IMPLEMENTING QVT IN A DOMAIN-SPECIFIC MODELING FRAMEWORK

István Madari, Márk Asztalos, Tamás Mészáros, László Lengyel and Hassan Charaf
Department of Automation and Applied Informatics, Budapest University of Technology and Economics
Magyar Tudósok körútja 2, Budapest, Hungary

Keywords: Bidirectional Model Transformations, Model-transformation Driven Synchronization, QVT.

Abstract: Meta Object Facility 2.0 Query/Views/Transformation (QVT) is OMG's standard for specifying model transformations, views and queries. In this paper we deal with the QVT Relations language, which is a declarative specification of model transformation between two models. The QVT Relations language specifies several great features in practice, such as implicit trace creation support, or bidirectional transformations. However, QVT lacks implementation because its specification is not final and far too complex. The main contribution of this paper is to show how we integrated QVT constructs in our domain-specific modeling environment to facilitate a later implementation of QVT Relations-driven bidirectional model transformation.

1 INTRODUCTION

Model transformation is an essential area of model-based development, applied for code generation, analysis, verification and simulation tasks. Model transformation can be implemented in different ways. In order to define a transformation, for instance, visual languages or textual languages can both be used. Languages can be declarative or imperative as well. In addition, the model transformation can be unidirectional or n-directional. (Czarneczki, 2003) provides further details on classifying model transformations.

In contrast with unidirectional approaches, n-directional transformations can be executed in multiple directions. However, with unidirectional transformations, multiple directions can be implemented as well (if each direction is assigned to a unidirectional transformation). Although, this solution has almost the same implementation challenges like n-directional approaches. Usually the reverse direction cannot be specified in conjunction with the original transformation, so that several transformation paths may exist between the given artifacts. Moreover, in many cases no applicable reverse direction exists. Thus, there is no clear, always applicable method for defining the reverse direction.

A special case of n-directional transformations is bidirectional transformation, where the transformation can be executed in two ways. In particular, the transformation language can define both the forward and backward direction in the transformation rules.

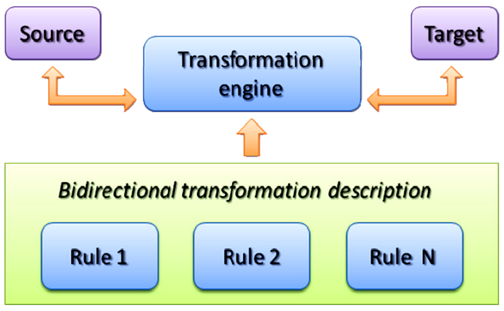

Figure 1: Bidirectional execution of model transformation.

From this definition, the transformation engine can execute the rules in each direction. Figure 1 shows an example for a bidirectional transformation: a source and a target model are depicted, and between the two models there is a transformation description, which can be executed both in the forward and the backward direction.

Beyond two-way execution, bidirectional transformation can be used in implementing incremental model synchronization solutions. In our approach we also use bidirectional transformations to implement our incremental model synchronization algorithm (Madari, 2009).

In order to implement bidirectional model transformations we have to express bidirectional rules in our modeling framework. Our approach utilizes QVT (Query/Views/Transformations) Relations language (Bast et al., 2005) to define bidirectional model transformations. QVT Relations is a declarative transformation language that can express bidirectional model transformation relations with their left-hand side (LHS) and right-hand side (RHS) structures. However, due to the complexity of the whole standard, QVT lacks implementation.

Our approach does not provide a completely new transformation engine. We use our unidirectional transformation mechanism to support the QVT Relation in such a way that we generate the corresponding forward and backward transformations with the appropriate control structures from the QVT description. Figure 2 depicts the generation process from a QVT transformation.

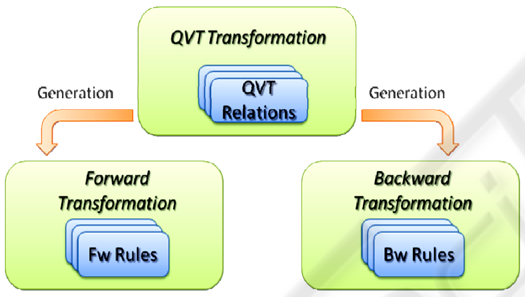

Figure 2: Generation of forward and backward transformations from QVT transformation.

Current paper discusses how we have implemented the QVT Relation language in our graph rewriting-based model transformation system (Visual Modeling and Transformation System, VMTS (VMTSSite)).

The rest of this paper is organized as follows: Section 2 provides background information including our modeling framework (VMTS). Furthermore, Section 2 gives an overview of OMG’s QVT language and triple graph grammars. Section 3 presents how QVT constructs are implemented within VMTS. Finally, Section 4 concludes the paper.

2 BACKGROUND AND RELATED WORK

Visual Modeling and Transformation System (VMTS) (Angyal et al., 2009) is a graph-based metamodeling system. The concept of metamodeling means that we can create models not only in predefined modeling languages, but also we can create new modeling languages as well. Models and transformation rules are formalized as directed, labeled graphs, which consist of individual attributed nodes and edges VMTS is a model transformation system, which transforms models by executing graph rewriting. VMTS is very flexible due to its plug-in architecture. Users can easily define new domains; as well as the graphical representation of instance models.

A formal description of bidirectional transformations can be given with triple graph grammars. Triple graph grammars (TGGs) were introduced in 1994 (Schürr, 1994). Triple graph grammar rules model the transformations of three separate graphs: source, target and correspondence graphs.

QVT (Query/Views/Transformations) is the OMG (Object Management Group) standard for the transformation of MOF (Meta-Object Facility) models. QVT defines a standard way to transform source models into target models. Both QVT and TGGs declaratively define the relation between two models. With this definition of relation, a transformation engine can execute a transformation in both directions and based on the same definition, can also propagate changes from one model to the other.

OMG published simplified *Unified Modeling Language* (UML) and *Relational Database Management System* (RDBMS) metamodels in the appendix of the QVT standard. The UML and RDBMS metamodels created in VMTS as well, to present the feasibility of our approach.

QVT relations can be extended with *Where* and *When* clauses. Both two clauses define conditions which need to hold in order to apply the current QVT relation. The difference between the two clauses is that *When* specifies which conditions are needed to hold before applying the current relation (i.e. it is like a pre-condition of the current relation), while *Where* defines the conditions that need to hold after apply the QVT relation (i.e. it is like a post-condition). The conditions can refer to other QVT relations, to an OCL expression or a custom function.

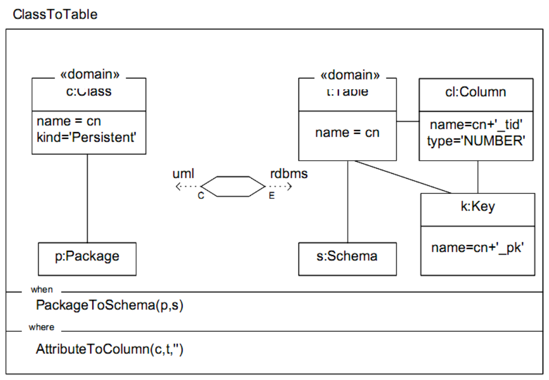

Figure 3: QVT relation with *When* and *Where* clauses.

In Figure 3 a QVT relation is depicted with *When* and *Where* clauses. As a matter of fact, the current relation in Figure 7 can be held if and only if the *PackageToSchema* relation has been held, and the *AttributeToColumn* relation can also be held.

3 REALIZATION OF QVT RELATION

In this section we give a detailed overview of how we implemented the QVT Relations in VMTS modelling framework.

In VMTS, new domain-specific languages can be easily created. QVT transformation is also a domain itself, thus the first step was to define the corresponding metamodels in VMTS. To realize the QVT Relation two metamodels had to be created: one for the QVT Relation and one for a composite domain that represents the whole QVT transformation. The QVT transformation consists of QVT relations, functions, input and output model definitions. The metamodel of a QVT transformation is depicted in Figure 4.

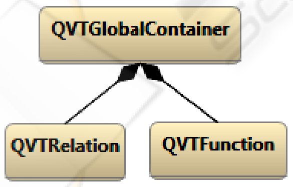

Figure 4: QVT transformation metamodel.

The *QVTGlobalContainer* element contains QVT relations and QVT functions (*QVTRelation* and *QVTFunction* in Figure 4). It is a high-level container, which also sets the transformation properties (such as the current input and output models). The QVT Relation domain is more complicated: it contains more nodes, attributes, and relationships. Figure 5 depicts the metamodel of the QVT relation. *QVTRelationContainer* is the topmost element of the QVT relation metamodel. Its purpose is to wrap every node of a QVT relation. The *QVTRelationContainer* has only one attribute: *RelationName*. A QVT relation contains only one *QVTRelationContainer* in each instance model.

QVTRelationContainer can contain *QVTElements* elements. It is a general model element because we never use it directly (i.e. it is not necessary to drop it into the diagram). In fact, nodes to be contained by the *QVTRelationContainer* are inherited from *QVTElement* thus the inherited nodes can also be contained by *QVTRelationContainer*.

QVTRegionParent is also a general node in the metamodel, which means that it is not used directly in the instance models. However, two elements inherit from QVT*RegionParent*: *QVTLHSRegion* and *QVTRHSRegion*. The regions are the LHS and RHS of QVT relations. The model elements (the regions) contain the relation nodes that describe the pattern to be checked or enforced during the transformation. Both of the regions have to belong to a domain, which determines the possible nodes that can be used in the regions. In other words, the LHS and the RHS can contain nodes only from the selected domains.

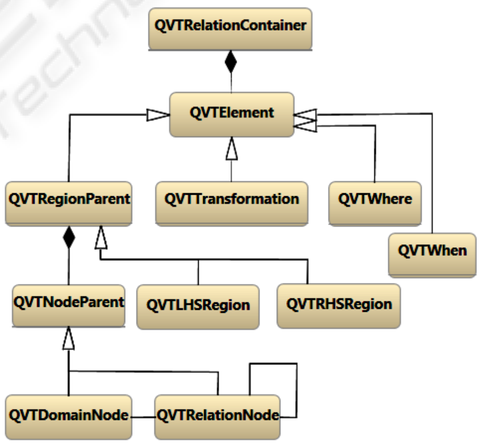

Figure 5: QVT relation with *When* and *Where* clauses.

QVTWhen and *QVTWhere* nodes contain clauses of the current relation. They can contain string type values such as OCL expressions, unique function names or QVT relation names. The values are stored in the *Expression* attribute of the *QVTWhere* and *QVTWhen* elements.

Nodes in regions are not yet shown. Two types of nodes can be distinguished in the LHS and RHS regions: *QVTDomainNode* and *QVTRelationNode*. To understand what the difference is between the

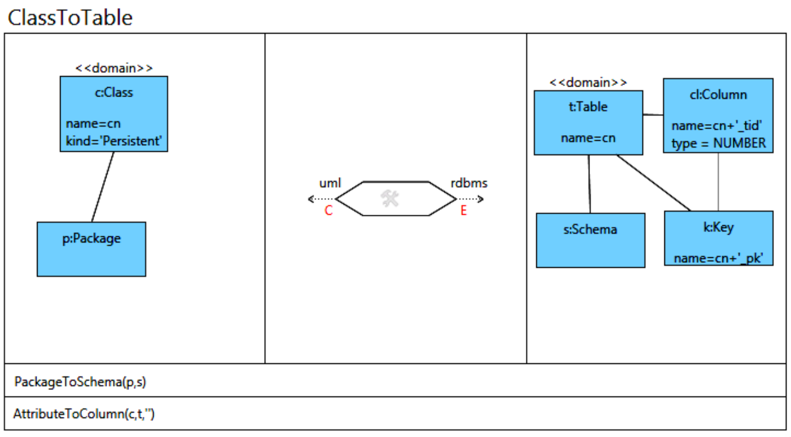

Figure 6: Relation *ClassToTable* in VMTS.

two types of nodes, see the relation in Figure 3. The *ClassToTable* relation defines nodes in the LHS and RHS patterns labeled with the "<<domain>>" string. This type of node in our metamodel is represented by the *QVTDomainNode* element. *QVTRelationNode* stands for the nodes without the "<<domain>>" identifier. Figure 6 shows a created QVT relation in VMTS.

4 CONCLUSIONS

In this paper we have briefly described the QVT Relation language with examples, as well as the VMTS modeling framework. We have discussed how the QVT Relation language has been implemented in our domain-specific modeling environment. Furthermore, it has been explained how to create the necessary metamodels, and implementation details of the concrete syntax have been given. With the presented approach we can define QVT relations and transformations in VMTS.

Our future research targets three important fields:

(i) Generating forward and backward VMTS transformations from the QVT relations instead of executing the QVT transformation directly is a major objective. The advantage of this approach is that developers can modify the backward and the forward directions independently.

(ii) The forward and backward transformations cannot necessarily be generated automatically from the QVT description. We have to analyze the QVT transformation properties to determine under which circumstances can the forward and backward transformations generated.

(iii) Formal validation of the synchronization and checking process in the generated unidirectional transformations is also a direction for future work.

ACKNOWLEDGEMENTS

This paper was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

REFERENCES

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

Schürr, A., 1994. Specifcation of graph translators with triple graph grammars. In *WG'94, Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in ComputerScience.*

Asztalos, M., Madari, I., 2009. Vtl: An improved model transformation language. In *AACS'09, Proceedings of Automation and Applied Computer Science Workshop.*

Madari, I., Angyal, L., Lengyel, L., 2009. Traceability-based Incremental Model Synchronization. In *WSEAS'09, WSEAS Transactions on Computers.*

Angyal, L., Asztalos, M., Lengyel, L., Levendovszky, T., Madari, I., Mezei, G., Mészáros, T., Siroki, L., and Vajk, T., 2009. Towards a fast, efficient and customizable domain-specific modeling framework. In *IASTED'09, In Proceedings of the IASTED International Conference.*

Bast, W., Belaunde, M., Blanc, X., Duddy, K., Griffin, C., Helsen, S., Lawley, M., Murphree, M., Reddy, S., Sendall, S., Steel, J., Tratt, L., Venkatesh, R., Vojtisek, D., 2005. *MOF QVT Final Adopted Specification.*

VMTSSite, Visual Modeling and Transformation System, *http://vmts.aut.bme.hu/*