A FORMAL LANGUAGE FOR MODEL TRANSFORMATION SPECIFICATION

Dan Song, Keqing He, Peng Liang, and Wudong Liu
The State Key Laboratory of Software Engineering, Wuhan University 430072, China

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) (Joaquin Miller, Jishnu Mukerji, 2003) 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 (Shane Sendall, Wojtek Kozaczynski, 2003).

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 (Tata Consultancy Services, 2003). As an effort to respond to this RFP, a large number of approaches have been proposed in the past two years (Krzysztof Czarnecki, Simon Helsen, 2003). 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 (DSTC, IBM, CBOP, 2003), 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 (Keith Daddy et al., 2003). Relations are non-executable and can

+This research project was supported by the National Natural Science Foundation of China under Grant No. 60373086; Wuhan Science & Technique Key Project under Grant No. 20021002043; Open Foundation of SKLSE under Grant No. 03-03; the Provincial Natural Science Foundation of Hubei under Grant No. 2002ABB037; Hubei Province Key Project under Grant No. 902130819.

be automatically refined into mappings. Mappings are typically uni-directional and must be consistent with the relations it refines.

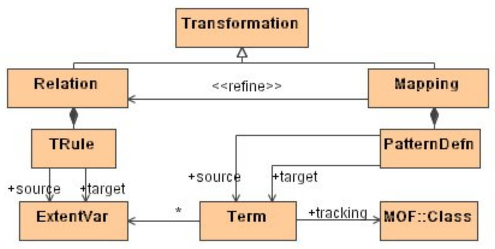

Figure 1: Transformation Model adapted from OMG.

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:
  $\forall (n_i, t_i), (n_j, t_j) \in A : n_i = n_j \Rightarrow t_i = t_j$

Def2 A meta-model mm is a collection of meta-classes {m$_0$, m$_1$,…,m$_n$}, with the property:

$\forall mc_i, mc_j \in mm :$
$mc_i.id = mc_j.id \Rightarrow mc_i = mc_j$

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:
  $\forall (a_i, v_i), (a_j, v_j) \in V : a_i = v_j \Rightarrow a_i = v_j$,and
  $\forall c, \forall (a, v) \in c.V, \exists (n, t) \in mc.A :$
  $a = n \wedge |c.V| = |mc.A|$, which ensure that the

class has all the attributes specified in the meta-class definition.

Def5 A model m(mm, CC)

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

Def6 A model fragment mf(m, CC')

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

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.

Def7 A model transformation rule r(mmf_0, mmf_1) denotes there is a certain equivalence between source model instance of mmf_0 and target model of mmf_1.

Def8 A relation R(mm_0, mm_1) is a model transformation rule set {r_0, r_1, …, r_n}, which is a finite set of model transformation rules, with the property: $\forall k \in \{0,1\}, \forall i, j \in \{0,1,\cdots n\}:$ $r_i.mmf_k.mm = r_j.mmf_k.mm$, which determines that all transformation rules of a model transformation rule set define transformations between the same two meta-models.

Def9 A mapping is a mathematical function containing a number of pattern definitions. A mapping extends a model transformation rule and can be described as follows:

```
$MAPPING(mf_0,\ mf_1)\ \ extends\ \ r$  {
$TRANSFORMATION\ \ (A,\ \ B)$
$SRC:\ s : mmf_0 :: A$
$TAR:\ t : mmf_1 :: B$
$PRE:\ --Boolean\ Expression--$
$POS:\ --Boolean\ Expression--$
$MAP:\ s.id \rightarrow t.id\ \ OR$
        $\exists\ SOME\ \ att \in s.V :$
        $FOR\ \ ALL\ \ att : s.att \rightarrow t.att$
$--other\ TRANSFORMATION\ \ pattern--$  }
```

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.

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 meageable(CC_a,CC_b) iff $\neg\exists c_i \in CC_a, c_j \in CC_b : c_i.id = c_j.id \wedge \exists(a_m, v_m) \in c_i.V, (a_n, v_n) \in c_j.V : (a_m = a_n \wedge v_m \neq v_n)) \vee c_i.mc \neq c_j.mc)$, i.e., two class collections are meageable only if there exists no class in both collections with an identical identifier but with contradictory attribute values or an contradictory meta-class.

Def11 merge(CC_a,CC_b)=CC' iff mergeable(CC_a,CC_b). The result of merge is a class collection containing all classes in the participating collection if these are mergeable. Therein, CC' is the set of classes which holds:

$$\forall c_j \in CC_a : c_i.id \notin CC_b.id \Rightarrow c_j \in CC'$$
$$\forall c_j \in CC_b : c_j.id \notin CC_a.id \Rightarrow c_j \in CC$$
$$\forall c_i \in CC_a, c_j \in CC_b, c_i.id = c_j.id, \exists c' \in CC':$$
$$(c'.id = c_i.id \wedge c'.id = c_j.id) \wedge$$
$$(c'.mc = c_i.mc \wedge c'.mc = c_j.mc) \wedge$$
$$(c'.V = c_i.V \wedge c'.mc = c_j.V)$$

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 mergeable(mf_a, mf_b) iff $mf_a.m = mf_b.m \wedge mergeable(mf_a.cc', mf_b.cc')$, 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 mfmerge(mf_a, mf_b)=mf' iff $mergeable\ (mf_a, mf_b) \wedge mf'.{CC'} = merge(mf_a.{CC'}, mf_a.{CC'})$, which merges two model fragments mf_a, mf_b to a new model fragment.

Def14 A transformation rule set R(mm_0, mm_1) is complete when it holds:

$$\forall i \in \{0,1,\cdots n), r_i \in R :$$
$$r_i.mmf_0.mm = mm_0 \wedge ri.mmf_1.mm = mm_1$$
$$\wedge\, merge(r_0.mmf_0, r_1.mmf_0, \cdots, r_n.mmf_0) = mm_0$$
$$\wedge\, merge(r_0.mmf_0, r_1.mmf_0, \cdots, r_n.mmf_0) = mm_1$$

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

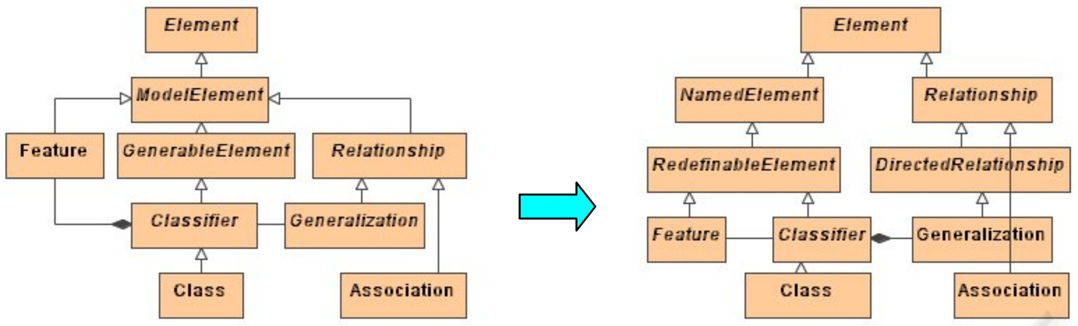

Figure 2: UML 1.4 Meta-model and UML 2.0 Meta-model.

semantics of models, serves for this purpose. The left part and right part of Figure 2 represent the core UML 1.4 meta-model (OMG, 2001) and corresponding core UML 2.0 meta-model (OMG, 2003) respectively.

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($mmf_{0,0}$, $mmf_{1,0}$), r_1($mmf_{0,1}$, $mmf_{1,1}$), r_2($mmf_{0,2}$, $mmf_{1,2}$) and r_3($mmf_{0,3}$, $mmf_{1,3}$). Thereof, mmf_0 and mmf_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:

MAPPING ($mf_{0,0}$, $mf_{1,0}$) ***extends*** r_0 {
TRANSFORMATION (Element, Element)
SRC: s: $mmf_{0,0}$::Element
TAR: t: $mmf_{1,0}$::Element
PRE: NONE
POS: NONE
MAP: s.id $\longrightarrow$ t.id
TRANSFORMATION (ModelElement, NamedElement)
SRC: s: $mmf_{0,0}$:: ModelElement
TAR: t: $mmf_{1,0}$:: NamedElement
PRE: s.parent instanceOf $mmf_{0,0}$:: Element
POS: t.parent instanceOf $mmf_{1,0}$:: Element
MAP: s.id $\longrightarrow$ t.id s.name $\longrightarrow$ t.name
……
}

5 CONCLUSION

In this paper, we are engaged in the effort of designing a formal language to describe model

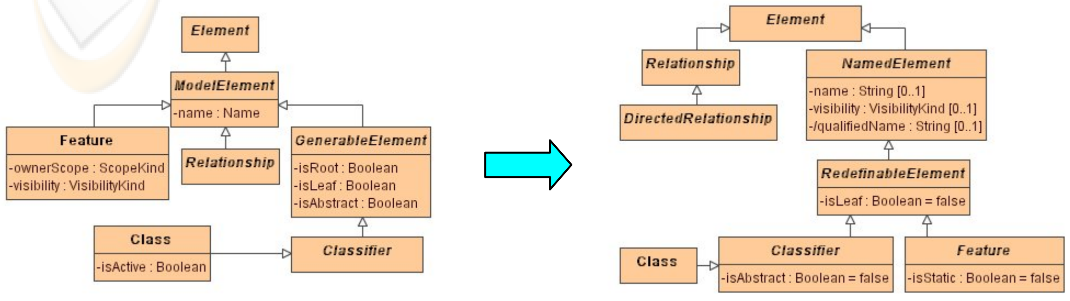

Figure 3: Transformation Rule r_0 in Rule Set R.

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

Joaquin Miller, Jishnu Mukerji, 2003. *MDA Guide*, OMG. U.S.A., Version 1.0.1.

Shane Sendall, Wojtek Kozaczynski, 2003. Model Transformation - the heart and soul of model-driven software development. *IEEE Software, Special Issue on Model Driven Software Development*, Vol.20, No. 5

Tata Consultancy Services, 2003. *Revised submission for MOF 2.0 Query / Views / Transformations RFP*, QVT-Partners. Version 1.1.

Krzysztof Czarnecki, Simon Helsen, 2003. Classification of Model Transformation Approaches. In *OOPSLA'03, Workshop on Generative Techniques in the Context of Model-Driven Architecture*.

DSTC, IBM, CBOP, 2003. *MOF Query / Views / Transformations*, QVT-Partners. 1st Revised Submission.

Keith Daddy, Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel, 2003. Model Transformation: A declarative, reusable patterns approach. In *EDOC'03, Seventh International Enterprise Distributed Object Computing Conference*.

OMG, 2001. *Unified Modelling Language Specification*, Version 1.4.

OMG, 2003. *UML 2.0 Superstructure Specification*, Version 2.0.