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

Yngve Lamo[1], Xiaoliang Wang[1], Florian Mantz[1], Øyvind Bech[1], Adrian Rutle[2]

[1]Faculty of Engineering, Bergen University College, Norway

[2]St. Francis Xavier University, Canada

5 October 2011
SPLST 2011, Tallinn, Estonia

HØGSKOLEN I BERGEN StFX UNIVERSITY

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 Metamodeling 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 Metamodeling approach
- Limitation
  - No support for multilevel metamodelling, i.e. cannot create instance of instances
  - Forced to introduce type-instance relation
    $\Rightarrow$ 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 metamodelling
  - Conformance check of models to their metamodels:
    - typing, by construction
    - diagrammatic constraints, at runtime

Outline

Introduction and Motivation

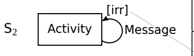
Diagram Predicate Framework

Sample: Business Process Modelling

Summary and Future Work

Diagram Predicate Framework

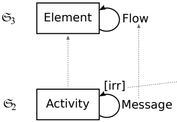
- A specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ consists of an *underlying graph* S together with a set of *atomic constraints* $C^{\mathfrak{S}}$
- Atomic constraints are specified by *predicates* from a predefined *(diagrammatic predicate) signature* Σ. A signature $\Sigma = (\Pi^{\Sigma}, \alpha^{\Sigma})$ consists of a collection of predicates, each having a symbol, an arity (or shape graph), a visualisation and a semantic interpretation.



Π^{Σ}	α^{Σ}	Proposed vis.	Semantic interpretation
[mult(m,n)]	$1 \xrightarrow{a} 2$	$\boxed{X} \xrightarrow[\text{[m..n]}]{f} \boxed{Y}$	$\forall x \in X : m \leq \lvert f(x) \rvert \leq n$, with $0 \leq m \leq n$ and $n \geq 1$
[irreflexive]	$1 \circlearrowleft a$	$\text{[irr]}\ \boxed{X} \circlearrowleft f$	$\forall x \in X : x \notin f(x)$

Diagram Predicate Framework

- The semantics of a specification is defined by the set of its instances (I, ι). An instance (I, ι) of $\mathfrak{S}$ is a graph I together with a graph homomorphism $\iota : I \to S$ that satisfies the atomic constraints $C^{\mathfrak{S}}$.



Π^Σ	α^Σ	Proposed vis.	Semantic interpretation
[mult (m,n)]	$1 \xrightarrow{a} 2$	$\boxed{X} \xrightarrow[\text{[m..n]}]{f} \boxed{Y}$	$\forall x \in X : m \leq \lvert f(x) \rvert \leq n$, with $0 \leq m \leq n$ and $n \geq 1$
[irreflexive]	1 a	[irr] $\boxed{X}$ f	$\forall x \in X : x \notin f(x)$

Outline

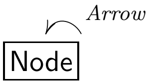
Introduction and Motivation

Diagram Predicate Framework

Sample: Business Process Modelling

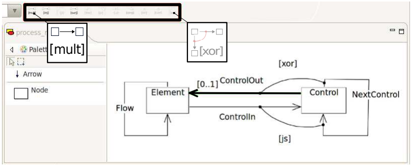
Summary and Future Work

Step 1: Deafult metamodel $\mathfrak{S}_4$ in DPF Editor



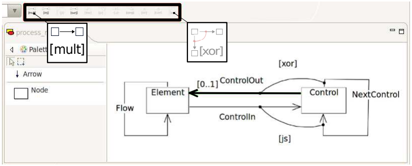
- DPF's default metamodel $\mathfrak{S}_4$ consisting of $Node$ and $Arrow$

Step 2: Business process meta-metamodel $\mathfrak{S}_3$



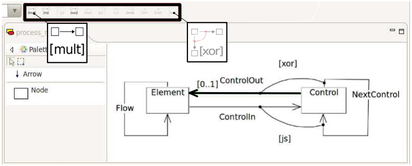
- Introducing the domain concepts:
  - $Element, Control : Node$
  - $Flow, NextControl, ControlIn, ControlOut : Arrow$

Step 2: Business process meta-metamodel $\mathfrak{S}_3$



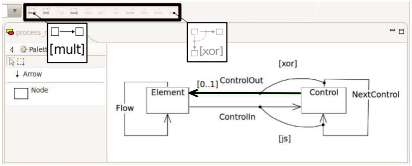
- Typing Conformance
  - The DPF Editor actually checks that there exists a graph homomorphism from the specification to its metamodel while creating a specification.

Step 2: Business process meta-metamodel $\mathfrak{S}_3$



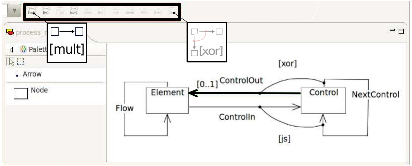
- Constraint Conformance
  - There is no constraint on the metamodel $\mathfrak{S}_4$
  - Hence $\mathfrak{S}_3$ is a valid instance of $\mathfrak{S}_4$

Step 2: Business process meta-metamodel $\mathfrak{S}_3$



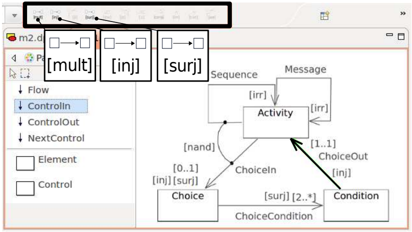
- Constraints on $\mathfrak{S}_3$:
  - "each control should have at least one incoming arrow from an element or another control"
  - $Constraint$: $[js]$ between the arrows $ControlIn$ and $NextControl$

Step 2: Business process meta-metamodel $\mathfrak{S}_3$



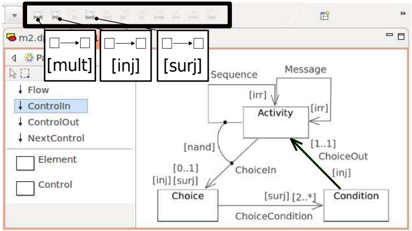
- Constraints on $\mathfrak{S}_3$:
  - "each control should be followed by either another control or by an element, not both"
  - $Constraint$: $[xor]$ between the arrows $ControlOut$ and $NextControl$

Step 3: Business process metamodel $\mathfrak{S}_2$



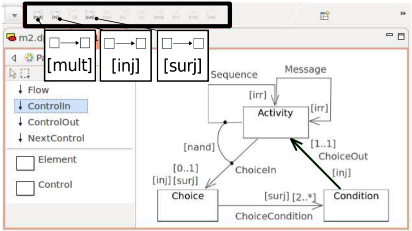
- We now generate an editor by using $\mathfrak{S}_3$ as metamodell
- Domain concepts:
  - $Activity : Element$
  - $Condition, Choice : Control$
  - $Sequence, Message : Flow$
  - $ChoiceCondition : NextControl$
  - $ChoiceIn : ControlIn$
  - $ChoiceOut : ControlOut$

Step 3: Business process metamodel $\mathfrak{S}_2$



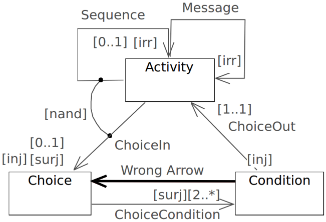
- Typing Conformance by construction:
  - For instance, when we create the $ChoiceIn$ arrow of type $ControlIn$, the tool ensures that the source and target of $ChoiceIn$ are typed by $Element$ and $Control$, respectively

Step 3: Business process metamodel $\mathfrak{S}_2$



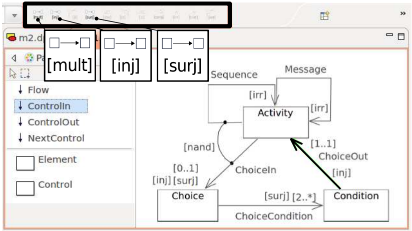
- Constraint Conformance
  - The constraints from $\mathfrak{S}_3$ are checked by corresponding validators
  - $Constraint$: $[js]$ between the arrows $ControlIn$ and $NextControl$, i.e. $Choice$ and $Condition$ has correct incoming arrows
  - $Constraint$: $[xor]$ between the arrows $ControlOut$ and $NextControl$

Step 3: Business process metamodel $\mathfrak{S}_2$



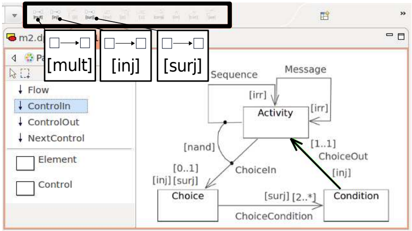
- An invalid instance of $\mathfrak{S}_3$:
  - $[xor]$ constraint on the arrows $ControlOut$ and $NextControl$ in $\mathfrak{S}_3$ is violated by the arrow $WrongArrow$
  - $Condition : Control$ is followed by a $Choice : Control$ and an $Activity : Element$, which violates the requirement "each control should be followed by either another control or an element, not both"

Step 3: Business process metamodel $\mathfrak{S}_2$



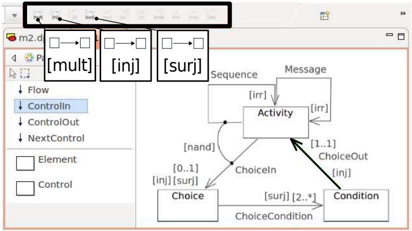
- Constraints on $\mathfrak{S}_2$
  - Each *Activity* may be connected to at most one *choice*
  - $Constraint$: $[0..1]$ on $ChoiceIn$

Step 3: Business process metamodel $\mathfrak{S}_2$



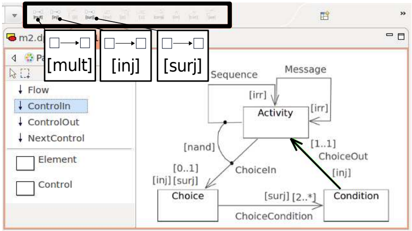
- Constraints on $\mathfrak{S}_2$
  - Each *choice* must be connected to at least two *conditions*
  - $Constraint$: $[2..*]$ on $ChoiceCondition$

Step 3: Business process metamodel $\mathfrak{S}_2$



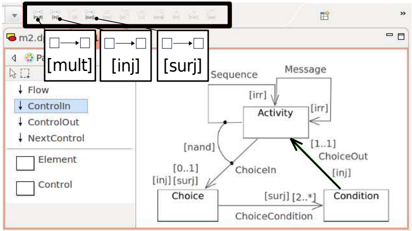
- Constraints on $\mathfrak{S}_2$
  - Each *Activity* may be connected either to a *Choice* or to another *Activity*, but not both.
  - $Constraint$: $[nand]$ between $ChoiceIn$ and $Sequence$

Step 3: Business process metamodel $\mathfrak{S}_2$



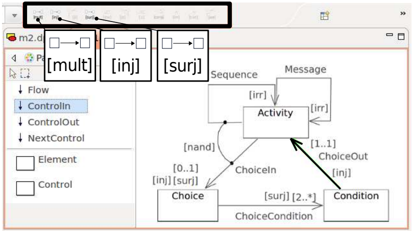
- Constraints on $\mathfrak{S}_2$
  - Each *Choice* must have exactly one *Activity* connected to it
  - $Constraint$: $[surj]$ on $ChoiceIn$

Step 3: Business process metamodel $\mathfrak{S}_2$



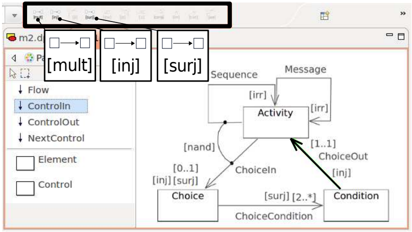
- Constraints on $\mathfrak{S}_2$
  - Each *Condition* must be connected to exactly one *Activity*
  - $Constraint$: $[1..1]$ on $ChoiceOut$

Step 3: Business process metamodel $\mathfrak{S}_2$



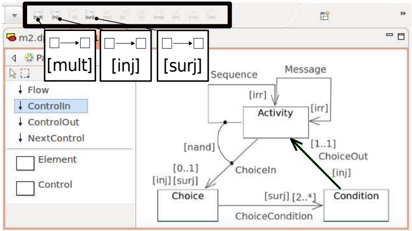
- Constraints on $\mathfrak{S}_2$
  - Each *Activity* must have exactly one *Condition* connected to it
  - $Constraint$: $[inj]$ on $ChoiceOut$

Step 3: Business process metamodel $\mathfrak{S}_2$



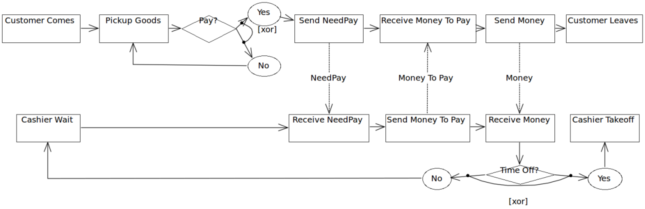
- Constraints on $\mathfrak{S}_2$
  - An *Activity* cannot send messages to itself
  - $Constraint$: $[irr]$ on $Message$

Step 3: Business process metamodel $\mathfrak{S}_2$



- Constraints on $\mathfrak{S}_2$
  - An *Activity* cannot be sequenced to itself
  - $Constraint$: $[irr]$ on $Sequence$

Step 4: Business process model $\mathfrak{S}_1$



- The model $\mathfrak{S}_1$ conforms to $\mathfrak{S}_2$
- 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

Related Work

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

Tool	Layers	Diag. Const.	Const. Lang.	Platform	GUI
EMF/GMF	2		OCL, EVL, Java	Java VM	✓
VMTS	∞		OCL	Windows	✓
AToM^3	2		OCL, Python	Python, Tk/tcl	✓
GME	2		OCL	Windows	✓
metaDepth	∞		EVL	Java VM	
DPF	∞	✓	Predefined validator	Java VM	✓

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?