Model Transformations

An overview

Pierre-Alain Muller
INRIA
Pierre-alain.muller@irisa.fr

Outline

► MDE basic principles
► What is a model-transformation?
► Typology of model-transformations
► Examples of transformations

MDA = MDE à la OMG

► OMG, Object Management Group
► MDA, Model-Driven Architecture
► PIM, Platform Independent Model
► PSM, Platform Specific Model
► (PDM, Platform Description Model)
► Transformation (PIM, PDM) --> PSM
  • RFP MOF Q/V/T Query, Views, Transformations
  • RFP MOF to Text

Model Transformations @ Google

Principles of Model-Driven Engineering

► A kind of (software) development approach
► Models as first class entities
► Everything is a model
► A model conforms to an other model (meta-model)
► A model transformation takes models and produces models
► A model-transformation is a model

Meta-modeling architecture
**Motivation**

- Model transformation is key to Model-Driven Engineering
  - Automation of the transition from Business models to Implementation models

**But also**

- Refining models
- Reverse engineering (code to models)
- Generating new views
- Applying design patterns
- Refactoring models

**Typical scope for transformations**

- Refining models
- Reverse engineering (code to models)
- Generating new views
- Applying design patterns
- Refactoring models

**Related fields**

- Program transformation and compiler techniques
- Meta-programming techniques
- Graph rewriting techniques

**MOF 2.0 Queries/Views/Transformations RFP**

- Define a language for querying MOF models
- Define a language for transformation definitions
- Allow for the creation of views of a model
- Ensure that the transformation language is declarative and expresses complete transformations
- Ensure that incremental changes to source models can be immediately propagated to the target models
- Express all new languages as MOF models

**Transformation Architecture**

- Define Transformation
  - Meta-model A
    - Conforms To
    - Model A
    - Apply Transformation
  - Transformation
  - Meta-model B
    - Conforms To
    - Model B
**Typical Example**

*From UML to RDBMS*

**Transformations as models**

- Composition of transformations
- Transformation of transformations

**Toward Model-Transformations**

- **CRUD on model elements**
  - Create, Read, Update, Delete

- **Transformation rules written in**
  - General purpose languages + API
  - Intermediate transformation language
  - Dedicated Model-Transformation languages

**General purpose language approach**

- **Java, VB, C++, C#,... Your favorite language!**
- Currently available in the tools via APIs
- No overhead to learn a new language
- Tool support to write the transformations
  
  => Monsieur Jourdain’s approach

- It’s going to be challenging to do better!

**Action Language**

- Use a general purpose action language
  - Better navigation facility (associations)
  - Get access to the types defined in the models
  - Procedural rule description

**Intermediate transformation language**

- Typically XML based
  - But XML (XMI) is verbose

- XSLT can be used to transform XML trees into other (XML) (trees)
  - More batch than interactive
  - Parameters are passed by values
  - XSLT transformations are not really easy to maintain
  - Better for simple transformations
Example of XSLT transformation

```xml
<xsl:template match="ClickscopeBusinessProcess耀眼exceptionGroup"> 
  <xsl:choose> 
    <xsl:when test="self::mode=\"isSynchronous\""> 
      <xsl:choose> 
        <xsl:when test="\$isSynchronous"> 
          <xsl:value-of select="concat(\"\$\", \"\$\")"/> 
        </xsl:when> 
        <xsl:otherwise> 
          <xsl:value-of select="concat(\"\$\", \"\$\")"/> 
        </xsl:otherwise> 
      </xsl:choose> 
    </xsl:when> 
    <xsl:otherwise> 
      <xsl:choose> 
        <xsl:when test="\$isSynchronous"> 
          <xsl:value-of select="concat(\"\$\", \"\$\")"/> 
        </xsl:when> 
        <xsl:otherwise> 
          <xsl:value-of select="concat(\"\$\", \"\$\")"/> 
        </xsl:otherwise> 
      </xsl:choose> 
    </xsl:otherwise> 
  </xsl:choose>
</xsl:template>
```

Dedicated Transformation Language

- Kind of DSL for transformation
- Simplify development and maintenance of model-transformations
- Higher expression power
- Enhanced structuration
  - Composition of rules
  - Interoperability

Dedicated transformation languages

- Terminology
- Features of model transformations

Query

- An expression evaluated over a model
  - Returns one or more instances of types defined either in the source model or by the query language
- OCL is an example of a query language

Examples of OCL queries

- Query: Has Pierre-Alain Muller sent a message about a given subject s?
  - `s.post->exists (author.name='Muller' and author.firstname='Pierre-Alain')`

- Query: Knowing that there is only one subject about QVT, I want to retrieve it.
  - `Subject.allInstances()->any (title = 'QVT')`

View

- A view is a model that is completely derived from another model
  - The meta-model of the view is typically not the same as the meta-model of the source
Transformation

A transformation generates a target model from a source model.

Source Model  \(\rightarrow\)  Transformation  \(\rightarrow\)  Target Model

May be bi-directional

Q vs V vs T

A query is a restricted kind of view

A view is a restricted kind of transformation
- The target model cannot be modified independently of the source model

A transformation generates a target model from a source model

Declarative

Declarative languages describe relationships between variables in terms of functions or inference rules and the language executor (interpreter or compiler) applies some fixed algorithm to these relations to produce a result.

Imperative

Any programming language that specifies explicit manipulation of the state of the computer system, not to be confused with a procedural language.

Declarative vs. Imperative Style

Declarative (what to do)
- Invariant relations between source and target models

Imperative (how to do it)
- How to derive a target from a source

May be combined via pre- and post-conditions

Execution Strategy

Invocation of the transformation rules
- Explicit, via invocation operations (Java like)
- Implicit, based on context and rules' signature (Prolog like)
Trace

- Trace associates one (or more) target element with the source elements that lead to its creation
  - For Round-trip development
  - Incremental propagation

- Rules may be able to match elements based on the trace without knowing the rules that created the trace

Rule

- Rules are the units in which transformations are defined
  - A rule is responsible for transforming a particular selection of the source model to the corresponding target model elements.

Declaration

- A declaration is a specification of a relation between elements in the LHS and RHS models

Implementation

- An implementation is an imperative specification of how to create target model elements from source model elements
  - An implementation explicitly constructs elements in the target model
  - Implementations are typically directed

Match

- A match occurs during the application of a transformation when elements from the LHS and/or RHS model are identified as meeting the constraints defined by the declaration of a rule
  - A match triggers the creation (or update) of model elements in the target model

Incremental

- A transformation is incremental if individual changes in a source model can lead to execution of only those rules which match the modified elements
Classification of model transformations

Model-to-Text Approaches

► Visitor-Based Approaches
► Template-Based Approaches

Model-to-Model Approaches

► Direct-Manipulation Approaches
► Relational Approaches
► Graph-transformation-based Approaches
► Structure-Driven Approaches
► Hybrid Approaches
► Other

M2T: Visitor-based

► Some visitor mechanisms to traverse the internal representation of a model and write code to a text stream
  ◦ Iterators
  ◦ Write ()

M2T: Template-Based

► A template consists of the target text containing slices of meta-code to access information from the source and to perform text selection and iterative expansion
  ◦ The structure of a template resembles closely the text to be generated
  ◦ Textual templates are independent of the target language and simplify the generation of any textual artefacts
M2M: Direct Manipulation
▶ Internal representation plus some API to manipulate it
▶ Object-oriented framework
▶ Rules and scheduling implemented from scratch using a programming language
▶ JMI (MOF-compliant Java Interface)
  * JSR-000040 Java™ Metadata Interface

M2M: Relational Approaches
▶ Declarative, based on mathematical relations
  * Good balance between flexibility and declarative expression
▶ Implementable with logic programming
  * Mercury, F-Logic programming languages
  * Predicate to describe the relations
  * Unification based-matching, search and backtracking

M2M: Graph-Transformation-Based
▶ Declarative, based on the theoretical work on graph transformations
  * Operates on typed, attributed, labeled graphs
  * Rule (LHS, RHS : Graph Pattern)
▶ Automated source element selection

JMI examples
```java
package javax.jmi.model;
import javax.jmi.reflect.*;
public interface Attribute extends StructuralFeature {
    public boolean isDerived();
    public void setDerived(boolean newValue);
}
```

Example of logic programming
▶ Excerpt of Mercury code
```
conditionaltask(id) :-
    conditionaltask_for_outputgroup_of_activity(id, _OutputGroup),
    outputgroup_of_activity(id, _Activity),
    map(_OutputGroup, id, conditionaltask_for_outputgroup, id),
    outputgroup_of_activity(_OutputGroup, _Activity) :-
    outputgroup_of_outputgroup(_OutputGroup),
    contains(Activity","id", _OutputGroup,"id",
    activity(Activity)).
```

About Graphs
▶ Web site of Reiko Heckel ☺
**M2M : Graph-Transformation-Based**

- Powerfull, but complex because of the non-determinism in scheduling and application strategy
  - Require careful consideration of termination of the transformation process and the rule application ordering
- It is unclear how practitioners will receive these complex approaches

**M2M : Structure-Driven Approaches**

- 1st Phase
  - Creation of hierarchical structure of target model
- 2nd Phase
  - Set the attributes and references in the target
- Users provide the transformation rules
- Framework determines the scheduling

**M2M : Structure-Driven Approaches**

- Pragmatic approaches developed in the context of EJB and Databases schema generation from UML models
- Strong support for 1-to-1 and 1-to-n correspondence between source and target
- Unclear how well these approaches can support other kinds of applications

**M2M : Hybrid Approaches - others**

- Any combination of different techniques
- Practical approaches are very likely to have the hybrid character

**Practically speaking**

- How many developers are familiar with the prolog-like style of rules writing?
- Where is the advantage of a dedicated explicit language vs. a general purpose language?
- Hybrid Languages or transformation libraries for general purpose languages...

**Tools**

- Generic transformation tools
- CASE tools scripting languages
- Dedicated model transformation tools
- Meta-modeling tools
**Generic transformation tools**

- XSLT
- Graph Transformation tools
  - Ask Reiko

**CASE tools scripting languages**

- Arcstyler from Interactive Objects
  - MDA-Cartridge, JPython (Python & Java)
- Objecteering from Objecteering Software
  - J language
- OptimalJ from Compuware
  - TPL language
- Fujaba (From UML to Java and Back Again)
  - Open Source

**Dedicated model transformation tools**

- Mia-Transformation from Mia-Software
  - Inference rules + Java
- PathMATE from Pathfinder Solutions
  - Easy to integrate with modeling tools
- Open-Source
  - ATL, MTL, AndroMDA, BOTL, Coral Mod-Transf, QVTEclipse or UMT-QVT

**Meta-modeling tools**

- MetaEdit+ from MetaCase
- XMF-Mosaic from Xactium
- Open-Source
  - KerMeta from INRIA
  - www.kermeta.org

**Coming soon**

- Model Transformations in Practice Workshop
  - October 3rd 2005
  - Part of the MoDELS 2005 Conference
- Comparing and contrasting various approaches

**References**

- CBOP, DSTC, and IBM. MOF Query/Views/Transformations, Revised Submission. OMG Document: ad/03/09-03
- C. Cleaveland. Program Generators with XML and Java. Prentice-Hall, 2001, see http://www.craigc.com/pg/
References

- Model Transformation – the Heart and Soul of Model-Driven Software Development, tech report 200352.