Bidirectional Transformations with Graphical Constraints

Juan de Lara, Esther Guerra
Univ. Autónoma de Madrid (Spain)
Introduction

- In Model-Driven Engineering, **models** are the core asset of the development:
  - Higher productivity through abstraction.
  - (re-)Generate code.
  - Testing, validation and verification.
  - ...

- Model manipulation is a key activity.
  - Model animation, model refactorings, etc.
  - Transformation into other languages (e.g. for analysis).
    - Batch or incremental. Synchronization.
  - Model comparison, model traceability, model merging, etc.

- Proliferation of programs or specifications stating how models should be related.
We call **inter-modelling** to the activity of modelling relations between two or more models:

- Model-to-Model transformation *specifications*.
- Definition of model matching and model traceability constraints.
- Inter-Model consistency maintainers, etc.

How to specify such model relations in a precise, high-level, bi-directional way?

Develop a *unified*, formal framework for inter-modelling, usable in practice and permitting analysis.
Declarative Inter-modelling specification

Approach

1. Specification Designer

2. Static Analysis
   - Pattern-pattern conflicts
   - Pattern-metamodel conflicts
   - Metamodel coverage
   - ...

3. Select op. scenario:
   - Batch transformation
   - Model traceability
   - Model matching

4. TGG rules

5. EOL

6. OCL

7. QVT-R

Check-only

Set of related models

Select check-only. scenario:
conformance
redundant traces
incorrect traces
enabled occurrences

act on...

check for...

Scenarios.

- An inter-modelling specification language.
- Applications
- Conclusions and Future Work.
Introduction

Model transformations

- Transform a model from a source to a target meta-model.
- Use domain specific languages to describe such transformation.
  - **Operational** languages. Two programs to transform source-to-target and target-to-source.
  - **Declarative/Bidirectional.** One specification can describe the transformation in both directions.
    - Compilation into operational mechanisms.
Operational Scenarios

Batch Transformations

- Generate model B from model A.

- Model B is empty at the beginning.

- There can be more than one model B that together with A satisfies the specification.

- Target-to-Source is the symmetrical situation.
Operational Scenarios

Batch Transformation (forwards)

Initial State

Operational mechanism

Final State

Transformation traces (or mappings). Maintain the correspondence between source and target elements. It is a model in its own right.
Operational Scenarios

Batch Transformation (backwards)

Initial State

Operational mechanism

Final State
Operational Scenarios

Incremental Transformation

- Model A was transformed into model B.

- Model A is modified, so that models do not satisfy the specification.

- Transformation is reapplied to modify B, so that consistency is obtained again.
Operational Scenarios

Incremental Transformation

Initial State

<table>
<thead>
<tr>
<th>Feature</th>
<th>c1: class</th>
<th>name=&quot;employee&quot;</th>
<th>persistent=true</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1: attribute</td>
<td>name=&quot;name&quot;</td>
<td>type=&quot;string&quot;</td>
<td>public=true</td>
</tr>
<tr>
<td>a: attribute</td>
<td>name=&quot;age&quot;</td>
<td>type=&quot;integer&quot;</td>
<td>public=true</td>
</tr>
</tbody>
</table>

Final State

<table>
<thead>
<tr>
<th>Feature</th>
<th>c1: class</th>
<th>name=&quot;employee&quot;</th>
<th>persistent=true</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1: attribute</td>
<td>name=&quot;name&quot;</td>
<td>type=&quot;string&quot;</td>
<td>public=true</td>
</tr>
<tr>
<td>a: attribute</td>
<td>name=&quot;age&quot;</td>
<td>type=&quot;integer&quot;</td>
<td>public=true</td>
</tr>
</tbody>
</table>

Operational mechanism

<table>
<thead>
<tr>
<th>Feature</th>
<th>t1: table</th>
<th>name=&quot;person&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>c: column</td>
<td>name=&quot;age&quot;</td>
<td>type=&quot;NUMBER&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>c1: column</th>
<th>name=&quot;name&quot;</th>
<th>type=&quot;CHAR(*)&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>c: column</td>
<td>name=&quot;age&quot;</td>
<td>type=&quot;NUMBER&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Operational Scenarios

Model Synchronization

- Model A was transformed into model B.

- Models A and B are modified, so that models do not satisfy the specification.

- Transformation is reapplied to modify A and B, so that consistency is obtained again.
Operational Scenarios

*Model Matching*

- Two models are given.
- Relate them in **all** possible ways (i.e. create all possible traces), according to the specification.
Operational Scenarios

Model Matching

Operational mechanism

Initial State
- c0: class
  - name="person"
  - persistent=true
- t0: table
  - name="person"
- a: attribute
  - name="age"
  - type="integer"
  - public=true

Final State
- c1: class
  - name="person"
  - persistent=true
- t1: table
  - name="person"
- a: attribute
  - name="age"
  - type="integer"
  - public=true
Operational Scenarios

Model Traceability

- Two models are given.
- Relate them in one possible way (i.e. create all possible traces), according to the specification.
- Several solutions could be possible.
Operational Scenarios

Model Traceability

Initial State

Final States

Operational mechanism
Check-Only Scenarios

Check satisfaction

- Check if models A and B are consistent according to the specification and scenario.
Operational Scenarios

Satisfaction

- Correctly traced.
- Incorrectly matched.
- Violates the specification forwards.
- Satisfies the specification backwards.
- Incorrectly synchronized.
Conclusion

- Given two languages related through an inter-modelling specification, there are many operational scenarios of interest that need to be solved.

- Avoid specifying each one of them by hand through a different operational specification.

- High-level language for inter-modelling, and automated synthesis of operational mechanisms for each different scenario!
Agenda

- Scenarios

  An inter-modelling specification language.
  - Triple Graphs, Constraints and Patterns.
  - Compilation.
  - Example.

- Applications

- Conclusions and Future Work.
Inter-modelling through constraints

- **Goal**: specify a set of model pairs, which we consider valid.

- We will use declarative (graph) constraints:
  - **Positive** (some structures should be present in source/target models).
  - **Negative** (some structure should not be present in source/target models).

- Different interpretations of the constraints depending on the scenario to be solved.
Triple Graphs

- Made of three graphs: source (S), target (T) and correspondence (C).
  - Plus mappings from the nodes in C to elements of S and T (two graph morphisms in the simplest case).

- We do not make any assumption about the kind of graphs in S, C, T.
  - from simple (V; E; src, tar: E→V)
  - to attributed, typed graphs.

- In particular, C is also a graph.
Constraint Triple Graphs

- Replace the set of attribute nodes (i.e., the values) by a finite set of variables.
- A formula constrains the possible values of variables.

A constraint triple graph (over signature $\Sigma=(S,\text{op})$) is made of:

- A triple graph.
- A finite set $\nu$ of $S$-sorted variables.
- A $\Sigma(\nu)$-formula $\alpha$. 

\begin{center}
\begin{tabular}{c|c|c}
T1: A & T: C & T2: B \\
\hline
a = x & y = x \cdot 2 & b = y \\
& & c = z \\
\hline
& y > 0 & \text{z = 3}
\end{tabular}
\end{center}
Everything is a constraint!

- A triple graph can be represented by a constraint where the formula assigns exactly one value to each attribute.
Relating constraints: morphisms

Now we need to relate constraints, e.g., to:

- Find the occurrence of a constraint in a set of models.
- Define the structure of patterns (pre-conditions)
Constraint Triple Graph Morphisms

Defined by:

- A structural morphism, with the condition that...
- … the formula in the morphism domain should be weaker (or equal) than the formula in the codomain:

\[ \forall f : \nu_2 \rightarrow \mathcal{A} \text{ s.t. } \mathcal{A} \models f \alpha_2 \]

\[ \mathcal{A} \models f \left( \alpha_2^S \Rightarrow m^\nu(\alpha_1^S) \right) \land \left( \alpha_2^T \Rightarrow m^\nu(\alpha_1^T) \right) \land (\alpha_2 \Rightarrow m^\nu(\alpha_1)) \]
Two kinds of patterns:

- Positive, with or without positive pre-conditions (parameters).

- Negative.

Specification = A conjunction of patterns.
Patterns may have pre-conditions

**Positive**

Attribute2Column = \langle \text{when} \rightarrow \text{Attribute-Column}, \, \mathcal{N}_{\text{pre}} = \emptyset \rangle

\[(p=false? \, x='__'+y: \, x=y) \quad \text{and} \quad \text{(Collection\{integer', 'float', 'double\}.exists(z \mid z=t \text{ and } t2='NUMBER') \text{ or } (t='string' \text{ and } t2='CHAR(*)'))} \]
Patterns may have pre-conditions

**Positive**

\[
\text{when } \quad \text{c:Class} \quad \text{C2T} \quad \text{t:Table} \\
\Rightarrow \\
\text{Attribute-Column} \\
\text{c:Class} \quad :\text{C2T} \quad \text{t:Table} \\
\text{a:Attribute} \quad :\text{A2C} \\
\text{name}=x \quad \text{type}=t \quad \text{public}=p \\
\text{co:Column} \\
\text{name}=y \quad \text{type}=t2 \\
\]

\[
(p=\text{false}? x='__'+y: x=y) \quad \text{and} \\
\left(\text{Collection\{integer', 'float', 'double}\}.\exists(z | z=t \quad \text{and} \quad t2='\text{NUMBER}') \quad \text{or} \quad (t='\text{string}' \quad \text{and} \quad t2='\text{CHAR(*)}')\right)
\]

\[
\text{Attribute2Column} = \langle \text{when} \rightarrow \text{Attribute-Column}, \ N_{\text{pre}} = \emptyset \rangle
\]
Patterns may have pre-conditions

Positive

\[
\text{Attribute2Column} = \langle \text{when} \rightarrow \text{Attribute-Column}, \ N_{\text{pre}} = \emptyset \rangle
\]

\[(p=false? x='__'+y: x=y) \text{ and } (\text{Collection\{integer', 'float', 'double'}\}.\exists(z \mid z=t \text{ and } t2='\text{NUMBER}') \text{ or } (t='\text{string}' \text{ and } t2='\text{CHAR(*)}')\)
Patterns may have pre-conditions

Class2Table = \langle \emptyset \rightarrow \text{ClassTable}, N_{pre} = \{\text{ClassTable} \rightarrow \text{Parent}\} \rangle
Patterns may have pre-conditions

Negative

Class2Table = 〈∅→ClassTable, N_{pre} = {ClassTable→Parent}〉
Satisfaction Model Transformation

- Synchronization = Forward sat. + Backward sat.

The semantics of one pattern is the set of all models that satisfy it.

Compositional semantics for specifications: intersection of the languages of all its patterns.
Other notions of satisfaction for the other scenarios.

Model Matching Example
Compilation
Forward Transformation/TGGs

- Pre/Post conditions (LHS/RHS).
- Based on rewriting.
- Application as long as possible.

Unfortunately, not so simple! : Parameterization.
Pre/Post conditions (LHS/RHS).
- Based on rewriting.
- Application as long as possible.

Unfortunately, not so simple! : Parameterization.
To achieve completeness of forward/backward transformation, we need to generate additional patterns.

- Increasingly bigger parameter.

- Then generate one rule for each one of the generated pattern.

- The transformation is in general not confluent, but it is complete: it finds all possible models satisfying the specification. In addition, it is always terminating.

- Heuristics: restrict parameterization to generate only some parameters.
Example

- RDBMS and XML representation of library information.
Example

Transformation Scenario: Forward Rules
Example

Transformation Scenario: Forward Rules
Example
Traceability Scenario.

S - relaxed deletion 1

L

s1: Subject

s2: Subject

R

s1: Subject

s2: Subject

NAC1

: Book

: B

: Book

s1: Subject
description = d1
d1 = d2

s2: Subject
description = d2

Book - tracing

Book.h2 - tracing

R = TNAC1^T = TNAC1^T

b1: Book

ISBN = i1

title = t1

publisher = p1

b2: Book

ISBN = i2

title = t2

p1 = Publisher

name = p2

p1 +

text = p2

b1: Book

ISBN = i1

title = t1

p1 = Publisher

name = p2

b2: Book

ISBN = i2

title = t2

p1 = Publisher

name = p2

BNAC2^T

b1: Book

ISBN = i1

title = t1

p1 = Publisher

name = p2

b2: Book

ISBN = i2

title = t2

p1 = Publisher

name = p2

Book

Book

S

Book

Publisher

desc = d1

d1 = 'Children'
d4 = 'Teen'

Book

Publisher

desc = d4

d4 = 'Teen'

Book

Publisher

desc = d3

d3 = 'Children'

d2 = 'Teen'

Publisher

name = p3
Example
Matching Scenario.

FlexibleSubject

: Book ← : B → : Book

: Subject ← : S → : Subject
desc = d1
desc = d2

(d1 = d2) or
Collection{ ‘Children’, ‘Kids’}.exists(x,y | x = d1 and y = d2) or
Collection{ ‘Teen’, ‘Teenagers’, ‘Youth’}.exists(x,y | x = d1 and y = d2)
Scenarios

An inter-modelling specification language.

Applications

- Inter-modelling
- Transformation contracts for testing
- QVT-R

Conclusions and Future Work.
Inter-Modelling

*Discovery of Design Patterns*

---

**Ecore**
- EClass
- EOperation
- EReference

**Instance**
- ClassBind
- OpBind
- RefBind

**Design Pattern Vocabulary**
- ClassRole
- OperationRole
- ReferenceRole

**DesignPattern**
- name: String
- type: String
- intent: String
- motivation: String
- consequence: String
- applicability: String

1..* + participants
Inter-Modelling

Discovery of Design Patterns
Inter-Modelling

*Discovery of Design Patterns*

- Apply the inter-modelling specification to particular models.
- Results shown in a three pane-window

UML model

traces identify instances of patterns

GOF pattern repository
Transformation contracts

OCL as checking mechanism

1. **contract (requirements)**

Part of a bigger effort to design a family of modelling languages to **engineer** model transformations (transML)

2. **transformation implementation**

3. **OCL correctness assertions**

4. **automated testing**

input model

output model

implementation + oracle

transform implementation
Transformation contracts
QVT-R as checking mechanism

1. \textbf{contract (requirements)}

2. \textbf{transformation implementation}

3. \textbf{compilation into QVT-Relations}

4. \textbf{automated testing}

\begin{itemize}
  \item [1] \textit{contract (requirements)}
  \begin{itemize}
    \item \textbf{preconditions} 
    \item \textbf{qualifying criteria}
    \item \textbf{transformation implementation under test}
  \end{itemize}
  \begin{itemize}
    \item \textbf{oracle function}
    \item \textbf{invariants, postconditions}
  \end{itemize}

  \item [2] \textit{transformation implementation}
  \begin{itemize}
    \item \textbf{using arbitrary transformation language}
  \end{itemize}

  \item [3] \textit{compilation into QVT-Relations}
  \begin{itemize}
    \item \textbf{compilation into QVT-Relations engine}
    \item \textbf{transformation execution}
    \item \textbf{source model}
    \item \textbf{target model}
  \end{itemize}

  \item [4] \textit{automated testing}
  \begin{itemize}
    \item \textbf{automated testing}
    \item \textbf{check}
  \end{itemize}
\end{itemize}
Transformation contracts

Class Metamodel

ModelElement
  name : String

Package
  namespace
  children
  parents
  classes
  ancestors

Class
  isPersistent : Bool
  attributes

context Class:
def ancestors: Set(Class)=self.parents->union(self.parents->collect(an| an.ancestors))

Attribute
  type : String

Correspondences

(Req1) Package -> Schema
(Req2) Class (isPersistent) -> Table
(Req3) Attributes -> Column
(Req4) Inherited Attributes -> Column

Relational Metamodell

ModelElement
  name : String

Schema
  schema
  tables
  columns

Table
  name : String
  columns

Resulting Relational Model

Exemplary Class Model

p1 : Package
  name = 'University'
  namespace

classes

attributes

a1 : Attribute
  name = 'name'
  type = 'String'

a2 : Attribute
  name = 'registriNo'
  type = 'Integer'

c1 : Class
  isPersistent = true
  name = 'Person'

ancestors

parents

children

c2 : Class
  isPersistent = true
  name = 'Student'

s1 : Schema
  name = 'Person'
  schema

co1 : Column
  name = 'name'
  type = 'String'

t1 : Table
  name = 'Student'
  columns

t2 : Table
  name = 'Student'
  columns

co2 : Column
  name = 'registriNo'
  type = 'Integer'

c03 : Column
  name = 'name'
  type = 'String'
Transformation contracts

**P(Class2Table)**
- **Class**
  - p: Package
    - name = X
  - c: Class
    - name = Y
    - isPersistent = true
  - t: Table
    - name = Y

**P(Attribute2Column)**
- **Class**
  - p: Package
    - name = X
  - c: Class
    - name = Y
    - isPersistent = true
  - a: Attribute
    - name = Z
    - type = T
- **Relational**
  - s: Schema
    - name = X
  - t: Table
    - name = Y
  - a: Attribute
    - name = Z
    - type = T

**P(InheritedAttributes)**
- **Class**
  - p: Package
  - c: Class
    - isPersistent = true
  - t: Table
    - name = C
  - a: Attribute
    - name = A
    - c.ancestors->includes(p)
- **Relational**
  - s: Schema
  - t: Table
    - name = C
  - a: Attribute
    - name = A
    - name = A

**N(NoRedefinedAttrs)**
- **Class**
  - pa: Package
    - name = X
  - c: Class
    - c.ancestors->includes(p)
- **Relational**
  - a: Attribute
    - name = X

**N(NoDuplicatedColumns)**
- **Class**
  - c: Column
    - name = X
- **Relational**
  - c: Column
    - name = X
  - e: Column
    - name = X
Translation into QVT-R

transformation checkPre(Source1:uml; Source2:uml) {
  top relation NoRedefinedAttrs{
    X : String;
    domain Source1 pa : Package{
      classes = p : Class{
        attribute = a : Attribute{ name=X }
      },
      classes = c : Class{
        attribute = ar : Attribute{ name=X }
      }
    }
    checkonly domain Source2 pa2 : Package();
    when{
      c<>p;
      ar<>a;
      c.general->includes(p); 
    } where{ false; } 
  }
}
transformation checkInv(Source:uml; Target:rel){
  top relation PackageAndSchema{
    Y: String;
    domain Source p: Package { name = Y };
    checkonly domain Target s: Schema { name = Y };
  }
}

top relation NoTableForTransientClass{
  X : String;
  domain Source p : Package {
    classes = c:Class {
      name = X,
      isPersistent = false
    }
  }
  checkonly domain Target s : Schema{};
  when { PackageAndSchema(p,s); }
  where { not NoTableForTransientClass2(p,s,X); }
}

relation NoTableForTransientClass2{
  X : String;
  domain Source p : Package {};
  checkonly domain Target s : Schema {
    tables = t:Table { name = X }
  }
  primitive domain X2: String;
  where { X = X2; }
}
} // end of transformation
Framework

[Diagram of the Framework process]

1. Specification of Requirements
   - GMF-based Editor
   - PaMoMo Model
   - ATL Engine

2. Generation of Concrete Syntax
   - QVT MM
   - ATL Transformation
   - QVT Model
   - XPand Transformation
   - XPand Engine
   - QVT Code

3. Configuration
   - Verification Editor
   - QVT Engine ModelMof
   - Test Suite
   - Verification Log

PaCo-Checker

implies:

- PaMoMo MM
- ATL
- QVT MM
- QVT Model
- XPand
- QVT Code
Framework

Specify your Verification Job

- **Source Metamodel and Model**: Select the source metamodel and an according source model.
- **Target Metamodel and Model**: Select the target metamodel and an according target model.

**Preconditions**
- Specify the preconditions that need to be fulfilled in order to execute a transformation.
- Validate preconditions.

**Invariants**
- Specify the invariants of the transformation.
- Validate invariants.

**Postconditions**
- Specify the properties that need to hold after the transformation.
- Validate postconditions.

**Specification of Metamodels and Models**
- Specification of Preconditions
- Specification of Invariants
- Specification of Postconditions
Framework

Test Input Model

Resulting Output Model

PAMoMo Contract

QVT Code

ModellMorf

Verification Results

Execution of Invariant 'Req1:' invariant succeeded
Execution of Invariant 'Req2:' invariant succeeded
Execution of Invariant 'Req3:' invariant succeeded
Execution of Invariant 'Req4:' invariant failed
Counter example:
Match not found for:
Package pa="University:Package"
pa.classes c="Student:Class"
Student.isPersistent T=true
Student.name C="Student"
pa.classes p="Person:Class"
p.attributes a="name:Class"
name.name A="name"
relation "Check.InheritedAttributes" does not hold.

Verification Log
QVT-Relations

Use constraints to express the semantics of QVT-Relations specifications in check-only mode.

\[
\text{PackageSchema (top)}
\]

\[
\langle\text{domain}\rangle \\
p: \text{Package} \\
\text{name=} X \\
persistent=true \\
\\n\text{where} \\
Y=\text{`S_`+X};
\]

\[
\text{AttributeColumn}
\]

\[
\langle\text{domain}\rangle \\
c: \text{Class} \\
\text{name=} X \\
type=T \\
\\n\text{where} \\
X1=X; \\
\text{Collection\{`int`,`float`,`double`\}.exists(z)\} \\
z=T \text{ and } T1=`\text{NUMBER}` \text{ or } (T=`\text{string}` \text{ and } T1=`\text{VARCHAR}`);
\]

\[
\text{ClassTable (top)}
\]

\[
\langle\text{domain}\rangle \\
p: \text{Package} \\
\text{name=} X \\
persistent=true \\
\\nc: \text{Class} \\
\text{name=} X \\
persistent=true \\
\\nt: \text{Table} \\
\text{name=} Y \\
\\n\text{when} \\
\text{PackageSchema (p, s);} \\
\text{where} \\
Y=`T_`+X; \\
\text{AttributeColumn (c, t);}
QVT-Relations
QVT-Relations
when and where dependencies

"An algebraic semantics for QVT-Relations check-only transformations". 2012. E. Guerra and J. de Lara. Fundamenta Informaticae (IOS Press), Volume 114. pp.: 1-29
Scenarios.

An inter-modelling specification language.

Applications

Conclusions and Future Work.
Conclusions

- A formal, declarative approach for inter-modelling.

- Different satisfaction notions for different scenarios.

- A single specification is able to generate operational mechanisms for several scenarios:
  - Batch forward and backwards transformation.
  - Model Matching.
  - Model Traceability.
  - Check-only scenario (fwd/bck semantics), useful for automated testing.

- Compilation into TGGs or EOL/OCL.

- Tool support based on Eclipse and Epsilon.

- Applications: inter-modelling, transformation specification, basis for other languages.
Future Work

- Further analysis methods for specifications.

- Use of constraint solvers for analysis of specifications.
  - UMLtoCSP, Alloy (satisfiability).

- Generation of operational code for forward/backwards transformations.
  - Integration of rewriting and constraint solvers.

- Generation of models for model transformation testing.
Bibliography


Applications:

- “An algebraic semantics for QVT-Relations check-only transformations”. 2012. E. Guerra and J. de Lara. Fundamenta Informaticae (IOS Press), Volume 114. pp.: 1-29
Thanks!

Questions?

Juan de Lara, Esther Guerra
{Juan.deLara, Esther Guerra}@uam.es

With acknowledgements to our collaborators on these topics: