Science of Computer Programming 104 (2015) 2–43

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Co-evolving meta-models and their instance models: A formal approach based on graph transformation

Florian Mantz [a,*], Gabriele Taentzer [b,a], Yngve Lamo [a], Uwe Wolter [c]

[a] Department of Computer Engineering, Bergen University College, Norway
[b] Department of Mathematics and Informatics, Philipps-University of Marburg, Germany
[c] Department of Informatics, University of Bergen, Norway

ARTICLE INFO

Article history:
Received 23 July 2013
Received in revised form 4 January 2015
Accepted 6 January 2015
Available online 14 January 2015

Keywords:
Meta-model evolution
Model migration
Graph transformation

ABSTRACT

Model-driven engineering focuses on models as primary artifacts of the software development process, which means programs are mainly generated by model-to-code transformations. In particular, modeling languages tailored to specific domains promise to increase the productivity of software developers and the quality of generated software. Modeling languages, however, evolve over time and therefore, existing models have to be migrated accordingly. The manual migration of models tends to be tedious and error-prone, therefore tools have been developed to (partly) automate this process. Nevertheless, the migration results may not always be well-defined.

In this article, we provide a formal framework for model migration which is independent of specific modeling approaches. We treat modeling languages, formalized by meta-models, as well as models as graphs and consider their co-evolutions as coupled graph transformations. In the same line, we study the conditions under which model migrations are well-defined. Existing solutions to model migration are either handwritten or default solutions that can hardly be customized. Here, we introduce a high-level specification approach, called model migration schemes, that supports automation and customization. Starting from a meta-model evolution rule, a default migration scheme can be automatically deduced and customized.

1. Introduction

Model-based, and particularly, model-driven software development considers models as primary artifacts of the software development process that have to be kept up-to-date throughout software evolution. Especially, in model-driven development, domain-specific modeling languages (DSMLs) are developed to lift recurring engineering tasks to a higher abstraction level and to provide sufficient information for automatic code generation. Hence, application-specific knowledge is described by models using domain-specific languages. These models serve as an input for not only code generation, but also validation and testing. Therefore, DSMLs can be considered as means to increase productivity and quality of software.

To maintain such advantages, DSMLs must evolve according to changing requirements and in line with target domains. This implies that existing models need to be adapted according to language changes. Since modeling languages are usually defined by so-called meta-models, this means that meta-model evolutions must essentially be reflected by model migrations

* Corresponding author.
E-mail addresses: fma@hib.no (F. Mantz), taentzer@informatik.uni-marburg.de (G. Taentzer), yla@hib.no (Y. Lamo), Uwe.Wolter@ii.uib.no (U. Wolter).

http://dx.doi.org/10.1016/j.scico.2015.01.002
0167-6423/

(see Fig. 1). This challenge has previously been addressed by (partial) automation of this tedious and error-prone process (see e.g. [1,2]).

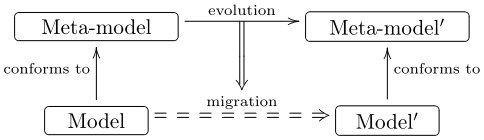

Fig. 1. Meta-model evolution and model migration.

Current research focuses on popular modeling frameworks such as the Eclipse Modeling Framework [3] and tool development. The conditions under which model migrations are well-defined have hardly been considered. Moreover, model migrations are typically programmed in a relatively low-level language. Here we explore an approach that raises the abstraction level of migration specifications allowing to specify them conveniently. In addition, migration definitions in this approach can be reused. Accordingly, the following requirements must be considered:

1. Migrated models must belong to the evolved modeling language. This property is usually called *soundness*. It subdivides into *well-typedness* and *well-definedness* w.r.t. language constraints. A migrated model is well-typed if all its elements are typed over the evolved meta-model. Moreover, all well-definedness rules of the evolved language have to be satisfied.
2. All models of the original modeling language can be migrated to the evolved language meaning that the migration is viable. This property is usually referred to as *completeness*.
3. Model migration should be specified on a high abstraction level. This means that a model migration is either automatically deduced from its meta-model evolution or specified using a high-level language.
4. The specification of model migrations is *reusable* (see also [1]). In particular, equivalent migration steps are specified only once.
5. General strategies for model co-evolution are formulated *independently of a specific meta-modeling approach*.

To address these requirements, we choose a formal approach to model transformations that is independent of model representations. Graph transformations [4,5] are well-established means to formally underpin model transformations. Moreover, the theory of graph transformations has been lifted to high-level structures, not necessarily being graphs [4]. In the following, we present a formal approach to model co-evolution based on graph transformations.

1. In our *formal approach*, meta-model evolutions and migrations of their instance models are specified by coupled graph transformations. This formal setting allows us to study *completeness* of model migrations and *well-typedness* of migrated models. Moreover, *well-definedness w.r.t. multiplicity constraints* is (partly) considered.
   In particular, we encode each meta-model evolution step by a graph transformation $tt_i : MM_i \to MM_{i+1}$. A model migration step is defined as graph transformation step $t_i : M_i \to M_{i+1}$ being typed over tt_i, i.e. M_i is typed over MM_i and M_{i+1} is typed over MM_{i+1} (see Fig. 2, black vertical edges denote typings).

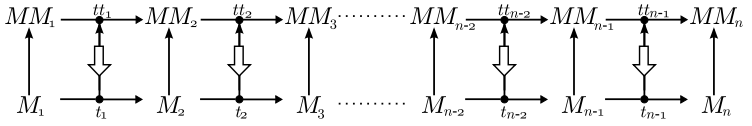

Fig. 2. Co-evolution steps.

2. To specify model migrations on a high abstraction level and for having the possibility to reuse them, we introduce *model migration schemes*. They support the pattern-based definition of model migrations. Once a model migration scheme is specified, it can be applied automatically to instance models yielding well-typed migrated results. Migration schemes lift the abstraction level of migration specification since they focus on the definition of migration patterns. Pattern recognition and their synchronized replacement do not have to be specified but are defined by the approach. In this context, *reuse* has two dimensions: Migration schemes can be used for (1) different instance models and also (2) for migrating models of different meta-models being evolved by the same meta-model evolution rules.
   To increase the level of automation, we show how *default migration schemes* can be derived from given meta-model evolution rules[1] by identifying related model patterns reflecting meta-model evolution steps. Since default migration schemes do not always reflect the intended semantics of evolution steps, we allow well-defined *customizations* of migration schemes as well.

[1] Which are defined manually or automatically.