A Formal Approach to Modeling and Model Transformations in Software Engineering

Adrian Rutle

Department of Computer Engineering
Faculty of Engineering
Bergen University College

27.05.2008
Formal Methods 2008, Doctoral Symposium, Turku, Finland
Research Area

- Research on MDD and MDD-related standards and approaches
- Definition of a formal framework to express formal models and model transformations
Outline

1. Introduction and Motivation
   - Model-driven Development (MDD)
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL

3. OMG standards and Metamodeling Levels
   - Metamodelling

4. Model Transformations

5. Tools and Contributions
What is MDD?

- Software development methodology in which models are first-class entities
- Development of software is started by building platform-independent models
- Models are used as input to model transformation tools
- Code generation can be seen as model transformation
MDD vs Traditional Development Processes

Traditional
- Requirements
  - Mostly text
- Analysis
  - Diagrams and text
- System Design
  - Diagrams and text
- Implementation
  - Code
- Testing
  - Code
- Deployment
  - Code

MDA
- Requirements
  - Mostly text
- Analysis
  - PIM
- System Design
  - PSM
- Implementation
  - Code
- Testing
  - Code
- Deployment
  - Code

MDA
- Requirements
  - Mostly text
- Analysis
- System Design
  - PIM
  - PSM
- Implementation
  - PSM
- Testing
  - Code
- Deployment
  - Code

PIM: Platform Independent Model
PSM: Platform Specific Model
What is Important for MDD?

Models are no longer only for documentation
Models should be **both**:

- easy for **humans** to intercept and reason about, and
- formal, such that different **machines** are able to understand them uniquely

⇒ Diagrammatic approach for formalization of modeling and transformation languages

Our approach ⇒ Diagrammatic Predicate Logic Framework
What is Important for MDD?

Models are no longer only for documentation
Models should be both:
- easy for humans to intercept and reason about, and
- formal, such that different machines are able to understand them uniquely

⇒ Diagrammatic approach for formalization of modeling and transformation languages

Our approach ⇒ Diagrammatic Predicate Logic Framework
What is Important for MDD?

Models are no longer only for documentation
Models should be **both**: 
- easy for **humans** to intercept and reason about, and
- formal, such that different **machines** are able to understand them uniquely

⇒ Diagrammatic approach for formalization of modeling and transformation languages

Our approach ⇒ Diagrammatic Predicate Logic Framework
Outline

1. Introduction and Motivation
   - Model-driven Development (MDD)
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL

3. OMG standards and Metamodeling Levels
   - Metamodelling

4. Model Transformations

5. Tools and Contributions
What is DPL?

- Graph-based specification format based on FOL and Category Theory
- In DPL, categorical sketches are generalized and adapted for modeling in software engineering
- DPL can be used as a mathematical formalism to formalize modeling languages and transformations between them
- \( \Rightarrow \) Investigate, Analyze, Adapt, Verify and Evaluate DPL
- Language-independent transformations
- Graph-based logic \( \Rightarrow \) compact relation between syntax and semantics
- Support for model integration and de-composition
- Support for reasoning about models and transformations
Concepts in DPL

- Diagrammatic predicate signatures = predicate names + arity requirements
- Diagrammatic Specifications = graph + constraints
### Signature – Example

<table>
<thead>
<tr>
<th>Π</th>
<th>α</th>
<th>visualization</th>
<th>intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>[total]</td>
<td>1 → 2</td>
<td>[ \text{A} \xrightarrow{f} \text{B} ]</td>
<td>( \forall a \in A :</td>
</tr>
<tr>
<td>[key]</td>
<td>1 → 2</td>
<td>[ \text{A} \xrightarrow{f \ [\text{key}]} \text{B} ]</td>
<td>( \forall a, a' \in A : a \neq a' \implies f(a) \neq f(a') )</td>
</tr>
<tr>
<td>[inv]</td>
<td>1 ( \xrightarrow{x} ) 2 ( \xleftarrow{y} )</td>
<td>[ \text{A} \xrightarrow{[\text{inv}]} \text{B} ]</td>
<td>( \forall a \in A, \forall b \in B : b \in f(a) \iff a \in g(b) )</td>
</tr>
<tr>
<td>[jointly-key] or [1-1]</td>
<td>1 ( \xrightarrow{x} ) 2 ( \xleftarrow{y} ) 3</td>
<td>[ \text{A} \xrightarrow{f} \text{B} \xleftarrow{g} \text{C} ]</td>
<td>( \forall a, a' \in A : a \neq a' \implies f(a) \neq f(a') ) or ( g(a) \neq g(a') )</td>
</tr>
</tbody>
</table>
Diagrammatic Specification – Example

Person

- name

Company

- address
- c_name

Employment

- worksFor
- hires

- employee
- employer

- start_date
- salary

[Person

[Int]

[Date]
Diagrammatic Specification – Example

Person

Employment

Company

Constraints S(Pi):

[inv]

[1-1]
Our Contributions so far

- Analysis of modeling techniques and MDD approaches/tools in view of DPL: EMF, QVT, Viatra, oAW etc
- Revision and development of DPL
- Formalization of metamodeling in view of DPL
- First studies of model transformations
- Tools development
Outline

1. Introduction and Motivation
   - Model-driven Development (MDD)
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL

3. OMG standards and Metamodeling Levels
   - Metamodeling

4. Model Transformations

5. Tools and Contributions
### OMG Metamodel Levels

<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 &quot;Person&quot; with attributes &quot;name&quot; and &quot;address&quot;</td>
</tr>
<tr>
<td>$M_0$: Instance</td>
<td>An instance of &quot;Person&quot;: &quot;Ola Nordmann&quot; living in &quot;Sot raveien 1, Bergen&quot;</td>
</tr>
</tbody>
</table>
DPL Methodology and Modeling Formalisms

- Modeling Formalism $F_2 = (\Sigma_1, M_2, \Sigma_2)$
- Meta-formalism $F_3 = (\Sigma_2, M_3, \Sigma_3)$
- $F_3$ is reflexive iff $(\Sigma_3 \subseteq \Sigma_2)$ and $t_{M_2} : G(M_2) \rightarrow G(M_3)$ is instance of $M_3$. 
DPL Methodology and Modeling Formalisms

- Modeling Formalism $F_2 = (\Sigma_1, M_2, \Sigma_2)$
- Meta-formalism $F_3 = (\Sigma_2, M_3, \Sigma_3)$
- $F_3$ is reflexive iff $(\Sigma_3 \subseteq \Sigma_2)$ and $t_{M_2} : G(M_2) \rightarrow G(M_3)$ is instance of $M_3$. 
Model Transformation Concepts

- Model Transformation = set of Transformation Rules + Coordinations (⇒ operational)
- Model Transaction = declarative definition of the transformation
- Transformation Engine = executes the rules
Transformation Rules

- Input and output patterns $P$ and $P'$
- Input and output instances $I$ and $I'$
- The models $M_1$ and $M_2$
- The matches $m$ and $\llbracket t \rrbracket (m)$

$$G(M_1) \xleftarrow{\iota} P \xrightarrow{\iota} I \xrightarrow{\text{trace}} I' \xrightarrow{\iota} G(M_2)$$

$$\llbracket t \rrbracket (m)$$
Transformation Rules

\[ G(M_1) \xleftarrow{\iota} I \xrightarrow{\text{trace}} I' \xrightarrow{\iota} G(M_2) \]

- Input and output patterns \( P \) and \( P' \)
- Input and output instances \( I \) and \( I' \)
- The models \( M_1 \) and \( M_2 \)
- The matches \( m \) and \( [t](m) \)
Tools

- Our tools are implemented as plug-ins to Eclipse
- Formalization of languages are done by designing diagrammatic signatures and metamodels for the languages
- Comparison and alignment of languages by definition of transformations between them
The MDD is a promising approach for software development processes.

The MDD requires tools for automatic model management.

Automatic model management requires formal modeling and model transformation languages.

Find out whether DPL can be used as a generic framework for formalization of modeling and model transformation languages.