Model Driven Engineering (MDE) and Diagram Predicate Framework (DPF)

Yngve Lamo[1,2]

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

[2] James Chair, St. Francis Xavier University, Antigonish, Canada

University of New Brunswick
Fredericton, Canada
May 30th, 2012

HØGSKOLEN I BERGEN

Outline

Intro

MDE

Modeling

Languages

Meta-Modeling

Model Transformations

DPF

DPF Workbench

Summary

About me

- Name: Yngve Lamo
- Associate Professor, Bergen University College , Norway
- James Chair, visiting Professor at St. FX University, Antigonish Canada, March-June 2012
- PhD in Computer Science (2003)
- Background in formal methods, algebraic specifications (logic and category theory)
- Working on foundations of model driven software engineering since 2004

Outline

Intro

MDE

Modeling

Languages

Meta-Modeling

Model Transformations

DPF

DPF Workbench

Summary

Model Driven Engineering (MDE)

- MDE is a promising approach for software engineering
- Models are the primary artifact
- MDE offers:
  - Domain specific abstraction mechanism i.e.:
    - Metamodelling
  - Automatic software development:
    - Code generation and
    - Model transformations
  - Improved productivity and correctness of software
- MDE needs to be further developed:
  - Concepts are not always clearly defined and (or) documented
  - Tool support is not mature enough to really benefit on MDE

MDE background

- Engineering technique where models are first-class entities
- Evolved from diagrammatic languages such as UML(1997) and ER(1976) and their popularity for specification and documentation of software systems
- Developed from CASE tools from the 80ies
- Aims to raise the abstraction level in software development from text to models
- The development process is based on software models, not on plain text
- Aims to generate as much as possible of the software from the models

Traditional use of models in SE

Models traditionally used in Software Engineering for:

- Software developers to **communicate** with domain experts during the system requirements phase
- Software architects to **specify** system requirements to developers during the implementation phase
- System **documentation** purposes during the maintenance phase

⇒ Models not in synchronization with the software development process

Key point for success for MDE

Models not only for documentation

- Models should be **both**:
  - **Intuitive** for software engineers and domain experts to **intercept** and work with
  - **Formal** and machine readable with precise meaning enhancing **reasoning** about the models

Key point for success for MDE

Models not only for documentation

- Models should be **both**:
  - **Intuitive** for software engineers and domain experts to **intercept** and work with
  - **Formal** and machine readable with precise meaning enhancing **reasoning** about the models

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

Modeling

Model, from Latin *modulus* meaning measure/standard

- A model are used in two ways either as a:

  Prescription model (usually a miniature) used to represent a system before creation, as a pattern for design

  Description model used to represent some major aspects of an existing item, object, system, or concept

- A model need to meet the following 3 criteria:

  Reflection The model reflects some properties of the original (system)

  Abstraction The model describes only some of the "interesting" properties of the original

  Substitution The model can be used instead of the original for some purposes

Modeling perspectives

Software models represents different perspectives of the system:

- External perspective, models the context of the system
- Interaction perspective, models the interaction between the system and its environment
- Structural perspective, models the structure of the system (or the data)
- Behavioral perspective, models the dynamic behavior of the system

General Purpose Modeling Languages (GPML)

- General purpose modeling Languages (GPML) are made to represent some aspects of a system (object of study)
- GPMLs are often graph based with limited support for constraints
  - In practice: UML+ OCL
- The semantics is often unprecise or undocumented
- GPMLs are often made to support a specific technology
  - UML object orientation
  - ER relational databases

Unified Modeling Language

- UML is defacto industry standard for modeling
- UML consists of 13 different diagrams, most important are:
  - Activity diagrams, activities in processes
  - Use case diagrams, interaction between system and environment
  - Sequence diagrams, interaction between system components (and actors)
  - Class diagrams, structure of the system
  - State diagrams, system dynamics

Unified Modelling Language

- UML can only express constraints on binary relations, basically UML specifies only cardinality constraints
- 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 text based logic, (Object Constraint Language, OCL)

Formal modeling languages

Pros:

- Nice semantics, several fundamental Software Engineering problems solved by use of formal methods
- Set based semantics (logic, algebras, type theory, . . .)

Cons:

- No concept of meta-modeling
- Hard for software engineers to apply in practice
- Lack of good software development tools support
- Only used by well trained experts
- High cost low productivity

Domain Specific Modeling Languages DSMLs

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

Domain Specific Modeling Languages DSMLs

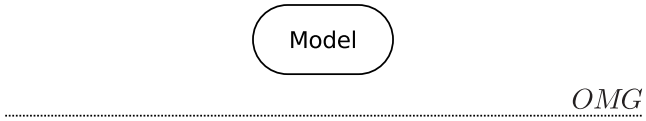
- Domain Specific Modelling Languages DSMLs are modelling languages made for a specific domain
- In MDE:
  - DSMLs usually specified by a graph-based metamodel + text-based Constraints
  - In practice: UML, UML profiles + OCL
- Problem
  - Typing and constraints are specified by different languages (having different metamodels)
  - Model transformations usually not constraint-aware
- Solution
  - Diagrammatic specification formalism where the metamodelling considers both typing and constraints

Meta Modeling

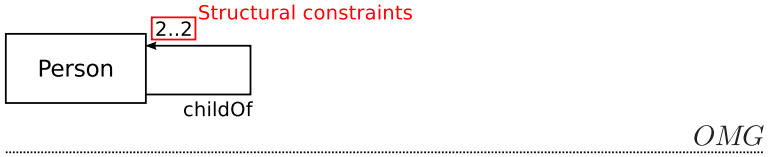
- A meta-model is a model of a modeling language
- Meta-models are used for defining modeling languages
- E.g. in OO modeling a person is an instance of class
- Meta-modeling is used to create Domain Specific Modelling Languages DSMLs, i.e. one create language constructs for important domain concepts, e.g. a student and a teacher is instances of persons

Meta-model example

Pattern



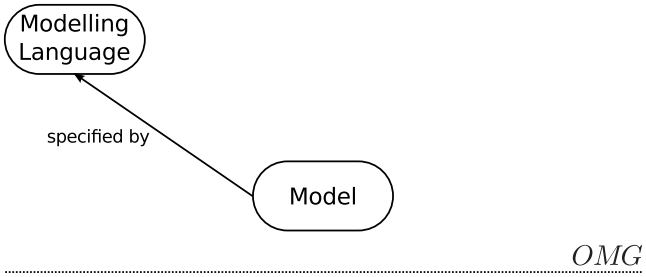
Example



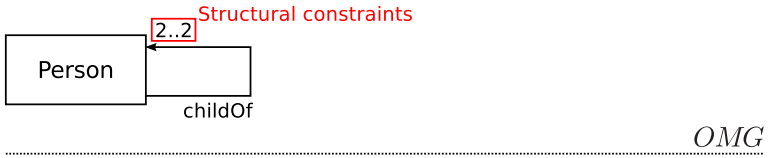
- Models: first class entities

Meta-model example

Pattern



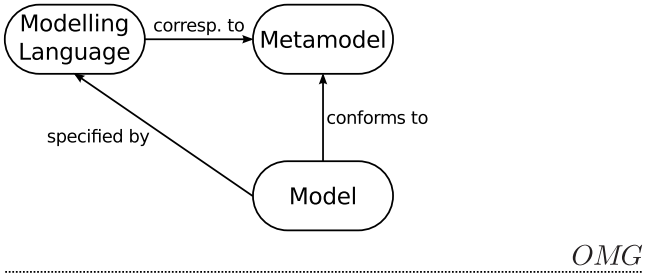
Example



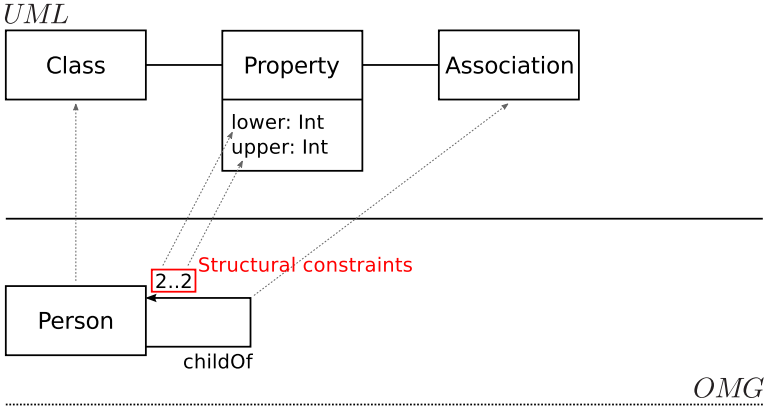
- Models: specified by means of a modelling language

Meta-model example

Pattern

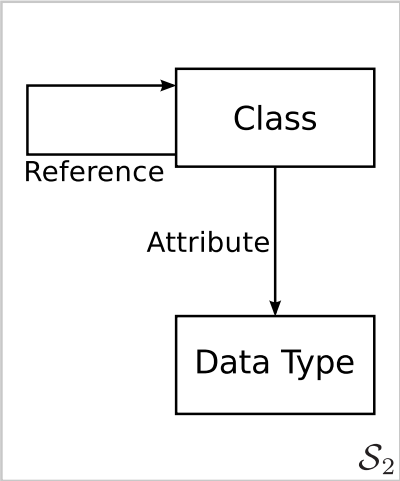


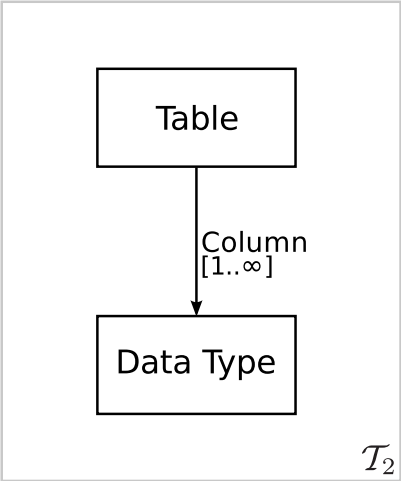
Example



- Modelling language: corresponding metamodel + semi-formal semantics

Meta-model for object orientation and databases





OMG Metamodel Levels

OMG levels	**OMG Standards/examples**
M_3: Meta-meta-model	MOF
M_2: Meta-model	UML language
M_1: Model	A UML model: Class "Person" with attributes "name" and "address"
M_0: Instance	An instance of "Person": "Ola Nordmann" living in "Sotraveien 1, Bergen"

Graph and Graph Homomorphism

A (directed) **graph** G is defined by $G = (N_G, E_G, src_G, trg_G)$ where

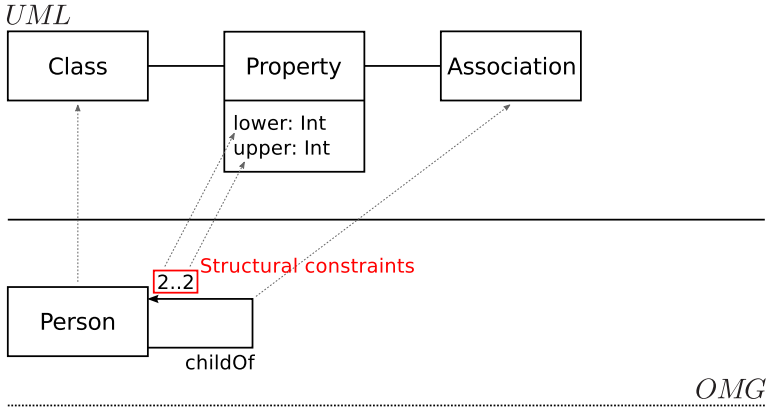
- N_G is a set of nodes
- E_G is a set of edges
- Function, $src_G : E_G \rightarrow N_G$, returns the source node of a edge
- Function, $trg_G : E_G \rightarrow N_G$, returns the target node of a edge

A **graph homomorphism** $\phi : G \rightarrow H$, between to graphs G, H is defined by two mappings $\phi_N : N_G \rightarrow N_H$ and $\phi_E : E_G \rightarrow E_H$, that preserves source and target.
It means that for each edge $e \in E_G$ we have that:
$src_H(\phi_E(e)) = \phi_N(src_G(e))$ and $trg_H(\phi_E(e)) = \phi_N(trg_G(e))$

Typing and instance

An instance of a graph G is a graph I together with a graph homomorphism $\iota : I \to G$, we say that I is typed by G

Example



Typing

- Person typed by Class
- childOf typed by Association
- 2..2 typed by Property

Conformance

An instance conforms to a model (graph) if it satisfies all constraints of the model. A person instance will conform to the model above if it has exactly two parents

Model transformation

- A model transformation is a mapping that takes models as input and creates new models (programs) as output
- Model transformations can be classified in:

  Homogenous transformation the metamodel of the input and the target model are equal

  Heterogenous transformations the metamodel of the input and output model are different

  Inplace transformation the target model is a modification of the input model

  Outplace transformation the target model are created from scratch

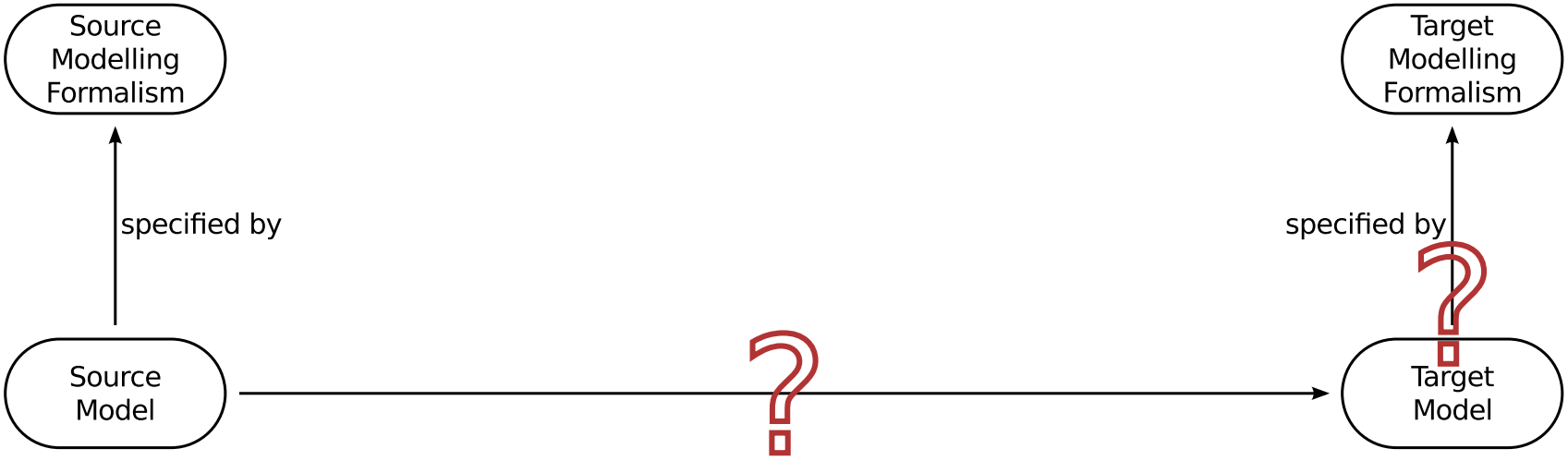
Model transformation

- Given source and target modeling formalisms and a source model ...

Model transformation

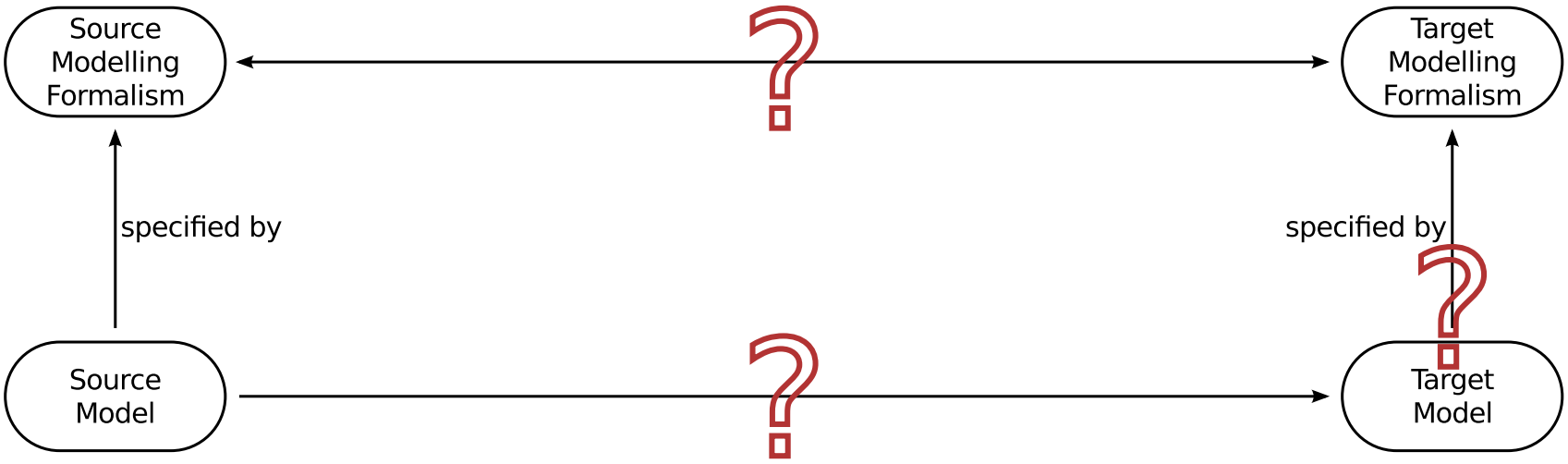
- ... we want to generate a target model

Model transformation



- How can we get the target model from the source model?

Model transformation



- We have to relate the modeling formalisms

Outline

Intro

MDE

Modeling

Languages

Meta-Modeling

Model Transformations

DPF

DPF Workbench

Summary

Diagram Predicate Framework (DPF)

- DPF is a fully diagrammatic approach to MDE, i.e.:
  - Modeling concepts and
  - Modeling constraints
- Are defined by graphs

Diagram Predicate Framework History

Formal foundations, 2004→

- 04→ Uwe Wolter and Yngve Lamo aims to apply Category Theory in Software Engineering, founded on work done by Zinovy Diskin et al. (Diskin currently at University of Toronto)
- 04-06 Ørjan Hatland (master) prototype tool Sketcher.NET
- 06→ Adrian Rutle, (PhD 2010) foundations of DPF (post doc. St. FX 2011-2012)
- 06-08 Stian Skjerveggen (master) prototype tool in Eclipse GMF
- 08→ Alessandro Rossini (PhD 2011) versioning and deep metamodelling

Diagram Predicate Framework History

Applying theory in tool development and applications, 2009→

09-11 Øyvind Bech (master) modeling tool for DPF (GEF)

10→ Florian Mantz (PhD student) model/metamodel co-evolution

10-12 Anders Sandven code generation from DPF models

11→ Xiaoliang Wang (PhD student) correctness of transformations

11→ Suneetha Sekhar (master student) DPF for services

11→ Ola Bråten (master student) storage structure for DPF

11→ Sidra Haroon (master student) visual syntax for DPF models

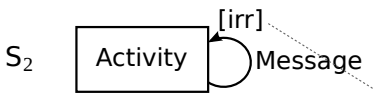
12→ Petter Barvik (master student) model transformation tool

Some DPF Results

- Domain Specific Modeling Languages, by using multilevel metamodelling hierarchies
- Diagrammatic constraint language
- Constraint aware model transformations
- Formalization of copy modify merge approach for model versioning
- Model-metamodel co-evolution
- Applications: Context modelling, Pattern Modelling, Security modelling, Service modelling, Behavioral modelling
- Tool support, (meta)modelling, code generation, constraint definition and validation

Diagram Predicate Framework

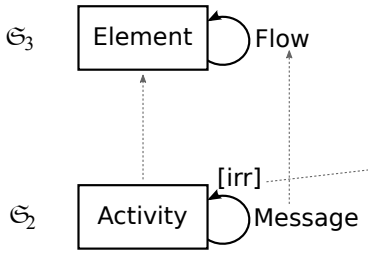
- A specification $\mathfrak{S} = (S, C^{\mathfrak{S}}{:}\Sigma)$ consists of an *underlying graph* S together with a set of *atomic constraints* $C^{\mathfrak{S}}$
- Atomic constraints are specified by *predicates* from a predefined *(diagrammatic predicate) signature* Σ
- A signature $\Sigma = (\Pi^{\Sigma}, \alpha^{\Sigma})$ 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`)]`	$1 \xrightarrow{a} 2$	X $\xrightarrow[\text{[m..n]}]{\text{f}}$ Y	$\forall x \in X : m \leq \lvert f(x) \rvert \leq n$, with $0 \leq m \leq n$ and $n \geq 1$
`[irreflexive]`	$1 \circlearrowleft a$	[irr] 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]`	$1 \circlearrowleft a$	[irr] $\boxed{X} \circlearrowleft f$	$\forall x \in X : x \notin f(x)$

Outline

Intro

MDE

Modeling

Languages

Meta-Modeling

Model Transformations

DPF

DPF Workbench

Summary

Tool Support

- Tool support needed to:
  - Facilitate the design and implementation of DSMLs
  - Provide an intuitive view of modelling
  - Runtime validation of models
  - Ensure soundness of transformations

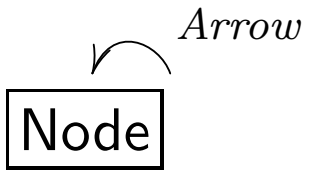
Tool Support

- Tool support needed to:
  - Facilitate the design and implementation of DSMLs
  - Provide an intuitive view of modelling
  - Runtime validation of models
  - Ensure soundness of transformations
- 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 Workbench

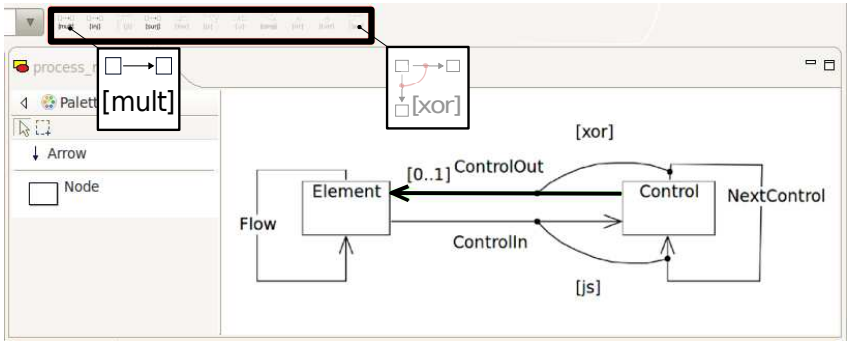
- DPF Workbench, an eclipse based modeling Tool for DPF offering:
  - Specification of (meta)models
  - Specification of domain specific signatures
  - Code generation from models
  - Generation of diagrammatic editors for (meta)models
  - Support for multilevel metamodelling
  - Conformance check of models to their metamodels:
    - Typing, by construction
    - Diagrammatic constraints, at runtime

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



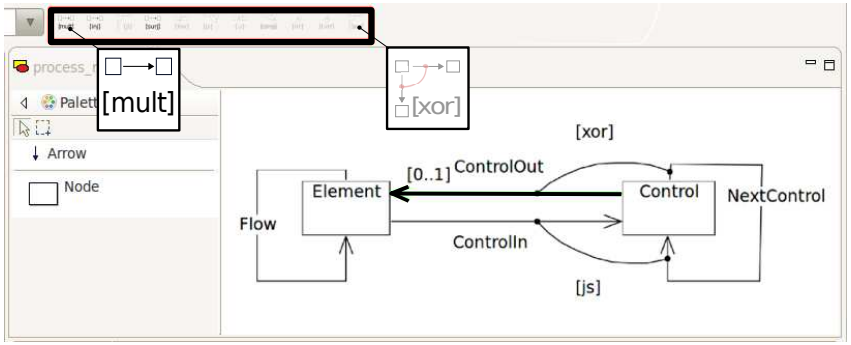
- DPF's default metamodel $\mathfrak{S}_4$ consisting of $Node$ and $Arrow$

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



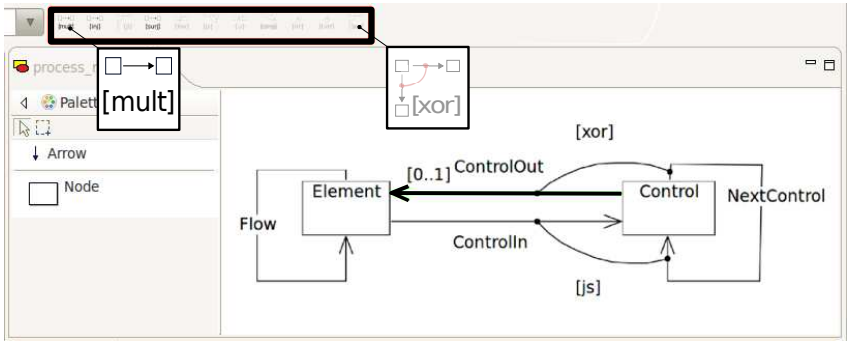
- Introducing the domain concepts:
  - $Element, Control : Node$
  - $Flow, NextControl, ControlIn, ControlOut : Arrow$

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



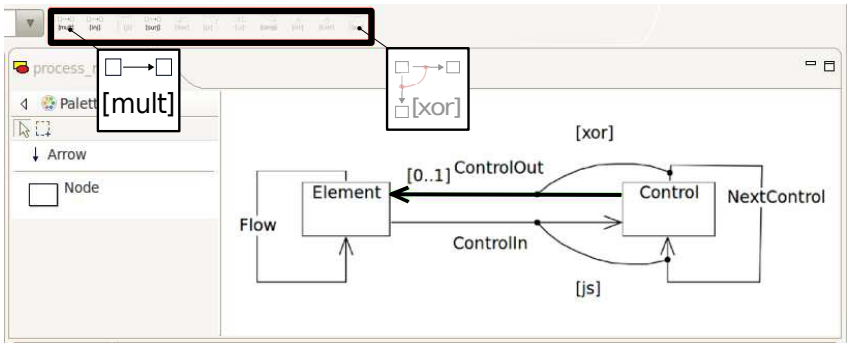
- Type Conformance
  - The DPF Editor actually checks that there exists a graph homomorphism from the specification to its metamodel while creating the specification.

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



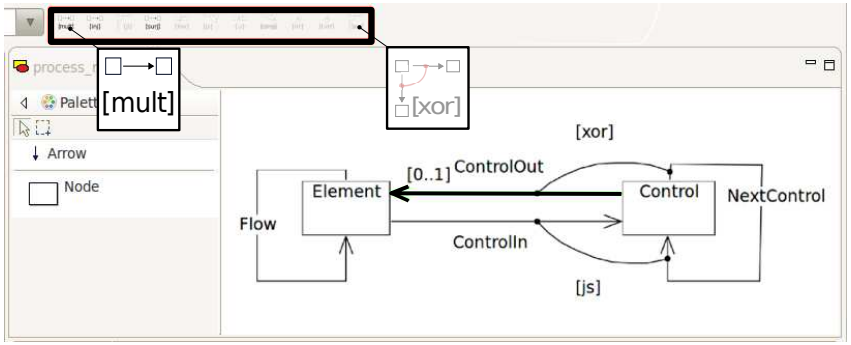
- Constraint Conformance
  - There is no constraint on the metamodel $\mathfrak{S}_4$
  - Hence $\mathfrak{S}_3$ is a valid instance of $\mathfrak{S}_4$

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



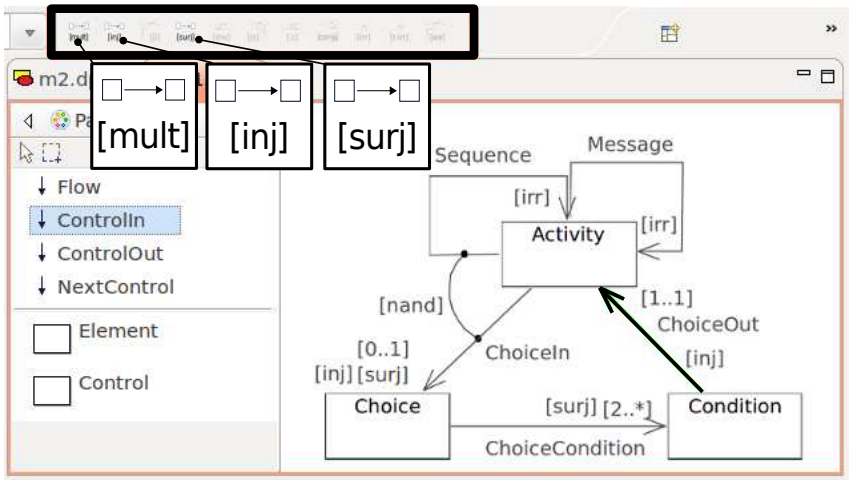
- Constraints on $\mathfrak{S}_3$:
  - Each control should have at least one incoming arrow from an element or another control
  - $Constraint$: $[js]$ between the arrows $ControlIn$ and $NextControl$

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



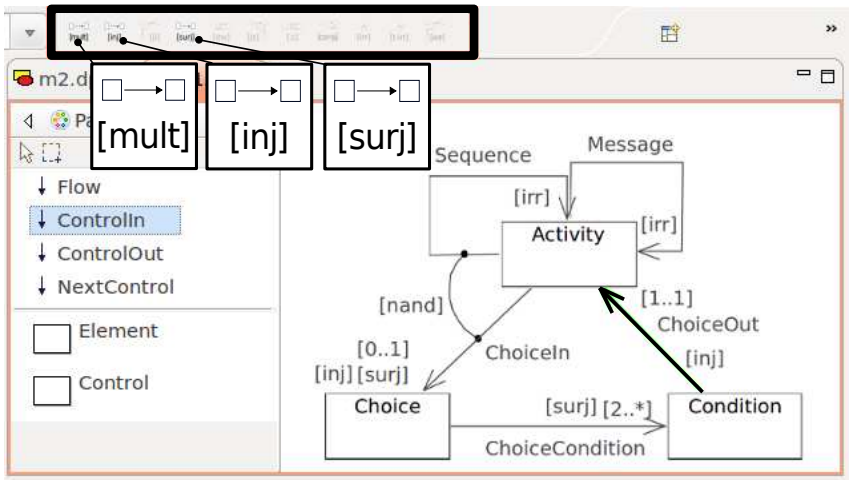
- Constraints on $\mathfrak{S}_3$:
  - Each control should be followed by either another control or by an element, not both
  - $Constraint$: $[xor]$ between the arrows $ControlOut$ and $NextControl$

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



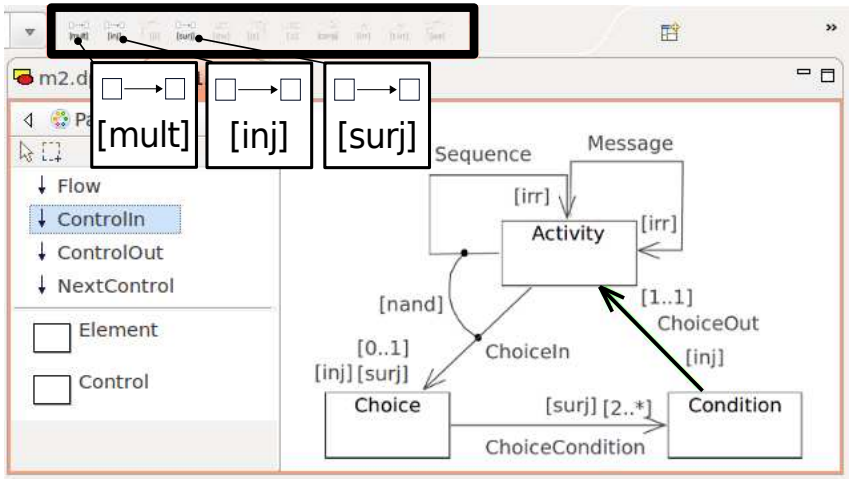
- We now generate an editor by using $\mathfrak{S}_3$ as metamodell
- Domain concepts:
  - $Activity : Element$
  - $Condition, Choice : Control$
  - $Sequence, Message : Flow$
  - $ChoiceCondition : NextControl$
  - $ChoiceIn : ControlIn$
  - $ChoiceOut : ControlOut$

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



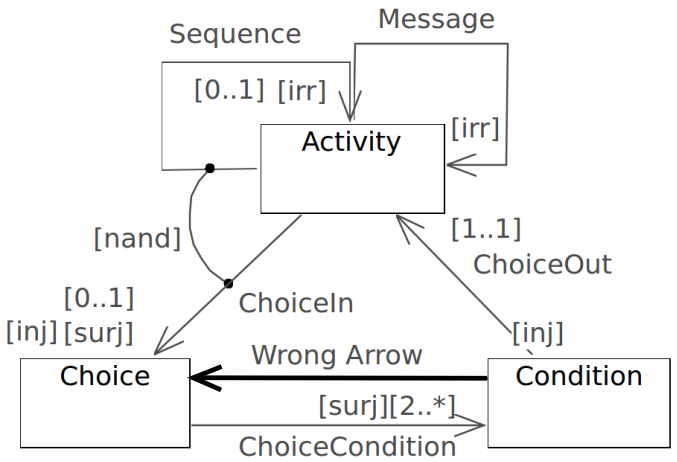
- Type Conformance by construction:
  - For instance, when we create the $ChoiceIn$ arrow of type $ControlIn$, the tool ensures that the source and target of $ChoiceIn$ are typed by $Element$ and $Control$, respectively

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



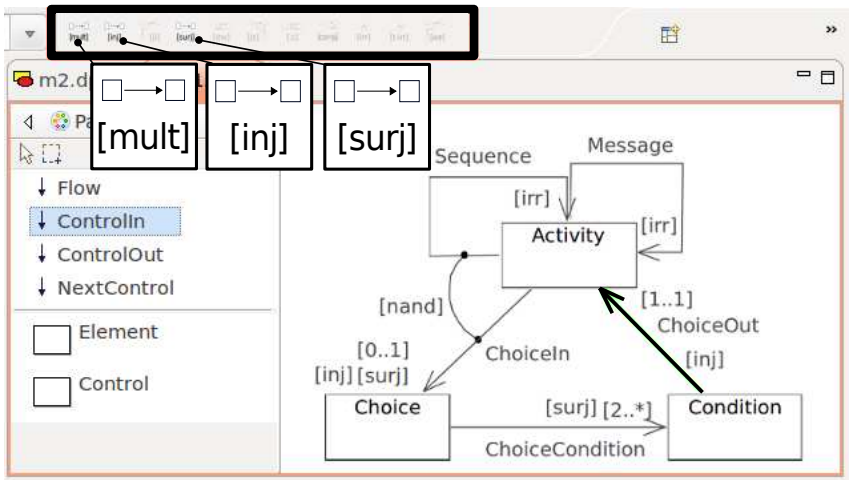
- Constraint Conformance
  - The constraints from $\mathfrak{S}_3$ are checked by corresponding validators
  - $Constraint$: $[js]$ between the arrows $ControlIn$ and $NextControl$, i.e. $Choice$ and $Condition$ has correct incoming arrows
  - $Constraint$: $[xor]$ between the arrows $ControlOut$ and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

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



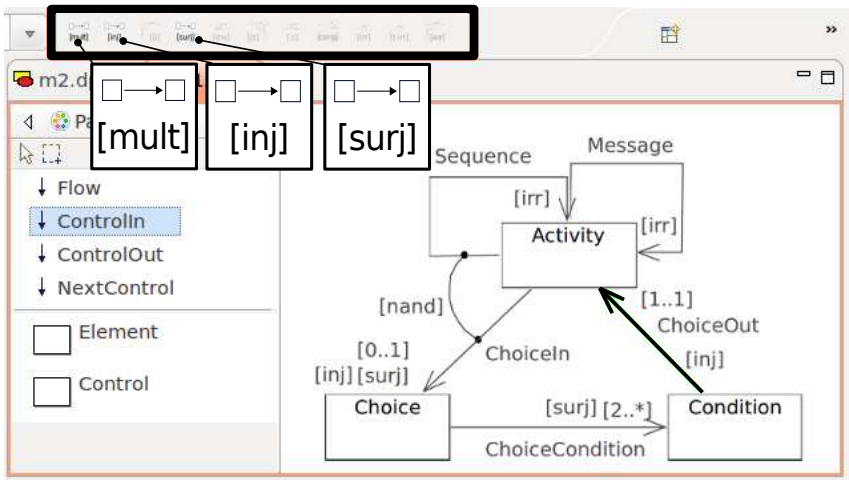
- An invalid instance of $\mathfrak{S}_3$:
  - $[xor]$ constraint on the arrows $ControlOut$ and $NextControl$ in $\mathfrak{S}_3$ is violated by the arrow $WrongArrow$
  - $Condition : Control$ is followed by a $Choice : Control$ and an $Activity : Element$, violating the requirement "each control should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

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



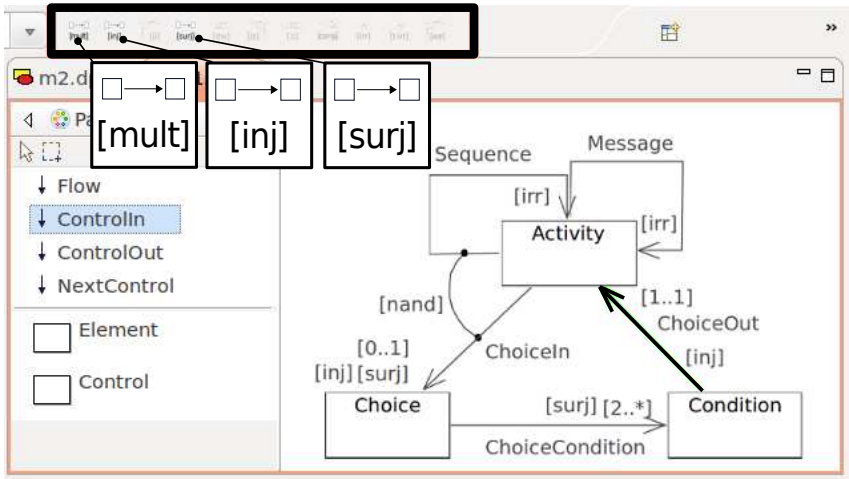
- Constraints on $\mathfrak{S}_2$
  - Each *Activity* may be connected to at most one *choice*
  - $Constraint$: $[0..1]$ on $ChoiceIn$

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



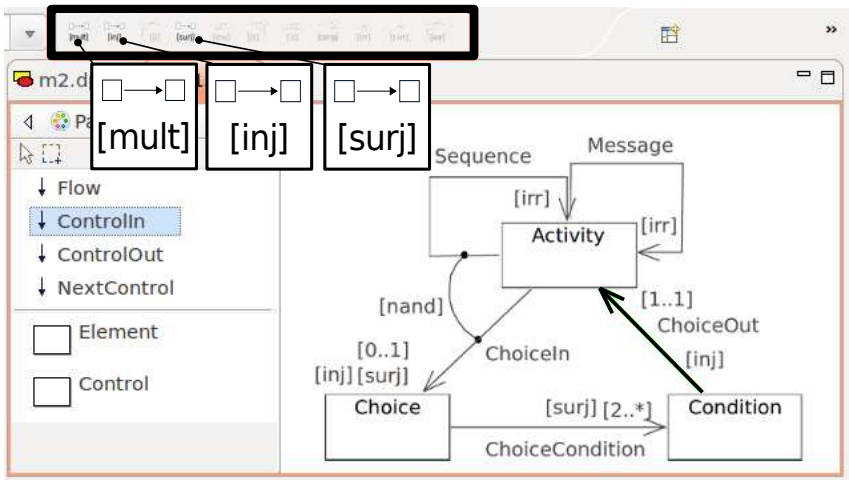
- Constraints on $\mathfrak{S}_2$
  - Each *choice* must be connected to at least two *conditions*
  - $Constraint$: $[2..*]$ on $ChoiceCondition$

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



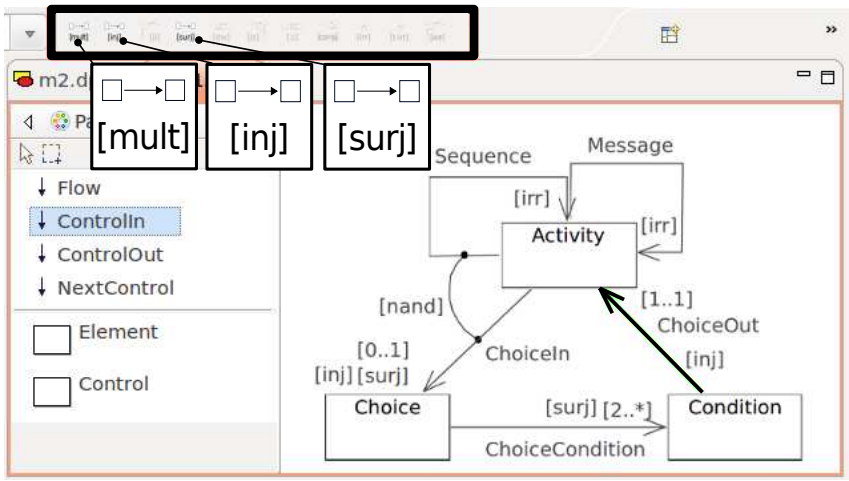
- Constraints on $\mathfrak{S}_2$
  - Each *Activity* may be connected either to a *Choice* or to another *Activity*, but not both.
  - $Constraint$: $[nand]$ between $ChoiceIn$ and $Sequence$

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



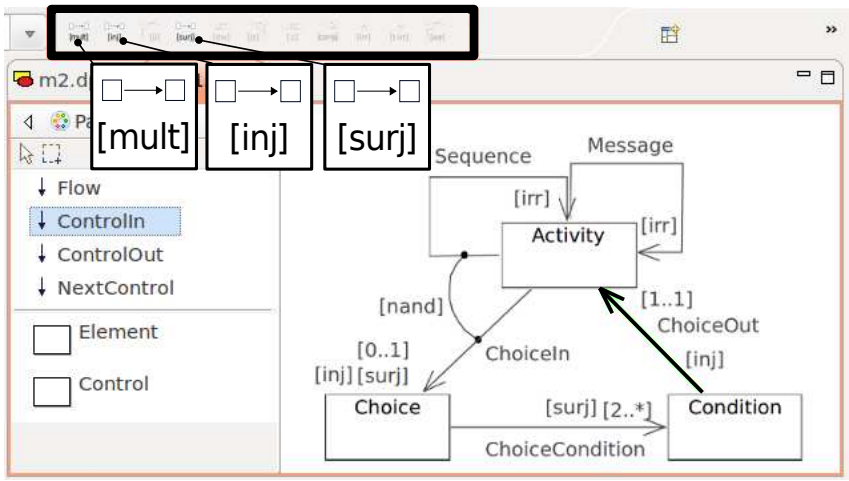
- Constraints on $\mathfrak{S}_2$
  - Each *Choice* must have exactly one *Activity* connected to it
  - $Constraint$: $[surj]$ and $[inj]$ on $ChoiceIn$

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



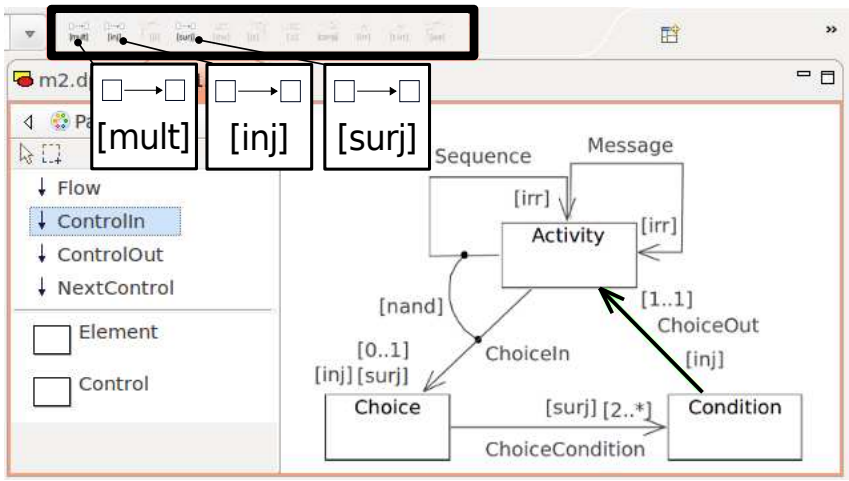
- Constraints on $\mathfrak{S}_2$
  - Each *Condition* must be connected to exactly one *Activity*
  - $Constraint$: $[1..1]$ on $ChoiceOut$

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



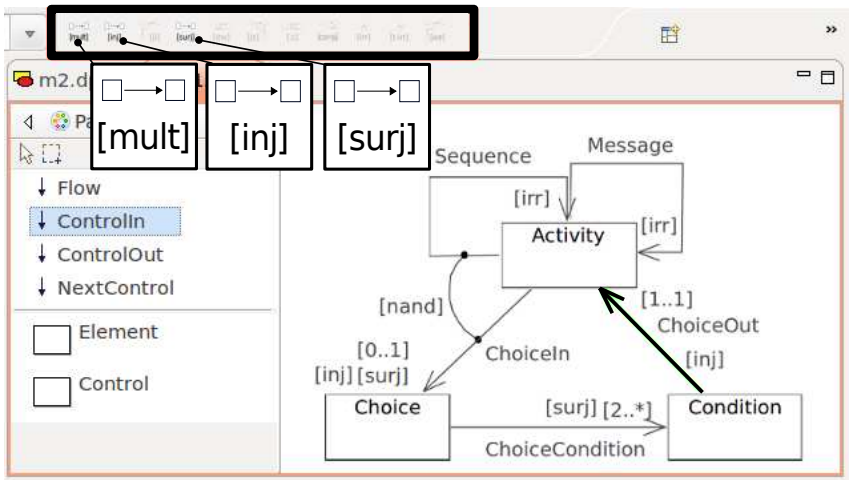
- Constraints on $\mathfrak{S}_2$
  - Each *Activity* must have exactly one *Condition* connected to it
  - $Constraint$: $[inj]$ on $ChoiceOut$

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



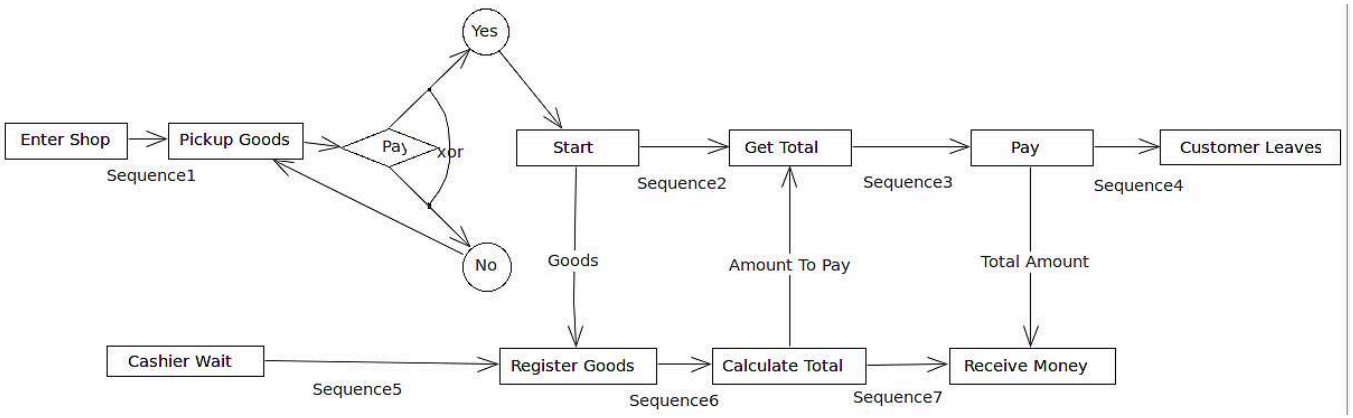
- Constraints on $\mathfrak{S}_2$
  - An *Activity* cannot send messages to itself
  - $Constraint$: $[irr]$ on $Message$

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:
  - $Enter\ Shop, Picup\ Goods : Activity$
  - $Yes, No : Condition$
  - $Goods, Amount\ To\ Pay : Message$
  - $Payment : Choice$
- Type Conformance
- Constraint Conformance

Outline

Intro

MDE

Modeling

Languages

Meta-Modeling

Model Transformations

DPF

DPF Workbench

Summary

Benefits of MDE

- Engineers can reason about system at different abstraction levels
- Platform independent models without concern of implementation details
- Less errors and faster development speed by automatic software generation
- Software development by (automatic) model transformations

Chalenges in MDE

- Modeling languages need to have right abstractions, i.e. one need domain specific modeling languages
- Constraint specification integrated in the graphical metamodeling approach
- MDE traditionally concerned by software architecture and behavior, need technologies for:
  - Model management (version control, meta-model evolution)
  - Model based security engineering
  - Model based testing, software dependencies, ...

State of the art in MDE

- Modeling; UML or EMF used as modeling language
- Model transformations; Rule based (e.g. Atlas) or add hoc transformations are used
- Meta modeling; Only tool support for 2 levels of meta-modeling
- Tool support; Eclipse based (EMF, GMF) tools
- Software constraints; Specified in text based language (OCL)

DPF Summary

- Fully diagrammatic metamodelling approach to MDE
- Constraint integrated in the metamodelling process
- Formalization of constraint aware modeltransformations
- Formalization of copy modify merge for models
- Tool support for multilevel (meta)modelling, code generation, constraint specification and validation
- ...
- for more about DPF see, `http://dpf.hib.no`

Future work on DPF

- **Process Modelling** (Work in progress, cooperation with group of Wendy McCaull St FX)
- **Tooling support for model management and versioning**
- **Model transformation tool**
- **Logical reasoning system**
- **Real application** Use DPF in a larger case study

Thanks for your attention

Questions?