A FORMAL LANGUAGE FOR MODEL TRANSFORMATION SPECIFICATION

Dan Song, Keqing He, Peng Liang, and Wudong Liu
State Key Laboratory of Software Engineering, Wuhan University 430072, China
Email: daneysong@tom.com, hekeqing@public.wh.hb.cn

Keywords: MDA, model transformation, meta-model, formal language

Abstract: Model transformation and its automation have been the core and major challenge of MDA; consequently OMG issued a QVT RFP to standardize its process. Though many approaches have been proposed, their efficiency cannot be validated and their application scope is still limited. The task of providing a reliable solution to model transformation is critical. This paper proposes a formal language for model transformation specification to achieve automatic implementation. The foundation of our work is explained and some basic elements of the language are defined. A concrete transformation example from UML 1.4 models to UML 2.0 models is presented using the formalism.

1 INTRODUCTION

The Object Management Group (OMG)’s Model Driven Architecture (MDA) [1] initiative has gained widespread reputation due to its adoption of a model-backboned approach for software system development. The process of building a final system can be regarded as a series of model transformations. Model transformation thus becomes the focus of model driven software development [2].

A major challenge of the model-driven development is to be able to achieve the automation of model transformation. While there exist well-established modelling standards, there lacks sound model transformation mechanisms and hence prevent it from automating. In order to change this embarrassment, the OMG initiated a standardization process by issuing a Request for Proposal (RFP) on Query/Views/Transformations (QVT) in 2002 [3]. As an effort to respond to this RFP, a large number of approaches have been proposed in the past two years [4]. However, their effectiveness cannot be guaranteed in practice. Additionally, automatic tools available only offer limited capabilities and lack proper theoretical foundation.

To achieve the maximum degree of automation, we may need a precise and unambiguous modal transformation language. A formal language, which is mainly based on mathematical logic and set theory can meet this goal. Formal language allows software engineers to create integral, conformant and unambiguous specifications. Additionally, formal language is easy for automation.

In this paper, we present our effort to design such a formal language for model transformation specification. The paper is organized as follows. Section 2 explains OMG’s transformation model, which is our working foundation. Section 3 gives the basic elements of our proposed formal language. Section 4 presents a concrete example of model transformation from the UML 1.4 model to UML 2.0 model. And Section 5 summaries our solution and discuss the future work.

2 TRANSFORMATION MODEL

We introduce the OMG’s simplified QVT transformation model [5], shown in Figure 1, to explain our working rationale. As we can see from the figure, there are two types of transformation: relation and mapping [6]. Relations are non-executable and can be automatically refined into
mappings. Mappings are typically uni-directional and must be consistent with the relations it refines.

A model transformation specification defines how a target model is derived from an existing source model. Since MDA requires its model to be well-formed, there are always two meta-models, i.e., source meta-model and target meta-model, to validate their conformance. What will be transformed are the actual data models, while rule sets are defined on meta-model level. The equivalence of source model and target model are reflected in the static relations of the corresponding meta-models.

On the lower level of Figure 1, we can see clearly that the relation consists of several transformation rules, represented by TRule. TRule is usually responsible for transforming a clipping of source model into a clipping of target model. So each TRule has a source ExtentVar and a target ExtentVar. An ExtentVar is an extent variable representing a meta-model or a meta-model fragment, since a transformation rule. Similarly, mapping is composed of several PatternDefn, which is a pattern definition describing the details of transformation. Each PatternDefn specifies how a source Term is transformed into a target Term. And a Term tracks to a MOF class in a model.

3 BASIC FORMALISM

As for the expression of specifications and implementations of transformation, it may need a specific modal transformation language. This language should be precise and unambiguous. A formal language based on mathematical logic and set theory can meet this goal. Formal language allows software engineers to create integral, conformant and unambiguous specifications. Additionally, formal language is easy for automation.

According to our designed transformation model, such a formal transformation language should provide at least definitions of three aspects:
- meta-model: a set of concepts to be matched for an instantiated model,
- model: a set of entities to be dealt with by transformation, and
- transformation: a functional mapping reflecting a set of relations and evolution of elements from the source to the target.

In the following, we will present the essential formal definitions of all the elements in the transformation model in part 2.

3.1 Definitions of Meta-models

These three consecutive concepts are defined on meta-model level. To be mentioned here, all the concepts have the most common sense. For example, the attribute of class here is a general concept, including universal attribute of the primitive type, association end of Object type and method of the class, and we use a full stop to attach it to its belonging class. So a meta-model, which actually contains meta-classes and associations between them, can be simplified to be a group of meta-classes.

**Def1** A meta-class mc(id, A)
- An identifier id
- A meta-attribute set A of tuples (n, t) with an identifier n and a type t, with the property:
  \[
  \forall (n, t), (n, t) \in A: n = n \Rightarrow t = t
  \]

**Def2** A meta-model mm is a collection of meta-classes \(\{m_0, m_1, \ldots, m_n\}\), with the property:
- \(m_c, m_c, m_c \in mm\)
- \(m_{c1}.id = m_{c2}.id \Rightarrow m_{c1} = m_{c2}\)

**Def3** A meta-model fragment mmf(mm, MCC)
- A meta-model mm which it belongs to
- A set MCC is a collection of meta-classes, with the property:
  \[
  \forall mmf : mmf.MCC \subseteq mmf.mm
  \]

3.2 Definitions of Models

**Def4** to **Def6** deals with the model and its containing elements. These definitions are dependent on the above definitions since models are the instances of meta-models. So, each of the following definitions attach a tag or mark reflecting its belonging meta source.

**Def4** A class c(id, mc, V)
- An identifier id
- A regarding meta-class mc
- An attribute set V of tuples (a, v), with the property:
  \[
  \forall (a, v), (a, v) \in V: a = a \Rightarrow v = v , and
  \forall c, (a, v) \in c.V, \exists (n, t) \in mc.A:
  a = n \wedge [V] = [mc.A],
  \]
  which ensure that the class has all the attributes specified in the meta-class definition.
3.3 Definitions of Model Transformations

Model transformations consist of a series of transformation rules. Each rule embodies the relation of source and target model. And each rule is refined by a mapping. A mapping is a mathematical function, described by several pattern definitions.

**Def5** A model $m(mm, \mathbb{CC})$

- A regarding meta-model $mm$
- A set $\mathbb{CC}$ is a collection of classes $\{c_0, c_1, \ldots, c_n\}$, with the property:
  \[ \forall c_i, c_j \in \mathbb{CC} : c_i.id = c_j.id \Rightarrow c_i = c_j \]

**Def6** A model fragment $m(m, \mathbb{CC})$

- A model $m$ which it belongs to
- A subset $\mathbb{CC}$ is a collection of classes, with the property:
  \[ \forall mf : mf.\mathbb{CC}' \subseteq mf.m.\mathbb{CC} \]

3.4 Other Auxiliary Definitions

As we can see from **Def9**, model can be decomposed into fragments and also can be integrated into a whole.

**Def10** $\text{mergeable}(\mathbb{CC}_a, \mathbb{CC}_b)$ iff
\[
-\exists c_i \in \mathbb{CC}_a \land c_i \in \mathbb{CC}_b : c_i.id \neq c_i.id \land \exists (a_0, b_0) \in c_i.V : (a_0 = a_0 \land b_0 \neq b_0) \lor c_i.mc \neq c_i.mc), \text{ i.e., two class collections are mergeable only if there exists no class in both collections with an identical identifier but with contradictory attribute values or an contradictory meta-class.}

**Def11** $\text{merge}(\mathbb{CC}_a, \mathbb{CC}_b) = \mathbb{CC}'$ iff $\text{mergeable}(\mathbb{CC}_a, \mathbb{CC}_b)$. The result of merge is a class collection containing all classes in the participating collection if these are mergeable. Therein, $\mathbb{CC}'$ is the set of classes which holds:
\[
\forall c_i \in \mathbb{CC}_a : c_i.id \neq c_i.id \Rightarrow c_i = \mathbb{CC}'
\]
\[
\forall c_i \in \mathbb{CC}_b : c_i.id \neq c_i.id \Rightarrow c_i = \mathbb{CC}'
\]
\[
\forall c_i \in \mathbb{CC}_a, c_i \in \mathbb{CC}_b, c_i.id \neq c_i.id \Rightarrow c_i \in \mathbb{CC}':
\]
\[
(c_i.id = c_i.id \land c_i.mc = c_i.mc \land c_i.mc = c_i.mc)
\]
\[
(c_i.id = c_i.id \land c_i.mc = c_i.mc \land c_i.mc = c_i.mc)
\]
\[
(c_i.id = c_i.id \land c_i.mc = c_i.mc \land c_i.mc = c_i.mc)
\]
\[
(c_i.id = c_i.id \land c_i.mc = c_i.mc \land c_i.mc = c_i.mc)
\]
\[
(c_i.id = c_i.id \land c_i.mc = c_i.mc \land c_i.mc = c_i.mc)
\]
\[\text{It denotes that to merge two class collections, and all the same classes in separate collection are merged into one and all the different classes are preserved in the new collection.}\]

**Def12** $\text{mergeable}(m_{fa}, m_{fb})$ iff
\[
m_{fa}.m = m_{fb}.m \land \text{mergeable}(m_{fa}.\mathbb{CC}', m_{fb}.\mathbb{CC}') , \text{ i.e., two model fragments are mergeable only if they are attached to the same model and their class collections are mergeable.}\]

**Def13** A model fragment merging $m_{fa}, m_{fb} = m'_f$ iff $\text{mergeable}(m_{fa}, m_{fb})$ and
\[
m_{fa}.\mathbb{CC} = m_{fb}.\mathbb{CC}
\]
\[
m_{fa}.\mathbb{CC} = m_{fb}.\mathbb{CC}
\]
\[\text{It merges two model fragments } m_{fa}, m_{fb} \text{ to a new model fragment.}\]

**Def14** A transformation rule set $R(mm_0, mm_1)$ is complete when it holds:
\[
\forall i \in \{0,1,\ldots,n\}, r_i \in R : \quad r_i.mm_0.mm = mm_1.\mathbb{CC} \land \text{merge}(r_i.mm_0.mm, r_i.mm_0.mm) = mm_0
\]
\[
\forall i \in \{0,1,\ldots,n\}, r_i \in R : \quad r_i.mm_0.mm = mm_1.\mathbb{CC} \land \text{merge}(r_i.mm_0.mm, r_i.mm_0.mm) = mm_0
\]
\[
\forall i \in \{0,1,\ldots,n\}, r_i \in R : \quad r_i.mm_0.mm = mm_1.\mathbb{CC} \land \text{merge}(r_i.mm_0.mm, r_i.mm_0.mm) = mm_0
\]
\[\text{It denotes a mapping is responsible for transforming a source model to a target model. Each area beginning with TRANSFORMATION label is called a pattern definition, in which the id or certain attributes of a source class is evaluated to the id or attributes of a target class.}\]

4 MODEL TRANSFORMATION FROM UML 1.0 TO UML 2.0

4.1 The Example Meta-models

To perform a model transformation requires clear understanding of the abstract syntax and the semantics of both the source and target. Meta-model, which is used to define the abstract syntax and
4.2 Transformation Rules

Transformation rules should clearly express how the source model and target model relates with each other. But since the UML meta-model is rather complex, this relation is not so distinctive. However, it is not too difficult for us to find that the core UML meta-model can be decomposed from such four aspects as backbone, relationship, feature and association. Accordingly, four rules can be customized.

Informally speaking, the first rule $r_0$ identifies the basic constructs required for elementary modelling. Rule $r_1$, as shown in Figure 3, identifies model elements that define relationships. Rule $r_2$ identifies various kinds of features of elements. Rule $r_3$ distinguish relationship between association and attribute. And all the four rules consist of the rule set $R$. In order to express the intuitive knowledge about how the models relate, graphical means are made use of to depict transformation rules.

4.3 Formal Description

As the formal language and customized rules have been presented, the relation between the UML 1.4 meta-model and UML 2.0 meta-model can be described as $R(mm_0, mm_1)$, in which, $mm_0$ is the given source extent variable, $mm_1$ is the target extent variable and $R$ is the transformation rule set $\{r_0, r_1, r_2, r_3\}$. Rules in $R$ can be respectively expresses as $r_0(mm_{0,0}, mm_{1,0})$, $r_1(mm_{0,1}, mm_{1,1})$, $r_2(mm_{0,2}, mm_{1,2})$ and $r_3(mm_{0,3}, mm_{1,3})$. Thereof, $mm_{0}$ and $mm_{1}$ represent the source and target meta-model fragment. Next, we should refine each rule by a mapping. Take the implementation of $r_0$ for example, the concrete and precise mapping and its pattern definitions can be written as:

$$\text{MAPPING} \ (mf_{0,0}, mf_{1,0}) \ extends \ r_0 \ {\{ \ \ \}$$

$$\text{TRANSFORMATION} \ (\text{Element, Element})$$

$$\text{SRC} : \ s : mm_{0,0}::\text{Element}$$

$$\text{TAR} : \ t : mm_{1,0}::\text{Element}$$

$$\text{PRE} : \ \text{NONE}$$

$$\text{POS} : \ \text{NONE}$$

$$\text{MAP} : \ s.id \rightarrow t.id$$

$$\text{TRANSFORMATION} \ (\text{ModelElement, NamedElement})$$

$$\text{SRC} : \ s : mm_{0,0}::\text{ModelElement}$$

$$\text{TAR} : \ t : mm_{1,0}::\text{NamedElement}$$

$$\text{PRE} : \ s.parent \ instanceOf mm_{0,0}::\text{Element}$$

Figure 2: UML 1.4 Meta-model and UML 2.0 Meta-model.

Figure 3: Transformation Rule $r_0$ in Rule Set $R$. 
5 CONCLUSION

In this paper, we are engaged in the effort of designing a formal language to describe model transformations. We argue that this formal mechanism promises such advantages: 1. Expressive power: formal definitions describe precise and unambiguous relationships between the source and models. 2. Incremental transformation: a model fragment that describes an incremental change to the source can be transformed to an additive model that describes the corresponding incremental change for the target, since transformation is based on independent rules. 3. Easy for automation: transformations are specified in declarative way without involving implementing details. Development tools can easily accept and understand the formal input, hence convenient for automation of transformation.

From the analysis, we believe that this solution has general-purpose and can be applied in many other specific domains. However, for the reason of easy illustration, concepts that are not important for our solution, such as method, multiplicity, inheritance and constraint, are left out in our formal language designing. And at present, the implementing framework does not yet have tools to support it. We hope that this formal language will be further perfected and our implementing framework can get full validation in the future.

REFERENCES
