Constraint-Aware Model Transformations

Adrian Rutle[1], Alessandro Rossini[2], Yngve Lamo[1] and Uwe Wolter[2]
[1]Faculty of Engineering, Bergen University College, Norway
[2]Department of Informatics, University of Bergen, Norway

ABSTRACT

This paper introduces a formal approach to the specification of constraint-aware model transformation which is suitable for language translation. The proposed approach extends graph transformations with the ability to specify and transform constraints by transformation rules. In particular, it uses non-deleting rules which are typed over the metamodel of a joined language which is constructed from the source and target languages. In addition, the application of transformation rules is formalised as pushout constructions which create a model that is typed over the metamodel of the joined language. Finally, the target model is obtained from the created model by a pullback construction.

1. INTRODUCTION

Models are first-class entities of the software development process in Model-Driven Engineering (MDE) and undergo a complex evolution during their life-cycles. In this regard, model transformation is one of the key techniques which is used to automate several model-based activities such as code generation, refactoring, optimisation, language translation etc. [11]. In this paper we focus on constraint-aware model transformation which is especially suitable for language translation – translate between models specified in different modelling languages.

In the MDE context, models are graph-based structures which are enriched with constraints. These constraints must be satisfied at any state of the software systems represented by the models. Models are defined by means of modelling languages and frameworks such as the Unified Modeling Language (UML) and Eclipse Modeling Framework (EMF). Each of these modelling languages has a corresponding metamodel – a model that defines the abstract syntax of models which can be specified by the modelling language. These metamodels, in turn, are specified by means of a metamodelling framework called the Meta-Object Facility (MOF). MOF-based modelling languages allow the specification of simple constraints such as multiplicity and uniqueness constraints, hereafter called *structural constraints* (see Fig. 1a). These constraints are usually specified by properties of classes in the metamodel of the modelling language.

However, structural constraints may not be sufficient to meet complex requirement's specifications. Hence, textual constraint languages such as the Object Constraint Language (OCL) are usually used to define complex constraints, hereafter called *attached OCL constraints* (see Fig. 1b). Unfortunately, these attached OCL constraints are ignored by existing transformation techniques [6]. This problem is close-
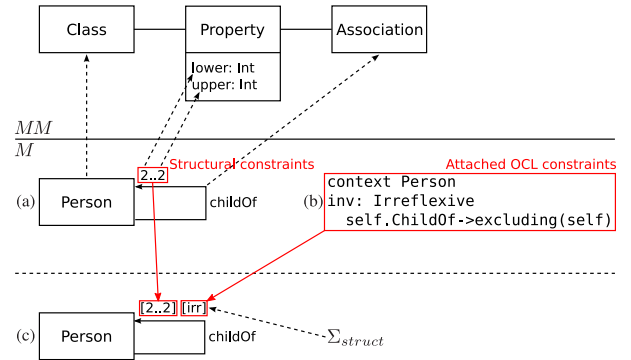

Figure 1: Constraints in MOF-based modelling languages: (a) structural constraints in UML (b) attached OCL constraints (c) integration of constraints in the DPF-based formalisation of MOF-based modelling languages

ly related to the fact that the *conformance* relation between models and metamodels is not formally defined for MOF-based modelling languages [1].

In this paper, a solution to this challenge is outlined. The solution reuses the formalisation approach to MOF-based modelling languages in which constraints are integrated in (meta)modelling [9]. The formalisation approach is based on the Diagram Predicate Framework (DPF) [9, 8, 10] which provides a formal diagrammatic approach to (meta)modelling and model transformation based on category theory. The DPF-based approach to model transformation provides an extension to the formal framework of graph transformations in the sense that it can be used to transform models as well as attached OCL constraints. This is done by integrating structural constraints and attached OCL constraints in modelling formalisms which represent the formalisation of MOF-based modelling languages (see Fig. 1c for a intuition of the DPF-based solution).

The outline of the paper is as follows. In section 2, we introduce the basics of the DPF-based formalisation approach to (meta)modelling. Section 3 outlines our approach to constraint-aware model transformation. Finally, Sections 4 and 5 addresses related work and concludes the paper.

2. DPF

In DPF, models are formalised as diagrammatic specifications. A diagrammatic specification $S = (G^S, C^S)$ consists of an underlying graph G^S *decorated by* a set of constraints

C^S. The graph represents the structure of the model, and predicates from a predefined *diagrammatic signature* are used to impose constraints on the model. In this paper, we use the terms "model" and "diagrammatic specification" interchangeably.

In this formalisation, we distinguish between two types of conformance relations: *typed over* and *conforms to*. A model is typed over a metamodel (called a typed specification) if its underlying graph is typed over the underlying graph of the metamodel; i.e. each model element is assigned a type in the metamodel by a *typing map*. In contrast, a model is said to conform to a metamodel if it is typed over the metamodel and, in addition, it satisfies all the constraints imposed by the metamodel.

Moreover, in DPF each modelling language is formalised as a modelling formalism [9], e.g. $(\Sigma_1^S, S_2, \Sigma_2^S)$ in Fig. 2a. A $(\Sigma_1^S, S_2, \Sigma_2^S)$-model, e.g. S_1, is a Σ_1^S-specification which conforms to the metamodel S_2, where S_2 is a Σ_2^S-specification. Moreover, in our transformation approach we use (and distinguish between) the concepts of specification morphisms and typed specification morphisms. A specification morphism between two Σ-specifications S_1 and S_1' is a graph homomorphism which preserves constraints; i.e. for all constraints in S_1 there are corresponding constraints in S_1'. In contrast, a typed specification morphism between two typed specifications is a specification morphism which respects the typing map.

3. MODEL TRANSFORMATION

Several approaches are used for the specification of model transformations (see [7] for a survey of these approaches). Regardless which transformation approach is chosen, the overall pattern is as follows: given a source and a target metamodel and (a set of) source models which conform to the source metamodel, create (a set of) target models which conform to the target metamodel. Often this is achieved by specifying a model transformation definition, which is then executed by a transformation engine. The transformation definition consists of a set (or sequence) of transformation rules, which in turn is declared by input and output patterns. Upon execution of a rule, the transformation engine's task is to create, for every match of the input pattern in a source model, a match of the output pattern in the corresponding target model.

The proposed formalisation approach to constraint-aware model transformation is organised in five steps. These steps are outlined as follows.

3.1 Joining Languages

The first step in transforming models defined in a language to models defined in another language is to relate the two languages. Since in our approach we formalise each modelling language as a modelling formalism, this step consists of creating a joined modelling formalism. The metamodel of the joined modelling formalism consists of the disjoint union of the source and target metamodels, where model elements from these metamodels are connected by means of a graph in a suitable way. In order to maintain the correspondence between elements coming from the source and target metamodels, additional predicates are needed to constrain the joined metamodel.

Roughly speaking, given the source $(\Sigma_1^S, S_2, \Sigma_2^S)$ and target $(\Sigma_1^T, T_2, \Sigma_2^T)$ modelling formalisms, a joined modelling
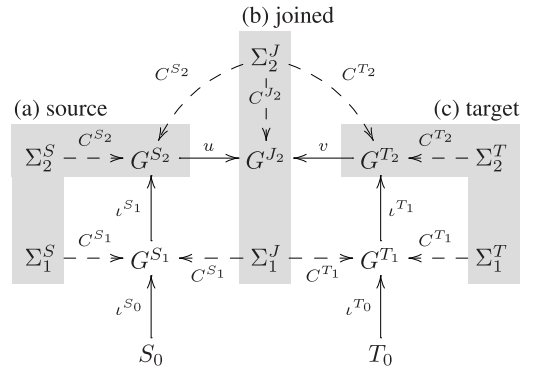

Figure 2: Two modelling formalisms with their metamodels joined together

formalism $(\Sigma_1^J, J_2, \Sigma_2^J)$ will be created (see Fig. 2b). In more detail, the source and target signatures are joined together to $\Sigma_1^J := \Sigma_1^S \uplus \Sigma_1^T$ and $\Sigma_2^J := \Sigma_2^S \uplus \Sigma_2^U \uplus \Sigma_2^T$, and the source and target metamodels are joined together to $J_2 := S_2 \uplus U_2 \uplus T_2$. Note that we denote by $\uplus$ the disjoint union operation. The signature Σ_2^U contains additional predicates which are used to constrain the joined metamodel as a whole. Moreover, U_2 is a Σ_2^J-specification which is used to connect the source and target metamodels.

Although the designer of the transformation is free to relate any model elements of the source and target languages, there is a *projection condition* which should be satisfied by the joined metamodel. The condition is that for any model J_1 conforming to the joined metamodel J_2 it should be possible to construct models S_1 and T_1 conforming to the source and target metamodels S_2 and T_2, respectively, by pullback constructions as depicted in Fig. 3 (see [2] for a detailed description and motivation for using pullbacks in model transformation). Having said this, the problem of language translation can be formulated as *the translation of $(\Sigma_1^S, S_2, \Sigma_2^S)$-specifications, e.g. a source model S_1, to $(\Sigma_1^T, T_2, \Sigma_2^T)$-specifications, e.g. a target model T_1.*

Figure 3: Projection condition for the joined metamodel J_2

3.2 Constraint-Aware Rules

The next step in model transformation is the definition of the transformation rules. Each transformation rule $r : L \hookrightarrow R$ is a typed specification morphism between an input L and an output R pattern. In our approach, which is an extension of non-deleting graph transformation rules [4], the input pattern is included in the output pattern. The input and output patterns of the rules are Σ_1^J-specifications which are typed over G^{J_2}.

3.3 Application of Transformations

After defining the transformation rules, we focus on the application of model transformations. This section outlines how a source model S_1 is transformed to a target model T_1 in the following three steps:

1. An initial step where the source model S_1 is extended to an intermediate G^{J_2}-typed Σ_1^J-specification J_1. This transformation is simply given by the composition $\iota^{S_1}; u$ (see Fig. 2). After this step, $J_1 = S_1 \uplus U_1 \uplus T_1$ with both U_1 and T_1 being empty specifications.

2. Then, stepwise application of the transformation rules to J_1. Upon the application of a rule $r : L \hookrightarrow R$, for every match of the input pattern L in a model $J_1 = S_1 \uplus U_1 \uplus T_1$, the U_1 and T_1 parts of J_1 will be extended by an appropriate copy of the new items in R, i.e., by those items in R that are not already in L. This step is formalised as a pushout construction.

3. Finally, obtaining the target model T_2 by constructing a pullback of the span $J_1 \rightarrow J_2 \leftarrow T_2$ (see Fig. 3). Note that this step is only applicable if J_1 is a $(\Sigma_1^J, J_2, \Sigma_2^J)$-specification. That is, in cases where J_1 is only a Σ_1^J-specification which is typed over G^{J_2}, this step will not yield a $(\Sigma_1^T, T_2, \Sigma_2^T)$-specification T_1, this is due to the projection condition.

4. RELATED WORK

The literature related to model transformation is becoming abundant. There are several approaches to model transformation which are closely related to our approach, however, a formal definition of constraint-aware transformation rules are not explicitly discussed in these approaches. In some cases, even multiplicity constraints are ignored in the definition and examples of transformation rules.

Graph Transformation System (GTS) [4] is widely used as the formal foundation for model transformation approaches. In our approach, we extend GTS by adding support for transformation of constraints. We extend also Triple Graph Grammars (TGG) [3, 5] by adding support for constraining the joined metamodel. This makes the specification and maintenance, i.e. the projection condition, between source and target models possible.

5. CONCLUSION

This paper proposes a formal approach to the definition of constraint-aware model transformation which provides an extension to the formal framework of graph transformations. In the proposed approach, the transformation process is organised into five steps. Firstly, the modelling languages are joined together. Then, the transformation rules are declared as input and output patterns where these patterns are typed over the joined metamodel. Afterwards, the source model is extended to a model which is typed over the joined metamodel. Next, the transformation rules are applied to the extended source model. Finally, the target model is obtained by a pullback construction. In this way, existing machinery from category theory is exploited to formalise model transformations. More precisely, pushout construction is used for the application of transformation rules, and pullback construction for the projection of target models.

6. REFERENCES

[1] A. Boronat and J. Meseguer. Algebraic Semantics of OCL-Constrained Metamodel Specifications. In *TOOLS 2009*, volume 33 of *LNBIP*, pages 96–115. Springer, 2009.

[2] Z. Diskin and J. Dingel. A metamodel Independent Framework for Model Transformation. Technical Report 1/2006, ATEM 2006, Johannes Gutenberg Universität Mainz, Germany, October 2006.

[3] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer. Information Preserving Bidirectional Model Transformations. In *FASE 2007*, volume 4422 of *LNCS*, pages 72–86. Springer, 2007.

[4] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, March 2006.

[5] A. Königs and A. Schürr. Tool Integration with Triple Graph Grammars – A Survey. *ENTCS*, 148(1):113–150, 2006.

[6] S. Marković and T. Baar. Refactoring OCL annotated UML class diagrams. *SoSyM*, 7(1):25–47, 2008.

[7] T. Mens and P. V. Gorp. A Taxonomy of Model Transformation. *ENTCS*, 152:125–142, 2006.

[8] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter. A Category-Theoretical Approach to the Formalisation of Version Control in MDE. In *FASE 2009*, volume 5503 of *LNCS*, pages 64–78. Springer, 2009.

[9] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter. A Diagrammatic Formalisation of MOF-Based Modelling Languages. In *TOOLS 2009*, volume 33 of *LNBIP*, pages 37–56. Springer, 2009.

[10] A. Rutle, U. Wolter, and Y. Lamo. A Diagrammatic Approach to Model Transformations. In *EATIS 2008*, pages 1–8. ACM, 2008.

[11] S. Sendall and W. Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5):42–45, 2003.