Automatic Definition of Model Transformations at Instance Level

Adrian Rutle\textsuperscript{1}, Alessandro Rossini\textsuperscript{2}, Yngve Lamo\textsuperscript{1}, Uwe Wolter\textsuperscript{2}

\textsuperscript{1}Faculty of Engineering, Bergen University College, Norway
\textsuperscript{2}Department of Informatics, University of Bergen, Norway

20 November 2008
Automatic Definition of Model Transformations at Instance Level

Outline

1. Introduction and Motivation
   - MDE and Model Transformations
   - Diagram Predicate Framework (DPF)

2. DPF and Model Transformation
   - DPF and Model Transformation
   - Automatisation of MT at Instance-Level
Model Driven Engineering (MDE)

- Engineering techniques where models are first-class entities
- Evolved from the popularity of diagrammatic languages such as UML and ER and their usage for specification and documentation of software systems
- Aims to raise the abstraction level of software development from code to models
- Uses model transformations for code generation, model refinement etc
4 Abstraction Levels in MDE

<table>
<thead>
<tr>
<th>OMG levels</th>
<th>OMG Standards/examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_3$: Meta-metamodel</td>
<td>MOF</td>
</tr>
<tr>
<td>$M_2$: Metamodel</td>
<td>UML language</td>
</tr>
<tr>
<td>$M_1$: Model</td>
<td>A UML model: Class “Person” with attributes “name” and “address”</td>
</tr>
<tr>
<td>$M_0$: Instance</td>
<td>An instance of “Person”: “Ola Nordmann” living in “Sotraveien 1, Bergen”</td>
</tr>
</tbody>
</table>
Model Transformations

- Model Transformation (MT) plays a central role in MDE
- Generation of target models from source models
- Application areas in MDE:
  - Development process: code generation, refinement etc
  - Model management: integration, decomposition etc
  - Migration: from a platform, implementation technology, programming language etc, to another
  - Technology mappings: e.g. JPA where Java classes are mapped to Relational tables, objects to rows in database tables
Model Transformations cont…

Definition at Metamodel level, application at Model level

$$M_2 \xrightarrow{MT-def.} M \xrightarrow{MT-exec.} M'$$

$$\text{Inst}(MM) \ni \iota_{MM}$$

$$\text{Inst}(MM') \ni \iota_{MM'}$$

$$MT-exec \approx \text{Apply } MT\text{-engine}(MT\text{-def})$$

Given $MT\text{-def}$ and $M$

We get $MT\text{-exec}(M)$ and $(M \xrightarrow{MT\text{-exec}(M)} M')$

We want also $M \xrightarrow{MT\text{-def}'(M)} M'$ so that …
Model Transformations cont...

Definition at Metamodel level, application at Model level
Automatic definition at Model level, application at Instance level

\[ M_2 \xrightarrow{\text{Inst}(MM) \ni \iota_{MM}} MM \xrightarrow{MT-\text{def.}} MM' \]
\[ M_1 \xrightarrow{\text{Inst}(M) \ni \iota_M} M \xrightarrow{MT-\text{exec.}} M' \xrightarrow{Auto.} M' \xrightarrow{MT-\text{engine}} I' \]

\[ I \xrightarrow{MT-\text{exec.}} I' \xrightarrow{I' \ni \iota_{M'} \in \text{Inst}(MM')} MM' \xrightarrow{MT-\text{def.}} MM \]
MDE and MT: State of the Art

- In general, modelling languages are diagrammatic because of graph-based nature of software models.
- Diagrammatic modelling languages: semi-formal (if formalised, not fully diagrammatic).
In general, modelling languages are diagrammatic because of graph-based nature of software models.

Diagrammatic modelling languages: semi-formal (if formalised, not fully diagrammatic)

Informal MT-languages, leading to:
- Correctness is difficult to prove
- No automatic reasoning
- Changes in MT-definitions requires re-transformation

Automatic definition at instance level is not possible yet

Diagrammatic constraints are not taken into account in MT
Outline

1. Introduction and Motivation
   - MDE and Model Transformations
   - Diagram Predicate Framework (DPF)

2. DPF and Model Transformation
   - DPF and Model Transformation
   - Automatisation of MT at Instance-Level
We use DPF as a diagrammatic approach for the formalisation of MDE.

Aims to combine the intuition from graphical modeling languages with the semantic rigor of formal methods.

Based on Sketches/Category theory.

Potentials to combine the machinery from category theory with first order logic.

Adaptation of FOL to diagrammatic specifications.
DPF’s Role in MDE and MT

- Diagrammatic: no need for mixing text with diagrams
- Enable diagrammatic reasoning
- Formalisation of MT
- Categorisation of MT based on their properties
- Automatic reasoning about models and MT
- Formalisation of the requirements for the automatisation of MT at instance-level
Models and their Instances in DPF

- An instance \((\iota_M, I)\) of a diagrammatic specification \(M\) is a graph homomorphism
- The elements of \(I\) are typed by \(G(M)\), the graph of \(M\)
- The constraints in \(M\) are satisfied
- \(\text{Inst}(M)\) denotes the set of all instances of \(M\)
Outline

1 Introduction and Motivation
   - MDE and Model Transformations
   - Diagram Predicate Framework (DPF)

2 DPF and Model Transformation
   - DPF and Model Transformation
   - Automatisation of MT at Instance-Level
DPF and Model Transformations

Recall that $MT : Inst(MM) \Rightarrow Inst(MM')$

An approach: model transformation definition (MT-def.) as a set of rules and a control

**Definition**

A model transformation is a set $MT = \{ r_j, \llbracket r_j \rrbracket \}$, where $r_j : P_j \leftarrow K_{r_j} \rightarrow P'_j$ is the set of transformation rules’ declarations and $\llbracket r_j \rrbracket : \text{Match}(P) \rightarrow \text{Match}(P')$ is the semantics of $r_j$. 
Transformation Rules

A rule $r : P \leftarrow K \rightarrow P'$
A pattern $P$
A match of $P$, $m(P)$
$\text{Match}^M(P) = \{ m | m : P \rightarrow M \}$
$\text{Match}(P) = \{ m \in \text{Match}^M(P) | \forall (M, \iota_{MM}) \in \text{Inst}(MM) \}$
Outline

1. Introduction and Motivation
   - MDE and Model Transformations
   - Diagram Predicate Framework (DPF)

2. DPF and Model Transformation
   - DPF and Model Transformation
   - Automatisation of MT at Instance-Level
Transformation Rules at Metamodel Level

\[ P \xleftarrow{\iota_P} K \xrightarrow{\text{in}_P} P' \]
\[ M \xleftarrow{\iota_{MM}} MM \]
\[ M' \xrightarrow{\iota_{MM'}} MM' \]
Automatic Construction of Transformation Rules
Automatic Construction of Transformation Rules

**Definition**

$f : \text{Match}^M(P) \rightarrow \mathcal{P}^M$ is a construction such that, $\forall m(P) \in \text{Match}^M(P)$, $f(m(P)) = P_m$ where $P_m$ is a pattern over $M$ such that $id(P_m) = P$ and $id; m = \iota_{P_m}$.

**Figure:** Creation of the pattern $P_m \in \mathcal{P}^M$ for a rule $r_i$ based on the match $m$ of $P$ in $M$. 

Automatic Definition of Model Transformations at Instance Level

DPF and Model Transformation

Automatisation of MT at Instance-Level
Summary

- MDE is a promising approach for software development
- Models and MT are important artefacts in MDE
- Models are graph-based (or diagrammatic) structures
- Category Theory as a methodological guideline for diagrammatic reasoning
- DPF “is” Category Theory for software engineers
- MT in DPF transforms also diagrammatic constraints
- Analysis of requirements for automatic specification of MT-def at model level based on the MT-def at metamodel level