Alignment of ATL and QVT

Ivan Kurtev

ATLAS group, INRIA & University of Nantes, France
http://www.sciences.univ-nantes.fr/lina/atl/
Context of this work

- The present courseware has been elaborated in the context of the MODELWARE European IST FP6 project (http://www.modelware-ist.org/).
- Co-funded by the European Commission, the MODELWARE project involves 19 partners from 8 European countries. MODELWARE aims to improve software productivity by capitalizing on techniques known as Model-Driven Development (MDD).
- To achieve the goal of large-scale adoption of these MDD techniques, MODELWARE promotes the idea of a collaborative development of courseware dedicated to this domain.
- The MDD courseware provided here with the status of open source software is produced under the EPL 1.0 license.
Prerequisites

To be able to understand this lecture, a reader should be familiar with the following concepts, languages, and standards:

- Model Driven Engineering (MDE)
- The role of model transformations in MDE
- QVT
- ATL
- MOF
Outline

- Need for ATL and QVT alignment
- Requirements alignment of ATL and QVT
- Architectural alignment of ATL and QVT
  - QVT architecture
  - ATL architecture
  - Aligning the architectural components
- Achieving interoperability between ATL and QVT
- Conclusions
A DSL Perspective on MDE

- MDE allows definition of small (and large) languages, focused on specific problems known as Domain Specific Languages (DSLs)
- Typical tasks in MDE also require DSLs: for example, model transformations and language definition
- A set of transformation languages exists
Problems

- Language fragmentation (Babylon tower)
  - Need for managing DSLs

- Multiple languages per problem domain
  - Problem domains often overlap
  - Need for choosing among DSLs

- Need for clear guidelines and knowledge for:
  - Matching the problem domain and the available solutions (DSLs)
  - Explicit comparison among languages to allow selection of the right tool
Aligning ATL and QVT

- Two model transformation languages for MDE
- Solving the same problem (at first glance)
  - Is that true?

Comparing ATL and QVT:
- At problem domain level: what problems can be solved, what are the requirements per language
- At architectural level: components and major capabilities
Requirements Alignment of ATL and QVT

- What are the problem domains of ATL and QVT?

- QVT is proposed in the context of OMG MDA approach
  - Targeted at software development

- ATL requirements evolved towards solving interoperability problems in data engineering
  - May solve software development problems
  - Deals with heterogeneous data
Characteristics of Software Development

• Transformations are mainly refinement of models;

• Models are expressed in a limited set of languages (e.g. UML)

• Languages often share the same conceptual foundation: Object-Oriented Principles

• Transformations should be semantic preserving

• Software development is iterative:
  • Change propagation should be supported

• Separation of concerns:
  • Need for model composition

• Reverse Engineering

• Refactoring
Characteristics of Data Engineering

- Heterogeneity of data and schemas (modeling languages):
  - RDBMS;
  - XML;
  - ORDBMS;
  - ER;
  - Many others;

- Need for data translation between heterogeneous data

- Need for declarative mappings for schema integration and query answering and translation

- Data migration (kind of data translation)
Requirements for QVT

An incomplete list (consult the QVT specification):

- Operates on MOF models (basically XMI-to-XMI transformations)
- Supports bidirectional transformations
- Declarative language (satisfied by Core and Relations languages)
- Checking the presence of certain relations among models
Requirements for ATL

Major requirement: ability to deal with various models expressed in different languages and technologies

Unification concepts:

- Everything is a model!
- Technologies are unified by the concept of Technical Space
- Heterogeneity is handled by the notion of Technical Projector
- Data translation between Technical Spaces is regarded as Bridging
Requirements alignment for ATL and QVT

- From the ATL perspective QVT solves transformational problems within the OMG/MDA Technical Space
- Without the concepts of Technical Projector and Bridging real interoperability problems cannot be solved (misses in QVT)
- ATL as a part of AMMA supports these concepts
Architectural Alignment of ATL and QVT

- Architecture: set of concepts and relations
- Alignment for the major components of ATL and QVT;
- Major goal:
  - Achieving interoperability between ATL and QVT
QVT Architecture

Contains three DSLs that form layers:

- Relations (declarative)
- Core (declarative, simpler than Relations)
- Operational Mappings (imperative)

![Diagram of QVT Architecture]

Diagram:
- Operational Mappings extends Relations
- Relations extends Core
- RelationsToCore Transformation extends Core
- Core extends Black Box
## Conformance Points for QVT Tools

<table>
<thead>
<tr>
<th>Language Dimension</th>
<th>Interoperability Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syntax Executable</td>
</tr>
<tr>
<td>Core</td>
<td></td>
</tr>
<tr>
<td>Relations</td>
<td></td>
</tr>
<tr>
<td>Operational Mappings</td>
<td></td>
</tr>
</tbody>
</table>

Note that **QVT conformance** is defined for **tools**. The term “**QVT compliant language**” is not defined in the spec.
ATL Architecture

Three-level architecture:

- AMW
- AMW to ATL transformations
- ATL compiler
- ATL VM
ATL Components

- **AMW (ATLAS Model Weaver):** Generic metamodel for establishing links among model elements (metamodel for model weaving)
  - Tool for defining domain specific transformation languages

- **ATL:** hybrid transformation language
  - Semantics of AMW extensions may be defined by ATL transformations

- **ATL VM:** virtual machine for executing model transformations
### ATL and QVT Component Alignment (1)

<table>
<thead>
<tr>
<th>Category</th>
<th>ATL</th>
<th>QVT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstraction Level of Transformation Specification</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMW</td>
<td></td>
<td>Relations</td>
</tr>
<tr>
<td>ATL Core, Operational Mappings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VM</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Transformation Scenarios</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model synchronization</td>
<td>Via a separate transformation</td>
<td>Relations, Core</td>
</tr>
<tr>
<td>Conformance checking</td>
<td>Via a separate transformation</td>
<td>Relations, Core</td>
</tr>
<tr>
<td>Model transformation</td>
<td>AMW, ATL, VM</td>
<td>Relations, Core, Operational Mappings</td>
</tr>
</tbody>
</table>

Abstractness
# ATL and QVT Alignment (2)

<table>
<thead>
<tr>
<th>Category</th>
<th>ATL</th>
<th>QVT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Paradigm</strong></td>
<td><strong>Declarative</strong></td>
<td>AMW</td>
</tr>
<tr>
<td></td>
<td><strong>Hybrid</strong></td>
<td>ATL</td>
</tr>
<tr>
<td></td>
<td><strong>Imperative</strong></td>
<td>VM</td>
</tr>
<tr>
<td><strong>Directionality</strong></td>
<td><strong>Multidirectional</strong></td>
<td>AMW</td>
</tr>
<tr>
<td></td>
<td><strong>Unidirectional</strong></td>
<td>ATL, VM</td>
</tr>
<tr>
<td><strong>Cardinality</strong></td>
<td>M-to-N</td>
<td>ATL, AMW, VM</td>
</tr>
<tr>
<td></td>
<td>M-to-1</td>
<td></td>
</tr>
<tr>
<td><strong>Traceability</strong></td>
<td>Automatic</td>
<td>ATL</td>
</tr>
<tr>
<td></td>
<td>User-specified</td>
<td>VM</td>
</tr>
<tr>
<td><strong>In-place Update</strong></td>
<td>ATL (in refining mode), VM</td>
<td></td>
</tr>
</tbody>
</table>
ATL and QVT Interoperability (1)

**Interoperability**: executing programs written in one language with the tools designed for another language.

**Motivations:**
- **Execution and support tools**
  - Assume you have a wonderful language without engine and a not so nice language with execution engine;
- **Compliance to standards**
  - If QVT programs run on the ATL VM then the ATL VM is QVT conformant!
ATL and QVT Interoperability (2)

Framework for reasoning about interoperability among ATL and QVT components

Legend:

<table>
<thead>
<tr>
<th>ATL components</th>
<th>our proposals</th>
</tr>
</thead>
<tbody>
<tr>
<td>QVT components</td>
<td>anticipated transformation</td>
</tr>
<tr>
<td>pre-existing</td>
<td></td>
</tr>
<tr>
<td>transformations</td>
<td></td>
</tr>
</tbody>
</table>

Forward engineering
### ATL and QVT Interoperability (3)

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>VM</th>
<th>OM</th>
<th>ATL</th>
<th>Core</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>N/A</td>
<td></td>
<td></td>
<td>Reverse engineering</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OM</td>
<td>OM-to-VM</td>
<td>N/A</td>
<td></td>
<td>issues</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATL</td>
<td>ATL compiler</td>
<td>ATL-to-OM</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td>Core-to-OM</td>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relations</td>
<td></td>
<td></td>
<td>Relations-to-Core</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

• ATL and QVT have common features but their problem domains are different:
  • QVT is mainly for software development;
  • ATL aims at solving data engineering transformation problems

• Architectural alignment between ATL and QVT shows that language interoperability is feasible via model transformations

• ATL tools may be called QVT conformant after the required transformations are provided

• It is interesting to generalize the framework for reasoning on ATL and QVT to other languages