Journal of Logical and Algebraic Methods in Programming 83 (2014) 427–458

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Towards the flexible reuse of model transformations: A formal approach based on graph transformation

Juan de Lara *, Esther Guerra

Computer Science Department, Universidad Autónoma de Madrid, Spain

ARTICLE INFO

Article history:
Received 27 February 2013
Received in revised form 31 July 2014
Accepted 13 August 2014
Available online 23 August 2014

Keywords:
Model-driven engineering
Graph transformation
Meta-modelling
Genericity
Reusability

ABSTRACT

Model transformations are the *heart and soul* of Model-Driven Engineering (MDE). However, in order to increase the adoption of MDE by industry, techniques for developing model transformations in the large and raising the quality and productivity in their construction, like reusability, are still needed.

In previous works, we developed a reutilization approach for graph transformations based on the definition of *concepts*, which gather the structural requirements needed by meta-models to qualify for the transformations. Reusable transformations are typed by concepts, becoming *transformation templates*. Transformation templates are instantiated by *binding* the concept to a concrete meta-model, inducing a retyping of the transformation for the given meta-model.

This paper extends the approach allowing heterogeneities between the concept and the meta-model, thus increasing the reuse opportunities of transformation templates. Heterogeneities are resolved by using algebraic adapters which induce *both* a retyping and an adaptation of the transformation. As an alternative, the adapters can also be employed to induce an adaptation of the meta-model, and in this work we show the conditions for equivalence of both approaches to transformation reuse.

1. Introduction

Model-Driven Engineering (MDE) [3,38] promotes an active use of models in the different phases of the software development. This involves the transformation of models between different languages – ranging from general-purpose to domain-specific modelling languages (DSMLs) – until code for the final application is generated.

MDE can be seen as a *reutilization* approach, where modelling languages and their associated transformations and code generators are reused across projects to describe different applications within a domain, but with certain variability that is configured through a model. However, it is also true that MDE is *type-centric* [7], because the different supporting artefacts (transformations and code generators) are defined over the types of a specific meta-model and cannot be reused for other meta-models, even if they share *essential* structural features. This rigidity hampers the adoption of MDE in industry because similar transformations have to be repeatedly developed, even for meta-models with only slight differences.

Taking ideas from generic programming, in previous works we proposed the definition of transformations over so-called *concepts* [7,16,17], instead of over concrete meta-models. In our context, a concept specifies the structural requirements that meta-models need to fulfill in order to be able to apply a certain model operation (e.g. a transformation) on their instances.

* Corresponding author.
E-mail addresses: Juan.deLara@uam.es (J. de Lara), Esther.Guerra@uam.es (E. Guerra).

http://dx.doi.org/10.1016/j.jlamp.2014.08.005
2352-2208/

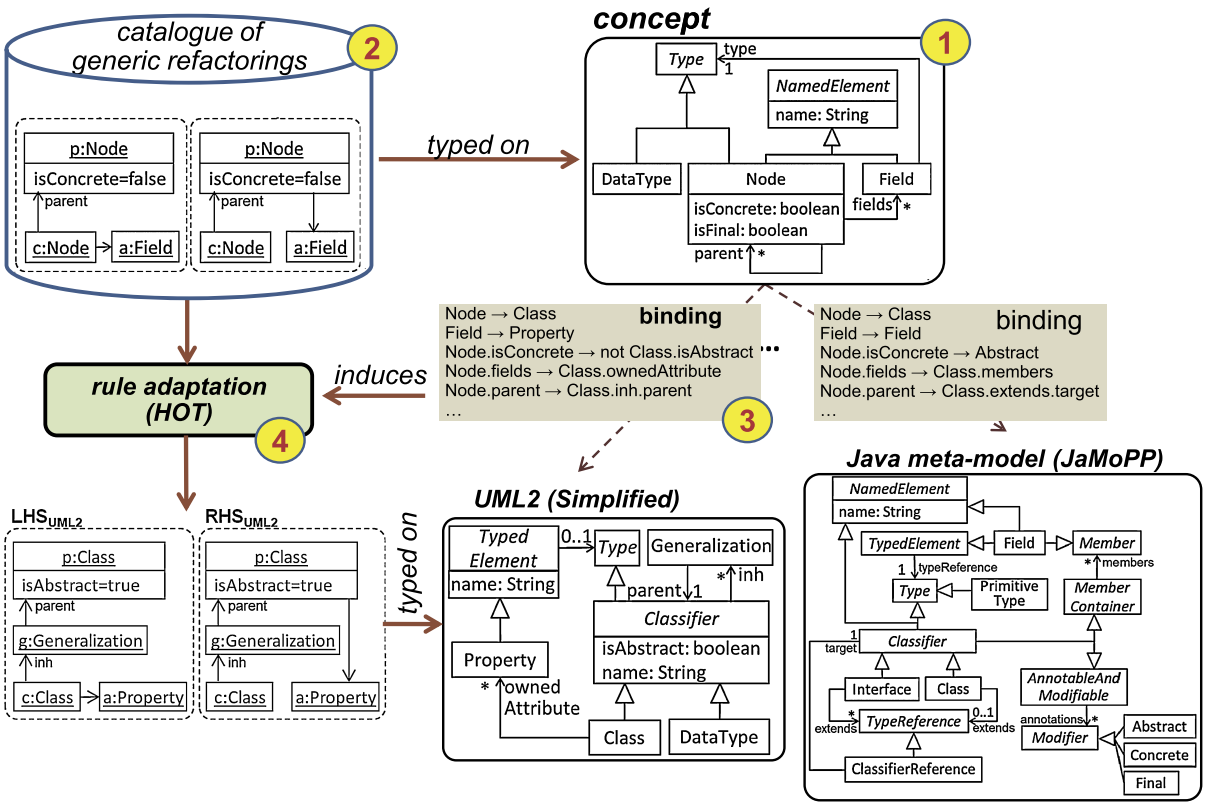

Fig. 1. Rule template defined over a concept, and instantiation via a binding.

Our concepts resemble meta-models, but their elements (classes, references, fields) are variables that need to be *bound* to concrete meta-model elements. In this way, similar to generic programming templates [16], a *transformation template* is defined over a concept and is instantiated for a specific meta-model via a *binding*.

In [6], we formalized these techniques using graph transformation (GT) [11] to express model transformations, and restricting the binding to simple injective mappings between the concept and the meta-model elements. In this paper, we expand the formalization by allowing a more flexible binding by means of algebraic *adapters*, which are able to resolve heterogeneities between the concept and the meta-model, in the line of [29]. This approach increases the reuse opportunities of transformation templates because their associated concepts can be bound to a wider set of meta-models. Interestingly, the formalization of our adapters involves building a "virtual view" that unifies the two main approaches to genericity in MDE (namely, adaptation of the transformation [29] and meta-model adaptation [18]) and enables the study of the conditions for their equivalence.

The rest of this paper is organized as follows. Section 2 presents an overview of our approach. Section 3 formalizes (meta-)models and concepts. Section 4 introduces binding adapters and Section 5 shows their use to instantiate a GT template and to build a derived model. Section 6 compares with related work and, finally, Section 7 concludes the paper and identifies lines of future work. Appendix A shows the details of the different proofs.

2. Motivation, overview and challenges

Assume we want to define a catalogue of refactorings for object-oriented notations [14] using GT rules. The first step is to define a meta-model so that the rules can be typed. However, this means that the rules will only be applicable to instances of such meta-model. This prevents the refactorings from being reused, as they cannot be applied to other object-oriented notations sharing common features – like UML class diagrams [37], KM3 [22] or Ecore [34] – but we need to encode slight variations of the same refactorings for each notation.

To overcome this limitation, we propose defining the rules over a so-called *concept*, as illustrated in Fig. 1. Label 1 depicts a concept for the refactoring of object-oriented notations. Label 2 shows one simple refactoring rule, which moves a field from a class to one of its parents. This rule is typed over the concept. A concept has the form of a standard meta-model, but it needs to be bound to some concrete meta-model, as shown in label 3. This binding induces an adaptation of the rule via a high-order transformation (HOT) to make it applicable to the meta-model instances, as shown in label 4. Hence, similar to generic programming [10], GT rules so defined become *templates* that need to be instantiated for particular meta-models. This approach promotes reusability because the same transformation can be applied to every meta-model to which we can