The Journal of Systems and Software 86 (2013) 412–436

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

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

Constraint-based specification of model transformations

K. Lano*, S. Kolahdouz-Rahimi

Department of Informatics, King's College London, United Kingdom

ARTICLE INFO

Article history:
Received 4 August 2011
Received in revised form 20 June 2012
Accepted 4 September 2012
Available online 25 September 2012

Keywords:
Model transformations
Model-driven development
Software synthesis

ABSTRACT

Model transformations are a central element of model-driven development (MDD) approaches. The correctness, modularity and flexibility of model transformations is critical to their effective use in practical software development. In this paper we describe an approach for the automated derivation of correct-by-construction transformation implementations from high-level specifications. We illustrate this approach on a range of model transformation case studies of different kinds (re-expression, refinement, quality improvement and abstraction transformations) and describe ways in which transformations can be composed and evolved using this approach.

1. Introduction

Model transformations are an essential part of development approaches such as model-driven architecture (MDA) (OMG, 2004) and model-driven development (MDD) (Gerber et al., 2002). Model-transformations are becoming large and complex and business-critical systems in their own right, and so require systematic development. In particular there is a need to verify the syntactic and semantic correctness of transformations, and to support evolution required by evolution of the languages which the model transformation operates on.

At present, a large number of different model transformation approaches exist, such as graph transformations (e.g., Viatra (OptXware, 2010)), declarative (QVT-Relations (OMG, 2005)), imperative (QVT-Operations, Kermeta (Kermeta, 2010)) and hybrid (ATL (Jouault and Kurtev, 2006)) languages (Czarnecki and Helsen, 2006). These are all primarily based around the concept of *transformation rules*, which define individual steps within a transformation process. The overall effect of a transformation is then derived from the implicit (QVT-Relations, ATL) or explicit (Kermeta, QVT-Operations, Viatra) combination of individual rule applications. These descriptions are closer to the level of designs, rather than specifications, and are also specific to particular languages, i.e., they are PSMs (platform-specific models) in terms of the MDA.

However for effective verification and reuse of transformations a higher level of structuring and specification is required, to define the complete behaviour of a transformation as a (black-box) process, for example, by pre and postconditions. This level of specification also provides support for the correct external composition of transformations, such as by the sequential chaining of transformations.

In this paper we will describe the following components of a systematic approach for specifying and developing model transformations to address these problems:

- A general model-driven development process for model transformations (Appendix B) based upon a formal semantics for UML and transformations (Appendix A).
- Specification of model transformations using OCL constraints and UML class diagrams (Section 2).
- Automated design and implementation strategies for these specifications. We give proofs of correctness of these strategies (Section 3).
- Structuring and composition techniques for specifications, to enable the effective evolution of transformations in response to evolving metamodels or changes in requirements (Section 4).

To minimise development costs, we implement transformation specifications by the automated derivation of (correct by construction) designs and executable implementations. This process has been implemented as part of the UML-RSDS toolset for MDD using UML. It could also be applied to other model transformation approaches, such as QVT or ATL.

Section 5 evaluates the approach. Section 6 describes related work, and Section 7 summarises the paper.

1.1. Categories of model transformation

Model transformations can be classified in different ways (Czarnecki and Helsen, 2006; Mens et al., 2005). At a syntactic level, we can differentiate between those transformations where

* Corresponding author. Tel.: +44 02078482832; fax: +44 02078482851.

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.09.006

the source and target languages S and T are entirely disjoint, or where they overlap, or where one is a sub-language of the other. Transformations may be *update-in-place*, i.e., they operate on a single model and produce the target model by modifying elements of the source model, or *separate models* if the source and target are distinct. The latter are also termed *input-preserving* if they do not modify the source model. *Endogenous* transformations have the same source and target language, whilst *exogenous* transformations have distinct languages (Czarnecki and Helsen, 2006).

Semantically, a transformation can be classified in terms of its role in a development process:

Refinement A transformation that replaces source model elements by target elements or structures to map an abstract model (such as a PIM) to a more specific version (such as a PSM).

Abstraction A transformation that produces an abstraction of a model, e.g., by discarding some details of the source model data.

Quality improvement/Restructuring A transformation that produces a target model at the same abstraction level as the source, but that re-organises a model to achieve some quality goal (e.g., removing duplicated attributes from a class diagram). Usually these are update-in-place and endogenous.

Re-expression A transformation that maps a model in one language into its equivalent in another language at the same level of abstraction, eg, migration transformations from one version of a language to another version.

We will consider the following examples of transformations from these categories:

- The well-known UML to relational database refinement transformation (OMG, 2010; Lano and Kolahdouz-Rahimi, 2011a), and the refinement example of Kurtev et al. (2006).
- A quality improvement restructuring to remove duplicated attributes from a class diagram (Kolahdouz-Rahimi et al., submitted for publication).
- Re-expression transformations to compute the transitive closure of a graph (Lano and Kolahdouz-Rahimi, 2011b) and of an association (Section 4.3).

These will be used to illustrate the development process and transformation patterns.

2. Model transformation specification

In this section and the following section we describe how the general model transformation development process of Lano and Kolahdouz-Rahimi (2011a) can be implemented in UML-RSDS. UML-RSDS is a model-driven development approach which has the following general principles:

- Systems are specified using declarative UML models and OCL constraints, at a CIM (computationally independent model) level, where possible.
- Designs and executable implementations are automatically derived by means of verified transformations, so that they are correct-by-construction with respect to the specification. Alternatively, developers can write explicit designs at a PIM level, similar to QVT-Operations or Kermeta in style.
- Capabilities for formal analysis are provided, for use by specialised users.

As an approach to transformation specification, this means that transformations are specified purely using UML notations, with no additional specialised syntax required.

Each transformation $\tau : S \rightarrow T$ from a source language S to a target language T is defined as a UML use case (Chapter 16 of OMG, 2009), with a set *Asm* of assumptions, which are the preconditions of the use case (ie, the *precondition* of the *BehavioralFeature* associated to the use case), and a set *Cons* of postconditions. Additional conditions *Ens* for the poststate should follow from *Cons*. Invariants *Inv* may be defined for the use case. The predicates *Asm*, *Cons*, *Inv* and *Ens* may involve both languages S and T.

Logically, the transformation is interpreted as achieving the conjunction of the postconditions, under the assumption that the conjunction of the preconditions holds at its initiation. Procedurally, the postcondition constraints *Cn* can be interpreted as transformation rules or statements *stat*(*Cn*) which establish *Cn* (Appendix C).

The precondition constraints *Asm* define checks which should be carried out on the source model, whilst the postconditions *Cons* also define consistency conditions that should hold (and be maintained) between the source and target models as a result of the transformation.

The structure and organisation of the constraints will be used to automatically derive the design and implementation of the transformation.

An example of such constraint-based specification is the well-known UML to relational database transformation. This transformation maps a data model expressed in UML class diagram notation into the more restricted data modelling language of relational database schemas. Modelling aspects such as inheritance and associations are removed from the source model and their semantics expressed instead using the language facilities (tables, primary keys and foreign keys) of relational databases.

Fig. 1 shows the source (on the left hand side) and target (on the right) metamodels. *Asm* consists of assertions that attribute names are unique within a class, that class and association names are unique within a package, and so forth (Lano and Kolahdouz-Rahimi, 2011a).

The transformation is defined as a use case operating on instance models of these metamodels. The formal specification *Cons* of the transformation as a global relation between the source and target languages can be defined by six postcondition constraints of the use case, of which the following three define the basic transformation of attributes to columns and classes to tables:

C1 "For each persistent attribute in the source model there is a unique column in the target model, of corresponding type":

$$\begin{array}{l} \forall a : Attribute \cdot a.owner.kind = \text{"}Persistent\text{"} \ implies \\ \quad \exists cl : Column \cdot cl.rdbId = a.umlId \ and \\ \qquad cl.name = a.name \ and \ cl.kind = a.kind \ and \\ \qquad (a.type.name = \text{"}Integer\text{"} \ implies \ cl.type = \text{"}NUMBER\text{"}) \ and \\ \qquad (a.type.name = \text{"}Boolean\text{"} \ implies \ cl.type = \text{"}BOOLEAN\text{"}) \ and \\ \qquad (a.type.name \neq \text{"}Integer\text{"} \ and \ a.type.name \neq \text{"}Boolean\text{"} \ implies \\ \qquad\quad cl.type = \text{"}VARCHAR\text{"}) \end{array}$$

C2 "For each persistent class in the source model, there is a unique table representing the class in the target model, with columns for each owned attribute":

$$\begin{array}{l} \forall c : Class \cdot c.kind = \text{"}Persistent\text{"} \ implies \\ \quad \exists t : Table \cdot t.rdbId = c.umlId \ and \ t.name = c.name \ and \\ \qquad t.kind = \text{"}Persistent\text{"} \ and \\ \qquad Column[c.attribute.umlId] \subseteq t.column \end{array}$$