The Added Value of Programmed Graph Transformations – A Case Study from Software Configuration Management

Thomas Buchmann, Bernhard Westfechtel, and Sabine Winetzhammer

Lehrstuhl Angewandte Informatik 1, University of Bayreuth
D-95440 Bayreuth, Germany
firstname.lastname@uni-bayreuth.de

Abstract. Model-driven software engineering intends to increase the productivity of software engineers by replacing conventional programming with the development of executable models at a high level of abstraction. It is claimed that graph transformation rules contribute towards this goal since they provide a declarative, usually graphical specification of complex model transformations. Frequently, graph transformation rules are organized into even more complex model transformations with the help of control structures, resulting in full-fledged support for executable behavioral models.

This paper examines the added value of programmed graph transformations with the help of a case study from software configuration management. To this end, a large model is analyzed which was developed in the MOD2-SCM project over a period of several years. The model was developed in Fujaba, which provides story diagrams for programming with graph transformations. Our analysis shows that the model exhibits a strongly procedural flavor. Graph transformation rules are heavily used, but typically consist of very small patterns. Furthermore, story diagrams provide fairly low level control structures. Altogether, these findings challenge the claim that programming with graph transformations is performed at a significantly higher level of abstraction than conventional programming.

1 Introduction

Model-driven software engineering is a discipline which receives increasing attention in both research and practice. Object-oriented modeling is centered around class diagrams, which constitute the core model for the structure of a software system. From class diagrams, parts of the application code may be generated, including method bodies for elementary operations such as creation/deletion of objects and links, and modifications of attribute values. However, for user-defined operations only methods with empty bodies may be generated which have to be filled in by the programmer. Here, programmed graph transformations may provide added value for the modeler. A behavioral model for a user-defined operation may be specified by a programmed graph transformation. A model instance being composed of objects, links, and attributes is considered as an attributed graph. A graph transformation rule specifies the replacement of a subgraph in a declarative way. Since it may not be possible to model the behavior of a user-defined

A. Schürr, D. Varró, and G. Varró (Eds.): AGTIVE 2011, LNCS 7233, pp. 198–209, 2012.

operation with a single graph transformation rule, control structures are added to model composite graph transformations.

But what is the added value of programmed graph transformations? Typical arguments which have been frequently repeated in the literature in similar ways are the following ones:

1. A graph transformation rule specifies a complex transformation in a rule-based, declarative way at a much higher level of abstraction than a conventional program composing elementary graph operations.
2. Due to the graphical notation, programming with graph transformations is intuitive and results in (high-level) programs which are easy to read and understand.

This paper examines the added value of programmed graph transformations by analyzing a large model which was developed in the MOD2-SCM project (Model-Driven and Modular Software Configuration Management) [3] over a period of several years. The model was developed in Fujaba [8], which provides story diagrams for programming with graph transformations. We analyze the MOD2-SCM model both quantitatively and qualitatively to check the claims stated above.

2 MOD2-SCM

The MOD2-SCM project [3] is dedicated to the development of a model-driven product line for Software Configuration Management (SCM) systems [1]. In contrast to common SCM systems, which have their underlying models hard-wired in hand-written program code, MOD2-SCM has been designed as a modular and model-driven approach which (a) reduces the development effort by replacing coding with the creation of executable models and (b) provides a product line supporting the configuration of an SCM system from loosely coupled and reusable components.

To achieve this goal, we used Fujaba [8] to create the executable domain model of the MOD2-SCM system. The main part of the work was to (1) create a feature model [4], that captures the commonalities and variable parts within the domain software configuration management and (2) to create a highly modular domain model whose loosely coupled components can be configured to derive new products. To this end, a model library consisting of loosely coupled components that can be combined in an orthogonal way has been built around a common core.

The success of a product line heavily depends upon the fact that features that have been declared as independent from each other in the feature model are actually independent in their realizing parts of the domain model. Thus, a thorough analysis of the dependencies among the different modules is crucial [3] in order to derive valid product configurations. In order to keep track of the dependencies in large domain models, a tool based upon UML package diagrams has been developed and integrated with Fujaba to support the modeler during this tedious task [2]. In the context of the MOD2-SCM project, graph transformations were used to specify the behavior of the methods that have been declared in the domain model.

In this paper, we will discuss the added value of graph transformations especially in the development of large and highly modular software systems. The added value