Towards Tool Support for the Configuration of Non-Functional Properties in SPLs

Julio Sincero, Wolfgang Schröder-Preikschat
Friedrich-Alexander University Erlangen-Nuremberg
{sincero,wosch}@cs.fau.de

Olaf Spinczyk
Dortmund University of Technology
olaf.spinczyk@tu-dortmund.de

Abstract—The configuration of NFPs (non-functional properties) is a crucial problem in the development of software-intensive systems. Most of the approaches currently available tackle this problem during software design. However, at this stage, NFPs cannot be properly predicted. As a solution for this problem we present the new extensions of the Feedback approach which aims at improving the configuration of NFPs in SPLs. We introduce our set of tools that are used to support the approach and show how to use them by applying it to the well-known SPL (The Graph Product Line) that was suggested as a platform for evaluating SPL technologies.

I. INTRODUCTION

A. Software Product Lines

The development of software product lines (SPLs) are considered [31] to be a new paradigm, as the result of its implementation, different members of a family of systems (product portfolio) can be generated. A SPL is, normally, comprised of a set of assets (software components, documentation, etc.) that can be assembled together to deliver products that satisfy the needs of a particular domain.

Several methodologies have been proposed in order to support the development of SPLs. Domain engineering [11] is the act of studying and documenting a specific domain in order to support the subsequent development phases (e.g. application engineering). Feature modeling is the approach used to capture the common, the variable and the interdependency’s present in a specific domain, the results are captured in a model (normally a represented in the form of a feature diagram) that determines which sets of features are valid product specifications. Hence, feature modeling is specially important during domain engineering and also during product configuration. Several feature implementation techniques have also been proposed in order to facilitate the actual implementation of the software assets of a SPL [27], [20], [15]. Moreover, techniques for scoping [19], testing [30], managing variability [17], are also available, business aspects like risk management [31], management and financial controls [29] are also addressed. The use of these, and, of course, other methodologies, in the context of SPL development, aim at reduced time to market, reduced costs, higher productivity, better product quality [31], etc.

B. Non-Functional Properties

We see non-functional properties (NFPs) as those that do not express what a piece of software is able to compute, but how, or in which circumstances it achieves its main goals (functional properties). Examples are: code size, memory footprint, performance, security, reliability, scalability, etc. Our experience developing families of operating systems show that these properties emerge from the complex interactions among the different modules that comprise the entire system [26], [25].

C. NFPs and SPLs

The idea to address the configuration of NFPs in the context of SPLs is, mainly, due to two reasons (1) NFPs are very much dependent on the components that comprise the product (2) SPLs are, normally, responsible for generating the components (choosing and configuring) that are required (through a product specification) to be present in the final product. We aim at taking advantage of the infrastructure that supports product generation in SPLs to improve the process of product configuration, specially, the configuration of NFPs.

D. Objective

The objective of this work is to present the extensions of the Feedback approach [36] which aims at improving the configuration of NFPs in SPLs. We introduce the new set of tools that are used to support the approach, and we show how to use them by applying it to a well-known [28] SPL that was suggested as a platform for evaluating SPL technologies. Finally, we evaluate the approach and discuss about applicability.

E. Outline

The remainder of this document is organized as follows. In the next section we start with the motivation by describing advantages of addressing the configuration of NFPs in SPLs. Subsequently, we present the related work. Section IV presents the basic concepts of the Feedback approach and introduces the new set of tools that are used to implement it. A case study is presented in Section VI. The final section summarizes and concludes the work.

II. MOTIVATION

Researchers have proposed many [9], [37], [2] approaches that integrate information regarding NFPs into the software development process. These techniques range from extensions of UML [9] diagrams with non-functional information up to
the design of software architectures that take into consideration the desired non-functional behavior.

However, these techniques take into account only information gathered during requirements engineering. Although we believe these approaches improve considerably the conformity of the resulting software with the non-functional requirements, we think that a significant amount information that can improve the configuration of NFPs is simply disregarded. Additionally, as mentioned above, we believe that NFPs are heavily influenced by the composition and implementation of the different software components that make up the final software product, and it cannot be precisely predicted during software design.

Therefore, our idea is to use the information captured both at product generation (e.g., compile-time, etc.) and at runtime to use it in the configuration process to enable the explicit configuration of NFPs.

Our vision is that during product configuration (feature selection) the application engineer should be informed about the impact of each feature on the NFPs of the final product.

III. RELATED WORK

Our work is comprised of pieces from three major fields. First, the configuration of NFP in SPLs. Second, tool support for the SPL development. Third, reasoning in feature models. The following sections present prominent work in these areas.

A. NFPs and SPLs

Norbert et. al. [34], [32] proposes the use of a semi-automated derivation (SAD) to assist developers selecting product features in SPLs with a large number of features. The basic idea is to hide variation points that are irrelevant due to non-functional requirements that should be met. They claim that traditional approaches do not consider non-functional properties, nor alternatives for a feature implementation. To solve this problem, they present a integrated software product line model (ISPLM) which integrates code units and their non-functional properties into the feature model.

Benavides et. al. [4] propose an extension to feature models (as proposed by Czarnecki [10]) to accommodate information about extra-functional features (NFPs, in our view). In this approach, attributes like price or development time can be assigned to features. The features and their attributes are transformed to a constraint satisfaction problem (CSP) so that an automated reasoning can be applied to it. Hence, Optimum products can be generated according to a determined criterion (the extra-functional features).

These methods take advantage of information about NFPs to improve product configuration. However, we think that both are more related to the field of variability management [33], [24]. They ease the configuration process in large SPLs but do not offer the ability to explicit configure NFPs or to inform the user about the real influence of a feature on the required NFP.

B. Tool Support

Pure::variants [7] is a tool (Eclipse plugin) that supports variant management of SPLs. It is independent of programming languages and it enables the definition of the problem domain by means of feature models, and the solution domain by family models. It also automates the process of product generation. Regarding the configuration of NFPs, the current version is able to assign bugs (from a bug-tracking system) to specific features, therefore, during product configuration, the application engineer is informed about the known bugs that will be present on the final product.

GEARS [23] is a tool and framework that enables the development and evolution of SPLs, it applies the 3-tiered SPL methodology [22]. In GEARS, SPLs are comprised of three elements, Software assets (source code, documentation, etc.), product feature profile (to model each product in the portfolio), and the gears configurator which automatically assembles products based on its specification.

FeaturePlugin [1] is an open-source Eclipse plugin for designing and configuring feature models. It supports the concepts of staged configuration [13] and feature cardinalities [12]. The tool focus on providing advanced techniques of feature modeling and not in supporting the whole process of SPL development.

FAMA (FeAture Model Analyser) [6] is an extensible framework for the automated analysis of feature models. It is able to denote feature models in several logic representations, therefore, different solvers can used in the analysis process. Currently, it supports CSP, SAT (boolean satisfiability problem) and BDD (binary decision diagrams), but it is flexible enough to have other solvers added to it. Cardinality-based feature models are allowed and the following operation are supported: finding out if a feature model is valid (it exists a valid selection that satisfies all constraints), finding the total number of valid products, list all valid configurations and calculating the commonality of features (the number of valid products it appears).

There are commercial, free and open-source alternatives for the design of feature models, only pure-variants is able to provide non-functional information during product configuration, at the moment very basic, though.

C. Reasoning in Feature Models

Important to our work is also the process of reasoning in feature models. A seminal work of Benavides et. al. [5], presents the mapping from feature model components (e.g. optional features, mandatory feature, and group features) to diverse logical representations, namely, SAT, BDD and CSP. Transforming feature models in these representations enables the use of off-the-shelf solvers that can perform several analysis very relevant for feature models (e.g. validity, number of solutions, etc.).

The relation of feature models and grammars have also been studied [3]. Batory, also motivated by the ability to use off-the-shelf satisfiability solvers, presented the mapping from feature models, firstly, to iterative tree grammars, and then
to propositional formulas. However, his main goal was to simplify the laborious task of debugging feature models.

Recently, the relation of feature models and logic representation have been further explored. Czarnecki et. al. [14] proposes a method for the inverse transformation, from propositional formulas to feature model representation. Janota et. al. [18] studies the representation of feature models in higher-order logic, a formalized meta-model is presented, however, no tool support is provided.

IV. THE Feedback APPROACH

The Feedback approach extends the traditional SPL development techniques in order to provide information regarding NFPs during product configuration. We introduced new structures and mechanisms so that the SPL infrastructure can be used to generate products that will be tested against the desired NFP. This information is saved, organized, and reinserted in the SPL, it enables the user to benefit from it on the configuration of further products.

This process is organized in three layers:

**SPL Repository** is comprised of the software components that can be assembled together to generate products. Additionally, components that are used merely to capture non-functional information from generated products are also available. We have shown [16] that aspects are very adequate for this task.

**User Configuration** is responsible for providing the mechanisms for product configuration. Besides the traditional configuration process by selecting features from a feature model, we provide the user with non-functional information. As NFPs are very specific to each product, or even features, this information can be displayed in different ways, for example, sliders, graphs, charts, etc. Moreover, during configuration the user can select the aforementioned components that are responsible for checking the NFPs of the product.

**Concrete Solution Domain** encompasses the generated product and the compile/runtime environment used to generate and test the product.

The organization of the SPL in this fashion aims at improving the configuration experience in the following ways:

(i) providing exact information about products that have been previously configured. (ii) using heuristics and regression techniques to provide approximated information about products that have been only partially configured. (iii) if more then one software component implement the same functional behavior, the user should be able to select the most appropriate by means of its non-functional characteristics.

V. TOOL-CHAIN ARCHITECTURE

This paper introduces our new set of tools that support the implementation of the Feedback approach. Basically, it consists of a feature modeling tool, a logic solver and a set of scripts. Figure 1 depicts the whole process and the tools involved, details are explained in the following sections.

A. Turning the LKC into a Feature Modeling Tool

Our approach is centered on feature modeling. However, our initial goal was not to develop a feature modeling tool from scratch, but to integrate the Feedback approach into an existing tool. The only feature modeling tool that, to the best of our knowledge, provides freely its source code is the FeaturePlugin [1]. Nevertheless, it requires the whole Eclipse project, as we did not want to force the use of Eclipse to deploy our approach, we had to turn this option down. Later, our work analyzing the Linux Kernel [35] as a SPL has shown that its configuration tool could be turned into a feature modeling tool.

The Linux Kernel Configurator (LKC) is a tool that is delivered within the Linux Kernel in order to enable its configuration (feature selection). Its first prototype was proposed in 2002, the current version is 1.3, and as the Linux Kernel, it is released under the GNU General Public License (GPL).

Table I presents the mappings that we propose for the feature models using the LKC language constructs.

Most of the mappings were relatively easy to perform. For the mandatory relation, the parent feature forces the selection of the child by the use of a reverse dependency (select). The optional relation is described by using a dependency between the child and the parent feature (depends on). The or group is designed by creating reverse dependencies between the children and the parent, this was done inside a menu definition in order to group the children together. The alternative group can be described by including configuration options (the children) inside a choice definition, which has the same semantic as of alternative group in feature models. Extra constrains like implies and excludes are defined using normal and negated dependencies (requires).

Using these mappings we were able to design several feature models. So far we did not find any feature model construction that could not be modeled with the LKC language. Hence, it was chosen as the feature configuration tool for the implementation of the Feedback approach.

B. Partial Configuration

As mentioned earlier, the Feedback approach enables the user to generate products, run tests on it and save real information about NFPs to improve the configuration of further products.

In our view, the collection of non-functional information can be performed in two ways. First, generating single products and performing tests on it (left side of Figure 1). Second, generating sets of products to be tested on the same scenario (right side of Figure 1).

To generate sets of product specifications we use the concept of partial configuration, which is a feature selection where features can be selected (the feature must be present) or blocked (must not be present). Using the partial configuration we generate the set of valid product specifications that respect the selected and blocked features. This process is shown on the right side of Figure 1. Using the feature modeling tool, the user performs a partial configuration, which is the input of our
generator (the feature model is translated to a binary decision diagram, using the translation as proposed by Benavides [5], and implemented using the BuDDy library [8]). The generator output is a set of specifications that are used to generate the corresponding set of products.

C. Running Tests

In our tool set, product specifications are simple text files with pre-processor directives (defines) to control the features that should be enabled or not. Using the specification generated as explained in the previous section, we are able to generate sets of products and run them for the evaluation of the required NFP.

As shown on the bottom part of Figure 1, the result of the product analysis can be associated to a feature (specific feature present in the model), to a spec (complete product configuration), or to a unit (implementation unit, like classes or aspects).

VI. CASE STUDY

In order to evaluate our approach we conducted one case study. We have chosen the Graph Product Line (GPL) [28] for several reasons, it was proposed as a platform for evaluating SPLs, the domain of graphs is well understood and the algorithms are well known, and finally, recent work [21] has used it for evaluating new techniques for implementing products lines.

A. The GPL Product Line

The GPL implements a family of classical graph applications. Its feature model is depicted in Figure 3, as it can be seen, basically, its variation points are the graph type (directed or undirected), the search algorithm (DFS or BFS) \(^1\), and edge type (weighted and unweighted). Additionally, a set of algorithms to handle graphs are also available:

- **Number (Number)**: assigns a unique number to each vertex a the result of a traversal.
- **Connected Components (Con.Comp.)**: computes the connected components of an undirected graph.
- **Strongly Connected Components (SCC)**: computes the strongly connected components of a directed graph.
- **Cycle Checking (CycleCheck)**: checks the existence of cycles in a graph.
- **Minimum Spanning Tree (MSTP. and MSTK)**: computes

\[^1\text{DFS=depth-first search, BFS=breadth-first search}\]
the minimum spanning tree, which is a spanning tree with weight less then or equal to the weight of every other spanning tree. Two implementations of this feature are available.

**Single-Source Shortest Path (SSP):** computes the shortest path from a source to all other vertices.

Table III shows the extra constraints that cannot be shown on the feature model, and, therefore, are implemented as an implies relation (shown on Table I). Each line shows which features can be associated to each algorithm. It may seem to be a very simple feature model, however, even with only 19 features and a couple of extra constraints the idea to provide non-functional information about every of its possible valid feature combinations, is a very challenging task.

### B. Targeted NFPs

So far in this document we have discussed about NFP in a very general way. This was due to the following reason: NFPs are totally dependent on scenarios and standpoints. For some stakeholders, the definition of performance or security may have completely different means than for other stakeholders. Therefore, we think that NFPs should be specifically defined. For example, the question “what is the performance of productX?”, is way too generic, and, in our opinion, cannot be satisfactorily answered. Nevertheless, a question like “what is the time required to perform the number algorithm in a graph with 300 vertices and 45k edges?”, is a very appropriate question, which shows a relevant way to represent one of the several ways that the NFP performance could be seen in this scenario. The Feedback aims at providing the visualization/configuration of NFPs defined in this manner during product configuration.

In order to test this concept, and the Feedback approach itself, we have defined the follow NFP to be tested: the performance of the number algorithm for three different workloads. That is, the time required to perform the algorithm in randomly generated graphs of the sizes: 300 vertices and 125k edges, 500 vertices and 45k edges, and, 1000 vertices and 500k edges. To be able to provide the information resulting from these tests during product configuration, we generated a set of variants of the GPL using the process described in Section V-B. We have set the feature number as selected and all other features under algorithm as blocked. Our generator output 8 different product configurations that were tested using the three different workloads. The results are shown on Table II (time in milliseconds). Our feature modeling tool shown in Figure 2 displays this information during feature selection. As it can be seen on the bottom of Figure 2, we are using rounded range control components (one for each workload) to display the performance (as defined above) of the current selection.

This process enables us to display exact information about NFPs that were specifically defined and tested. We believe that this information helps the user on the process of deciding which features should be included or not, or, at least, informing

---

The table and comments are not directly transcribed due to their complexity and the need for understanding the context to interpret them accurately. However, the primary focus is on the minimum spanning tree, its properties, and the implications of different feature models on non-functional properties (NFPs).
about the non-functional behavior that should be expected in the final product.

VII. CONCLUSIONS AND FUTURE WORK

There are many ways to address the configuration of NFPs of a system. Most of the approaches annotate models during software design with non-functional information/requirements that are of interest. However, the real behavior of a piece of software can be verified only at runtime and when deployed in real scenarios.

To tackle this problem we devised the Feedback approach. We developed mechanisms to generate sets of products so that desired NFPs can be tested in a real scenario. Additionally, this information can be saved for further reuse. The application engineer using our configuration tool is able to see what is the impact of feature selection on NFPs that were previously tested.

In our case study we have shown the feasibility of the idea, and also, that with appropriate tool support it is relatively easy to implement.

In order to extend and further evaluate our work, we plan to improve the integration of our tools and to carry out a case study in a large scale SPL with thousands of features. The analysis of other types of NFPs (security, latency, etc.) in operating systems are also under work. Moreover, we are also investigating the inverse way to configure a product, enabling the user to set constraints regarding the tested NFPs, and the configuration tool will generate the sets of products that are able to fulfill the requirements. Regarding selections that were not tested, we are applying regression techniques to predict its behavior.

REFERENCES


<table>
<thead>
<tr>
<th>Conf.</th>
<th>Workload1</th>
<th>Workload2</th>
<th>Workload3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>110</td>
<td>378</td>
<td>2868</td>
</tr>
<tr>
<td>2</td>
<td>111</td>
<td>380</td>
<td>2876</td>
</tr>
<tr>
<td>3</td>
<td>153</td>
<td>501</td>
<td>3298</td>
</tr>
<tr>
<td>4</td>
<td>161</td>
<td>493</td>
<td>3361</td>
</tr>
<tr>
<td>5</td>
<td>109</td>
<td>385</td>
<td>2848</td>
</tr>
<tr>
<td>6</td>
<td>111</td>
<td>377</td>
<td>2856</td>
</tr>
<tr>
<td>7</td>
<td>157</td>
<td>496</td>
<td>3340</td>
</tr>
<tr>
<td>8</td>
<td>153</td>
<td>513</td>
<td>3358</td>
</tr>
</tbody>
</table>
Table III

<table>
<thead>
<tr>
<th>Search</th>
<th>Graph Type</th>
<th>Edge Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>None</td>
<td>BPS</td>
</tr>
<tr>
<td>NUMBER</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CC</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SCC</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CYCLE</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>MST</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SSP</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>


