Trace-Based Propagation of Variability Annotations¹

Bernhard Westfechtel², Sandra Greiner³

Abstract: This contribution presents a mechanism to extend single- to multi-variant model transformations based on traces created during the transformation. The approach tackles a problem typically occurring in model-driven software product line engineering. Models are the key artifacts of such product lines and annotated with variability annotations in the case an annotative approach towards product line engineering is followed. Although model transformations are well-developed by now and a key facility when developing model-driven software, they are not capable to handle the variability annotations of product lines. Consequently, they transform modeled artifacts of a product line but ignore their annotations. We propose to propagate variability annotations a posteriori, using the traces of transformation execution. This approach is generic and may be applied to heterogeneous tools.

Keywords: Model transformation, software product line, annotative variability

1 Background

Model-Driven software product line engineering (MDPLE) combines two disciplines, model-driven software engineering (MDSE) and software product line engineering (SPLE), both aiming at increasing the level of productivity when developing software-intensive systems. On the one hand, SPLE follows the paradigms of organized reuse and variability [PBvdL05]. In annotative approaches [Ap09] software artifacts carry presence conditions, we refer to as annotations below, which are boolean expressions defining the visibility of an artifact in a product. Typically, feature models capture the common and varying parts of the product line. A feature configuration, providing each feature with a selection state, allows to derive customized products. On the other hand, MDSE aims at increasing the level of abstraction and automation by using models instead of source code artifacts. Model transformations are the common means to convert one model representation into another, e.g., to generate source code in a model-to-text transformation. In MDPLE, model transformations may be required in all development phases. In particular, they may become necessary when the product line is designed, for instance, with class diagrams which should be turned into source code (or its modeled representation). However, state-of-the-art model transformations cannot handle annotations attached to model elements out-of-the-box. Consequently, the product line engineer needs to annotate the target representation manually, which contradicts the aim of automation and increased productivity in MDPLE.

¹ This paper is an extended abstract of [WG18].
² Universität Bayreuth, Lehrstuhl für Angewandte Informatik, Universitätsstraße 30, 95440 Bayreuth, Germany Bernhard.Westfechtel@uni-bayreuth.de
³ Universität Bayreuth, Lehrstuhl für Angewandte Informatik, Universitätsstraße 30, 95440 Bayreuth, Germany Sandra1.Greiner@uni-bayreuth.de

doi:10.18420/SE2020_16
2 Contribution

In order to address the problem outlined above, we propose automatic trace-based propagation of variability annotations [WG18]: The trace of a single-variant model transformation is used to propagate annotations from source to target model elements. This a posteriori approach is defined generically, i.e., independent of the transformation language as well as the tool environment. As a consequence, it may be integrated in heterogeneous MDPLE tool suites. We formally prove its correctness based on the commutativity criterion. It postulates that the two branches transform-filter (1) and filter-transform (2) should commute: For each feature configuration, transforming a multi-variant model and filtering the multi-variant target model afterwards (branch 1) should result in the same model as on branch 2, where the multi-variant source model is filtered first, followed by transforming the source model by the reused single-variant model transformation.

3 Results and Outlook

The commutativity proof assumes that single-variant model transformations conform to a computational model with certain properties and traces provide sufficient information to propagate annotations correctly. Current model transformation languages and tools partially violate these assumptions. Since we intend to support heterogeneous tool suites, we have not addressed the development of a new transformation language and tool guaranteeing commutativity. Rather, we are designing extensions to trace-based propagation that address violations of the assumptions underlying the commutativity proof. For example, in [GW19] we address incomplete traces, resulting in annotations of target model elements that are initially incomplete. Various algorithms are proposed to make these annotations complete.

Bibliography


