M. Tichy, E. Bodden, M. Kuhrmann, S. Wagner, J.-P. Steghöfer (Hrsg.): SE 2018,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 155

Projectional Editing of Software Product Lines (Extended Abstract)

Benjamin Behringer[1], Jochen Palz[2], Thorsten Berger[3]

Abstract: A software product line is a portfolio of software variants. To implement its common and variable features, a variety of techniques emerged, representing the feature implementation either as annotated code or as dedicated feature modules. While each technique provides distinct advantages, developers need to choose one feature representation and then adhere to it when the product line evolves. In this extended abstract, we describe PEoPL, an approach [BPB17] that combines the advantages of different feature representations. Developers can engineer features using the most-suited representation, switch it seamlessly, and even use different representations (editable views) in parallel.

Keywords: projectional editing; software product lines; variability mechanisms

A software product line (SPL) is a portfolio of system variants engineered in an application domain. It implements common and variable features using implementation techniques called *variability mechanisms*. Many such mechanisms have emerged, typically classified into annotative (e.g., preprocessors) and modular (e.g., feature modules) mechanisms. Each represents a feature's artifacts differently, with distinct advantages and disadvantages. For instance, annotative mechanisms are easy to apply, but annotations clutter source code, obscure control flows, and force developers to work on all variants in parallel. They also lead to scattered and tangled feature implementations, challenging program comprehension, maintenance, and evolution. In contrast, feature modules ease the latter by realizing features modularly in cohesive units. Unfortunately, creating feature modules imposes substantial engineering overhead, while their interaction with other features is more difficult to comprehend, both hindering their adoption in practice.

Although these representations are complementary, existing SPL engineering approaches typically focus on one representation. Most importantly, developers need to choose one representation per feature and to adhere to it for maintenance and evolution. While refactorings were proposed for migrating between annotative and modular representations, they do not allow to quickly switch the representation during evolution and maintenance. Ideally, developers could always select the best-suited representation.

[1] htw saar, Germany, benjamin.behringer@htwsaar.de
[2] htw saar, Germany, jochen.palz@htwsaar.de
[3] Chalmers | University of Gothenburg, Sweden, thorsten.berger@chalmers.se

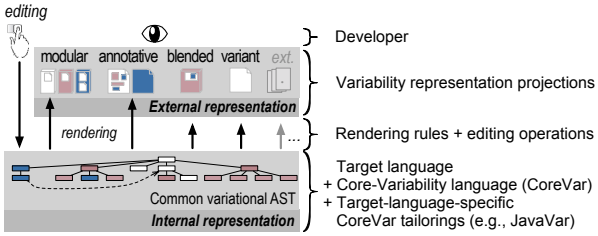

Abb. 1: PEoPL separates internal and external variability representations

In our main publication [BPB17], we present PEoPL, an approach that combines the benefits of annotative and modular variability mechanisms. PEoPL allows developers to quickly switch representations and even use different representations side-by-side. The core idea of PEoPL is to establish an internal representation of the SPL and separate it from the external representations that developers use. Fig. 1 illustrates the approach. Internally, PEoPL persists an abstract syntax tree (AST) adhering to a programming language composed with our languages CoreVar (to internally represent feature artifacts) and a programming-language-specific tailoring of CoreVar (to realize well-formedness constraints). We currently provide such tailorings for Java, C, and fault trees. Externally, the AST is rendered into editable projections used by developers. According to the projectional-editing paradigm, the editing gestures of developers directly change the AST, without any involvement of parsing [Be16]. The AST is rendered into concrete syntax using projections. We currently provide projections showing artifacts as *textual annotations* (e.g., #ifdef), *visual annotations* (colored bars), *feature modules*, *annotations blended into modules*, *fade-in modules* (i.e., show external code within a module), *individual variants* (i.e., hide non-selected features), and *reused code snippets* (e.g., to support cross-cutting aspects) [Be17]. We realize PEoPL's concepts in a full-fledged IDE based on JetBrains Meta Programming System (MPS).

We evaluate PEoPL by adopting and implementing eight Java-based SPLs, such as the Berkeley DB (70kLOC, 42 features, 218 classes). Our evaluation shows the feasibility of the approach, especially that internal and external feature representations can be separated, and that it scales. A preliminary user study confirms the benefits of our external representations and of seamlessly switching them. Controlled experiments are subject to our future work.

Literaturverzeichnis

[Be16] Berger, Thorsten; Völter, Markus; Jensen, Hans Peter; Dangprasert, Taweesap; Siegmund, Janet: Efficiency of Projectional Editing: A Controlled Experiment. In: 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE). 2016.

[Be17] Behringer, Benjamin: Projectional Editing of Software Product Lines—The PEoPL Approach. Dissertation, University of Luxembourg, 2017.

[BPB17] Behringer, Benjamin; Palz, Jochen; Berger, Thorsten: PEoPL: Projectional Editing of Software Product Lines. In: 30th International Conference on Software Engineering (ICSE). 2017.