Journal of Logical and Algebraic Methods in Programming 85 (2016) 173–199

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Resource–usage–aware configuration in software product lines

Damiano Zanardini [a,*], Elvira Albert [b], Karina Villela [c]

[a] Technical University of Madrid, Spain
[b] Complutense University of Madrid, Spain
[c] Fraunhofer IESE Kaiserslautern, Germany

ARTICLE INFO

Article history:
Received 2 August 2014
Received in revised form 16 July 2015
Accepted 26 August 2015
Available online 6 September 2015

ABSTRACT

Deriving concrete products from a product-line infrastructure requires resolving the variability captured in the product line, based on the company market strategy or requirements from specific customers. Selecting the most appropriate set of features for a product is a complex task, especially if quality requirements have to be considered. *Resource–usage–aware configuration* aims at providing awareness of resource–usage properties of artifacts throughout the configuration process. This article envisages several strategies for resource–usage–aware configuration which feature different performance and efficiency trade-offs. The common idea in all strategies is the use of resource–usage estimates obtained by an off-the-shelf *static resource–usage analyzer* as a heuristic for choosing among different candidate configurations. We report on a prototype implementation of the most practical strategies for resource–usage–aware configuration and apply it on an industrial case study.

1. Introduction

One increasing trend in the market of Software Engineering is the need to develop multiple, similar software products instead of just a single individual product. *Software Product-Line Engineering* (SPLE) [41] offers a solution to this trend based on explicitly modeling what is common and what differs among product variants, and on building a reuse infrastructure, a so-called *product-line infrastructure*, that can be instantiated and possibly extended to build the desired similar software artifacts (the products).

Deriving concrete products from a product-line infrastructure requires resolving the variability captured in the product line according to a company's market strategy or the requirements from specific customers. *Feature models* [35,21] have been the main approach for capturing the commonality and variability in product lines. The process of *product configuration* usually consists in selecting those features that are applicable to the desired product, so that this product can be assembled from the product-line assets. One of the most difficult tasks is the translation of market or customer requirements and goals into the concrete set of features that best match them. Several aspects affect feature selection for a certain product: dependencies and constraints among features, the desired degree of *product quality*, and economic cost. Moreover, different stakeholders are capable of selecting external (visible to the customers and/or marketing people) and internal features (necessary to realize external features, but not visible). In product lines with a large number of features, which are very common in practice, feature selection becomes an increasingly difficult task, and may result in *invalid*, *inappropriate* or *inefficient* configurations.

* Corresponding author.
E-mail address: damiano@fi.upm.es (D. Zanardini).

http://dx.doi.org/10.1016/j.jlamp.2015.08.003
2352-2208/

Table 1
Support for feature selection.

Main characteristic	Support type	NF concerns	Underlying technology
Multi-level staged [22]	Interactive	Security	Specialized FMs
Probabilistic [23]	Interactive	No	Conditional probabilities and legal Joint Probability Distributions
Dynamic [39]	Automatic	No	Binding analysis and reconfiguration strategy
Multi-step [57,58]	Automatic	Cost	Constraint Satisfaction Problem
Polynomial-time [56]	Automatic	Yes	Multi-dimensional Multi-choice Knapsack Problem
Fast selection time [28]	Automatic	Yes	Genetic algorithm (repair operator and penalty function)
Business concern annotation [51]	Automatic	Yes	Hierarchical Task Network
Multi-view [31,1,32]	Interactive	No	Workflow management tool
Feature-wise and variant-wise properties [47]	Mostly automatic	Yes	Constraint Satisfaction Problem
Domain experts' judgment [59]	Interactive	Yes	Analytic Hierarchical Process

Several authors have contributed to the research on feature selection (Table 1). We have analyzed the proposed approaches in terms of the type of support (either interactive or automatic), the non-functional concerns that are taken into consideration, and the underlying problem-solving technology.

Concerning *Support Type*, interactive product configuration uses the rules provided by the feature model to propagate configuration choices made by the user [23], whereas automatic product configuration provides a set of configurations that satisfy the rules and the user's requirements and constraints. The selection of features in our resource–usage–aware configurator is mainly automatized. However, the user has a central role providing not only information on concerns (e.g., memory consumption) and constraints (e.g., that the cost has to be lower than x), but also on the key features of the product. *Key features* are those features which are required by the customer as a crucial part of the desired products, similarly to user-selected features included in the input partial configurations of the tool implemented by Sincero et al. [48]. If their presence does not infringe any rule, then the configurator will not propose deselecting them in any of the provided solutions. On the one hand, this information is essential for the efficiency and effectiveness of a product configurator. On the other hand, it provides an interesting balance between automatic and interactive configurations. Tun et al. [53] proposed an approach to systematically relate requirements to features that uses three separate feature models (requirements, world context and specifications) and respective links between them. Our approach addresses this issue by asking the user for key features and quality concerns (requirements and world context), and proposing configurations (specifications) that include such key features and optimize the quality concerns.

As regards *Non-Functional Concerns*, several approaches take into consideration cost constraints, but only few of them consider quality concerns [22,56,51,28,47] as we do. There are two crucial aspects in this context: (1) quality-aware configurations require modeling quality variability; and (2) it is necessary to provide support or guidance on how to obtain quality indicators. Etxeberria et al. [24] presented a survey on existing approaches for specifying variability in quality attributes. The six approaches (Goal-based model [26], F-SIG [33], COVAMOF [14], Extended Feature Model [12], Definition Hierarchy [38], and Bayesian Belief Network [60]) are compared according to the requirements defined by the authors for a quality-variability modeling approach. Our resource–usage–aware configurator adopts the Extended Feature Model approach, because this approach does not require the learning of additional/new notations by practitioners, which will promote the adoption of our approach in practice.

Regarding the third aspect used to compare the approaches in Table 1, namely, the *Underlying Technology*, our case study was built upon the CSP (Constraint-Satisfaction-Problem) solver called *Choco Java*, because: (1) the mapping of the product-configuration problem into CSP [58] is intuitive; and (2) there are translators from CSP into *Satisfiability Modulo Theories* (SMT), which can be adopted to address quality issues of the underlying technology, if required. However, any other underlying technology capable of dealing with quality annotation of features (e.g., the one used by Soltani et al. [51]) could have been used, which includes visualization and exploration techniques such as the ones proposed in [40].

This paper focuses on obtaining quality indicators of performance for features and/or product configurations that can be used to guide product configuration. *Performance* (a.k.a. *resource consumption* or *resource usage*) is a frequently desired quality for software artifacts. In our implementation and case study, the quality metrics we use to estimate the degree of performance of a product are either the amount of allocated memory (*memory consumption*) or the number of *executed instructions*. It is important to point out from the beginning that, unlike related work, the presented techniques rely on *static* resource–usage analysis, i.e., quality indicators are obtained without actually executing the code and refer to all possible inputs (not just to a few specific workloads, as in existing approaches).

We discuss and compare four strategies for *resource–usage–aware configuration* of software product lines. Many ideas behind such strategies are well-known; one of them is actually infeasible and is only presented in order to start the discussion. However, all these strategies are applied with static analysis in mind, which is something not discussed in existing works. The common idea in all strategies is the use of resource–usage estimates as a *heuristic* for guiding the automatic selection of features. The crux is the use of an automated static resource–usage analyzer (e.g., [27,30,6]) providing estimates