Flexible Modeling and Product Derivation in Software Product Lines

Jorge Barreiros\textsuperscript{1,2}
\textsuperscript{1} Instituto Superior de Engenharia
Instituto Politécnico de Coimbra
Coimbra, Portugal
jmsousa@isec.pt

Ana Moreira \textsuperscript{2}
\textsuperscript{2} Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Caparica, Portugal
amm@fct.unl.pt

Abstract—Software Product Line development entails planned reuse of development assets for creating applications in a specific domain. SPL development can benefit from incorporating soft constraints in both Domain and Application Engineering. Increased expressiveness is attained and important domain knowledge that would otherwise be lost can be included, allowing improved configuration support to be provided. The stakeholders’ goals for a specific product can also be represented with soft constraints, allowing incomplete and inconsistent specifications to be inputted and then be automatically processed and analyzed. The approach is supported by a tool, which is capable of detecting inconsistencies, identifying the required trade-offs and explaining them to the stakeholder, who can then make an informed trade-off decision.

Keywords - Feature Models; Software Product Lines; Soft Constraints; Configuration Support; Feature Consistency; Feature Interactions; Domain Knowledge

I. INTRODUCTION

In Software Product Line (SPL) development, feature models annotated with domain constraints are often used to represent domain variability and commonalities \cite{1, 2}. Domain constraints specify stringent conditions that must be verified by all products within the scope of the SPL. Automated support for the derivation of products can use this information to assist the developer in the creation of viable configurations of products. Regrettably, there are situations where these constraints clash with a reality that is not all black and white and stubbornly, albeit reasonably and incontrovertibly, refuses to conform precisely to the exacting specifications of our models.

A direct consequence of this state of affairs is that developers must chose to either drop domain constraints representing useful domain knowledge for the sake of flexibility, or accept that the potential scope of the SPL must be crippled, lest under-specification of constraints allow unintended incursions into the realm of unfeasibility. The first solution strips relevant information from the model, impairing automation and support for product derivation: the developer bears the burden of creating meaningful and feasible configurations with reduced assistance. The second solution may be too conservative: by playing it safe, security is gained but flexibility is lost. Rather than agonizing over the choice of the lesser of two evils, we propose that the concept of soft constraints (SC) is embedded into SPLs to address these concerns. SCs represent useful information that is relevant to the configuration of products, but is not of forcing nature and can be ignored if convenient. They serve the primary and important purpose of allowing automated advice and suggestions to be provided during product derivation. Although the developer may choose to act against this advice, he is making an informed rather than blind choice (as would be the case had those constraints been completely ignored and no support provided because of it). Collateral but relevant benefits include the possibility of identifying required trade-offs of conflicting stakeholders goals.

Using these techniques improves domain modeling capabilities and product derivation support. In this paper we present a method and tools, based on the soft constraint model described in \cite{3}, for supporting the use of feature models annotated with SCs in SPL development. Goals of our approach include the following functionalities:

- Provide configuration suggestions based on constraints and feature tree structure, in addition to configuration propagation and completion services.
- Identify conflicted features, that is, features for which antagonic configuration advice is offered.
- Provide an explanation for conflicted features so that the user (stakeholder) can better understand the required trade-offs.

Hence, the main contributions include a detailed description of an inconsistency analysis algorithm and the development of the corresponding tool support for SC-assisted SPL configuration.

II. BACKGROUND

A. Software Product Lines

SPL development is an approach for developing software products within a specific domain of application \cite{1, 2} that promotes planned reuse of core assets (e.g., specifications, code, models). Two distinct roles can be identified in SPL development: the domain engineer and the application engineer. The domain engineer defines the scope of the product line by identifying the common and variable features of the products to be created. He also provides the composable assets that are used to create those products. The application engineer is responsible for the derivation of
specific products. These are created by configuring the variation points according to stakeholders’ specifications and composing the corresponding assets. Feedback is provided to the domain engineer so that evolution of the SPL scope and core assets is possible.

B. Feature Modeling

The domain engineer typically represents product commonality and variability using feature models. These identify valid product configurations, that is, configurations corresponding to a variant that can be created by the application engineer. 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). Feature models can be represented formally using logic expressions according to well-known transformations described in [4, 5]. A feature model is said to be consistent if at least one valid configuration exists.

To allow better modularization and management of complex products, a product configuration is often created by the application engineer in an incremental process called staged configuration, where features are progressively included or dropped from the product [6]. This allows automated feedback to be provided by interactive configuration tools that assist the application engineer in obtaining a valid configuration. Interactive configuration support for staged configuration automatically propagates user choices, by selecting or deselecting open features as required, according to the partial configuration inputted by the user and the feature model and its restrictions. This is achieved by computing the configuration domain after each selection made by the user, by identifying features that become dead (must be always deselected) and features that are common (must be always selected) in the current partial configuration. Techniques and tools to assist feature model configuration have been described in [5, 7].

C. Hard and Soft Constraints

Domain constraints can be categorized as hard or soft. While hard constraints are mandatory and must be satisfied, SCs are used to indicate optional but preferential or common configurations [3].

While all hard constraints must be satisfied in a consistent feature model, the same does not apply to soft constraints. This opens up the possibility of valid models containing conflicting soft constraints, that is, soft constraints that may be impossible to satisfy simultaneously. In such cases, a trade-off between conflicting constraints exists and must be resolved. It is worth pointing out that hard constraints cannot conflict by definition; otherwise the feature model would become inconsistent.

Soft constraints can also be prioritized. This allows a wider range of scenarios to be represented, by making it possible to establish relative degrees of importance. In this case, satisfaction of higher priority constraints takes precedence over those of lower priority.

III. CONFLICT ANALYSIS AND CONFIGURATION SUGGESTIONS

In this Section we describe the overall strategy for generating configuration suggestions and provide a detailed description of the configuration suggestion algorithm. In the SPL lifecycle, SCs originate in domain modeling or can be used to model user configuration goals. The overall strategy to automate conflict analysis is as follows:

1. Identify the set of active SCs with highest priority.

2. Temporarily consider the SCs identified in step 1 as hard constraints and verify consistency of the solution, by checking the satisfiability of the combined expression of feature model, partial configuration and the active constraint set.

   a. If the result is consistent, it is possible to find a solution that satisfies all active SCs. Proceed to generate configuration suggestions, by performing standard common and dead feature identification (but do not change the configuration). These are outputted as selection/deselection suggestions, correspondingly.

   b. If the result is inconsistent, then it is not possible to simultaneously satisfy all active SCs. The inconsistency must be analyzed to explicitly detect the conflicting constraints and how they conflict in terms of feature (de)selection. This algorithm is described in detail in the sequel.

3. The user selects or deselects a feature belonging to the domain of the active SCs.

4. Propagate the choice performed in the previous step by selecting all features that became common and deselecting all features that became dead.

The inconsistency analysis algorithm is detailed in Figure 1. The overall strategy is based on identifying a minimal set of conflicting SCs by local search branching out from a consistent base model conjoining the feature tree, hard constraints and partial configuration. Neighborhoods of this model, including all k-permutations of SCs, are progressively explored, in increasing k order, until an inconsistent model is found. Ultimately, the process is ensured to end when the entire set of active SCs is considered (as these have already been identified as being inconsistent). Conflicts are identified by finding common and dead features (i.e., suggestions for selection and deselection) for different k-permutations of SCs. If conflicting advice is found, a conflict is identified.

By considering the effect of k-permutations of SCs in increasing order of k, the algorithm will find the simplest explanations of the conflict (that is, explanations involving the smallest number of SCs). This is motivated for efficiency

---

1 We designate soft constraints as active if they have not been falsified or satisfied in the current partial configuration of a staged configuration process.
sake, as the majority of inconsistencies can be assumed to originate in the interactions between a reduced number of SCs. In scenarios where the assumption does not hold, efficiency is degraded but correct operation is still preserved.

### Algorithm 1: Analyze Inconsistency

```
Function analyzeInconsistency
   input : Partial configuration config, feature model fm and constraints
   done ← false
   k ← 1
   while ~ done do
      for each k-combination S of active soft constraints do
         dead ← computeDeadFeatures (fm - S, config)
         common ← computeCommonFeatures (fm - S, config)
         /* associate soft constraint satisfaction to required (de)selection of features. */
         if not empty dead then
            foreach feature f in dead do
               deselect [f]← true
               Add S to deselectionExplanation [f];
            end foreach
         end if
         if not empty common then
            foreach feature f in common do
               select [common]← true
               Add S to selectionExplanation [f];
            end foreach
         end if
         if done then
            k ← k + 1
         end if
      end for
   end while
   computeMetric (deletionExplanation, selectionExplanation)
```

Figure 1. Inconsistency Analysis Algorithm.

Our algorithm associates a metric to each feature, in the range -1..1, which can be interpreted as a deselection/selection recommendation. This function is generically dependent on feature trade-off information as described above. Any number of reasonable possibilities can be considered. Currently, we map the percentage of recommendations for selection vs. total number of recommendations into the -1..1 interval. Other methods could easily be adopted, based on factors such as an importance weight attributed to each SC.

### IV. ILLUSTRATIVE EXAMPLE AND TOOL DEMONSTRATION

We developed a tool that supports SPL configuration. Rather than devising a feature model specifically tailored for illustrating our approach and supporting tool, we decided to adapt and use models freely available by industrial and academic sources in the SPLOT feature model online repository [7]. For this, we selected the Web Portal SPL feature model, a medium-high complexity feature model, in terms of number of features and constraint density, originally described in [9]. This feature model describes a product line for creating web servers with differentiated capabilities such as support for alternative protocols, advertisement and search services, security features, etc.

#### A. Annotation of Feature Model with Soft Constraints

Usage of SCs in the Domain Engineering phase of SPL development entails annotating a feature model with SCs. This is achieved by transforming relevant domain knowledge into appropriate set of SCs. In our example, Figure 2 shows the SCs corresponding to domain knowledge such as:

- Although popups are possible, they are not well tolerated by users and are discouraged.
- If banners are used, a Flash based version is advisable.
- It is recommended that authentication should be conducted through secure https connections.

Other concerns are also represented as well.

#### B. Application Engineering

File StakeholderSpecification.xml in Figure 2 represents the stakeholder’s desired configuration represented with SCs, with the ms, data_storage, advanced, dynamic and https features selected and the nttp feature deselected. These SCs are composed with the annotated feature model. As seen in Figure 3, the tool provides configuration suggestions (+100% is a strong selection suggestion, -100% is a strong deselection suggestion) with conflicts being indicated by intermediate scores. In this case, a conflicted feature is https. The conflict stems from the impossibility of simultaneously satisfying constraints cfg00 and cfg05, since domain restrictions indicate that https support precludes ms level performance. In this case, the stakeholder must resolve a trade-off by deciding to drop either one of cfg00 or cfg05. Once this decision is made, configuration can proceed by following the tools suggestions: since no other inconsistencies are found, the configuration is successfully completed.

This example highlights how soft constraints have been used throughout the development process to successfully create a product, from an initial specification that is neither complete nor consistent. Automated support was able to detect the inconsistency in the specification and provided a clear indication of what the required trade-off is. Even though the stakeholder’s specification is not complete, a complete configuration with desirable properties was achieved.
SCs and no specific support for conflict resolution is described.

VI. CONCLUSIONS

This paper presented an approach for integrating the use of SCs in the SPL development process, with tool support being provided for relevant activities. This process allows richer representation of domain knowledge, allowing for improved configuration support in the form of automated configuration suggestions. Also, by representing stakeholder goals for a product as SCs during the application engineering phase, not only automated identification of conflicting goals is possible, but additional information can also be provided to the stakeholders, aiding them in making the required trade-off decision. Our approach and corresponding tool support have been demonstrated by using published feature models available from a public repository. Future work may include the development of systematic domain analysis techniques with SC support. Another promising field of application of SCs that warrants further study is the representation of quality attributes and their interplay in feature models.

REFERENCES