Chapter 26
MoDSEL:
Model-Driven Software Evolution Language

Ersin Er
Hacettepe University, Turkey

Bedir Tekinerdogan
Bilkent University, Turkey

ABSTRACT

Model-Driven Software Development (MDSD) aims to support the development and evolution of software intensive systems using the basic concepts of model, metamodel, and model transformation. In parallel with the ongoing academic research, MDSD is more and more applied in industrial practices. Like conventional non-MDSD practices, MDSD systems are also subject to changing requirements and have to cope with evolution. In this chapter, the authors provide a scenario-based approach for documenting and analyzing the impact of changes that apply to model-driven development systems. To model the composition and evolution of an MDSD system, they developed the so-called Model-Driven Software Evolution Language (MoDSEL) which is based on a megamodel for MDSD. MoDSEL includes explicit language abstractions to specify both the model elements of an MDSD system and the evolution scenarios that might apply to model elements. Based on MoDSEL specifications, an impact analysis is performed to assess the impact of evolution scenarios and the sensitivity of model elements. A case study is provided to show different kind of evolution scenarios and the required adaptations to model elements.

INTRODUCTION

In traditional, non-model-driven software development the link between the code and higher level design models is not formal but intentional. Required changes are usually addressed manually using the given modeling language. Because of the manual adaptation the maintenance effort is not optimal and as such sooner or later the design models become inconsistent with the code since changes are, in practice, defined at the code level. One of the key motivations for introducing model-driven software development (MDSD) is the need to reduce the maintenance effort and as such support evolution. MDSD aims at achieving this goal through defining model elements as

DOI: 10.4018/978-1-4666-4301-7.ch026

first class abstractions, and providing automated support using model transformations. For a given change requirement the code is not changed manually but automatically generated or regenerated, thereby substantially reducing maintenance effort. Further, because of the formal links between the models and the code the evolution of artefacts in the model-driven development process is synchronized. The link between the code and models is formal. In fact, there are only models, and as such, 'the documentation is the code'. Research on MDSD is continuing to improve the expressiveness of the three key abstractions of model, metamodel and transformation (Kleppe, 2008). As such even better and more automated support to cope with changing requirements and as such to provide reuse, portability, interoperability, and maintenance. Because of the promising benefits for development and evolution, MDSD is more and more applied in industrial projects (Hästbacka, 2011; Fieber, 2009; Maurmaier, 2008). Albeit, MDSD provides from one perspective better support for evolution, it also introduces new dimensions and challenges for software evolution (Visser, 2007; Briand, 2003). Like conventional code, models, metamodels and transformations might be subject to changing requirements and as such require to evolve in due time. Moreover, changes to the metamodels and transformations might render the terminal models invalid.

The software evolution problem in MDSD needs to address different challenges. One of the initial and key issues in considering evolution in MDSD is the impact of changes to the existing systems. To understand evolution in MDSD we have provided a *megamodel*, that consists of both a model for MDSD, the model for adapting model elements, and the model for scenarios that reflect the evolution process. Based on the megamodel we propose a scenario-based approach for analyzing the impact of changes that apply to model-driven development systems. For modeling the required changes we define the notion of so-called *evolution scenario,* which is defined as a description of the need for changes due to concerns of stakeholders. The concept *evolution scenario* has been inspired from the method called Scenario-based Analysis of Software Architecture that focuses on a more general use of scenarios (Kazman, 1996). Each evolution scenario will usually have an impact on the MDSD system and require changes to the models, metamodels or transformations.

To provide automated support for both documenting and analysis of evolution scenarios on MDSD projects we have developed the so-called domain-specific Model-Driven Software Evolution Language (MoDSEL). MoDSEL includes explicit language abstractions to specify model elements and evolution scenarios that apply to model elements. Based on MoDSEL specifications an impact analysis is provided to measure the impact of evolution scenarios and the sensitivity of model elements to the given evolution scenarios. We have supported the analysis process with a set of metrics (Fenton, 1997) that measure the impact of the defined scenarios as well as the sensitivity of each model element with respect to these scenarios. Once the system and the scenarios are modeled the metric values are automatically generated. The result of the measurement based on these metrics can support the decision in the design and refactoring of the system.

A case study for web-based conference management is used to show different kind of evolution scenarios and the required adaptations to model elements. The scope of our study includes the evolution in forward engineering that aims to derive concrete implementation from higher level abstractions. We do not consider evolution in reverse engineering projects.

The remainder of the chapter is organized as follows: In the second section we describe the example case of web-based conference management system. The third section defines the megamodel for modeling model-driven systems. The fourth section describes the design and implementation of MoDSEL. The fifth section presents the scenario-based impact analysis process. The sixth