Modeling Variability in Template-based Code Generators for Product Line Engineering

Timo Greifenberg[1] , Klaus Müller[1], Alexander Roth[1], Bernhard Rumpe[1], Christoph Schulze[1], Andreas Wortmann[1]

Abstract: Generating software from abstract models is a prime activity in model-driven engineering. Adaptable and extendable code generators are important to address changing technologies as well as user needs. However, they are less established, as variability is often designed as configuration options of monolithic systems. Thus, code generation is often tied to a fixed set of features, hardly reusable in different contexts, and without means for configuration of variants. In this paper, we present an approach for developing product lines of template-based code generators. This approach applies concepts from feature-oriented programming to make variability explicit and manageable. Moreover, it relies on explicit variability regions (VR) in a code generator's templates, refinements of VRs, and the aggregation of templates and refinements into reusable layers. A concrete product is defined by selecting one or multiple layers. If necessary, additional layers required due to VR refinements are automatically selected.

Keywords: Model-Driven Engineering, Code Generator Development, Variability Modeling

1 Introduction

Engineering complex software systems introduces a conceptual gap between the problem domains and the solution domains of discourse [FR07]. Model-driven engineering (MDE) aims to bridge this gap by lifting abstract models to primary development artifacts. Deriving executable software from models requires extensive handcrafting or code generators. Thus, generating software from abstract models is a prime activity in MDE and many domains have adopted code generation [RR15].

Although reuse is of essence in software engineering, most code generators are monoliths developed for a very specific purpose (such as a certain target platform with specific features) that do not consider reuse or variability as their primary focus. Reusing such code generators in different contexts with different requirements or features is hardly feasible and thus impedes code generator development. One approach to handle variability in such monolithic code generators is to create code generator variants via informal reuse [Jö13] such as copy-paste. In this scenario, the original code generator variant is copied and all required changes are applied to the copy of the variant. The main downside of this approach is that generator changes might need to be applied to all generator copies. This is laborious and error-prone. An alternative to that is to use specific code generation frameworks with built-in support for handling variability [Ac15, Xt15]. Even though this alternative does not result in monolithic code generators, the resulting code generator variants are bound

[1] RWTH Aachen University, Software Engineering, Germany, http://www.se-rwth.de