Enabling comprehensive tool support for QVT

Jörg Kiegeland, Hajo Eichler

ikv++ technologies ag, Bernburger Strasse 24-25, 10963 Berlin, Germany

[kiegeland,eichler]@ikv.de

Abstract. ikv++ technologies implemented OMG's QVT Relations specification and also provides an editor and debugger for developing model-to-model transformations. With the goal to build up a framework for tool suppliers to build further plug-ins for working with QVT, an integration strategy must be established. The attention must thereby be turned to a set of interfaces that allows connection between a QVT engine and tools using this engine.

Motivation

Of particular importance to a model-driven toolchain is the notion of model transformation. A specific standardized language for model transformation has been defined by OMG called QVT (Queries/Views/Transformations). ikv++ technologies has developed a QVT engine which implements the QVT Relations language.

The success of a newly introduced language like QVT depends strongly on the tool landscape that supports the users applying the language for their purposes. This issue is initially targeted by providing a QVT editor and debugger with the QVT engine itself.

Still, there is enormous space for more tool support when working with QVT, e.g. an editor for QVT's graphical notation or a tool measuring the code quality of a QVT script. Thus, this paper will outline base requirements allowing other tool producer to plug-in in a given engine of a standardized language.

medini QVT engine

As an example for a very basic interface, the ikv QVT engine is independent from concrete technologies for model representation. When using EMF as model technology in the Eclipse world, transformations can be executed transactionally using the EMF transaction framework and the transformation result can be checked on constraints using the EMF validation framework. Metamodels can be provided to the engine directly by ecore-files or as generated Java model code.

One main feature of the ikv QVT engine is the usage of traces to enable incremental updates: When the target model of a transformation is manually changed, a re-transformation shall preserve changes to some degree. In our experience, this was always the most complicated task when programming model-to-model transformations with programming languages like C++ or Java. However using the abstraction of QVT, the "programmer" does not need to care about it. Tools developers could use the generated traces to visualize which elements in the source model are mapped on which elements in the target model by QVT relations.

Another main feature of the ikv QVT engine is the support for bidirectional transformations, again by using traces. Changes in a target model can be propagated back to the source model. This is perhaps the most outstanding concept of QVT in general, since a transformation written in a traditional language like Java can be used only for one transformation direction. A tool could be written which visualizes modifications in the source and in the target model which need to be synchronized.

The ikv QVT engine does not contain features not conforming to the QVT specification. We have used our QVT engine in many customer projects and we were able to express difficult transformation tasks by using pure QVT. This conformance also enables us to offer clean and non-proprietary interfaces to tool developers.

The scripting of a QVT transformation can be accelerated by using the ikv QVT editor and the ikv QVT debugger, which both use respective Eclipse frameworks. In the same way as one can program and debug Java applications, one can also program and debug QVT.

A small overview of the QVT editor features which use interfaces of the QVT engine:

- Syntax highlighting of QVT and OCL keywords
- Syntactical and semantic errors are displayed in the Eclipse Problems view
- Code completion (e.g. the attributes and operations for a model element)
- Refactoring (e.g. rename variables, format source code)
- Outline View displaying the AST of the QVT script
- You can inspect pseudo code for a QVT relation, making the functioning of QVT very transparent

The ikv QVT debugger offers:

- Breakpoints and breakpoint conditions
- Watch view to evaluate OCL expressions (with code completion)
- Variable view to see all variables in the scope of the current stack frame, i.e. all variables defined by the current relation. Variables can be modified.

Requirements for integration of further tools

Tools like the QVT editor or the QVT debugger use interfaces provided by the ikv QVT engine. Though these tools are already provided by ikv itself, 3rd party tools can certainly use these public interfaces of the engine, too. It becomes clear that public interface definitions provided by the QVT engine is the key for enabling a multi tool provider landscape.

As an example, a GMF-based graphical QVT editor could use the QVT metamodel as domain model to graphically edit QVT scripts. Via the validation interface of the ikv engine, validation errors and warnings can be reported to the user in the Eclipse Problems view. Further, by using the serialization/deserialization interface of the ikv QVT engine, the QVT model can be persisted as QVT code in contrast to XMI.

We want to summarize requirements to an API which should be offered by a QVT engine:

- Debugging API to execute a QVT transformation in debug mode by an Eclipse debugger tool.
- Serialization / deserialization (i.e. parsing) interface.
- Semantic and syntactical analysis facility for QVT scripts / QVT AST instances, including a data structure to communicate problems in detail.
- Interfaces for the AST and the mapping from the AST to line/column information of the parsed QVT script, e.g. to facility code completion in a QVT editor.

Summary

This paper shows which interfaces should be provided by a QVT engine so that other Eclipse tools can make use of it. Closer looking at the ikv QVT engine, we presented already existing tools using interfaces of the engine and outlined a scenario how the engine could be used by a graphical QVT editor. A standardized API for QVT engines would facilitate to use QVT tools with different QVT engines.