DPF Editor: A Multi-Layer Diagrammatic (Meta)Modelling Environment

Yngve Lamo¹, Xiaoliang Wang¹, Florian Mantz¹, Øyvind Bech¹, Adrian Rutle²

¹Faculty of Engineering, Bergen University College, Norway

²St. Francis Xavier University, Canada

5 October 2011
SPLST 2011, Tallinn, Estonia
Outline

Introduction and Motivation

Diagram Predicate Framework

Sample: Business Process Modelling

Summary and Future Work
Introduction

- Domain Specific (modelling) Languages DSLs are (modelling) languages made for a specific domain
- In MDE
  - DSLs usually specified by a graph-based metamodel + text-based Constraints
  - In practice: UML, UML profiles + OCL
Introduction

- Domain Specific (modelling) Languages DSLs are (modelling) languages made for a specific domain
- In MDE
  - DSLs usually specified by a graph-based metamodel + text-based Constraints
  - In practice: UML, UML profiles + OCL
- Problem
  - typing and constraints are specified by different languages (having different metamodels)
  - model transformations usually not constraint-aware
Introduction

- Domain Specific (modelling) Languages DSLs are (modelling) languages made for a specific domain
- In MDE
  - DSLs usually specified by a graph-based metamodel + text-based Constraints
  - In practice: UML, UML profiles + OCL

- Problem
  - typing and constraints are specified by different languages (having different metamodels)
  - model transformations usually not constraint-aware

- Solution
  - diagrammatic specification formalism where the metamodelling considers both typing and constraints
Tool Support

- Tool support needed to:
  - Facilitate the design and implementation of DSLs
  - Provide an intuitive view of modelling
  - Runtime validation of models
  - Check soundness of transformation
Tool Support

- Tool support needed to:
  - Facilitate the design and implementation of DSLs
  - Provide an intuitive view of modelling
  - Runtime validation of models
  - Check soundness of transformation

- Today MDE technologies:
  - OMG provides MOF + OCL
  - Industrial reference implementation of (essential)MOF is given by EMF
  - Two-layers Metamodelling approach
Tool Support

- Tool support needed to:
  - Facilitate the design and implementation of DSLs
  - Provide an intuitive view of modelling
  - Runtime validation of models
  - Check soundness of transformation

- Today MDE technologies:
  - OMG provides MOF + OCL
  - Industrial reference implementation of (essential)MOF is given by EMF
  - Two-layers Metamodelling approach

- Limitation
  - No support for multilevel metamodelling, i.e. cannot create instance of instances
  - Forced to introduce type-instance relation
  ⇒ Mixture of domain concepts with language concepts
DPF Editor

- Modeling Tool for
  - Specification of (meta)models
  - Generation of Diagrammatic Editors for those (meta)models
  - Support Multilevell metamodeling
  - Conformance check of models to their metamodels:
    - typing, by construction
    - diagrammatic constraints, at runtime
Outline

Introduction and Motivation

Diagram Predicate Framework

Sample: Business Process Modelling

Summary and Future Work
**Diagram Predicate Framework**

- A specification $\mathcal{G} = (S, C^\mathcal{G}; \Sigma)$ consists of an *underlying graph* $S$ together with a set of *atomic constraints* $C^\mathcal{G}$.

- Atomic constraints are specified by *predicates* from a predefined *(diagrammatic predicate) signature* $\Sigma$. A signature $\Sigma = (\Pi^\Sigma, \alpha^\Sigma)$ consists of a collection of predicates, each having a symbol, an arity (or shape graph), a visualisation and a semantic interpretation.

<table>
<thead>
<tr>
<th>$\Pi^\Sigma$</th>
<th>$\alpha^\Sigma$</th>
<th>Proposed vis.</th>
<th>Semantic interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{[mult}(m, n)\text{]}$</td>
<td>$1 \xrightarrow{a} 2$</td>
<td>$\begin{array}{c} X \xrightarrow{f} [m..n] Y \end{array}$</td>
<td>$\forall x \in X : m \leq</td>
</tr>
<tr>
<td>$\text{[irreflexive]}$</td>
<td>$\begin{array}{c} \circ \xrightarrow{a} \end{array}$</td>
<td>$\begin{array}{c} [m] \xrightarrow{f} X \end{array}$</td>
<td>$\forall x \in X : x \notin f(x)$</td>
</tr>
</tbody>
</table>

![Diagram](image)
The semantics of a specification is defined by the set of its instances \((I, \iota)\). An instance \((I, \iota)\) of \(\mathcal{G}\) is a graph \(I\) together with a graph homomorphism \(\iota : I \rightarrow S\) that satisfies the atomic constraints \(C^\mathcal{G}\).

<table>
<thead>
<tr>
<th>(\Pi^\mathcal{G})</th>
<th>(\alpha^\Sigma)</th>
<th>Proposed vis.</th>
<th>Semantic interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\text{mult}(m, n)])</td>
<td>(1 \xrightarrow{a} 2)</td>
<td>(X \xrightarrow{f} Y)</td>
<td>(\forall x \in X : m \leq</td>
</tr>
<tr>
<td>([\text{irreflexive}])</td>
<td>(1 \xrightarrow{a} X)</td>
<td>([\text{irr}] X \xrightarrow{f} X)</td>
<td>(\forall x \in X : x \notin f(x))</td>
</tr>
</tbody>
</table>
Outline

Introduction and Motivation

Diagram Predicate Framework

Sample: Business Process Modelling

Summary and Future Work
Step 1: Default metamodel $S_4$ in DPF Editor

- DPF’s default metamodel $S_4$ consisting of *Node* and *Arrow*
Step 2: Business process meta-metamodel $\mathcal{G}_3$

- Introducing the domain concepts:
  - *Element, Control : Node*
  - *Flow, NextControl, ControlIn, ControlOut : Arrow*
Step 2: Business process meta-metamodel $S_3$

- Typing Conformance
  - The DPF Editor actually checks that there exists a graph homomorphism from the specification to its metamodel while creating a specification.
Step 2: Business process meta-metamodel $S_3$

- Constraint Conformance
  - There is no constraint on the metamodel $S_4$
  - Hence $S_3$ is a valid instance of $S_4$
Step 2: Business process meta-metamodell $\mathcal{G}_3$

- **Constraints on $\mathcal{G}_3$:**
  - “each control should have at least one incoming arrow from an element or another control”
  - *Constraint*: $[js]$ between the arrows $ControlIn$ and $NextControl$
Step 2: Business process meta-metamodel $\mathcal{S}_3$

- **Constraints on $\mathcal{S}_3$:**
  - "each control should be followed by either another control or by an element, not both"
  - *Constraint*: $[\text{xor}]$ between the arrows $\text{ControlOut}$ and $\text{NextControl}$
Step 3: Business process metamodel $\mathcal{G}_2$

- We now generate an editor by using $\mathcal{G}_3$ as metamodell.
- Domain concepts:
  - Activity: Element
  - Condition, Choice: Control
  - Sequence, Message: Flow
  - ChoiceCondition: NextControl
  - ChoiceIn: ControlIn
  - ChoiceOut: ControlOut
Step 3: Business process metamodel $\mathcal{G}_2$

- Typing Conformance by construction:
  - For instance, when we create the *ChoiceIn* arrow of type *ControlIn*, the tool ensures that the source and target of *ChoiceIn* are typed by *Element* and *Control*, respectively.
Step 3: Business process metamodel $\mathcal{G}_2$

- Constraint Conformance
  - The constraints from $\mathcal{G}_3$ are checked by corresponding validators
  - *Constraint*: $[js]$ between the arrows ControlIn and NextControl, i.e. Choice and Condition has correct incoming arrows
  - *Constraint*: $[xor]$ between the arrows ControlOut and NextControl
An invalid instance of $S_3$:

- $[\text{xor}]$ constraint on the arrows ControlOut and NextControl in $S_3$ is violated by the arrow WrongArrow
- Condition : Control is followed by a Choice : Control and an Activity : Element, which violates the requirement “each control should be followed by either another control or an element, not both”
Step 3: Business process metamodel $\mathcal{S}_2$

- Constraints on $\mathcal{S}_2$
  - Each *Activity* may be connected to at most one *choice*
  - Constraint: $[0..1]$ on *ChoiceIn*
Step 3: Business process metamodel $\mathcal{S}_2$

- Constraints on $\mathcal{S}_2$
  - Each *choice* must be connected to at least two *conditions*
  - *Constraint*: $[2..\ast]$ on *ChoiceCondition*
Step 3: Business process metamodel $\mathcal{G}_2$

- **Constraints on $\mathcal{G}_2$**
  - Each *Activity* may be connected either to a *Choice* or to another *Activity*, but not both.
  - *Constraint:* $[\text{nand}]$ between *ChoiceIn* and *Sequence*
Step 3: Business process metamodel $\mathcal{S}_2$

- **Constraints on $\mathcal{S}_2$**
  - Each *Choice* must have exactly one *Activity* connected to it
  - *Constraint*: $[\text{surj}]$ on *ChoiceIn*
Step 3: Business process metamodel $\mathcal{G}_2$

- **Constraints on $\mathcal{G}_2$**
  - Each *Condition* must be connected to exactly one *Activity*
  - *Constraint*: $[1..1]$ on *ChoiceOut*
Step 3: Business process metamodel $\mathcal{G}_2$

- Constraints on $\mathcal{G}_2$
  - Each Activity must have exactly one Condition connected to it
  - Constraint: $\text{[inj]}$ on ChoiceOut
Step 3: Business process metamodel $\mathcal{G}_2$

- Constraints on $\mathcal{G}_2$
  - An Activity cannot send messages to itself
  - Constraint: $[\text{irr}]$ on Message
**Step 3: Business process metamodel $\mathcal{G}_2$**

- **Constraints on $\mathcal{G}_2$**
  - An *Activity* cannot be sequenced to itself
  - *Constraint:* $[\text{irr}]$ on *Sequence*
Step 4: Business process model $\mathcal{S}_1$

- The model $\mathcal{S}_1$ conforms to $\mathcal{S}_2$
- Domain concepts:
  - *Customer Comes*: Activity
  - *Yes, No*: Condition
  - *NeedPay, Money*: Message
  - *Pay, TimeOff*: Choice
- Typing Conformance
- Constraint Conformance
Outline

Introduction and Motivation

Diagram Predicate Framework

Sample: Business Process Modelling

Summary and Future Work
Summary

- Diagram Predicate Framework (DPF) is well established on the theoretical level, see http://dpf.hib.no
- This is the first publication of its corresponding prototype tool
- The tool is developed in Java and runs as a plug-in on the Eclipse platform, using EMF and GEF technologies
Table: Comparison of DPF to other metamodelling tools, EVL stands for Epsilon Validation Language, and the current predefined validator in DPF is implemented in Java

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EMF/GMF</td>
<td>2</td>
<td></td>
<td>OCL, EVL, Java</td>
<td>Java VM</td>
<td>✓</td>
</tr>
<tr>
<td>VMTS</td>
<td>∞</td>
<td></td>
<td>OCL</td>
<td>Windows</td>
<td>✓</td>
</tr>
<tr>
<td>AToM^3</td>
<td>2</td>
<td></td>
<td>OCL, Python</td>
<td>Python, Tk/tcl</td>
<td>✓</td>
</tr>
<tr>
<td>GME</td>
<td>2</td>
<td></td>
<td>OCL</td>
<td>Windows</td>
<td>✓</td>
</tr>
<tr>
<td>metaDepth</td>
<td>∞</td>
<td></td>
<td>EVL</td>
<td>Java VM</td>
<td></td>
</tr>
<tr>
<td>DPF</td>
<td>∞</td>
<td>✓</td>
<td>Predefined validator</td>
<td>Java VM</td>
<td>✓</td>
</tr>
</tbody>
</table>
Future work

- Code generation
- Signature and constraint definition editors
- Enhance the layout and routing algorithms
- Extend the tool for model transformation
Thanks for your attention

Questions?