Automatic Definition of Model Transformations at Instance Level

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

20 November 2008

HØGSKOLEN I BERGEN

Outline

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

$$\begin{tikzcd}
M_3 \arrow[loop above, "conformsTo"] \\
M_2 \arrow[u, "conformsTo"] \\
M_1 \arrow[u, "conformsTo"] \\
M_0 \arrow[u, "conformsTo"]
\end{tikzcd}$$

OMG levels	**OMG Standards/ examples**
M_3: Meta-metamodel	MOF
M_2: Metamodel	UML language
M_1: Model	A UML model: Class “Person” with attributes “name” and “address”
M_0: Instance	An instance of “Person”: “Ola Nordmann” living in “Sotraveien 1, Bergen”

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

$$
\begin{tikzcd}
M_2 & MM \arrow[rr, "MT-def."'] \arrow[drr, phantom, "\Downarrow MT-engine"] & & MM' \\
M_1 & M \arrow[u, "Inst(MM) \ni \iota_{MM}"] \arrow[rr, Rightarrow, "MT-exec."'] & & M' \arrow[u, "\iota_{MM'} \in Inst(MM')"']
\end{tikzcd}
$$

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

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

We want also $M \xrightarrow{MT-def'(M)} M'$ so that ...

Model Transformations cont...

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

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
M_2 & MM \arrow[r, "MT-def."'] \arrow[dr, phantom, "\Downarrow MT-engine"] & MM' \\
M_1 & M \arrow[u, "Inst(MM)\ni\iota_{MM}"] \arrow[r, Rightarrow, "MT-exec."'] \arrow[r, bend right=50, "MT-def."'] \arrow[r, phantom, shift right=3ex, "\Downarrow \color{blue}{Auto.}"] \arrow[dr, phantom, "\Downarrow MT-engine"] & M' \arrow[u, "\iota_{MM'}\in Inst(MM')"'] \\
M_0 & I \arrow[u, "Inst(M)\ni\iota_{M}"] \arrow[r, Rightarrow, "MT-exec."'] & I' \arrow[u, "\iota_{M'}\in Inst(M')"']
\end{tikzcd}
$$

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)

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)
- 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

Diagram Predicate Framework (DPF)

- 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

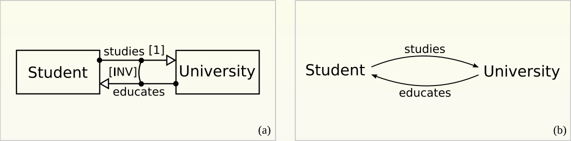

Figure: (a) $M = (G(M), M(\Pi))$, (b) its graph $G(M)$.

- An instance (ι_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
- $Inst(M)$ denotes the set of all instances of M

$$\begin{tikzcd} \alpha(p) \arrow[r, "\delta"] & G(M) \\ & I \arrow[u, "\iota_M"'] \end{tikzcd}$$

Outline

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 $MT = \{r_j, [\![r_j]\!]\}$, where $r_j : P_j \hookleftarrow K_{r_j} \hookrightarrow P'_j$ is the set of transformation rules' declarations and $[\![r_j]\!] : Match(P) \to Match(P')$ is the semantics of r_j.

Transformation Rules

$$
\begin{tikzcd}
 & P \arrow[dl, "\iota_P"'] \arrow[d, "m"] & K \arrow[l, hook', "in_P"'] \arrow[r, hook, "in_{P'}"] & P' \arrow[d] \arrow[dr, "\iota_{P'}"] & \\
MM \arrow[r, phantom, "{[=]}"] & M \arrow[l, "\iota_{MM}"] & m \arrow[r, mapsto, "[\![r]\!]"] & m' & M' \arrow[r, "\iota_{MM'}"'] \arrow[r, phantom, "{[=]}"] & MM'
\end{tikzcd}
$$

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

Outline

Transformation Rules at Metamodel Level

$$
\begin{tikzcd}
 & P \arrow[dl, "\iota_P"'] \arrow[d, "m"] \arrow[dl, phantom, shift left=4ex, "{[=]}"] & K \arrow[l, hook', "in_P"'] \arrow[r, hook, "in_{P'}"] & P' \arrow[d, "m'"'] \arrow[dr, "\iota_{P'}"] \arrow[dr, phantom, shift right=4ex, "{[=]}"] & \\
MM & M \arrow[l, "\iota_{MM}"] & & M' \arrow[r, "\iota_{MM'}"'] & MM'
\end{tikzcd}
$$

$$m \xmapsto{[\![r]\!]} m'$$

Automatic Construction of Transformation Rules

$$
\begin{tikzcd}
& P \arrow[dl, "\iota_P"'] \arrow[d, "m"] & K \arrow[l, hook', "in_P"'] \arrow[r, hook, "in_{P'}"] & P' \arrow[d, "m'"'] \arrow[dr, "\iota_{P'}"] & \\
MM \arrow[r, phantom, bend left, "{[=]}"] & M \arrow[l, "\iota_{MM}"] \arrow[d, Rightarrow, "f"] \arrow[rr, phantom, "m \overset{\llbracket r \rrbracket}{\longmapsto} m'"] & & M' \arrow[r, "\iota_{MM'}"'] \arrow[d, Rightarrow, "g"'] \arrow[r, phantom, bend left, "{[=]}"] & MM' \\
{} \arrow[r, phantom, "{[=]}"] & P_m \arrow[d, bend left, "n"] & K_{r_i} \arrow[l, hook'] \arrow[r, hook] & P'_{m'} \arrow[d, bend right, "n'"'] \arrow[r, phantom, "{[=]}"] & {} \\
& I \arrow[uul, "\iota_M;\iota_{MM}"] \arrow[uu, bend left, "\iota_M"'] \arrow[rr, phantom, "n \overset{\llbracket r_i \rrbracket}{\longmapsto} n'"] & & I' \arrow[uu, bend right, "\iota_{M'}"] \arrow[uur, "\iota_{M'};\iota_{MM'}"'] &
\end{tikzcd}
$$

Automatic Construction of Transformation Rules

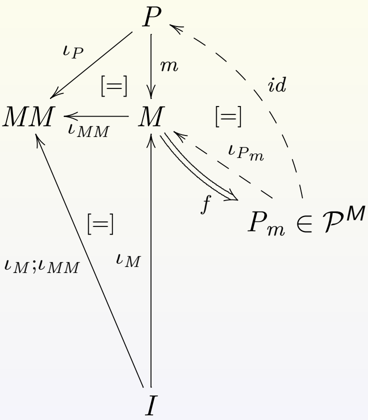

Figure: Creation of the pattern $P_m \in \mathcal{P}^M$ for a rule r_i based on the match m of P in M.

Definition

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

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