Model Driven Engineering (MDE) and Diagrammatic Predicate Logic (DPL)

Yngve Lamo

Department of Computer Engineering
Faculty of Engineering
Bergen University College
NORWAY

06.06.2008
Institute of Mathematics and Informatics, Vilnius, LITHUANIA
Project goal

- To apply and adapt formal methods to software engineering
  - Actual research on MDE and MDE-related standards and approaches
  - Develop a formal framework to express software models and model transformations
  - Develop tools that justifies the framework
Outline

1. Introduction and Motivation
   - Personal Motivation
     - Model-driven Engineering (MDE)
     - Modeling languages state of the art
     - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL
   - Specification
   - Example
   - Instance

3. OMG standards and Metamodeling Levels
   - Metamodeling
   - DPL meta model
   - Ecore meta model in DPL

4. Tools

5. Model Transformations
History of our project:

- Ph.d in Theoretical Computer Science 2003, algebraic specifications, logic and category theory
- Discussions with professor Uwe Wolter since 2000 about how to use theory in practice
- In 2000 Wolter introduced me to the Generalized Sketches framework, created in Latvia in the 90’s, by Diskin ++
- 2003, we decided to try to use the GS approach for practical software engineering
2004 master student Ørjan Hatland started to implement a tool for GS, based on the .NET framework, (graduated 2006)

Early 2006 Ph.d student Adrian Rutle started to investigate the subject and we get interested in MDE and the Eclipse framework

2006 Master student Stian Skjervegen started to implement a graphical editor in Eclipse framework, (will graduate June 2008)
Outline

1 Introduction and Motivation
   - Personal Motivation
   - Model-driven Engineering (MDE)
     - Modeling languages state of the art
     - Diagrammatic Predicate Logic (DPL)

2 Concepts in DPL
   - Specification
   - Example
   - Instance

3 OMG standards and Metamodeling Levels
   - Metamodeling
   - DPL meta model
   - Ecore meta model in DPL

4 Tools

5 Model Transformations
Engineering techniques where models are first-class entities

Evolved from the popularity of diagrammatic languages such as UML and ER and their popularity for specification and documentation of software systems

Aims to raise the abstraction level of software development from text to models

The development process is based on software models, not on plain text
Model Driven Architecture (MDA) approach from Object Management Group (OMG) where:

- First step is building platform independent domain models, PIM’s
- PIM’s are refined to platform specific models PSM’s
- Refinement process more or less automatized as model transformations
- Code generation and model refinement can be seen as model transformations
- Reuse of software models by using model transformations
MDD vs Traditional Development Processes

- **Traditional**
  - requirements
  - analysis
  - system design
  - implementation
  - testing
  - deployment

- **MDA**
  - requirements
  - analysis
  - system design
  - implementation
  - testing
  - deployment

- **MDA**
  - requirements
  - analysis
  - system design
  - implementation
  - testing
  - deployment

**PIM**: Platform Independent Model

**PSM**: Platform Specific Model

Yngve Lamo
MDE and DPL
Key point for success for MDE

Models are no longer only for documentation

- Models should be both:
  - intuitive for **software engineers** to intercept and work with
  - formal such that **machines** could have precise understanding and reason about the models

⇒ Diagrammatic approach enhance human understanding
⇒ Challenge to combine intuition with formalization
Key point for success for MDE

Models are no longer only for documentation

- Models should be **both**:
  - intuitive for **software engineers** to intercept and work with
  - formal such that **machines** could have precise understanding and reason about the models

⇒ Diagrammatic approach enhance human understanding
⇒ Challenge to combine intuition with formalization
Key point for success for MDE

Models are no longer only for documentation

- Models should be **both**:
  - intuitive for **software engineers** to intercept and work with
  - formal such that **machines** could have precise understanding and reason about the models

⇒ Diagrammatic approach enhance human understanding
⇒ Challenge to combine intuition with formalization
Outline

1. Introduction and Motivation
   - Personal Motivation
   - Model-driven Engineering (MDE)
   - Modeling languages state of the art
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL
   - Specification
   - Example
   - Instance

3. OMG standards and Metamodeling Levels
   - Metamodeling
   - DPL meta model
   - Ecore meta model in DPL

4. Tools

5. Model Transformations
Some facts about UML

- UML is defacto industry standard for modeling
- UML can only express constraints on binary relations
- UML has serious issues regarding:
  - Semantics; UML models may be ambiguous and have semi-formal semantics
  - Complexity; UML uses 13 different types of diagrams
  - Expressibility; To express constraints over higher order relations one need to use string based logic, (Object Constraint Logic, OCL)
Formal modeling languages

- Nice semantics, proven abilities and several fundamental Software Engineering problems solved by use of formal methods
- Set based semantics (logic, algebras etc...)
- Hard for software engineers to apply in practice
- Lack of good software development tools based on formal approaches
- Only used by well trained experts
- High cost low productivity
Outline

1. Introduction and Motivation
   - Personal Motivation
   - Model-driven Engineering (MDE)
   - Modeling languages state of the art
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL
   - Specification
   - Example
   - Instance

3. OMG standards and Metamodeling Levels
   - Metamodeling
   - DPL meta model
   - Ecore meta model in DPL

4. Tools

5. Model Transformations
DPL

- Aims to combine the intuition from graphical modeling languages with the semantic rigor of formal methods
- Based on Generalized Sketches
- Sketches is a graphical representation of category theory, constraints expressed by universal constructions (limits and co-limits) of diagrams
- Generalized Sketches is sketches extended with constraints on arbitrary diagrams, not only universal constructions
- In DPL, sketches are generalized and adapted to model driven software engineering
DPL will be used as a framework for defining modeling languages and transformations between them in a formal graphical way.

We work continuously on Investigating, Analyzing, Adapting, Verifying and Evaluating DPL.

Potential to use the machinery from category to implement MDE tools.
Outline

1. Introduction and Motivation
   - Personal Motivation
   - Model-driven Engineering (MDE)
   - Modeling languages state of the art
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL
   - Specification
   - Example
   - Instance

3. OMG standards and Metamodelling Levels
   - Metamodelling
   - DPL meta model
   - Ecore meta model in DPL

4. Tools

5. Model Transformations

Yngve Lamo  MDE and DPL
Diagrammatic specifications

- **Signature**
  - Set of constraint (predicate) names $\Pi$ with
  - An arity function $\alpha : \Pi \rightarrow \text{graph}$

- **Diagrammatic Specifications** consist of
  - the underlying graph $G(S)$ together with a
  - set of constraints $C_S$ i.e. diagrams $\delta : \alpha(c_S) \rightarrow G$, where $c_S \in \Pi$
Outline

1. Introduction and Motivation
   - Personal Motivation
   - Model-driven Engineering (MDE)
   - Modeling languages state of the art
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL
   - Specification
   - Example
   - Instance

3. OMG standards and Metamodeling Levels
   - Metamodeling
   - DPL meta model
   - Ecore meta model in DPL

4. Tools

5. Model Transformations

Yngve Lamo  MDE and DPL
## 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>$1 \xrightarrow{x} 2$</td>
<td><img src="total.png" alt="Diagram" /></td>
<td>$\forall a \in A :</td>
</tr>
<tr>
<td>[key]</td>
<td>$1 \xrightarrow{x} 2$</td>
<td><img src="key.png" alt="Diagram" /></td>
<td>$\forall a, a' \in A : a \neq a' \implies f(a) \neq f(a')$</td>
</tr>
<tr>
<td>[inv]</td>
<td>$1 \xrightarrow{x} 2$</td>
<td><img src="inv.png" alt="Diagram" /></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>$1 \xrightarrow{x} 2$</td>
<td><img src="1-1.png" alt="Diagram" /></td>
<td>$\forall a, a' \in A : a \neq a' \implies f(a) \neq f(a') \text{ or } g(a) \neq g(a')$</td>
</tr>
</tbody>
</table>
Diagrammatic Specification – Example

Person

\[1\]

\[\text{key}\]

p_name

\[\text{worksFor}\]

\[\text{hires}\]

\[\text{[inv]}\]

Employment

\[1\]

Employee

[Date]

start_date

[Int]

salary

Company

\[1\]

\[\text{address}\]

\[\text{c_name}\]

[1-1]

[1-1]

[1]
Diagrammatic Specification – Example

- **Person**
  - [String] `p_name`
  - 1 `worksFor`
  - 1 `hires` [inv]
- **Company**
  - [String] `c_name`
  - 1 `address`
- **Employment**
  - 1 `employee`
  - 1 `employer`
- **Constraints S(Pi):**
  - [inv]
  - [1-1]

Graph G(S):

- `Person` -> `Employment`
- `Employment` -> `Company`
Outline

1. Introduction and Motivation
   - Personal Motivation
   - Model-driven Engineering (MDE)
   - Modeling languages state of the art
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL
   - Specification
   - Example
   - Instance

3. OMG standards and Metamodelling Levels
   - Metamodelling
   - DPL meta model
   - Ecore meta model in DPL

4. Tools

5. Model Transformations

Yngve Lamo  MDE and DPL
Instance of Diagrammatic Specification – Example
The semantic interpretation of $\Sigma := (\Pi, \alpha)$ assigns to each $p \in \Pi$ a set $\llbracket p \rrbracket$ of graph homomorphisms $\tau : O \rightarrow \alpha(p)$, where $O$ may vary over all graphs.

An instance of a specification $S$ is a graph $I$ together with a graph homomorphism $\iota : I \rightarrow G(S)$, such that $\iota^* \in \llbracket p \rrbracket$, i.e. $\iota^*$ is a valid instance of $p$ for each constraint $(p, \delta)$ where $\delta : \alpha(p) \rightarrow G(S)$.

$\iota^*$ is given by the pullback diagram:

\[
\begin{array}{ccc}
\alpha(p) & \xrightarrow{\delta} & G(S) \\
\downarrow{\iota^*} & & \downarrow{\iota} \\
O^* & \xrightarrow{\delta^*} & I \\
\end{array}
\]
Outline

1. Introduction and Motivation
   - Personal Motivation
   - Model-driven Engineering (MDE)
   - Modeling languages state of the art
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL
   - Specification
   - Example
   - Instance

3. OMG standards and Metamodelling Levels
   - Metamodelling
   - DPL meta model
   - Ecore meta model in DPL

4. Tools

5. Model Transformations
**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;Sotraveien 1, Bergen&quot;</td>
</tr>
</tbody>
</table>
Outline

1. Introduction and Motivation
   - Personal Motivation
   - Model-driven Engineering (MDE)
   - Modeling languages state of the art
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL
   - Specification
   - Example
   - Instance

3. OMG standards and Metamodeling Levels
   - Metamodeling
   - DPL meta model
   - Ecore meta model in DPL

4. Tools

5. Model Transformations
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 $(\Sigma_3 \subseteq \Sigma_2)$
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 $(\Sigma_3 \subseteq \Sigma_2)$
Outline

1. Introduction and Motivation
   - Personal Motivation
   - Model-driven Engineering (MDE)
   - Modeling languages state of the art
   - Diagrammatic Predicate Logic (DPL)

2. Concepts in DPL
   - Specification
   - Example
   - Instance

3. OMG standards and Metamodeling Levels
   - Metamodeling
   - DPL meta model
   - Ecore meta model in DPL

4. Tools

5. Model Transformations
Ecore is the metamodel of the Eclipse Modeling Framework, EMF

- Meta-formalism $E_3$ (EClassifier, EReference), essentially Graph
- Modeling Formalism, $E_2$ (EPackage), typed over $E_3$:
  \[
  t(EPackage) = t(EClassifier) = t(EClass) = t(EDataType) = EClassifier,
  \]
  \[
  t(eClassifiers) = t(eType) = t(eAttributes) = t(eReferences) = eStructuralFeatures
  \]
Ecore is the metamodel of the Eclipse Modeling Framework, EMF

- **Meta-formalism** $E_3$ (EClassifier, EReference), essentially **Graph**
- **Modeling Formalism**, $E_2$ (EPackage), typed over $E_3$:
  
  $t(EPackage) = t(EClassifier) = t(EClass) = t(EDataType) = EClassifier,$
  
  $t(eClassifiers) = t(eType) = t(eAttributes) = t(eReferences) = eStructuralFeatures$
Ecore model expressed in DPL

Ecore is the metamodel of the Eclipse Modeling Framework, EMF

- Meta-formalism $E_3$ (EClassifier, EReference), essentially **Graph**
- Modeling Formalism, $E_2$ (EPackage), typed over $E_3$:

  $t$ (EPackage) = $t$ (EClassifier) =
  $t$ (EClass) = $t$ (EDataType) = EClassifier,

  $t$ (eClassifiers) = $t$ (eType) =
  $t$ (eAttributes) = $t$ (eReferences) =
  eStructuralFeatures
Tools

- Tools implemented as plug-ins to the Eclipse platform
- Need tools for:
  - Drawing of models, partly finished June 2008
  - Serialization of models, work started
  - Validation of models, future research project
  - Model transformation tools, research started
Drawing tool

- Built over Eclipse Graphical Modeling Framework, GMF
- Will build separate tool for building signatures, conforms to a modeling language
- Tool for creating models over signatures, i.e. modeling tool in the chosen language
- Today drawing tool still some issues with diagrams with more than 3 nodes
Summary

- The industry have a lot of hidden knowledge, they have no need for formalization
- Modeling community work with fibrations, not index as we are trained to in mathematics
- Practical problems seems easy in theory but you really get dirt on your fingers
- We have obtained good understanding on modeling, metamodelling and model transformation
Further work

- Work on classification of model transformations, what properties has different class of transformations?
- Implement different formalisms in DPL (UML, XML, ...)
- Implement model transformation tool with code generation and cross platform translation
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 $\llbracket t \rrbracket(m)$
Transformation Rules

\[
\begin{align*}
G(M_1) & \xrightarrow{\iota} P & \xrightarrow{\iota} P' & \xrightarrow{\iota} G(M_2) \\
\xrightarrow{=} & m & \xrightarrow{[t](m)} & \xrightarrow{=} \\
I & \xrightarrow{trace} I' & \xrightarrow{trace} \\
\end{align*}
\]

- 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 \( [t](m) \)