Towards a Formal Diagrammatic Framework for Model Driven Architecture (MDA)

Adrian Rutle

Department of Computer Engineering
Faculty of Engineering
Bergen University College
Norway

08.07.2008
ECOOP 2008, Paphos, Cyprus
Motivation

- Raise of abstraction level of programming languages ⇒
- Modeling vs coding ⇒ MDA ⇒ Models and Model Transformations

```java
public class Person{
    public String p_name;
    public Company worksFor;
    //constraints here
    public Person(...){...}
}

public class Company{
    public String c_name;
    public String address;
    public List<Person> hires;
    //constraints here
    public Company(...){...}
}

public class Employment{
    public Date start_date;
    public int salary;
    public Company employer;
    public Person employee;
    //constraints here
    public Employment(...){...}
}
```
Motivation

- Raise of abstraction level of programming languages ⇒ Modeling vs coding ⇒ MDA ⇒ Models and Model Transformations
Motivation

- Raise of abstraction level of programming languages ⇒ Modeling vs coding ⇒ MDA ⇒ Models and Model Transformations

- Need for OCL to put constraints ⇒ Mixing text and diagrams

```java
public class Person {
    public String p_name;
    public List<Company> worksFor;
    //constraints here
    public Person(...) {...}
}

public class Company {
    public String c_name;
    public String address;
    public List<Person> hires;
    //constraints here
    public Company(...) {...}
}
```
**Motivation**

- Raise of abstraction level of programming languages \( \Rightarrow \) Modeling vs coding \( \Rightarrow \) MDA \( \Rightarrow \) Models and Model Transformations

- Need for OCL to put constraints \( \Rightarrow \) Mixing text and diagrams

```java
public class Person{
    public String p_name;
    public List<Company> worksFor;
    //constraints here
    public Person(...){...}
}

public class Company{
    public String c_name;
    public String address;
    public List<Person> hires;
    //constraints here
    public Company(...){...}
}
```

Instances:
- p1
- p2
- p2
- c1
- c2
- c1
- c1
Motivation

- Raise of abstraction level of programming languages ⇒
- Modeling vs coding ⇒ MDA ⇒ Models and Model Transformations

```
public class Person{
    public String p_name;
    public Company worksFor;
    //constraints here
    public Person(...){...}
}

public class Company{
    public String c_name;
    public String address;
    public List <Person> hires;
    //constraints here
    public Company(...){...}
}

public class Employment{
    public Date start_date;
    public int salary;
    public Company employer;
    public Person employee;
    //constraints here
    public Employment(...){...}
}
```
Motivation

- MDA \(\Rightarrow\) Models and Model Transformations \(\Rightarrow\) Code generation, model integration, model versioning, reverse engineering ...

```java
Public class Person{
    public String p_name;
    public Company worksFor;
    //constraints here
    public Person(...){...}
}

Public class Company{
    public String c_name;
    public String address;
    public List <Person> hires;
    //constraints here
    public Company(...)\{...\}
}

Public class Employment{
    public Date start_date;
    public int salary;
    public Company employer;
    public Person employee;
    //constraints here
    public Employment(...)\{...\}
}
```
Motivation

- MDA ⇒ Models and Model Transformations ⇒ Code generation, model integration, model versioning, reverse engineering ...

```java
public class Person{
    public String p_name;
    public Company worksFor;
    //constraints here
    public Person(...){...}
}

public class Company{
    public String c_name;
    public String address;
    public List<Person> hires;
    //constraints here
    public Company(...){...}
}

public class Employment{
    public Date start_date;
    public int salary;
    public Company employer;
    public Person employee;
    //constraints here
    public Employment(...){...}
}
```

DPL, MDA and MT (http://gs.hib.no)
Key-points for the Success of MDA

Models are no longer only for documentation

- Models should be both:
  - intuitive to ease their interception by software engineers and domain experts
  - formal such that machines can have a precise understanding of (and able to reason about) the models

⇒ Diagrammatic approach enhances human understanding
⇒ One challenge is to combine intuition with formalization
⇒ Another challenge is to implement the formalization approach
Key-points for the Success of MDA

Models are no longer only for documentation

- Models should be both:
  - intuitive to ease their interception by software engineers and domain experts
  - formal such that machines can have a precise understanding of (and able to reason about) the models

⇒ Diagrammatic approach enhances human understanding
⇒ One challenge is to combine intuition with formalization
⇒ Another challenge is to implement the formalization approach
Key-points for the Success of MDA

Models are no longer only for documentation

- Models should be both:
  - intuitive to ease their interception by software engineers and domain experts
  - formal such that machines can have a precise understanding of (and able to reason about) the models

⇒ Diagrammatic approach enhances human understanding
⇒ One challenge is to combine intuition with formalization
⇒ Another challenge is to implement the formalization approach
Key-points for the Success of MDA

Models are no longer only for documentation

- Models should be **both**:
  - intuitive to ease their interception by *software engineers* and *domain experts*
  - formal such that *machines* can have a precise understanding of (and able to reason about) the models

⇒ Diagrammatic approach enhances human understanding
⇒ One challenge is to combine intuition with formalization
⇒ Another challenge is to implement the formalization approach
Outline

1. Introduction and Motivation
   - Modeling Languages – state of the art

2. Approach
   - Diagrammatic Predicate Logic (DPL)
   - Concepts in DPL
     - Signatures and Diagrammatic Specifications

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

4. Model Transformations
   - Concepts
   - MT Example

5. Implementation
UML vs Formal Modeling Languages

UML ...

- is considered *de facto* industry standard for modeling
- has serious issues regarding semantics, complexity, expressibility (uses OCL for constraints on higher order relations)

Formal Modeling Languages ...

- has precise semantics and proven abilities
- has set based semantics, while models are graph based
- is hard for software engineers to apply in practice
- lacks good software development tools
Outline

1. Introduction and Motivation
   - Modeling Languages – state of the art

2. Approach
   - Diagrammatic Predicate Logic (DPL)
   - Concepts in DPL
     - Signatures and Diagrammatic Specifications

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

4. Model Transformations
   - Concepts
   - MT Example

5. Implementation
Our Approach: Diagrammatic Predicate Logic (DPL)

- Aims to combine the intuition from graphical modeling languages with the semantic rigor of formal methods
- Based on Generalized Sketches/Category Theory
- In DPL, sketches are generalized and adapted for use in MDA
- Potentials to use the machinery from category theory to implement MDE tools
**Outline**

1. **Introduction and Motivation**
   - Modeling Languages – state of the art

2. **Approach**
   - Diagrammatic Predicate Logic (DPL)
   - Concepts in DPL
     - Signatures and Diagrammatic Specifications

3. **OMG standards and Metamodeling Levels**
   - Metamodelling
   - Ecore in DPL

4. **Model Transformations**
   - Concepts
   - MT Example

5. **Implementation**
### Signature – Example

<table>
<thead>
<tr>
<th>Signature</th>
<th>visualization</th>
<th>intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>[total]</td>
<td><img src="image1" alt="Diagram" /></td>
<td>( \forall a \in A :</td>
</tr>
<tr>
<td>[key]</td>
<td><img src="image2" 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><img src="image3" 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><img src="image4" alt="Diagram" /></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

Employment

Company

Employer

Employee

[start_date]

[start_date]

[salary]

[salary]

[StreamReader]

[StreamReader]

[StreamReader]

[StreamReader]

[StreamReader]
Diagrammatic Specification – Example

Person

Employment

Company

[1-1]

worksFor

hires

employee

employer

[String]

[p_name]

[1]

[key]

[address]

[1]

[c_name]

[1]

Employee

Employment

Company

[Date]

[start_date]

[1]

[Salary]

[1]

salary

start_date

[1-1]

graph G(S)

Constraints S(\Pi):

[inv]

[1-1]

DPL, MDA and MT (http://gs.hib.no)
Instance of Diagrammatic Specification – Example
Outline

1. Introduction and Motivation
   - Modeling Languages – state of the art

2. Approach
   - Diagrammatic Predicate Logic (DPL)
   - Concepts in DPL
     - Signatures and Diagrammatic Specifications

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

4. Model Transformations
   - Concepts
   - MT Example

5. Implementation
DPL Methodology and Modeling Formalisms

- $M_3 \conformsto M_2\conformsto M_1 \conformsto M_0$
- $\Pi_3 \rightarrow G(M\text{Metamodel})\rightleftharpoons \Pi_2 \rightarrow G(M\text{etamodel})\rightleftharpoons \Pi_1 \rightarrow G(M\text{odel})$

Adrian Rutle
DPL, MDA and MT (http://gs.hib.no)
Outline

1. Introduction and Motivation
   - Modeling Languages – state of the art

2. Approach
   - Diagrammatic Predicate Logic (DPL)
   - Concepts in DPL
     - Signatures and Diagrammatic Specifications

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

4. Model Transformations
   - Concepts
   - MT Example

5. Implementation
Ecore model expressed in DPL

- `EPackage`
  - `eClassifiers`
- `EClassifier`
  - `eSuperType` (disjoint)
  - `eSuperType`
- `EClass`
  - `eReferences`
  - `eAttributes`
- `EDataType`
Ecore model expressed in DPL
Ecore model expressed in DPL
Model Transformation

MT Definition

MT Language

MT Execution

conformsTo

Adrian Rutle

DPL, MDA and MT (http://gs.hib.no)
Model Transformation

MT Definition

<table>
<thead>
<tr>
<th>MT Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT Definition</td>
</tr>
<tr>
<td>MT Execution</td>
</tr>
</tbody>
</table>

conformsTo

<table>
<thead>
<tr>
<th>MT Definition</th>
</tr>
</thead>
</table>

conformsTo

<table>
<thead>
<tr>
<th>MT Execution</th>
</tr>
</thead>
</table>

MT Example

Adrian Rutle

DPL, MDA and MT (http://gs.hib.no)
Outline

1. Introduction and Motivation
   - Modeling Languages – state of the art

2. Approach
   - Diagrammatic Predicate Logic (DPL)
   - Concepts in DPL
     - Signatures and Diagrammatic Specifications

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

4. Model Transformations
   - Concepts
   - MT Example

5. Implementation
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 \([t](m)\)
Transformation Rules

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

- 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 \([t](m)\)
Outline

1. Introduction and Motivation
   - Modeling Languages – state of the art

2. Approach
   - Diagrammatic Predicate Logic (DPL)
   - Concepts in DPL
     - Signatures and Diagrammatic Specifications

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

4. Model Transformations
   - Concepts
   - MT Example

5. Implementation

Adrian Rutle
DPL, MDA and MT (http://gs.hib.no)
Model Transaction: Ecore to RDB
The Input EMF Model

- **Person:** EClass
  - **hires:** EReference
    - **upperBound:** -1
  - **eReferenceType:**
- **Company:** EClass
  - **worksFor:** EReference
    - **upperBound:** 1
  - **eReferenceType:**
- **name:** EAttribute
  - **eAttributeType:** String
  - **eAttribute**

Adrian Rutle
DPL, MDA and MT (http://gs.hib.no)
Rule 1: EClass to Table + PrimaryKey

1: EClass

2: Column

3: PrimaryKey

1: Table

[1] ownedby

cols

[1] [key]

table

[⊂]

pcols

Adrian Rutle
DPL, MDA and MT (http://gs.hib.no)
Rule2: EAttribute to Column

1:EClass \( \rightarrow \) eAttributes \( \rightarrow \) 2:EAttribute \( \rightarrow \) eAttributeType \( \rightarrow \) 3:EDataType

1:Table \( \rightarrow \) ownedby \( [1] \) cols \( \rightarrow \) 2:Column

3:DataType \( \rightarrow \) type
Rule 3: EReference to Foreign Key

1: EClass
   └── eReferenceType

2: EClass
   └── eReferences
       └── upperBound
           └── 0|1|-1

3: EReference

4: ForeignKey
   └── [1-1]

5: PrimaryKey
   └── ref
       └── [1-1]

6: Column
   └── cols

1: Table
   └── ownedby

3: Column
   └── owner

2: Table
   └── ownedby

4: ForeignKey
   └── pcol

5: PrimaryKey
   └── ref

6: Column
   └── cols

Adrian Rutle
DPL, MDA and MT (http://gs.hib.no)
The Output RDB Model

Adrian Rutle
DPL, MDA and MT (http://gs.hib.no)
The Output RDB Model

- **Person:** EClass
  - **name:** EAttribute
    - **eAttributeType**
      - **String:** EDataType
        - **upperBound**
          - **1**
          - **worksFor:** EReference
            - **eReferenceType**
              - **Person:** EClass
                - **Company:** EClass
                  - **eReferenceType**
                    - **hires:** EReference
                      - **upperBound**
                        - **-1**

- **Person:** Table
  - **String:** DataType
    - **type**
      - **name:** Column
        - **ownedby**
          - **[1]**
          - **worksFor:** Column
            - **col**
              - **Pers-id:** Column
                - **pcol**
                  - **[1]**
                  - **table**
                    - **[key]**
                      - **[1]**
                      - **Person:** Table
                        - **C-pk:** Primarykey
                          - **[1]**
                          - **ref**
                            - **PC-fk:** Foreignkey
                              - **pcol**
                                - **Comp-id:** Column
                                  - **[1]**
                                  - **ownedby**
                                    - **[1]**

Tools

- Tools implemented as plug-ins to the Eclipse platform
- Need tools for:
  - Drawing of models (GMF-based editor), partly finished June 2008
  - Serialization of models, work started
  - Validation of models, future research project
  - Model transformation tools, research started
Generation of GMF-based Editors

1. Create GMF Project
2. Develop Domain Model
3. Develop Graphical Definition
4. Develop Tooling Definition
5. Develop Mapping Model
6. Create Generator Model
7. Generate Diagram Plug-in

Adrian Rutle
DPL, MDA and MT (http://gs.hib.no)
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.

Using DPL as a generic framework for formalization of modeling and model transformation languages.