The Journal of Systems and Software 83 (2010) 283–302

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Verification and validation of declarative model-to-model transformations through invariants

Jordi Cabot [a,1], Robert Clarisó [a], Esther Guerra [b,*], Juan de Lara [c]

[a] Estudis d'Informàtica, Multimèdia i Telecomunicació, Univ. Oberta de Catalunya, Spain
[b] Computer Science Department, Universidad Carlos III de Madrid, Spain
[c] Polytechnic School, Universidad Autónoma de Madrid, Spain

ARTICLE INFO

Article history:
Received 3 March 2009
Received in revised form 28 July 2009
Accepted 6 August 2009
Available online 15 August 2009

Keywords:
Model-to-model transformation
Model-Driven Development
OCL
Verification and validation
Triple Graph Grammars
QVT

ABSTRACT

In this paper we propose a method to derive OCL invariants from declarative model-to-model transformations in order to enable their verification and analysis. For this purpose we have defined a number of invariant-based verification properties which provide increasing degrees of confidence about transformation correctness, such as whether a rule (or the whole transformation) is satisfiable by some model, executable or total. We also provide some heuristics for generating meaningful scenarios that can be used to semi-automatically validate the transformations.

As a proof of concept, the method is instantiated for two prominent model-to-model transformation languages: Triple Graph Grammars and QVT.

1. Introduction

Model-Driven Development (MDD) is a software engineering paradigm where models are the core asset (Völter and Stahl, 2006). They are used to specify, simulate, test, verify and generate code for the application to be built. Many of these activities include the specification and execution of model-to-model (M2M) transformations, that is, the transformation of a model conformant to a meta-model into another one conformant to a different meta-model.

There are two main approaches to M2M transformation: *operational* and *declarative*. The former is based on rules or instructions that explicitly state how and when creating the elements of the target model from elements of the source one. Instead, in declarative approaches, some kind of visual or textual patterns describing the relations between the source and target models are provided, from which operational mechanisms are derived e.g. to perform forward and backward transformations. These declarative patterns are complemented with additional information to express relations between attributes in source and target elements, as well as to constrain when a certain relation should hold. The Object Constraint Language (OCL) standard (Object Management Group, 2003) is frequently used for this purpose (OMG, 2007).

The increasing complexity of modelling languages, models and transformations makes urgent the development of techniques and tools that help designers to assure transformation correctness. Whereas several notations have been proposed for specifying M2M transformations in a declarative way (Akehurst et al., 2003; Jouault et al., 2006; OMG, 2007; Schürr, 1994), there is a lack of methods for analysing their correctness in an integral way, taking into account the relations expressed by the transformation, as well as the meta-models and their well-formedness rules.

In this paper we propose verification and validation techniques for M2M transformations based on the analysis of a set of OCL invariants automatically derived from the declarative description of the transformations. These invariants state the conditions that must hold between a source and a target model in order to satisfy the transformation definition, i.e. in order to represent a valid mapping. We call these invariants, together with the source and target meta-models, a *transformation model* (Bézivin et al., 2006). To show the wide applicability of the technique, we study how to create this transformation model from two prominent M2M transformation languages: Triple Graph Grammars (TGGs) (Schürr, 1994) and QVT (OMG, 2007).

Once the transformation model is synthesized, we can determine several correctness properties of the transformation by analysing the generated transformation model with any available

* Corresponding author.
E-mail addresses: jcabot@uoc.edu (J. Cabot), rclariso@uoc.edu (R. Clarisó), eguerra@inf.uc3m.es (E. Guerra), Juan.deLara@uam.es (J. de Lara).
[1] Rbla. del Poblenou 156, E-08018 Barcelona, Spain.

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.08.012

tool for the verification of static UML/OCL class diagrams (see Anastasakis et al., 2007; Brucker and Wolff, 2006; Cabot et al., 2008; Queralt and Teniente, 2006; Straeten et al., 2003). In particular, we have predefined a number of verification properties in terms of the extracted invariants, which provide increasing confidence on the transformation correctness. For example, we can check whether a relation or the whole transformation is applicable in the forward direction (i.e., whether there is a source model enabling a relation), forward weak executable (if we can find a pair of source and target models satisfying the relation and the meta-model constraints), forward strong executable (if a relation is satisfied whenever it is enabled), or total (whether all valid source models can be transformed). In order to illustrate this analysis, we show the use of the UMLtoCSP tool (Cabot et al., 2008) to perform the verification. The tool translates the transformation model into a constraint satisfaction problem, which is then processed with constraint solvers to check different aspects of the model.

The transformation model can also be used for validation purposes. Given the transformation model, tools like UMLtoCSP can be used to automatically generate valid pairs of source and target models, or a valid target model for a given or partially specified source model. These generated pairs help designers in deciding whether the defined transformation reflects their intention, thus helping to uncover transformation defects. Additionally, we have devised heuristics to partially automate the validation process by means of generating potentially relevant scenarios (representing corner cases of the transformation) that the designer may be specially interested in reviewing.

This paper extends our preliminary work in Cabot et al. (2008). Here, we propose a new way of handling OCL attribute conditions in TGGs which avoids algebraic manipulations; provide a new way of generating invariants, so as to make the resulting TGG and QVT invariants more uniform, easing its portability to other languages; present a detailed formalization of the extraction of invariants from QVT; provide a comprehensive list of formalized verification properties; and present a semi-automatic method for validation.

Paper organization. Section 2 introduces TGGs and our proposal for handling OCL attribute conditions. Section 3 presents the method for extracting invariants from TGGs. Sections 4 and 5 present such method for QVT. Section 6 shows the use of the invariants and UML/OCL analysis tools for the verification and validation of transformations. Section 7 compares with related work and Section 8 draws the conclusions. As running example we use a transformation between class diagrams and relational schemas (OMG, 2007). The appendix includes all the invariants for the example.

2. Triple Graph Grammars

Triple Graph Grammars (TGGs) (Schürr, 1994) were proposed by A. Schürr as a formal means to specify transformations between two languages in a declarative way. TGGs are founded on the notion of graph grammar (Rozenberg, 1997). A graph grammar is made of rules having graphs in their left and right hand sides (LHS and RHS), plus the initial graph to be transformed. Applying a rule to a graph is only possible if an occurrence of the LHS (a *match*) is found in it. Once such occurrence is found, it is replaced by the RHS graph. This is called *direct derivation*. It may be possible to find several matches for a rule, and then one is chosen at random. The execution of a grammar is also non-deterministic: at each step, one rule is randomly chosen and its application is tried. The execution ends when no rule can be applied.

Even though graph grammar rules rely on pre- and post-conditions, and on pattern matching, when used for model-to-model transformation, they have an operational, unidirectional style, as the rules specify how to build the target model assuming the source already exists. On the contrary TGGs are declarative and bidirectional since, starting from a unique TGG specifying the synchronized evolution of two graphs, it is possible to generate forward and backward transformations as well as operational mechanisms for other scenarios (Königs and Schürr, 2006).

TGGs are made of rules working on triple graphs. These are made of two graphs called *source* and *target*, related through a *correspondence* graph. Any kind of graph can be used for these three components, from standard unattributed graphs $(V; E; s, t : E \to V)$ to more complex attributed graphs, e.g. E-graphs (Ehrig et al., 2006). The nodes in the correspondence graph (the *mappings*) have morphisms[2] to the nodes in the source and target graphs. Triple morphisms are defined as three graph morphisms that preserve the correspondence functions. They are used to relate the LHS and RHS of a TGG rule, to identify a match of the LHS in a graph, and to type a triple graph.

Definition 1 (*Triple graph and morphism*). A triple graph $TrG = (G_s, G_c, G_t, cs : V_{G_c} \to V_{G_s}, ct : V_{G_c} \to V_{G_t})$ is made of two graphs G_s and G_t called source and target, related through the nodes of the correspondence graph G_c.

A triple graph morphism $f = (f_s, f_c, f_t) : TrG^1 \to TrG^2$ is made of three graph morphisms $f_x : G_x^1 \to G_x^2$ (with $x = \{s, c, t\}$) such that the correspondence functions are preserved.

In the previous definition, V_{G_x} is the set of nodes of graph G_x. Morphisms *cs* and *ct* relate two nodes *x* and *y* in the source and target graphs iff $\exists n \in V_{G_c}$ with $cs(n) = x$ and $ct(n) = y$. We often depict a triple graph by $\langle G_s, G_c, G_t \rangle$, and use TrG_x (for $x = \{s, c, t\}$) to refer to the *x* component of *TrG*. In this way, $\langle G_s, G_c, G_t \rangle_s = G_s$.

Fig. 1 shows a triple graph, taken from the class-to-relational transformation (OMG, 2007), which we use as a running example. The source graph is a class diagram with a package and a class, the target one is a relational schema model with one schema node, and the correspondence includes a mapping between the package and the schema. Note that "source" and "target" are relative terms, as we could also use source for the relational schema and target for the class diagram.

A triple graph is typed by a *meta-model triple* (Guerra and de Lara, 2007) or TGG schema, which contains the source and target meta-models and declares allowed mappings between both. Fig. 2 shows the meta-model triple for our running example. The correspondence meta-model declares five classes: `P2S` maps packages and schemas, `A2Co` maps attributes and columns, and `CT` and its specializations `C2T` and `C2TCh` relate classes and tables. In particular `C2TCh` is used to relate a children class with the table associated to its parent class. The dotted arrows specify the allowed morphisms from the correspondence to the source and target models, and can be treated as normal associations with cardinality 1 on the side of the source/target class. The meta-model includes OCL constraints ensuring uniqueness of attribute names for each class and table, as well as same persistence for a class and its children. As an example, the triple graph in Fig. 1 conforms to the meta-model in Fig. 2.

A typed triple graph is formally represented as $(TrG, type : TrG \to MM)$, where the first element is a triple graph and the second a morphism to the meta-model triple. Morphisms between typed triple graphs must respect the typing morphism and can take inheritance into account, as in Guerra and de Lara (2007). For simplicity of presentation, we omit the typing in the following definitions.

Besides a meta-model triple, a M2M transformation by TGGs consists of a set of declarative rules that describe the synchronized

[2] A morphism corresponds to the mathematical notion of total function between two sets, or in general between two structures (graphs, triple graphs, etc.).