Constraint-Aware Model Transformations

Adrian Rutle$^1$, Alessandro Rossini$^2$, Yngve Lamo$^1$, Uwe Wolter$^2$

$^1$Faculty of Engineering, Bergen University College, Norway

$^2$Department of Informatics, University of Bergen, Norway

16 October 2009
NWPT 2009, Lyngby, Denmark
Outline

1. Introduction and Motivation
   - Motivating Example

2. Constraints and Model Transformation

3. Summary and Future Work
MDE and model transformation

- Generation of target models from source models
- Model Transformation (MT) plays a central role in MDE
- Application areas in MDE:
  - Development process: code generation, refinement etc
  - Model management: integration, decomposition etc
  - Migration: from a platform, an implementation technology etc, to another
  - Language translation: from a modelling language to another
What is model transformation?

- $L_S, L_T$ source and target modelling languages
  - Usually graph-based languages e.g. UML or EMF
  - Have a corresponding metamodel $MM_S, MM_T$

- $S, T$ source and target models
  - Specified by the modelling languages
  - Conform to the metamodel of the languages

\[
\begin{align*}
MM_S & \xrightarrow{\text{rules}} MM_T \\
S & \xrightarrow{MT} T
\end{align*}
\]
Challenge
Constraint-Aware Model Transformations

Introduction and Motivation

Challenge

- **Class**
- **Property**
  - lower: Int
  - upper: Int
- **Association**

**MM**

**M**

(a) **Person**

(b) **childOf**

2..2

**Structural constraints**

**Attached OCL constraints**

context Person
inv: Irreflexive
self.ChildOf->excluding(self)
Challenge
Our contribution

- Diagrammatic modelling framework
  - Integration of constraints in modelling
- Enables support for specification of constraint-aware rules
Modelling formalisms

- Modelling language \( \text{formalisation} \) → modelling formalism in DPF
- A modelling formalism \((\Sigma_1, S_2, \Sigma_2)\)

\[
\begin{align*}
\Sigma_2 & \rightarrow C^{S_2} G^{S_2} \\
\Sigma_1 & \rightarrow C^{S_1} G^{S_1}
\end{align*}
\]
Constraint-aware model transformation

(a) source

\[ \Sigma_S^2 \quad C^{S_2} \quad G^{S_2} \]
\[ \Sigma_S^1 \quad C^{S_1} \quad G^{S_1} \]
\[ S_0 \]

(c) target

\[ G^{T_2} \quad C^{T_2} \quad \Sigma_T^2 \]
\[ G^{T_1} \quad C^{T_1} \quad \Sigma_T^1 \]
\[ T_0 \]
Constraint-aware model transformation

(a) source

(b) joined

(c) target

\[ \Sigma^S_2 \rightarrow \sum J \rightarrow \sum^T_2 \]

\[ \Sigma^S_1 \rightarrow \sum J \rightarrow \sum^T_1 \]

\[ C^{S_2} \rightarrow C^{J_2} \rightarrow C^{T_2} \]

\[ C^{S_1} \rightarrow C^{T_1} \]

\[ S_0 \rightarrow T_0 \]
Outline

1. Introduction and Motivation
   - Motivating Example

2. Constraints and Model Transformation

3. Summary and Future Work
Recall the motivating example

UML class diagram
Recall the motivating example

**UML class diagram**

- **Employee**
  - *employees
- **Enrolment**
  - *enrolments
- **Department**
  - 1.*departments
  - department 1
- **Project**
  - 1.project

**OCL**

```ocl
class Enrolment
  inv rule6: self.department.employees -> includesAll(self.employee)
  inv rule7: Let enrolments: Set(Enrolment) = Enrolment.allInstances in
  (not enrolment -> exists(enr |
    enr.project = self.project
    and enr.department = self.department
    and enr.employees = self.employees))
```
Recall the motivating example

**UML class diagram**

**OCL**

```ocl
class Enrolment
  inv rule6: self.department.employees -> includesAll(self.employee)
  inv rule7: Let enrolments: Set(Enrolment) = Enrolment.allInstances in
    (not enrolment -> exists(enr | enr.project = self.project
      and enr.department = self.department
      and enr.employees = self.employees))
```

**DPF**
Recall the motivating example

Source: structural model

\( \sum_{\text{struct}} \text{-specification} \)
Recall the motivating example

**Source: structural model**

\[ \Sigma_{\text{struct}-\text{specification}} \]

**Target: relational model**

\[ \Sigma_{\text{rel}-\text{specification}} \]
## Signature $\Sigma_{rel}$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\alpha_{rel}(p)$</th>
<th>Proposed visualisat.</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>[primary-key]</td>
<td><img src="primary-key.png" alt="Diagram" /></td>
<td><img src="primary-key.png" alt="Diagram" /></td>
<td>$f$ is $\text{[mult}(1,1)]$, $\text{[injective]}$</td>
</tr>
<tr>
<td>[foreign-key]</td>
<td><img src="foreign-key.png" alt="Diagram" /></td>
<td><img src="foreign-key.png" alt="Diagram" /></td>
<td>$f(X) \subseteq g(Y)$</td>
</tr>
<tr>
<td>[image-equal]</td>
<td><img src="image-equal.png" alt="Diagram" /></td>
<td><img src="image-equal.png" alt="Diagram" /></td>
<td>$f(X) = g(Z)$</td>
</tr>
</tbody>
</table>
Outline

1. Introduction and Motivation
   - Motivating Example

2. Constraints and Model Transformation

3. Summary and Future Work
Steps

1. Join the modelling formalisms
2. Define transformation rules
3. Extend $S$ to be typed by the joined metamodel
4. Apply the transformation rules
5. Projection of $T$
Joined modelling formalism

- Create \((\Sigma^J_1, J_2, \Sigma^J_2)\)
- \(\Sigma^J_1 := \Sigma^S_1 \cup \Sigma^T_1\) and \(\Sigma^J_2 := \Sigma^S_2 \cup \Sigma^U_2 \cup \Sigma^T_2\)
- \(J_2 := S_2 \cup U_2 \cup T_2\)

\[
\Sigma^J_1 := \Sigma^S_1 \cup \Sigma^T_1 \quad \text{and} \quad \Sigma^J_2 := \Sigma^S_2 \cup \Sigma^U_2 \cup \Sigma^T_2
\]
Joined modelling formalism

- Create \((\Sigma^J_1, J_2, \Sigma^J_2)\)
- \(\Sigma^J_1 := \Sigma^S_1 \cup \Sigma^T_1\) and \(\Sigma^J_2 := \Sigma^S_2 \cup \Sigma^U_2 \cup \Sigma^T_2\)
- \(J_2 := S_2 \cup U_2 \cup T_2\)
Example of join

\[ \Sigma_{\text{struct}} \]

\[ \Sigma^S_2 \]

\[ \Sigma_{\text{struct}} \cup \Sigma_{\text{rel}} \]

\[ \Sigma^J_2 \]

\[ \Sigma_{\text{rel}} \]

\[ \Sigma^T_2 \]

\[ R \]

\[ C \]

\[ A \]

\[ DT \]

\[ S_2 \]

\[ R \]

\[ C \]

\[ A \]

\[ DT \]

\[ \Sigma_2 \]

\[ \Sigma^J_2 \]

\[ T \]

\[ Col \]

\[ Ty \]

\[ J_2 \]

\[ T \]

\[ Col \]

\[ Ty \]

\[ T_2 \]

\[ \Sigma_{\text{struct}} \]

\[ \Sigma^S_1 \]

\[ \Sigma_{\text{struct}} \cup \Sigma_{\text{rel}} \]

\[ \Sigma^J_1 \]

\[ \Sigma_{\text{struct}} \cup \Sigma_{\text{rel}} \]

\[ \Sigma^T_1 \]
Signature $\Sigma_{\text{join}}$

[commutative]

\[
\forall x \in X : g'(f(x)) = f'(g(x))
\]
Projection condition

\[ G^{S_2} \xrightarrow{u} G^{J_2} \xleftarrow{v} G^{T_2} \]

- \text{[bijective] on arrows in } G^{U_2}
Projection condition

\[ G^{S_2} \rightarrow^u G^{J_2} \leftarrow^v G^{T_2} \]
\[ G^{S_1} \rightarrow^{\pi_{S_1}} G^{J_1} \leftarrow^{\pi_{T_1}} G^{T_1} \]

- [bijective] on arrows in \( G^{U_2} \)
Define constraint-aware rules

Both \( L \) and \( R \) are \( G^{J2} \)-typed \( \Sigma^J_1 \)-specifications

\( r \) is a \( G^{J2} \)-typed \( \Sigma^J_1 \)-specification morphism

Application of \( r \) via a match \( m \)

Specification Morphism
Define constraint-aware rules

- Both $L$ and $R$ are $G^{J_2}$-typed $\Sigma^{J_1}_1$-specifications
- $r$ is a $G^{J_2}$-typed $\Sigma^{J_1}_1$-specification morphism
- Application of $r$ via a match $m$

Application of Rule

\[
\begin{array}{c}
L \\ \\
\downarrow m \\
J_1 \\
\end{array} \xrightarrow{r} \begin{array}{c}
\langle r, m \rangle \\
P.O. \\
m^* \\
J_1' \\
\end{array} \xrightarrow{m^*} \begin{array}{c}
R \\
\end{array}
\]
Constraint-aware rules: \( r_1 \) and \( r_2 \)

<table>
<thead>
<tr>
<th>Rule ( r_1 ). Class to table</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c:C )</td>
</tr>
<tr>
<td>( a:A )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule ( r_2 ). Attribute to column</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c:C \rightarrow t:T \rightarrow [pk] ty:Ty )</td>
</tr>
<tr>
<td>( a:A \rightarrow dt:DT \rightarrow [pk] ty_2:Ty )</td>
</tr>
</tbody>
</table>
Rule $r_3$: Reference to foreign key

- Multiplicity one-to-many
Rule $r_4$: Reference to foreign key

- Multiplicity many-to-many
Rule $r_5$: Reference to foreign key

- Multiplicity many-to-many and surjective
One rule application to $J_1$
Applying all the rules to this $S_1$...
... will lead to this $T_1$ after projection
Outline

1. Introduction and Motivation
   - Motivating Example

2. Constraints and Model Transformation

3. Summary and Future Work
Related work

- **Graph Transformation System (GTS)** *Hartmut Ehrig et al*
  - Adding support for transforming constraints
- **Triple Graph Grammar (TGG)** *Schürr et al*
  - Adding support for constraining the joined modelling formalism
- **Refactoring** *Fondement and Baar*
  - Transformation of constraints in refactoring UML/OCL models
  - Adding support for a more general case
- **VTMS** *Lengyl and Levendovzsky*
  - Validation of transformations by means of OCL constraints
  - Puts constraints on rules to narrow down matches
  - A match is a metamodel instantiation
- **Analyses of GTS** *Cabot et al*
  - Rules and their properties e.g. conflicts and rule applicability to OCL expressions
Summary

- DPF: a formal diagrammatic specification framework
  - Integration of constraints in modelling
- Extending GTS and TGG: supporting transformation of attached constraints
Future work

- Controlling and scheduling transformation rules
- Dependency between predicates
- Signature morphisms: “better” joining
- Prototype tool
- Comparison to other tools: AGG, GReAT, PROGRESS, VMTS, Epsilon, Fujaba etc
- Real-size case study: noark (Norwegian Archive standard)
Thank you!

Questions?
Instances

Instances of Predicates
A semantic interpretation of a signature $\Sigma = (\Pi, \alpha)$ is given by a mapping that assigns to each $p \in \Pi$ a set $[[p]]$ of graph homomorphisms $\tau : O \rightarrow \alpha(p)$ called valid instances of $p$, written $\tau \models p$, where $O$ may vary over all graphs.

Instance of Specification
An instance of a diagrammatic specification $S = (G^S, C^S)$ is a graph $I$ together with a graph homomorphism $\iota : I \rightarrow G^S$, written $(I, \iota)$, such that for each constraint $(p, \delta) \in C^S$ we have $\iota^* \in [[p]]$, where $\iota^* : O^* \rightarrow \alpha(p)$ is given by the following pullback diagram

\[
\begin{array}{ccc}
\alpha(p) & \xrightarrow{\delta} & G^S \\
\downarrow^{\iota^*} & & \downarrow_{\iota} \\
O^* & \xrightarrow{\delta^*} & I
\end{array}
\]
Semantics of diagrammatic specification
Challenges in modelling

- Mixing graph-based structures with textual constraints
  - Different technical spaces
    - checking models in two different engines/steps
    - model-constraint synchronisation problem
    - violation of “everything-is-a-model” vision of MDE
- Challenge for domain experts who do not understand OCL
Formalisation approach

- Based on category theory
  - Sketches formalism: define semantics of diagrams (thus graph-based models)
    - models: graphs (nodes and edges)
    - model properties: universal properties (limit, colimit, commutative diagrams)
  - Generalized sketches formalism
    - not only universal properties
    - user-defined diagrammatic predicate signatures
- DPF: specification formalism based on generalized sketches