by considering that:

- Any configuration valid in the constrained model will also be valid in the unconstrained model. That is, *F(A,B,…)* should hold.
- The soft constraint should be upheld *(A ⇒ B)*, but not at the cost of invalidating the configuration. Knowing that the soft constraint fails to hold when *A* is selected and *B* is not, this will only be acceptable if switching the value of *B,* in this scenario, would not be allowed according to the unconstrained model[1], that is, $\neg F(A,\neg B,...)$.

These considerations lead to the following formulation:

$$F_S(A,B,...) = F(A,B,...) \wedge ((A \Rightarrow B) \vee \neg F(A,\neg B,...)) \quad (1)$$

where F_S is the resulting feature model expression.

Standard feature model techniques can be applied to analyze the resulting feature expression, e.g., satisfiability-based techniques are commonly applied to the analysis of feature model expressions [8], for tasks such as finding dead features. This can be also done in a feature model annotated with soft constraints by considering the relevant F_S.

[1] Strictly speaking, it would also be possible to satisfy the constraint by deselecting *A* rather than selecting *B*, but we find that solution to be counter-intuitive, with respect to the compositional approach inherent in the feature selection process. In other words, a constraint may force the selection of a feature the user has not selected, but will not force the deselection of a previously selected feature.

Equation (1) can be applied iteratively with respect to all soft constraints, in some priority order, to obtain the feature expression corresponding to a feature model with multiple soft constraints. Nevertheless, one difficulty must be pointed out. If *F* is in clause normal form (CNF), the standard input format for most SAT solvers, then the number of clauses will increase exponentially as additional normative soft constraints are composed. This makes it much more challenging to analyze normative soft constraints rather than their annotational counterparts. Fortunately, annotational soft constraints include valuable information that can be more efficiently subjected to automated analysis.

D. Soft Constraint Analysis

The inclusion of soft constraints in a feature model brings additional semantics that allow improved consistency and sanity checks to be performed. Annotational soft constraints do not alter in any way the space of admissible configurations. Nevertheless, the question of whether or not the soft constraints themselves can be upheld is relevant. In the remaining text, we assume an annotational interpretation of soft constraints.

When introducing a soft constraint into a feature model, all configurations previously valid will remain so. However, if the soft constraint impacts the feature model meaningfully, at least some of those configurations will fail to hold all the soft constraints (or else the soft constraint will be reproducing information already present in the feature model: an example would be a suggestion of inclusion of a parent feature). If no configuration exists where all the soft constraints are upheld, the soft constraints are inconsistent, in the sense that their simultaneous satisfaction is impossible (this is not the same as feature model inconsistency, as defined in the introduction).

An analysis procedure may be used to identify such situations. We begin by defining a constraint as *active* if its implicant (e.g: *A* in *A⇒B*) is true according to the expressions defined in Table II. It may be impossible to simultaneously activate all constraints according to the feature model. In this case, the constraint set is *orthogonal*. In this case, the constraints may not be satisfied simultaneously, because its implicants cannot be verified simultaneously. A more interesting situation occurs when all constraints can be active simultaneously, but satisfaction of the soft constraints is not possible. In this case, the soft constraints are said properly *inconsistent*.

Inconsistencies and orthogonally can be analyzed by verifying the satisfiability of Boolean propositions composed from the feature expression and soft constraint expressions. Although verifying the satisfiability of a proposition is an NP-Complete problem, SAT solvers have proven to be efficient tools for the majority of expression of practical interest for feature modeling [8]. Let **F** be the feature expression, **E** the conjunction of the soft constraint expressions and **P** the conjunction of the soft constraint implicants according to the expressions found in Table I:

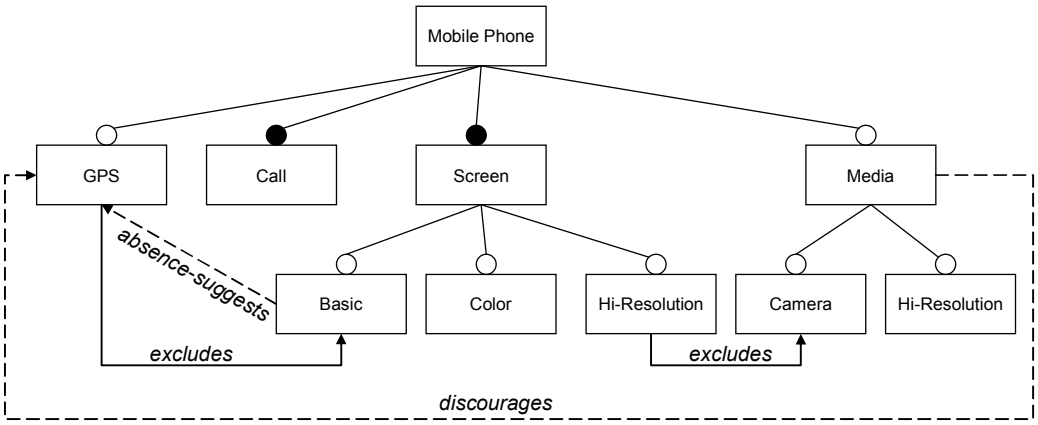

Figure 5. Feature model annotated with soft constraints (source: adapted from www.splot-research.org)

1. Check for satisfiability of the conjunction **F P**. If it is unsatisfiable, then the soft constraints are inconsistent due to orthogonality.

2. A proper inconsistency in a non-orthogonal set of constraints can be identified by assessing the satisfiability of **F P E**. If that expression is found to be unsatisfiable, then an inconsistency is detected.

Inconsistencies are not the only interesting interaction between soft constraints and feature models. In fact, any constraint of the format presented in Table II may be satisfied by falsifying the implicant. It may be the case where a soft constraint may only be satisfied by this recourse. In this case, we say the implicant is *hidden* by the soft constraint, in the sense that it may never be made true if the soft constraint is to be upheld. This is relevant, as it makes it possible to identify specific configuration profiles that must be upheld for satisfaction of the soft constraint to be possible. In the sequel, we will refer to hidden implicants as hidden features, although strictly speaking the implicant may not correspond to the selection of a single feature.

Hidden features can be identified in the following way. Let **F** be the feature expression, **E** the conjunction of the soft constraint expressions, and $\mathbf{I_c}$ the implicant of soft constraint **c**. Then:

1. If **F E** is satisfiable, then at least one configuration exists that satisfies the feature model and soft constraints, and proceed to step 2.
2. For all $\mathbf{I_c}$: If **F E** $\mathbf{I_c}$ is not satisfiable, then no configuration exists that satisfies the feature model and soft constraints with $\mathbf{I_c}$ being true. $\mathbf{I_c}$ is therefore hidden by **c**.

IV. PATTERNS OF SOFT CONSTRAINT USE

In this section, we propose some typical patterns for the use of soft constraint annotations in feature models, with respect to the topological structure of the annotated feature model. We strived for identifying such patterns for multiple reasons:

- Identifying typical patterns of use improves understanding of the subject matter and provides insight into potential applications of soft constraints.

- If typical patterns of application are identified, with respect to the topological structure of the feature model, it becomes possible to automatically annotate feature models with such constraints for the purposes of generating test cases for experimenting and validating the automated analysis techniques we describe.

In this way, we have identified three patterns that describe specific cases of application of soft constraints. Naturally, this list cannot be considered exhaustive in any way, but it is sufficient for the purposes of providing a basic understanding of soft constraint use and allowing the automated creation of test cases. The description of these three patterns follows:

A. Soft Constraint Pattern: Reversed Constraint Suggestion

The pattern *Reversed Constraint Suggestion* (RCS) describes a situation where a feature model includes a hard constraint *C*, specifying a *requires* or *excludes* relation between two features (or their absence). The *RCS* of constraint *C* is a soft constraint that specifies that the reversed relation should also hold, that is, RCS(A⇒B) = B *suggests* A.

Examples of RCS can be found in Fig. 5 and Fig. 6. The feature model in Fig. 6 is an adaptation of a simple automobile product line described in [9]. A hard constraint determines that the presence of the "Lateral Parking" feature requires selection of the "Lateral Range Finder" feature. The RCS of this hard constraint can be found in the feature model: using the lateral range finder feature suggests the use of lateral parking. In Fig. 5, the "Basic *absence-suggests* GPS*"* soft constraint is the RCS of the "GPS *excludes* Basic".

The conceptual interpretation of the *RCS* pattern is based on the intuitive notion that, in some cases, if all the requirements for a certain feature (as specified by hard constraints) are met, then it may be sensible to opportunistically select it. For example, in Fig. 6 example, the suggestion of selecting "Lateral Parking" in the presence

of the "Lateral Range Finder" sensor can be interpreted as a suggestion to take maximum advantage of all the potential capabilities provided by the installed hardware in the vehicle. A similar perspective is that the "Lateral Range Finder" sensor can be considered of reduced utility if the "Lateral Parking" feature is not selected.

B. Soft Constraint Pattern: Group Selection Suggestion

The pattern *Group Selection Suggestion* (GSS) is related to the preferential configuration options of grouped features. Given a feature group G with subfeatures G_1, G_2,…, a GSS is a soft constraint that describes preferred group configurations (e.g.: GSS(G) ={G⇒G_x}). An example can be found in Fig. 7. The GSS represents the notion of preferred configuration options for feature groups.

C. Soft Constraint Pattern: Optional Selection Suggestion

The pattern *Optional Selection Suggestion* (OSS) encompasses a broad range of situations where domain-specific interdependencies affect the configuration of optional features. An *OSS* represents one such situation, by interlinking the configuration of two optional features via a soft constraint.

One such situation is represented in Fig. 5, where selection of the optional "Media" feature discourages selection of the optional "GPS" feature. This soft constraint could be understood to represent domain-specific constraints being embedded in the feature model: in this specific case, this soft constraint could be in place because of hardware performance limitations that could entail a non-negligible degradation of performance if both features are selected.

The OSS pattern represents domain-specific dependencies between optional features and as such has very generic scope. Specializations of this pattern may be devised if such domain-specific knowledge is considered. However, for the purposes of understanding typical structural patterns, it is sufficient.

V. AUTOMATED TEST CASE GENERATION

Although a large number of feature models can be obtained from online repositories [10], successfully applying these to the validation of the techniques proposed in this work entails the need to individually annotate the models with soft constraints. These models, however, are concerned with a large diversity of domains of application, making it extremely difficult to seek the help of independent domain experts for all are relevant areas of expertise. Consequently, manual annotation may be feasible for only a relatively small number of models in specific areas of expertise. However, it is still a significantly time consuming task, where arbitrary decisions, that may put into question the credibility of the results, are unavoidable. Also, only a fraction of all available models may be considered, wasting a significant portion of potentially available resources and putting into question the representativity of any results that are obtained.

To address these difficulties, we have chosen to annotate all the feature models by automatically injecting soft constraints according to the usage patterns (RCS, GSS, and OSS) described in Section VI. This approach has significant benefits:

- It allows any models in repository to be used for validation and test purposes.
- It speeds up test case generation significantly, and multiple configurations of soft constraint annotations for each model can be generated, allowing for a large test set to be created with the corresponding emergence of observable statistical properties and trends.

A. Test Case Generation

The following strategy was used to annotate a given base feature model with soft constraints according to the RCS, GSS, and OSS patterns.

Given the number of constraints N_c, number of groups N_G, the number of optional features N_O of the base feature model, and the configurable density parameters D_{RCS}, D_{GSS}, and D_{OSS},

1. Randomly select a constraint in the base feature model, generate the corresponding soft constraint according to RCS and insert it into the model. Do this N_c *D_{RCS} times.
2. Randomly select one group in the base feature model and one subfeature belonging to that group. Generate the corresponding soft constraint according to GSS and insert it into the model. Do this N_G *D_{GSS} times.
3. Randomly select two optional features in the base feature model, generate the corresponding soft constraint according to OSS and insert it into the model. Do this N_O *D_{OSS} times.

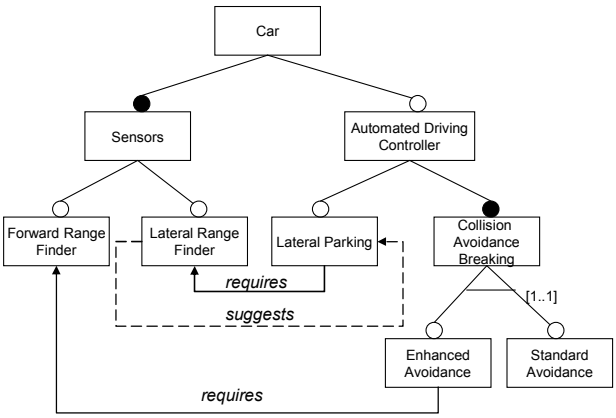

Figure 6. Example of a Reverse Constraint Suggestion, based on a feature model of a simple automobile product line.

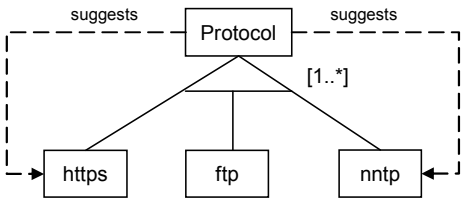

Figure 7. Example of Group Selection Suggestions

The density parameters control the number of soft constraints introduced. Nevertheless, fluctuations may occur because degenerate or nonsensical soft constraints (such as a feature suggesting one of its (grand-) parents) are ignored so they do not pollute or bias the results. Duplicate soft constraints may also be generated, so the actual number of soft constraints injected into the feature model may be less than N_c *D_{RCS} + N_G *D_{GSS} + N_O *D_{OSS}. This approach is effective and simpler than trying to always generate valid, distinct soft constraints in sufficient number (which may very well be impossible, depending on the chosen density parameters and structural properties of the base feature model).

VI. EXPERIMENTAL RESULTS

A. Tool description

We developed a Java based tool that processes and analyses feature models annotated with soft constraints. The SAT4J package was used for resolving satisfiability problems. The SXFM (Simple XML Feature Model) file format [10], used for storing feature model descriptions, was extended to include soft constraint information. The partial contents of a SXFM file containing soft constraint information can be observed in Fig. 8. Our tool is also capable of injecting soft constraints into feature models and conducting the analysis described in Section IV.

B. Test and Validation

For test and validation purposes, we have selected to use all the feature models with more than 40 features currently available in the SPLOT feature model repository [10]. These were provided by the site users and include models from both academic and industrial origin. This provided us with a large set of feature models with diverse characteristics and relevant dimension to validate our work.

Table III presents the relevant characteristics of the feature models. The descriptions were taken *verbatim* from their entries in the online feature repository.

```
<feature_tree>
:r Car
  :o Automated Driving Controller
    :m Collision Avoidance Braking
      :g [1,1]
        : Standard Avoidance
        : Enhanced Avoidance (enhanced_avoidance)
    :o Parallel Parking (parallel_parking)
  :m Sensors
    :o Lateral Range Finder (lateral_range_finder)
    :o Forward Range Finder (forward_range_finder)
</feature_tree>
<constraints>
c1: ~enhanced_avoidance or forward_range_finder
c2: ~parallel_parking or lateral_range_finder
</constraints>
<softconstraints>
s1: lateral_range_finder suggests parallel_parking
s2: forward_range_finder suggests enhanced avoidance
</softconstraints>
</feature_model>
```

Figure 8. Extract of SXFM file extended with soft constraint information.

The purpose of the experiments is to observe to what extent the extent inclusion of soft constraints in feature models may lead to hidden features and inconsistencies as described in Section V, as well as assessing the effectiveness of the analysis algorithm. To this effect, soft constraints were injected in these models and the analysis algorithm was run to identify inconsistencies and hidden features. Although weak real world representativity is always a risk when using automated test case generation, this concern is mitigated by employing typical patterns of usage to guide soft constraint injection.

Different test sets were created by injecting soft constraints with increasing density parameters D_{RCS}, D_{GSS}, and D_{OSS}. All density parameters were set to the same value in each test set, and four different test sets were created, with density values of 0.125, 0.25, 0.5, and 0.75.

The results in Fig. 9 represent the aggregate results of running soft constraint injection and analysis for the models in Table III, while Table IV presents the results for each individual model. Because feature injection is a stochastic process, experiments were run 5 times for each feature model for each different setting of the density parameters, for a total of 20 runs per feature model.

The injection algorithm fails to inject any soft constraint into three feature models (*Thread*, *Datbase Tool*, and *DS Sample*) at the lowest density setting, because of their specific topological properties. For preserving homogeneity, results for these three models are not represented in Table IV, since only higher density results are available; comparison with other results would not be meaningful.

Results in Fig. 9 illustrate that, as can be expected, inconsistencies noticeably increase with higher densities of soft constraints. The number of inconsistencies seems to increase linearly with the number of soft constraints, while the number of orthogonal constraint sets increases more rapidly and appears to converge to a value in the vicinity of 80%. The number of unaffected feature models decreases correspondingly, until it drops bellow the number of inconsistencies at densities of approximately 65%. An important observation is that a significant number (approximately 20%) of inconsistent soft constraints is found even for low densities of soft constraints. This highlights the usefulness of automated analysis procedures for validating feature models annotated with soft constraints.

Results in Table IV show that adding soft constraints to two specific feature models (*PFTeste1* and *DELL Laptop/Notebook Computers*) systematically resulted in the appearance of an inconsistent set of soft constraints set. Analyzing the characteristics of these two models, it is easily observed that one common distinguishing feature is the very high number of hard constrains in each (even after normalizing according to the number of features). It can be concluded that constraint density, and not feature model dimension or other factors, is the main contributing factor for the appearance of inconsistent soft constraint suggestions.

Hidden features were also identified. Table V presents the percentage of soft constraints hiding a feature as a percentage of the total number of soft constraints. For most feature models, the percentage of hidden features increases

with the density of soft constraints. The average results displayed in the last row of Table IV confirm this tendency. No hidden features were identified in *PFTeste1* or *DELL Laptop/Notebook Computers*, for the simple reason that those feature models systematically generated inconsistencies precluding satisfaction of constraints. The presence of hidden implicants is found to be prevalent enough in most models so that automatic detection and report can be considered useful.

The analysis algorithm was found to be very efficient. Experiments were conducted in a netbook with 1 Gb of RAM memory, taking approximately 5-10s for loading, injecting soft constraints and analyzing once every one of the feature models considered in this test (between 0.15 and 0.3 s per feature model).

VII. Related Work

In [6], probabilistic feature models are described that use soft constraints as descriptions of features that have high probabilities of being concurrently selected in the same configuration. Probabilistic feature models and corresponding samples spaces are suited to represent feature models obtained through feature mining processes, because complete feature joint probability distributions must be obtained. Incomplete specifications must be handled by complementary mechanisms such as Bayesian networks. In our work, the use of standard Boolean propositional logic capitalizes on established tool support and improves accessibility to the developer. This allows soft constraints to be more readily used to represent important domain knowledge in feature models.

"Encourages" and "discourages" constraints have been proposed for feature models in [11]. However, no precise semantics have been provided, precluding automated analysis and reasoning as described in our work.

In [12], fuzzy logic is applied to relate feature configurations to costumer profiles. Although it is a significant departure from standard feature modeling approaches familiar to developers, Fuzzy logic is a powerful alternative tool for handling uncertainty.

While we focus this work on detection of inconsistencies and semantical analysis (e.g: detection of hidden features) of feature models annotated with Boolean soft constraints, in [7] improved configuration support is described.

Soft constraint frameworks have been studied in the context of constraint programming. These approaches focus on the search of a optimal variable assignment with respect to a set of quantified soft constraint expressions, as opposed to semantical and consistency analysis [13].

VIII. Conclusions

We have experimentally demonstrated the usefulness and viability of automated analysis of soft constraints in feature models. Typical patterns of soft constraint use were described. These were injected in publicly available feature models. In this process, inconsistencies and hidden features were introduced. These situations can correspond to potential semantic errors and should be reported back to the user for further inspection. Our tool was applied and was effective in identifying these potential problems. This demonstrates that a framework for handling soft constraints in feature models using propositional logic can be a valuable tool for feature modeling. Future work will be conducting in identifying additional patterns of soft constraint use. The role of soft constraint usage in typical development tasks such as refactoring or domain modeling will also be investigated.

TABLE III. Feature Models

Description	Number of Features	Optional Features	Number of Groups	Hard Constraints
AndroidSPL	45	8	9	5
Arcade Game PL	61	4	9	34
bCMS system	66	6	8	2
Billing	88	45	2	59
Car Selection	72	10	19	21
Consolas de Videojuegos	41	11	2	5
Database Tool	40	7	7	0
DATABASE_TOOLS	70	20	7	2
DELL Laptop/Notebook Computers	46	1	8	110
Documentation_Generation	44	3	9	8
DS Sample	41	0	6	0
Electronic Drum	52	1	11	0
E-science application	61	7	16	2
HIS	67	10	6	4
Hotel Product Line	55	31	7	0
J2EE web architecture	77	26	11	0
Letovanje	43	3	13	2
Linea de Experimentos	52	11	4	4
Meshing Tool Generator	40	8	11	17
Model_Transformation	88	11	25	0
OW2-FraSCAti-1.4	63	39	2	46
PFTest1	56	5	8	90
Plone Meeting	57	13	9	0
Printers	172	1	28	0
Reuso – UFRJ – Eclipse1	72	40	7	1
Smart Home	56	36	4	0
Smart Home v2.2	60	30	6	2
SmartHome_vConejero	59	33	0	3
SPL SimulES, PnP	59	8	14	0
Thread	44	0	7	0
Video Player	53	17	9	2
Video Player	71	12	5	0
Web_Portal	43	17	6	6
Xtext	137	95	0	1

TABLE IV. INCONSISTENCY RESULTS PER FEATURE MODEL

Description	Unaffected	Orthogonal Inconsistent	Non-orthogonal Inconsistent
Model_Transformation	50%	50%	0%
OW2-FraSCAti-1.4	25%	75%	0%
Documentation_Generation	65%	35%	0%
SPL SimulES, PnP	45%	55%	0%
PFTest1	0%	0%	100%
DELL Laptop/Notebook Computers	0%	0%	100%
Linea de Experimentos	50%	50%	0%
Letovanje	50%	45%	5%
Xtext	0%	100%	0%
Smart Home v2.2	30%	60%	10%
SmartHome_vConejero	20%	80%	0%
bCMS system	60%	40%	0%
E-science application	50%	50%	0%
DATABASE_TOOLS	40%	60%	0%
Reuso - UFRJ - Eclipse1	55%	45%	0%
Hotel Product Line	20%	75%	5%
Electronic Drum	30%	70%	0%
Video Player	55%	45%	0%
Billing	40%	60%	0%
Smart Home	40%	60%	0%
Plone Meeting	25%	75%	0%
Meshing Tool Generator	15%	85%	0%
AndroidSPL	40%	60%	0%
Arcade Game PL Feature Model	40%	40%	20%
Web_Portal	35%	60%	5%
Consolas de Videojuegos	95%	5%	0%
Car Selection	40%	60%	0%
Printers	50%	35%	15%
HIS	30%	10%	60%
J2EE web architecture	30%	70%	0%

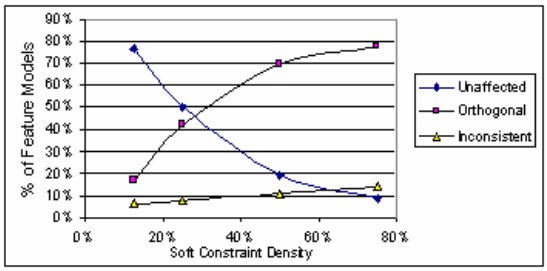

Figure 9. Aggregate results: unaffected, orthogonal and inconsistent feature models.

TABLE V. HIDDEN FEATURE RESULTS PER FEATURE MODEL

Description	Percentage of constraints hiding features			
	density 0,125	density 0,25	density 0,5	density 0,75
Model_Transformation	0%	0%	0%	79%
OW2-FraSCAti-1.4	4%	56%	98%	91%
Documentation_Generation	0%	0%	11%	10%
SPL SimulES, PnP	0%	33%	67%	69%
PFTest1	0%	0%	0%	0%
DELL Laptop/Notebook Computers	0%	0%	0%	0%
Linea de Experimentos	0%	0%	0%	20%
Letovanje	0%	11%	29%	36%
Xtext	3%	6%	10%	21%
Smart Home v2.2	0%	0%	32%	33%
SmartHome_vConejero	0%	0%	30%	32%
bCMS system	0%	33%	13%	0%
E-science application	0%	0%	21%	63%
DATABASE_TOOLS	0%	0%	8%	49%
Thread	-	0%	22%	47%
Reuso - UFRJ - Eclipse1	0%	6%	30%	28%
Hotel Product Line	0%	0%	17%	11%
Database Tool	-	0%	6%	43%
Electronic Drum	0%	0%	67%	100%
Video Player	0%	0%	0%	6%
Billing	45%	41%	100%	100%
Smart Home	0%	0%	8%	9%
Plone Meeting	0%	7%	10%	33%
Meshing Tool Generator	0%	21%	0%	14%
AndroidSPL	0%	13%	58%	88%
Arcade Game PL Feature Model	0%	0%	0%	2%
Web_Portal	0%	11%	27%	32%
Consolas de Videojuegos	0%	22%	21%	25%
Car Selection	0%	0%	15%	0%
Printers	0%	0%	43%	41%
DS Sample	-	0%	0%	67%
HIS	0%	8%	27%	12%
J2EE web architecture	0%	8%	6%	20%
AVERAGE	**2%**	**8%**	**23%**	**36%**

REFERENCES

[1] J. Barreiros and A. Moreira, "Soft Constraints in Feature Models," in *International Conference in Software Engineering Advances, ICSEA'11* Barcelona, 2011.

[2] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*: Addison-Wesley Professional, 2000.

[3] P. Clements and L. Northorp, *Software Product Lines:Practices and Patterns*: Addison-Wesley, 2001.

[4] D. Batory, "Feature-Oriented Programming and the AHEAD Tool Suite," in *26th International Conference on Software Engineering*: IEEE Computer Society, 2004.

[5] K. Czarnecki and A. Wasowski, "Feature Diagrams and Logics: There and Back Again," in *11th International Software Product Line Conference (SPLC)* Kyoto, 2007, pp. 23-34.

[6] K. Czarnecki, S. She, and A. Wasowski, "Sample Spaces and Feature Models: There and Back Again," in *Software Product Lines, 12th International Conference, SPLC* Limerick, Ireland, 2008, pp. 22-31.

[7] J. Barreiros and A. Moreira, "Configuration Support for Feature Models with Soft Constraints " in *ACM Symposium on Applied Computing (in press)* Coimbra, 2013.

[8] M. Mendonça, A. Wasowski, and K. Czarnecki, "SAT-based analysis of feature models is easy," in *Software Product Lines, 13th International Conference, SPLC 2009*, San Francisco, California, USA, 2009, pp. 231-240.

[9] J. White, B. Dougherty, and D. C. Schmidt, "Automated reasoning for multi-step feature model configuration problems," in *Software Product Line Conference 2009* San Francisco, USA, 2009.

[10] M. Mendonça, M. Branco, and D. Cowan, "S.P.L.O.T - Software Product Lines Online Tools," in *24th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages and Applications, OOPSLA 2009* Orlando, USA, 2009.

[11] H. Wada, J. Suzuki, and K. Oba, "A feature modeling support for non-functional constraints in service oriented architecture.," *IEEE Computer Society,* pp. 187-195, 2007.

[12] S. Robak and A. Pieczynski, "Employment of fuzzy logic in feature diagrams to model variability in software families.," in *10th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2003)* Huntsville, AL, USA, 2003, pp. 305-311.

[13] F. Rossi, P. V. Beek, and T. Walsh, "Handbook of Constraint Programming," in *Foundations of Artificial Intelligence*, J. Hendler, H. Kitano, and B. Nebel, Eds.: Elsevier, 2006.