Model Transformation Techniques
(or: Why I'd like write programs that write programs rather than write programs)

Prof. Jean-Marc Jézéquel
(Univ. Rennes 1 & INRIA)
Triskell Team @ IRISA
Campus de Beaulieu
F-35042 Rennes Cedex
Tel : +33 299 847 192 Fax : +33 299 847 171
e-mail : jezequel@irisa.fr
http://www.irisa.fr/prive/jezequel

Modeling and Weaving

Challenges:
- Automatic Weaving
- Product Lines
- Reuse design know-how reified as Model Transformations
Model-to-Text vs. Model-to-Model

- **Model-to-Text Transformations**
  - For generating: code, xml, html, doc.
  - Should be limited to syntactic level transcoding

- **Model-to-Model Transformations**
  - PIM to PSM a la OMG MDA
  - Refining models
  - Reverse engineering (code to models)
  - Generating new views
  - Applying design patterns
  - Refactoring models
  - Deriving products in a product line
  - ... any model engineering activity that can be automated...

Model-to-Text Approaches

- For generating: code, xml, html, doc.
  - Visitor-Based Approaches:
    » Some visitor mechanisms to traverse the internal representation of a model and write code to a text stream
    » Iterators, Write()
  - Template-Based Approaches
    » 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
Transformation Architecture

- Meta-model A
  - Conforms To
  - Model A
  - Define Transformation
  - Meta-model B
  - Conforms To
  - Model B
  - Apply Transformation

Model-to-Model: Typical Example

From UML to RDBMS
M2M: Reuse Engineering Know-How (Design/Test/…)

Design pattern application (parametric collaboration)

Element stereotype

…and also Tagged values & Contracts

The result we want: design patterns application
**Why complex transformations?**

- Example: Air Traffic Management
  - “business model” quite stable & not that complex
- Various modeling languages used beyond UML
  - As many points of views as stakeholders
- Deliver software for (many) variants of a platform
  - Heterogeneity is the rule
- Reuse technical solutions across large product lines (e.g. fault tolerance, security…)
- Customize generic transformations
- Compose reusable transformations
- Evolve & maintain transformations for 15+ years!

---

**Classification of Model Transformation Techniques**

1. General purpose programming languages
   - Java/C#...
2. Generic transformation tools
   - Graph transformations, XSLT...
3. CASE tools scripting languages
   - Objecteering, Rose...
4. Dedicated model transformation tools
   - OMG QVT style
5. Meta-modeling tools
   - Metacase, Xactium, Kermeta...
1. General purpose language approach

- Java, VB, C++, C#, ... Your favorite language!
- Currently available in the tools via APIs
- Rules and scheduling implemented from scratch using the programming language
- Example:
  - JMI (MOF-compliant Java Interface)
    » JSR-000040 Java™ Metadata Interface

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

```java
package javax.jmi.model;
import javax.jmi.reflect.*;
public interface Operation extends BehavioralFeature {
    public boolean isQuery();
    public void setQuery(boolean newValue);
    public java.util.List getExceptions();
}
```

JMI examples
General purpose language approach: Conclusion

- No overhead to learn a new language
- Tool support to write the transformations

=> Monsieur Jourdain’s approach

But wait:
- We resort to modeling (before programming) for mastering complexity, right?
- Does it mean that transformations never get complex?
  » Or that the problem only appears for non toy applications…

2. Generic transformation tools

- Awk-like (inc. sed, perl…)
- XSLT
- Graph Transformation tools
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:param name="a"/>
  <xsl:variable name="ct" select="concat($omii.id,'.condTask')"/>
  <xsl:choose>
    <xsl:when test="self::node()[$isSynchronous = 'true']">
      <xsl:call-template name="condTaskTemplate">
        <xsl:with-param name="ct" select="$ct"/>
        <xsl:with-param name="a" select="$a"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="asynCompoundTaskInputGroupOrActivityOutputGroup">
        <xsl:with-param name="a" select="$a"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```
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, non-determinism in scheduling and application strategy

- Well known technology, albeit hard to master

Generic transformation tools: Conclusion

- Awk-like (inc. sed, perl...) SE Limit: ~100 LOC

- XSLT SE Limit: ~1000 LOC
  - Good for syntactic transcoding, not for semantic manipulations

- Graph Transformation tools SE Limit: PhD needed!
  - Powerful, 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
3. CASE tool 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
- ...

CASE tools scripting languages

- **Pro**
  - Good level of maturity
  - Excellent integration with their CASE tool
- **Drawbacks**
  - Proprietary languages and/or tight coupling with the CASE
  - Often developed as a second thought, not central
  - Many limitations when model transformation get complex
    » Structuration, modularity, reuse problem
    » Configuration management issue when they need to be evolved and maintained for long periods
4. Dedicated Transformation Language

- OMG QVT style
  - Kind of DSL for transformation

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

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
## 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

## 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 target models from source models.

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 target models from source models.
Classification

- Several approaches
  - Graph-transformation-based Approaches
  - Relational Approaches
  - Structure-Driven Approaches
  - Hybrid Approaches

- Commercial
  - Mia-Transformation (Mia-Software), PathMATE (Pathfinder Solutions)

- Many academic tools
  - ATL & MTL (INRIA), AndroMDA, BOTL (Bidirectional Object oriented Transformation Language), Coral (Toolkit to create/edit/transform new models/modeling languages at run-time), Mod-Transf (XML and ruled based transformation language), QVTEclipse (preliminary implementation of some ideas of QVT in Eclipse) ou encore UMT-QVT (UML Model Transformation Tool)

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

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
Example of logic programming

- Excerpt of Mercury code

```mercury
conditionaltask(Id) :-
    conditionaltask_for_outputgroup_of_activity(Id, _OutputGroup).

conditionaltask_for_outputgroup_of_activity(Id, OG) :-
    outputgroup_of_activity(OG, _Activity),
    mapId(OG^og_id, conditionaltask_for_outputgroup, Id).

outputgroup_of_activity(OutputGroup, Activity) :-
    outputgroup(OutputGroup),
    contains(Activity^a_id, OutputGroup^og_id),
    activity(Activity).
```

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
Dedicated model transformation tools: Conclusion

- 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…

5. Meta-modeling tools

- Build (OO) Models of Transformations
- Use MDE to run them

**Commercial tools:**
- MetaEdit+ from MetaCase
- XMF-Mosaic from Xactium

**Open-Source**
- KerMeta from INRIA
- www.kermeta.org
“Programming style” Issues

- The transformation is simply the model of an object-oriented program that manipulates model elements
  - Navigation through model is first class though (like in OCL)
- OO techniques
  - Customizability through inheritance/dyn. binding
  - Pervasive use of GoF like Design Patterns
Defining the metamodels

Visual / Textual

package RDBMSMM;
require kermeta;
using kermeta::standard;

class Table{
  attribute name : String;
  attribute cols : Column[1..*];
  reference pkey : Column[1..*];
  attribute fkeys : FKey[0..*];
}
class FKey{
  reference references : Table;
  reference cols : Column[1..*]
}
class Column{
  attribute name : String;
  attribute type : String;
}
class RDBMSModel{
  attribute table : Table[1..*];
}
### UML2RDBMS template method

- **Create tables**
  - Tables are created from classes marked as persistent in the input model.

- **Create columns**
  - For each persistent class, process all attributes and outgoing associations to create corresponding columns. The foreign keys are created but the `cols` property cannot be filled and the corresponding columns cannot be created because primary keys of `references` table cannot be known before it has been processed.

- **Update foreign-keys**
  - The foreign-key columns are created in the table that contains the foreign-key and the property `cols` of foreign-keys is updated.

```
=> Handle details/variability into subclasses
```

### Writing the transformation

```
package Class2RDBMS;
require kermeta // The kermeta standard library
require "trace.kmt" // The trace framework
require "../metamodels/ClassMM.ecore" // Input metamodel in ecore
require "../metamodels/RDBMSMM.kmt" // Output metamodel in kermeta

class Class2RDBMS
{
  /** The trace of the transformation */
  reference class2table : Trace<Class, Table>

  /** Set of keys of the output model */
  reference fkeys : Collection<FKey>

  [...]}
```

Loading ECore and Kermeta metamodels
operation transform(inputModel : ClassModel) : RDBMSModel is do

// Initialize the trace
class2table := Trace<Class, Table>.new
class2table.create
fkeys := Set<FKey>.new

result := RDBMSModel.new

// Create tables
getAllClasses(inputModel).select{ c | c.is_persistent }.each{ c |
  var table : Table init Table.new
table.name := c.name
class2table.storeTrace(c, table)
result.table.add(table)
}

// Create columns
getAllClasses(inputModel).select{ c | c.is_persistent }.each{ c |
  createColumns(class2table.getTargetElem(c), c, "")
}

// Create foreign keys
fkeys.each{ k | k.createFKeyColumns }

Getting the input model

![Image of the input model]

Resource Set

- platform/resource:https://class2RDBMSmodels/ClassModel.xml
  - ClassModel
    - Class Customer
      - Attribute name
    - Class Order
      - Attribute order_no
    - Class Address
      - Attribute addr
      - PrimitiveDataType string
    - PrimitiveDataType integer
  - Association Customer
    - Association address
      - platform/resource:https://class2RDBMSmodels/ClassModel.xml

Console

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>does</td>
<td>Class Customer</td>
</tr>
<tr>
<td>name</td>
<td>Customer</td>
</tr>
<tr>
<td>src</td>
<td>Class Order</td>
</tr>
</tbody>
</table>
Executing the transformation

Generated output model
Object-orientation

- Classes and relations, multiple inheritance, late binding, static typing, class genericity, exception, typed function objects
- OO techniques such as patterns, may be applied to model transformations
  - Template method as above
  - Command, undo-redo
    » Refactorings example

abstract class RefactoringCommand
{
  operation check() : Boolean is abstract
  operation transform() : Void is abstract
  operation revert() : Void is abstract
}

Composition of transformations

- Packages, classes, operations and methods, inheritance and late bindings
- Rule recursivity is handled by function recursivity
Robustness and error handling

- Kermeta is statically typed, and the code can be fully checked for correctness at compilation time.
- For unexpected behavior at runtime, the language provides exception handling.

Design variations, libraries vs. DSLs

- A final design reflects a set of tradeoffs made by the developer
- The variation of the designs may be more or less constraint by the amount of pre-design and reuse provided by the language environment
Software Engineering Concerns

- Modularity in the small and the large
  - classes & packages
- Reliability
  - static typing, typed function objects and exception handling
- Extensibility and reuse
  - inheritance, late binding and genericity
- V & V
  - test cases

Conclusion

- Transformations are Assets
  - apply sound SE principles: Modeling!
    » from requirements, analysis, design, to implementation, V&V, and Configuration Management
- Developing Model Transformations in-the-large is not different from developing software
  - Unclear Requirements, Bugs,
  - Learning Curve (yet another language)
- A specific language does not help much to address these generic SE issues
  - Rely on proven techniques: OO languages, patterns, frameworks

Model-Driven Approach to Model Transformation