Round-trip Engineering UML Class Models and Java Models: A Real-world Use Case for Bidirectional Transformations with QVT-R

Sandra Greiner, University of Bayreuth, Bayreuth, Germany
Thomas Buchmann, University of Bayreuth, Bayreuth, Germany

ABSTRACT

Model transformations constitute the key technology for model-driven software development, a software engineering discipline which became more and more important during the last decade. While tool support for unidirectional batch transformations is rather mature, bidirectional and incremental transformations are only weakly investigated. Nevertheless, several usage scenarios demand for incremental and bidirectional transformations, like round-trip engineering between UML class models and Java source code. This paper presents a bidirectional transformation between UML class models and a Java model which is obtained from Java source code. The transformation is written in QVT Relations, a declarative model transformation language provided by the OMG. While the case study demonstrates that it is possible to specify bidirectional transformations between heterogeneous metamodels in a single relational specification, it also reveals some inherent limitations of the language and the corresponding tool support.

KEYWORDS

Bidirectional Transformations, Java, Model-Driven Engineering, Model Transformations, QVT-R, Round-trip Engineering, Software Engineering, UML

1. INTRODUCTION

The goal of model-driven software development (MDSD) (Schmidt, 2006) is to increase the productivity of software engineers by replacing low-level programming with the development of high-level models. Over the years, the Unified Modeling Language (UML) (Object Management Group [OMG], 2015a) has been established as the standard modeling language for MDSD. The model-driven architecture (MDA) (Mellor, Kendall, Uhl, & Weise, 2004) process starts with an initial model containing the requirements and uses a series of subsequent models being refined over multiple levels of abstraction until the system is eventually implemented. For a full-fledged support of MDSD, model transformations are required as a key enabling technology.

A wide variety of languages and tools for model transformations has been developed (Czarnecki & Helsen, 2006). Available languages differ with respect to underlying computational paradigms (procedural, functional, rule-based, object-oriented), directionality (unidirectional, bidirectional), granularity (batch, incremental) and the source/target model relationship (in-place, out-place).

The QVT (Query-View-Transformation) standard (OMG, 2015b) issued by the OMG, defines a family of model transformation languages residing at different levels of abstraction. QVT Relations (QVT-R) is the most high-level language providing means for a declarative specification of transformations between MOF (OMG, 2008) based models. Furthermore, it reuses the Object
Constraint Language (OCL) (OMG, 2012), an expression language for MOF models. A wide spectrum of model transformation scenarios is addressed by QVT-R, including enforcing and checking, unidirectional and bidirectional, batch and incremental, and n:1 transformations.

This article, which is a revised version of a conference paper (Greiner, Buchmann, & Westfechtel, 2016), presents a case study dealing with incremental round-trip engineering of UML class models and Java source code. Furthermore, it presents an alternative implementation of the bidirectional transformation, which avoids a combinatorial explosion of the number of rules. Using the MoDisco (Brunliere, Cabot, Dupé, & Madiot, 2014) framework, Java source code may be represented as a model. Thus, QVT-R is used to formalize a bidirectional model-to-model transformation between UML class models and the MoDisco Java model. Specifying bidirectional transformations in QVT-R allows software engineers to provide a single relational specification, which may be executed in either direction, rather than writing two unidirectional transformations separately. Moreover, QVT-R was chosen because of its declarative nature, allowing the transformation developer to focus on relations and dependencies between metamodel elements rather than on single execution steps.

The chosen application scenario serves as a real-world use case which involves complex metamodels and allows to evaluate the benefits and drawbacks of the QVT-R standard in general and the provided tool in particular.

The article is structured as follows: In Section 2, background information for this case study is briefly presented. Section 3 provides insights in selected rules of the bidirectional transformation specification while results are shown with the help of an example in Section 4. Related work is discussed in Section 5. A discussion of the results and the conclusion is presented in Section 6.

2. OVERVIEW

2.1. The Relations Language

This section briefly enumerates key features of the Relational QVT language specified in (OMG, 2015b).

Every transformation consists of a set of relations and optionally some quer(y)ies. A relation declares at least two domains which may be specified as checkonly, enforce or primitive. The first keyword hints that a consistency check reports an error if correspondences are missing whereas target elements are created or deleted in enforce domains. A primitive domain serves as variable. If two domains are enforced, a bidirectional transformation is defined, as opposed to a unidirectional one where only one enforce domain is present. A relation that is decorated with the keyword top is executed automatically, otherwise it is called explicitly. This might happen in when- and where-clauses which provide pre- and postconditions for relations respectively. To avoid generating one element more than once, key properties can be defined which identify the element uniquely.

QVT-R is supposed to support different execution strategies. Executing the transformation in checking mode is meant to report possible inconsistencies between two existing models for both directions. When executing in enforcing mode, on the other hand, one direction must be picked. Here, the target model is either constructed anew (batch transformation) or possible inconsistencies of domains that are enforced are fixed (incremental transformation) by updating the respective element. Missing elements are added and elements in the target model will be deleted if they have no corresponding element in the source model.

In model transformations a trace may be present to store which elements of source and target model are linked. The QVT standard does not require that such a trace is persisted nor does it provide a language construct to access the trace explicitly in a domain. However, a transient trace is available
Related Content

A Semantic Approach for Multi-Agent System Design
www.igi-global.com/chapter/a-semantic-approach-for-multi-agent-system-design/117302?camid=4v1a

CbSSDF: A Two-Layer Conceptual Graph Approach to Web Services Description and Composition – A Scenario Based Solution Analysis and Comparison with OWL-S
www.igi-global.com/article/cbssdf-two-layer-conceptual-graph/58646?camid=4v1a
Web-Wide Application Customization: The Case of Mashups
[www.igi-global.com/article/web-wide-application-customization/61394?camid=4v1a](www.igi-global.com/article/web-wide-application-customization/61394?camid=4v1a)

Representation of Situational Methods: Incorporating ISO/IEC 24744 into a Domain-Based Framework
[www.igi-global.com/article/representation-of-situational-methods/80195?camid=4v1a](www.igi-global.com/article/representation-of-situational-methods/80195?camid=4v1a)