The Journal of Systems and Software 106 (2015) 42–58

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Safe evolution templates for software product lines

L. Neves [a], P. Borba [a], V. Alves [b], L. Turnes [b], L. Teixeira [a,*], D. Sena [c], U. Kulesza [c]

[a] Informatics Center, Federal University of Pernambuco, Av. Jornalista Anibal Fernandes, s/n 50740–560 Recife, PE, Brazil
[b] Computer Science Department, University of Brasilia, Brazil
[c] Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal, Brazil

ARTICLE INFO

Article history:
Received 11 February 2014
Revised 3 March 2015
Accepted 3 April 2015
Available online 16 April 2015

Keywords:
Software product lines
Refinement
Evolution

ABSTRACT

Software product lines enable generating related software products from reusable assets. Adopting a product line strategy can bring significant quality and productivity improvements. However, evolving a product line can be risky, since it might impact many products. When introducing new features or improving its design, it is important to make sure that the behavior of existing products is not affected. To ensure that, one usually has to analyze different types of artifacts, an activity that can lead to errors. To address this issue, in this work we discover and analyze concrete evolution scenarios from five different product lines. We discover a total of 13 safe evolution templates, which are generic transformations that developers can apply when evolving compositional and annotative product lines, with the goal of preserving the behavior of existing products. We also evaluate the templates by analyzing the evolution history of these product lines. In this evaluation, we observe that the templates can address the modifications that developers performed in the analyzed scenarios, which corroborates the expressiveness of our template set. We also observe that the templates could also have helped to avoid the errors that we identified during our analysis.

1. Introduction

A software product line is a set of related software products that are systematically generated from reusable assets. Products are related in the sense that they share functionality or behavior. Assets correspond to artifacts such as classes and property files, which we compose or instantiate in different ways to specify or build products. This kind of reuse targeted at a specific set of products can bring productivity and time-to-market improvements (van der Linden et al., 2007; Pohl et al., 2005). To obtain these benefits with reduced upfront investment and risks, we can minimize the initial upfront cost of software product line adoption by extracting a software product line from existing products (Clements & Northrop, 2001; Krueger, 2002). Similar processes apply to evolving a software product line, both when just improving the product line design and when adding new functionality and products, which often requires extracting variations from parts previously shared by a set of products.

The activity of manually extracting different software product line assets when evolving it requires substantial effort, especially for checking necessary conditions to make sure the extraction is correctly performed. In fact, the lack of specific guidelines and development tools to support software product line evolution makes this process error-prone. Extractions might lead to unintended modifications to the behavior of existing products, affecting product line users and compromising the promised benefits in other dimensions of costs and risks. The associated defects are more difficult to track because they are only present in specific products. Generating and testing these different products may help to discover and correct the mentioned issues. However, as the number of products can be high, testing all of them can be expensive and impact productivity.

To avoid these problems and evolve a software product line in a *safe* way (in the general sense that behavior is preserved, not specifically referring to conventional safety properties), we could resort to a formal notion of software product line refinement or safe evolution (Borba, 2009; Borba et al., 2012). By basically requiring preservation of the observable behavior of existing products, this notion guarantees that changes to a software product line do not impact its existing users. So users of the products that could be generated before the changes can use the new modified products without noticing any difference. This notion applies when we need to introduce new products to the software product line without changing existing ones, or when we want to improve the product line design without modifying the behavior of existing products. To support such change scenarios, safe evolution considers that software product line specific artifacts, like feature models (Czarnecki & Eisenecker, 2000; Kang et al., 1990) and configuration knowledge (Czarnecki & Eisenecker, 2000), often

* Corresponding author.
E-mail addresses: lmn3@cin.ufpe.br (L. Neves), phmb@cin.ufpe.br (P. Borba), valves@unb.br (V. Alves), lmt@cin.ufpe.br (L. Teixeira), demostenes.sena@ifrn.edu.br (D. Sena), uira@dimap.ufrn.br (U. Kulesza).

http://dx.doi.org/10.1016/j.jss.2015.04.024
0164-1212/

coevolve with assets (Neves et al., 2011; Passos et al., 2013; Seidl et al., 2012).

With the goal of better understanding the process involved in software product line safe evolution, in this work we describe empirical studies that lead to the discovery and analysis of 67 concrete safe evolution scenarios from five different software product lines. Each scenario is characterized by a commit and one of its subsequent commits in the evolution history of the product lines repositories.

Based on the evolution history from two of the aforementioned software product lines, we identify and precisely describe a number of safe evolution templates that abstract, generalize, and factorize the analyzed scenarios, and also conform to the refinement notion (Borba et al., 2012) we rely on. These templates are generic transformations that developers can safely apply when maintaining compositional and annotative software product lines. They specify transformations that go beyond program refactoring notions (Fowler, 1999; Roberts, 1999), which deal with simple programs, by considering both sets of reusable assets that do not necessarily correspond to valid programs, and extra software product line artifacts such as feature models and configuration knowledge. For each template, we describe its structure and the necessary conditions for proper application. We also show examples of correct application of the templates in evolution scenarios mined from existing software product lines. This way we hope to provide extra, concise and explicit guidance to evolve a software product line in a safe way. The templates can also be used as a basis to automate support for safe software product line evolution.

We evaluate the proposed templates by analyzing the evolution history of the five aforementioned software product lines. In this evaluation, we could observe that the proposed templates can address the modifications that developers performed in the analyzed scenarios, which corroborate the expressiveness of our template set. As a secondary result, we observe that the templates could be used to avoid some defects introduced during the evolution history of some product lines. Such defects were caused by modifications that were supposed to be safe, but actually changed the behavior of existing products.

In summary, with the aim of discovering more safe evolution templates and assessing whether they could be useful to justify existing evolution scenarios, this article extends our previous conference paper (Neves et al., 2011) in two main ways:

- study of annotative software product lines: we go beyond our previous study on compositional software product lines by analyzing and presenting five additional templates (Section 3.2.2) that deal with an extended configuration knowledge notion—mapping feature expressions to transformations involving assets—necessary to address evolution scenarios that involve preprocessor-based variability management in annotative software product lines (Kästner et al., 2008);
- further evaluation: we bring additional evidence of the expressiveness of the proposed templates, evaluating the evolution history of three additional software product lines, namely, a product line of research group management systems, a product line of product line derivation tools, and a product line of academic information systems.

We organize the rest of the text as follows. Section 2 introduces the main concepts used in this work, such as feature models, configuration knowledge, asset mappings, and the product line refinement notion. Section 3 presents the safe evolution templates for software product lines. It also shows some examples of the templates being applied in different software product lines. Section 4 presents the results of a study performed to evaluate the expressiveness of our template set. We discuss related work in Section 5 and conclude with Section 6.

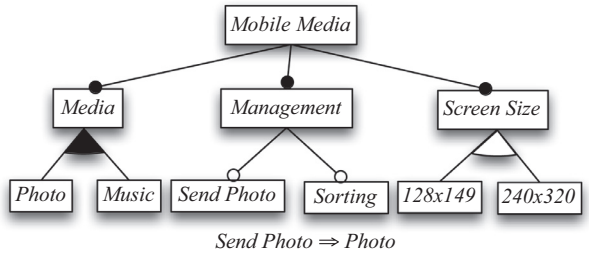

Fig. 1. MobileMedia FM example.

2. Software product line concepts

To enable the automatic generation of products from assets, software product lines, hereafter product lines, often rely on artifacts such as Feature Models (FM), Configuration Knowledge (CK) (Czarnecki & Eisenecker, 2000), and assets, which we briefly describe in what follows. To guide the product line evolution analysis and identify the evolution scenarios, we rely on a product line refinement notion (Borba et al., 2012), which formalizes our intuition about safe evolution, drives the analysis of the evolution scenarios and justifies the product line transformation templates we propose in this article. Essentially, we say that a product line L' refines a product line L whenever L' is able to generate products that behaviorally match L products. This way, users of a product from L cannot observe behavioral differences when using the corresponding product of L'. This is exactly what guarantees safety when improving a product line design by changing its FM, CK or assets.

2.1. Feature models

A FM is usually represented as a tree, containing features and information about how they relate to each other. Features have different names and abstract groups of associated requirements, both functional and non-functional. In this work, we use the notation by Czarnecki and Eisenecker (2000) to express relationships between a parent feature and its child features.

Besides these relationships, the notation we consider may also contain propositional logic constraints over features. We use feature names as atoms to indicate feature selection. So, negation indicates that a feature should not be selected. For instance, the formula below the tree in Fig. 1 states that feature *Photo* must be present in every product that has feature *SendPhoto*. So {*Photo, SendPhoto*, 240×320}, together with the mandatory features, which hereafter we omit for brevity, is a valid feature selection (product configuration), but {*Music, SendPhoto*, 240×320} is not. A product configuration is a valid feature selection that satisfies all FM constraints, specified both graphically and through formulae. Each product configuration corresponds to a product from the product line, expressed in terms of the features it supports. This captures the intuition that the FM denotes the set of products in a product line (Schobbens et al., 2007).

2.2. Assets

In a product line, we specify and implement features with reusable assets. So, we must consider different languages for specifying and implementing assets such as requirements documents, design models, code, tests, data files, and so on. For simplicity, in the text we focus on code assets for the examples and concepts, as they are equivalent to the other kinds of assets with respect to our interests on product line refinement. This way, we can focus on the essential concepts these languages should support. The important issue here is not the