Model Transformation Semantic Analysis by Transformation

K. Lano[1], S. Kolahdouz-Rahimi[2], S. Yassipour-Tehrani[1]

[1]Dept of Informatics, King's College London, Strand, London, UK; [2]Dept of Software Engineering, University of Isfahan, Isfahan, Iran

Abstract. In this paper we show how translation from QVT-R to an intermediate semantic representation can be used to support analysis of QVT-R specifications. We use the UML to RDB case study to illustrate the approach.

1 Introduction

Verification of model transformations has been hindered by the large number of different MT languages which are currently in use, such as ATL [6], QVT-R [15], QVT-O, ETL [8], Flock [16] and UML-RSDS [10]. This results in a multiplication and duplication of verification effort and the need to construct separate analysis tools for each different language. In [13] we proposed a general transformation framework using a language-independent transformation metamodel $\mathcal{TMM}$ to express transformation semantics. In [11] we applied these techniques to analysis of standard mode ATL. Subsequently the approach has been applied to ATL refining mode, and to the Epsilon languages ETL [8] and Flock [16]. In this paper we apply the approach to QVT-R. QVT-R presents particular difficulties for semantic analysis because of its multiple execution modes, including aspects such as implicit deletion, change propagation and in-place execution [14]. Various proposals have been made for QVT-R semantics [3, 5, 17], however deficiencies and ambiguities remain in the language semantics. Via the translation to $\mathcal{TMM}$ we provide a logical semantics for QVT-R with a direct computational model that supports proof of transformation properties including confluence and syntactic correctness.

Figure 1 (extended from [11]) shows the metamodel $\mathcal{TMM}$ of our semantic representation for transformations. *Property*, *Operation*, *Constraint* and *Activity* are from the metamodel of the UML-RSDS subset of UML. A *TransformationSpecification* is similar to a UML Use Case: it is an owner of structural and behavioural features, and it has an associated behavior, with parameters, preconditions and postconditions. In our experience this is an appropriate entity type for the representation of transformations in declarative and hybrid MT languages.

Mappings from ATL, ETL, Flock and QVT-R to $\mathcal{TMM}$ have been implemented as transformations from the MT language metamodels to $\mathcal{TMM}$ using the UML-RSDS toolset [10]. Syntactic and data-dependency analysis can be performed on the $\mathcal{TMM}$ representation to check definedness and confluence.

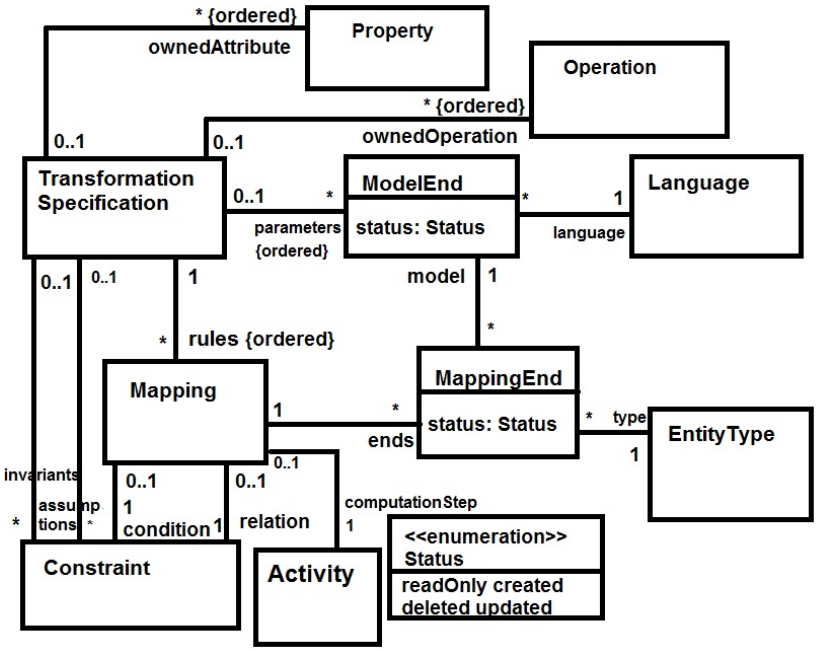

Fig. 1. Transformation specification metamodel $\mathcal{TMM}$

Mappings to B, Z3 and SMV from $\mathcal{TMM}$ have also been automated in the UML-RSDS tools. This approach means that only one semantic mapping needs to be defined and verified for each formal analysis language, rather than semantic maps for each different MT and target formalism.

Rule inheritance and transformation import/superposition relationships are not present in $\mathcal{TMM}$, instead such MT composition mechanisms are semantically expressed via the translations to $\mathcal{TMM}$. The reason for this is that the semantics of these mechanisms varies between MT languages [9].

In addition to enabling verification, the mappings can help to clarify semantic issues in the MT languages (such as the ambiguities of in-place semantics in QVT-R), and identify possibilities to extend the MT languages in semantically consistent ways, eg., to extend ATL with direct update-in-place capabilities, to extend ETL with multiple source parameter rules, or to extend MT languages with explicit specifications of transformation invariant predicates.

2 Analysis of QVT-R

QVT is an OMG standard language for model transformation [15], consisting of relational (QVT-R), operational (QVT-O) and core (QVT-Core) languages. In this section we describe the semantic mapping from QVT-R to $\mathcal{TMM}$. This

mapping provides a set-theoretic formal semantics of QVT-R, and an execution semantics based on [15], and it also takes account of the semantic issues identified in [4, 17]. All of QVT-R is covered except for collection templates and rule overriding (the semantics of overriding is not defined by the QVT-R standard: http://www.omg.org/issues/issue15524.txt). QVT-R supports bidirectionality and change-propagation for transformations. These aspects need therefore to be modelled in the $\mathcal{TMM}$ representation. We do this by deriving cleanup $\tau^{\times}$, and change propagation τ^{Δ} versions of a given transformation specification $\tau : \mathit{TransformationSpecification}$.

An example transformation, from UML to relational databases, is given in [15]:

```
transformation tau(uml1 : SimpleUML, rdb1 : SimpleRDMS) {
  top relation Class2Table
  { checkonly domain uml1 c : Class {name = n};
    enforce domain rdb1 t : Table {name = n};
    where { Attribute2Column(c,t) }
  }

  relation Attribute2Column
  { checkonly domain uml1 c : Class { attribute =
        a : Attribute {name = an, type = typ:Type { name = tn }}};
    enforce domain rdb1 t : Table { column =
        col : Column {name = an, coltype = tn} };
  }
}
```

The *name* attributes of *Class* and *Table* are identities/keys, and *attribute* and *column* are set-valued roles.

2.1 Semantic mapping from QVT-R to $\mathcal{TMM}$

Table 1 expresses the correspondence between QVT-R and $\mathcal{TMM}$ language elements. We assume that there are no cycles in the dependencies between relations via when/where clauses.

A computation step of a QVT-R transformation consists of the application of a top relation to specific model elements matching its source domains. The application of non-top relations invoked directly or indirectly from this top relation application are included in the computation step. For simplicity, in the following we assume that a QVT-R transformation is executed in the direction of the *enforce* domains as targets, with the *checkonly* domains as sources. Other execution variations, including checkonly mode, can be modelled in a similar way. We will use the QVT-R metamodel classes such as *TemplateExp* and *Relation* from [15] to describe the mapping. A full formal definition of the mapping can be found in [10], and it is implemented in the UML-RSDS toolset.

An *ObjectTemplateExp ote* of the form

```
e : E { f1 = val1, ..., fn = valn }
```

QVT-R element	$\mathcal{TMM}$ *representation*
RelationalTransformation	TransformationSpecification
modelParameter : Sequence(TypedModel)	parameters : Sequence(ModelEnd)
Relation (isTopLevel = true)	Mapping
Relation (isTopLevel = false)	Owned operation of TransformationSpecification
Relation variable	Variable defined in Mapping condition
RelationDomain (isEnforceable = false)	Mapping condition and readOnly MappingEnd
RelationDomain (isEnforceable = true)	Mapping relation and updated MappingEnd
Check-before-enforce semantics	Unique instantiation semantics [12]
Key	Identity attribute

Table 1. Correspondence of QVT-R and $\mathcal{TMM}$

occurring in a *checkonly* domain is interpreted as a Mapping condition predicate *cpred*(*ote*) formed as a conjunction of the interpretations of each *PropertyTemplateItem* *fi* = *vali*. For enforce domains, the interpretation of *ote* as a Mapping relation predicate is *epred*(*ote*) [10]. The $E{\rightarrow}exists(e \mid P)$ quantifier in Mapping relations is interpreted as "create a new $e : E$ and establish P for e, unless there already exists an $e : E$ satisfying P". The Unique Instantiation pattern of [12] is used to implement QVT-R 'check before enforce' semantics [15]. If E has a key attribute, and an E instance already exists with the key value specified in P, then that instance is updated to satisfy the remainder of P.

The interpretation of a QVT top-level *Relation* r with multiple (at least one) checkonly domains, and multiple (at least one) enforce domains is then a Mapping Con_r which has *readOnly* ends for each checkonly domain of r, and *updated* ends for each enforce domain of r. Con_r has condition the conjunction of predicates $e : E$ & *cpred* for each checkonly domain

```
checkonly domain src e : E { f1 = val1, ..., fn = valn }
```

and relation the conjunction of predicates *epred* for each enforce domain

```
enforce domain trg e : E { f1 = val1, ..., fn = valn }
```

In addition, the *when* clause is interpreted as a predicate *whenp* and conjoined to the Mapping condition, and the *where* clause is interpreted as a predicate *wherep* and conjoined to the relation. An implicit *where* predicate, always incorporated into the relation of Con_r for separate-models model transformations, is the creation of a trace object for this rule application:

$$r\$trace{\rightarrow}exists(rx \mid rx.e1 = e1 \ \& \ ... \ \& \ rx.en = en)$$

where the *ei* are all the root variables of the relation domains of r. $r\$trace$ is an auxiliary entity which has attributes consisting of these variables[1]. A *RelationCallExp* of $r(e1, ..., en)$ in a *when* clause is also interpreted by the same

[1] "Relations ... implicitly creates trace instances to record what occurred during a transformation execution" (Page 9 of [15])

predicate, as a condition query expression. The main use of these trace relations is to control execution order by means of *when* conditions [7], [4].

The complete interpretation of $Class2Table$ as a Mapping $Con_{Class2Table}$ is:

$$c : Class \ \& \ n = c.name \ \Rightarrow$$
$$Table \rightarrow exists(t \mid t.name = n \ \& \ attribute2column\$op(c, t) \ \&$$
$$Class2Table\$trace \rightarrow exists(class2table \mid$$
$$class2table.c = c \ \& \ class2table.t = t))$$

where the condition is written on the LHS of $\Rightarrow$, and the relation on the RHS.

A QVT-R relational transformation τ is interpreted as a TransformationSpecification which has rules Con_r for each top-level *Relation* r of τ. We treat the order of these rules as significant. For non-top-level relations r, a predicate Op_r is used to form the postcondition of an update operation op_r which has parameters the root variables of the relation domains of r. Op_r is formed as for Con_r, but without antecedent conjuncts $e : E$ for the root variables of the checkonly relation domains, and without quantifiers $E \rightarrow exists(e \mid ...)$ for the root variables of the enforce relation domains of r. It is assumed that all the operation parameters are bound at the point of call. These operations are added as owned operations of the TransformationSpecification representing the transformation.

QVT-R has the semantic feature of *implicit deletion*: if a relation is not satisfiable for an existing element of the target model, which matches the target domain pattern of the relation, then this target element should be deleted. We express this semantics by deriving transformation *invariants* from the rules, which are constraints which are expected to hold at the start and throughout the transformation execution. The invariants express that all target model elements can be derived from source model elements via at least one top-level relation and some possible execution[2]. If an invariant fails for a given target enforce domain element, then the target element should be deleted (Page 21 of [15]). The invariants are derived as duals of the forward top-level rules: an invariant constraint Inv_r is derived from a top relation r by exchanging the roles of *checkonly* and *enforce* domains in the derivation process for Con_r. Both *where* and *when* clause predicates are placed in the succedent of Inv_r, and trace constructions/tests are omitted[3]. For the above example $Class2Table$ this produces the Mapping

$$t : Table \ \& \ n = t.name \ \Rightarrow$$
$$Class \rightarrow exists(c \mid c.name = n \ \& \ attribute2column\$inv(c, t))$$

If the condition P: $n = t.name$ of this mapping holds for a $t : Table$ in the target model, but the relation (succedent) Q does not hold, then t should be deleted. This semantics is expressed by a Mapping with condition $P \ \& \ not(Q)$ and relation $t \rightarrow isDeleted()$. Each of the enforced domain root variables involved in

[2] These consistency properties are logical conditions, independent of traces/execution orders.

[3] For the *when* clause a test on a top relation $r1$ is interpreted as Inv_{r1}

the QVT-R relation is deleted, and the tuple of domain variables is also removed from the relation trace. This deletion form of Inv_r is denoted $CleanTarget_r$. For non-top relations r, a boolean-valued query operation $r\$inv$ is produced whose postcondition is based on the dual of Op_r. The default value of such a query operation is *false*. The invocation of r in either a *where* or *when* clause is interpreted as a call of $r\$inv$. For QVT-T transformations t with separate source and target models, a cleanup version $\tau^{\times}$ of its $\mathcal{TMM}$ representation τ is formed with its rules consisting of the $CleanTarget_r$ for the top relations r of t. The complete semantics of t is then the sequential composition $\tau^{\times}; \tau$. Similarly, a change-propagation version τ^{Δ} of τ can be defined which uses the trace relations to propagate incremental source model changes to the target model, including deletion of target elements derived from deleted source elements [10].

2.2 Analysis techniques

After translation to $\mathcal{TMM}$, analyses of determinacy, confluence and termination of a QVT-R transformation based on syntax and data-dependency can be carried out, using the UML-RSDS tools. For the above specification, *Class2Table* is identified as confluent and of type 1, meaning that it can be correctly implemented by a bounded loop, instead of by a fixed-point iteration [13]. If the called relation was changed to:

```
relation Attribute2Column
{ checkonly domain uml1 c : Class { name = n,  attribute =
      a : Attribute { name = an, type = typ : Type { name = tn }};
  enforce domain rdb1 t : Table { name = n, column = col : Column
       { name = an + t.column->size(), coltype = tn }};
}
```

then the translated constraint would be identified as being of type 2 (it both reads and writes *Table* :: *column*), and a warning would be issued to the developer that it may be semantically incorrect: mapping transformations should normally consist of type 1 constraints only. Required postconditions can be proved as transformation invariants, using B [13] or can be derived from the transformation invariants. For example, the syntactic correctness property that the names of columns of tables are unique within the table can be established from $Inv_{Class2Table}$ and the corresponding property for attributes of classes. Model-level semantic preservation can also be proved by invariant-based reasoning. The semantics can also be used to justify refactorings of QVT-R transformation specifications by showing that the semantics of the original and refactored transformations are the same. If all the rules are type 1 and satisfy the condition that rule applications never over-write data read by rule applications preceding them in execution order, then semantic correctness of the transformation can be deduced.

We have evaluated our semantics on a number of examples, in particular the checkonly examples of [17]. Our semantics for a checkonly top relation r in the direction of domain d is Inv_r evaluated with d as the enforce domain

(existentially quantified). This agrees with the semantics of [17]. To ensure the correctness of the semantics for update transformations, the translation clauses have been defined based very closely on the (informal) semantics presented in [15]. Update examples such as the UML to RDB example have been used to check that our semantics agrees with that given in [15]. One major difference between our semantics and the Medini QVT-R semantics is that the OCL semantics used in $\mathcal{TMM}$ uses classical logic. Thus specifications which use explicit null and invalid values cannot be given a semantics.

2.3 In-place transformations

The semantics of in-place transformations is described very briefly in [15] and it is unclear how this semantics interacts with aspects such as implicit deletion and change propagation. The translation to $\mathcal{TMM}$ described above can be used to give a semantics for in-place QVT-R specifications, but with a fixed-point execution model: the computationStep of a Mapping is applied repeatedly until the Mapping relation is satisfied for all model elements that satisfy its condition. Relation traces may not be functional. Implicit deletion cannot be expressed by invariants, instead 'default deletion' rules are defined, which express that if no rule creates or copies an element, then the element is deleted. Change-propagation may be implemented by re-execution of the transformation on the modified model. An example of translation and analysis of an in-place QVT-R transformation is given in [10].

3 Related work

Some other works consider the analysis of multiple transformation languages by transformation [2, 3]. We explicitly represent the semantics of MT languages within our language-independent representation, such semantic representation is not detailed in other approaches such as [2]. Compared to [3, 14, 17] we include an execution semantics for QVT-R, with explicit representation of transformation steps, trace relations and of implicit deletion semantics. This is necessary for proof of syntactic correctness and model-level semantic preservation using transformation invariants. This representation is also necessary to express the semantics of in-place transformations. In our approach, confluence and termination may be proved for QVT-R specifications based on syntactic and data-dependency analysis of the semantic representation, rather than by using state-space exploration as in [5]. Stevens [17] identifies the deficiencies of existing QVT-R semantics and tools, and provides a game-theoretic semantics for checkonly transformations. Stevens prefers the declarative interpretation of *where*/*when* clauses for both checkonly and enforce mode. However the operational interpretation is used in practice for enforce mode, eg., in [15], [7], [4]. We adopt the declarative interpretation for checkonly mode. In terms of the classifications of [1], we address the verification properties of termination, determinism, conformance and semantic preservation, using transformation-dependent input-independent verification techniques.

4 Conclusions

We have described techniques which provide language-independent verification capabilities for model transformations via translations to a common semantic representation. Translations to $\mathcal{TMM}$ have also been implemented for ATL, Flock and ETL [10], these semantic interpretations highlight the semantic commonalities and differences between MT languages, and identify where problematic areas exist in their semantics. Our approach can also be extended in principle to other MT languages, such as TGG, GrGen.NET and QVT-O.

References

1. M. Amrani, B. Combemale, L. Lucio, G. Selim, J. Dingel, Y. Le Traon, H. Vangheluwe, J. Cordy, *Formal verification techniques for model transformations: a tridimensional classification*, JOT, 2011.
2. V. Bollati, J. Vara, A. Jimenez, E. Marcos, *Applying MDE to the (semi-)automatic development of model transformations*, Information and Software Technology, 2013.
3. J. Cabot, R. Clariso, E. Guerra, J. De Lara, *Verification and Validation of Declarative Model-to-Model Transformations Through Invariants*, Journal of Systems and Software, 2010.
4. T. Goldschmidt, G. Wachsmuth, *Refinement transformation support for QVT relational transformations*, FZI, Karlsruhe, Germany, 2011.
5. E. Guerra, J. de Lara, *Colouring: execution, debug and analysis of QVT-Relations transformations through coloured Petri Nets*, SoSyM vol. 13, no. 4, Oct 2014.
6. Eclipsepedia, *ATL User Guide*, http://wiki.eclipse.org/ATL/ User_Guide_-_The_ATL_Language, 2014.
7. J. Kiegeland, H. Eichler, *Medini-QVT*, http://projects.ikv.de/qvt, 2014.
8. D. Kolovos, R. Paige, F. Polack, *The Epsilon Transformation Language*, in ICMT 2008, LNCS Vol. 5063, pp. 46–60, Springer-Verlag, 2008.
9. A. Kusel, J. Schonbock, M. Wimmer, G. Kappel, W. Retchitzegger, W. Schwinger, *Reuse in model-to-model transformation languages: are we there yet?*, Sosym vol. 14, no. 2, 2015.
10. K. Lano, *The UML-RSDS Manual*, www.dcs.kcl.ac.uk/staff/kcl/uml2web/umlrsds.pdf, 2014.
11. K. Lano, S. Kolahdouz-Rahimi, T. Clark, *Language-independent Model Transformation Verification*, VOLT 2014, York, July 2014.
12. K. Lano, S. Kolahdouz-Rahimi, *Model Transformation Design Patterns*, IEEE Transactions in Software Engineering, 2014.
13. K. Lano, S. Kolahdouz-Rahimi, T. Clark, *A framework for model transformation verification*, BCS FACS journal, 2014.
14. N. Macedo, A. Cunha, *Implementing QVT-R bidirectional model transformations using Alloy*, FASE 2013.
15. OMG, *MOF 2.0 Query/View/Transformation Specification v1.1*, 2011.
16. L. Rose, D. Kolovos, R. Paige, F. Polack, *Model migration with Epsilon Flock*, International Conference on Model Transformations 2010, Springer-Verlag.
17. P. Stevens, *A simple game-theoretic approach to checkonly QVT-Relations*, Softw. Syst. Model vol. 12, no. 1, 2013, pp. 175–199.