A Framework to Formalise the MDE Foundations

Xavier Thirioux, Benoît Combemale,
Xavier Crégut and Pierre-Loïc Garoche

IRIT Institut de Recherche en Informatique de Toulouse (CNRS UMR 5505),
University of Toulouse, France
first_name.last_name@enseeiht.fr

June 25, 2007

Outline

1. Motivations
   - Expressing execution semantics
   - Consistency of multiple semantics
   - Issues
2. Our Framework
   - Intuitive approach
   - Formal definitions
3. Application to EMOF and OMG pyramid issues
   - EMOF Core as a Reference Model
   - Example of an EMOF model: PetriNet
   - EMOF Metacircularity
   - Definition of the MDA Pyramid
4. Conclusion & Future Works

Plan

Our general goals

Context: The TOPCASED Project (http://topcased.org)

- An open source CASE environment for critical embedded systems
- Based on Model-Driven Engineering

Goals: execution semantics of models for:

- simulation
- formal verification
- code generation

Requirement

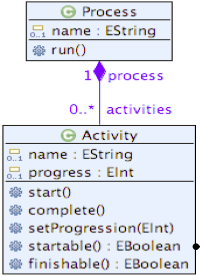
- Definition of DSL's semantics (structural and behavioral).

Experiments on a Simplified Process Description Language (SimplePDL) and on UML 2 statemachines (work in progress).

DSL semantics using operational semantics

May be achieved thanks to :

- meta-programming language (kermeta, action language. . .)




```
operation startable() : Boolean is do
  var start_ok : kermeta::standard::Boolean
  var previousActivities : seq Activity [0..*]
  var prevPrecedes : seq Precedes [0..*]
  if progress==-1 then
    // Getting the activities which have to be started
    prevPrecedes := previous.select{p | p.kind ==
      PrecedenceKind.pk_start_start }
    previousActivities := prevPrecedes.collect{p | p.before}
    start_ok := previousActivities.forAll{a | a.progress >= 0}
    // Getting the activities which have to be finished
    prevPrecedes := previous.select{p | p.kind ==
    PrecedenceKind.pk_finish_start }
    previousActivities := prevPrecedes.collect{p | p.before}
    start_ok := start_ok and
      (previousActivities.forAll{a | a.progress==100})
    result := start_ok or (previous.size() == 0)
  else
    result := false
  end
end
```

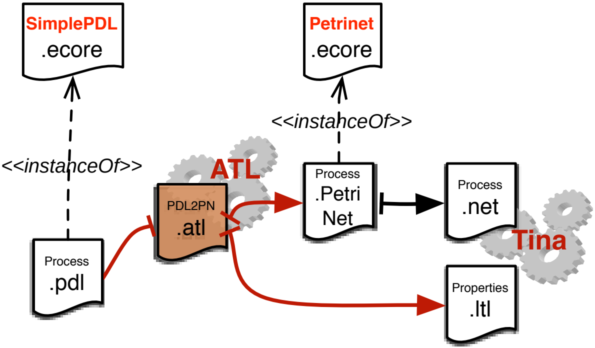
startable() Operation : Kermeta code

- endogenous transformations (ATL. . .)

Main advantage: Deals with concepts related to the DSL.

DSL semantics using translational semantics

Example : Mapping a SimplePDL model into a time Petri net one to use the TINA toolkit.



Advantage: reuse the tools available in the target technical space.

Consistency of multiple semantics

Usefulness of several semantics

1. Define a reference semantics using operational semantics.
2. Define translational semantics to reuse tools in other technical spaces

Problem

- How to assert that all the defined semantics are consistent?

Our solution

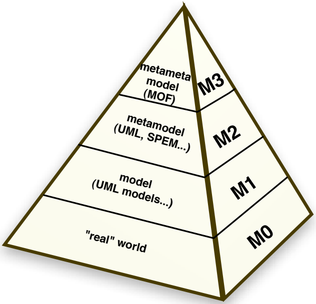
Defining a framework based on formal tools like the COQ proof assistant to

1. define operational semantics of the DSL (reference semantics)
2. define operational semantics of the technical space (semantic domain)
3. express the mapping from the DSL to the semantic domain
4. prove the equivalence of translational semantics and reference semantics

Issues

- How to formally express the concepts of models, metamodels, meta-metamodels. . . ?
  - ⇒ what are their various types ?
  - ⇒ what is the encoding in a formal domain semantics ?
- With this encoding, how to express the structural and behavioral semantics ?
  - ⇒ does a model conform to its language ?
  - ⇒ are two languages equivalent from a structural or behavioral point of view ?

Warning: *the OMG vision being one of the possible MDE view... The framework must be more general.*



Content of this talk

Content of this talk

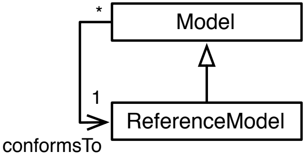
- describe a formal framework to formalise MDE foundations
- apply it to the formalisation of a subset of EMOF Core

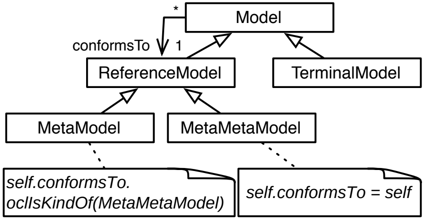
Warning

At the moment, only formalisation of static semantics is addressed

Plan

Jouault & Bézivin formalisation [FMOODS'06]





- Notion of *ReferenceModel*
  - "A Model conforms to a ReferenceModel"
  - "A ReferenceModel is a Model"

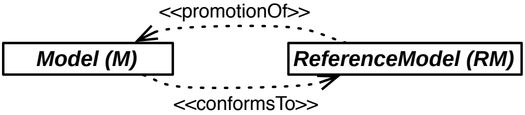
    $\Rightarrow$ A *ReferenceModel* conforms to a *ReferenceModel*,

    $\Rightarrow$ An initial ReferenceModel has to be reflexive.
- Illustrated with the formalisation of KM3
- Implemented in Prolog with some limitations

ReferenceModel and Model

Intuitive approach

- REFERENCEMODEL ($\langle concepts, relations, semantics \rangle$):
  - modelling language from which one can define a family of models,
  - specifies the semantic properties of its models.
- MODEL ($\langle objects, links \rangle$): the instance level.



- A model MUST conform to a RM.
- A RM may be directly defined.
- A RM may be obtained as the promotion of a model.

Conformity and promotion

Intuitive approach

Conformity

1. Every object *o* in *M* is the instance of a class *C* in *RM*;
2. Every link between two objects is such that it exists, in *RM*, a reference between the two classes typing the two elements.
3. Every semantic property defined in *RM* is satisfied in *M*.

Promotion

1. Identify the different concepts among the model elements.
2. Identify relations between the previous concepts.
3. Define the different semantic properties that must hold on the models that conform to the Reference Model.

Formal Approach
General Definitions

Let us consider:

Definition

- `Classes` the set of all possible classes,
- `References` the set of reference labels,
- `Objects` the set of instances of such classes.

Definition

- $\mathscr{C} \subseteq \texttt{Classes}$ be a set of classes,
- $\mathscr{R} \subseteq \{\langle c_1, r, c_2\rangle \mid c_1, c_2 \in \mathscr{C}, r \in \texttt{References}\}$ be the set of references among classes where
$$\forall c_1 \in \mathscr{C}, r \in \texttt{References}, card\{\langle c_1, r, c_2\rangle \in \mathscr{R}\} \leq 1.$$

Formal Approach
Model and ReferenceModel

Definition (Model)

A model $\langle MV, ME \rangle \in \mathtt{Model}(\mathscr{C}, \mathscr{R})$ is a multigraph built over a finite set *MV* of typed objects and a finite set *ME* of typed edges such that:

$$MV \subseteq \{\langle o, c\rangle \mid o \in \mathtt{Objects}, c \in \mathscr{C}\}$$
$$ME \subseteq \left\{ \langle\langle o_1, c_1\rangle, r, \langle o_2, c_2\rangle\rangle \;\middle|\; \langle o_1, c_1\rangle, \langle o_2, c_2\rangle \in MV, \langle c_1, r, c_2\rangle \in \mathscr{R} \right\}$$

Definition (ReferenceModel)

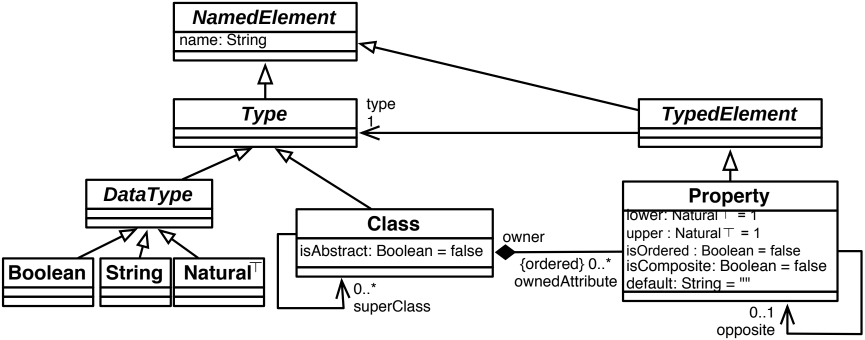
A reference model $\langle (RV, RE), conformsTo\rangle$ is a **multigraph** built over a finite set *RV* of classes and a finite set *RE* of references, with **semantic properties** over the instances of both classes and references.

$$RV \subseteq \mathtt{Classes}$$
$$RE \subseteq \{\langle c_1, r, c_2\rangle \mid c_1, c_2 \in RV, r \in \mathtt{References}\}$$
$$conformsTo : Model(RV, RE) \longrightarrow Bool$$

Plan

EMOF Core as a Reference Model

Traditional notation (class diagram notation)



EMOF Core as a Reference Model

Formal notation

Definition (EMOF Core)

The EMOF Core Reference Model is $\langle\langle RV, RE\rangle, conformsTo\rangle$ where :

$$RV \triangleq \{ NamedElement, Type, TypedElement, DataType, Boolean, String, Natural^{\top}, Class, Property \}$$

$$RE \triangleq \{ \langle Class, \texttt{ownedAttribute}, Property\rangle, \langle Class, \texttt{isAbstract}, Boolean\rangle, \langle Class, \mathrm{inh}, Type\rangle, \ldots \}$$

$$\begin{aligned} conformsTo(\langle MV, ME\rangle) \triangleq\ & \langle MV, ME\rangle \in \texttt{Model(RV, RE)} \\ & \wedge\ lower(TypedElement, \texttt{type}, 1) \\ & \wedge\ upper(TypedElement, \texttt{type}, 1) \\ & \wedge\ \text{and other semantic properties (next slide)}\ldots \end{aligned}$$

EMOF Core as a Reference Model: semantics

Definition (Lower Property)

$$lower(c_1 \in RV, r_1 \in RE, n \in Natural^{\top}) \triangleq \langle MV, ME\rangle \mapsto \forall \langle o, c\rangle \in MV, c = c_1 \Rightarrow card(\{m_2 \in MV \mid \langle\langle o, c_1\rangle, r_1, m_2\rangle \in ME\}) \geq n$$

Definition (Opposite Property)

$$isOpposite(r_1, r_2 \in RE) \triangleq \langle MV, ME\rangle \mapsto \forall m_1, m_2 \in MV, \langle m_1, r_1, m_2\rangle \in ME \Leftrightarrow \langle m_2, r_2, m_1\rangle \in ME$$

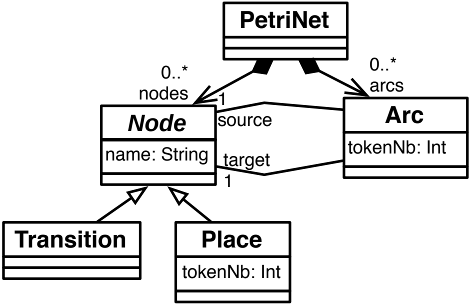
Definition (Abstract Classes)

$$isAbstract(r \in RE, c_1 \in RV) \triangleq \langle MV, ME\rangle \mapsto \forall \langle o, c\rangle \in MV, c = c_1 \Rightarrow \exists c_2 \in RV, \langle\langle o, c_2\rangle, r, \langle o, c_1\rangle\rangle \in ME$$

And also: upper, inheritance, *composite*, *ordered*. . . .

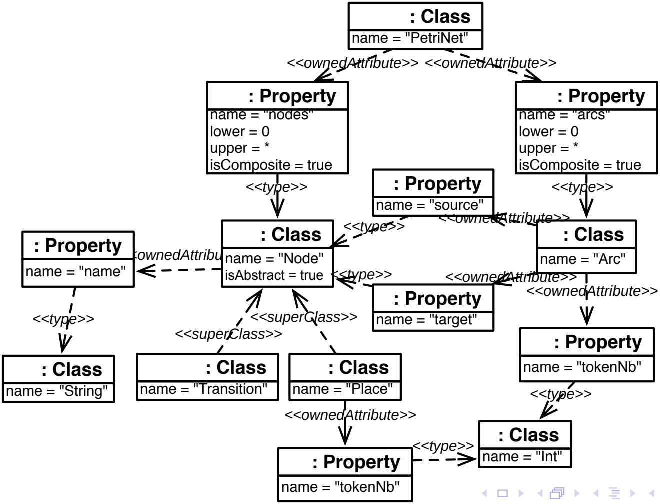
Example of an EMOF model: PetriNet

Class diagram graphical notation



Example of an EMOF model: PetriNet

Object diagram notation



Example of an EMOF model: PetriNet

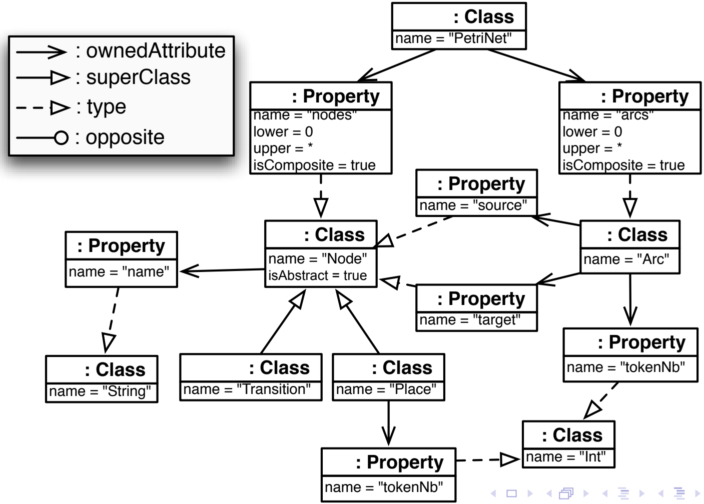
Formal notation

$$
\begin{aligned}
MV \triangleq \quad & \{ \langle PetriNet, Class \rangle, \\
& \langle arcs, Property \rangle, \\
& \langle nodes, Property \rangle, \\
& \langle Node, Class \rangle, \ldots \}
\end{aligned}
$$

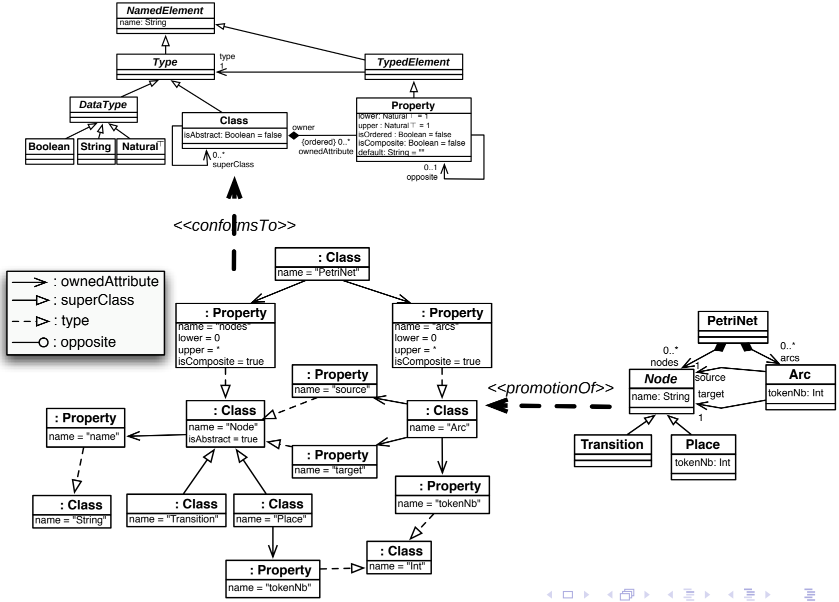
$$
\begin{aligned}
ME \triangleq \quad & \{ \langle \langle PetriNet, Class \rangle, \texttt{ownedAttribute}, \langle arcs, Property \rangle \rangle, \\
& \langle \langle arcs, Property \rangle, \texttt{type}, \langle Node, Class \rangle \rangle, \\
& \langle \langle PetriNet, Class \rangle, \texttt{ownedAttribute}, \langle nodes, Property \rangle \rangle, \\
& \ldots \}
\end{aligned}
$$

Example of an EMOF model: PetriNet

Other graphical notation

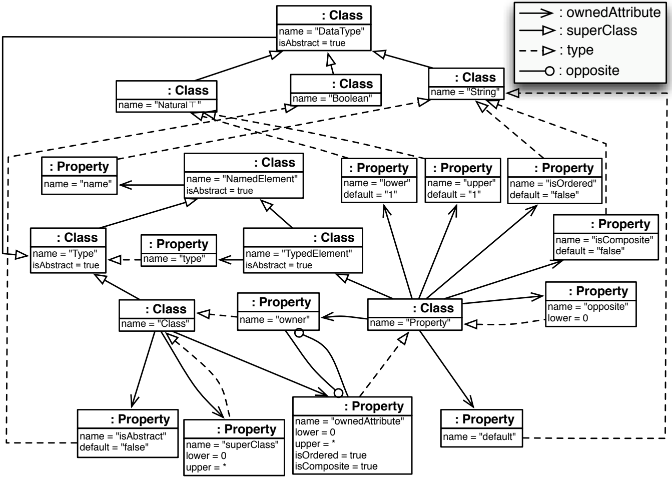


Example of an EMOF model: PetriNet
Conformity and promotion



Validation of the EMOF Metacircularity

Structural Promotion, the EMOF Model



Validation of the EMOF Metacircularity

Structural Promotion, the EMOF (RM) graph definition

let (MV, ME) be the graph of the model

Definition (Nodes)

The set *RV* of reference model vertices is defined such that,

$$RV = \{o.name | \langle o, Class\rangle \in MV\}$$

Definition (Edges)

The set *RE* is the set of edges in the reference model such that,

$$RE \triangleq \{ \ \langle o_1.name, o_2.name, o_3.name\rangle \ \Big| \ Rule(o_1, o_2, o_3) \ \}$$

Where,

$$\begin{aligned} Rule(o_1, o_2, o_3) \triangleq \quad & \langle\langle o_1, Class\rangle, \texttt{ownedAttribute}, \langle o_2, Property\rangle\rangle \in ME \\ & \wedge \langle\langle o_2, Property\rangle, \texttt{type}, \langle o_3, Class\rangle\rangle \in ME \end{aligned}$$

Validation of the EMOF Metacircularity

Definition of the semantics

Definition (Lower (resp. Upper) Property)

$$\bigwedge \left\{ \; lower(o_1.name, o_2.name, o_2.\mathtt{lower}) \;\middle|\; Rule(o_1, o_2, o_3) \; \right\}$$

Definition (Opposite Property)

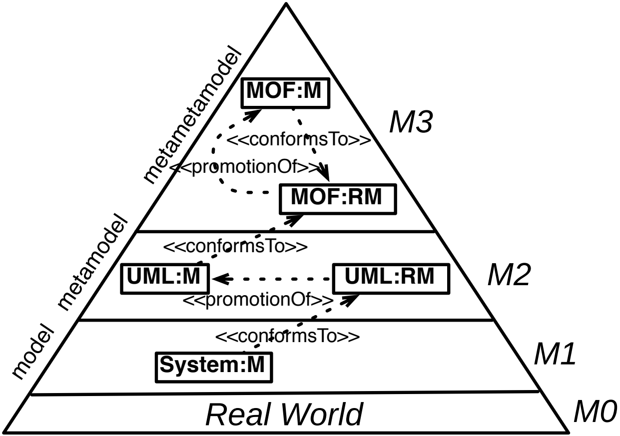
$$\bigwedge \left\{ \; isOpposite(o_1.name, o_2.name) \;\middle|\; \langle\langle o_1, Property\rangle, \mathtt{opposite}, \langle o_2, Property\rangle\rangle \in ME \right.$$

Definition (Abstract Classes)

$$\bigwedge \left\{ \; isAbstract(o.name) \;\middle|\; \langle\langle o, Class\rangle, \mathtt{isAbstract}, \langle \mathtt{true}, Boolean\rangle\rangle \in ME \; \right\}$$

And also: *superclass*, *composite*, *ordered*....

Definition of the MDA Pyramid



Plan

Conclusion & Future Works

Conclusion:

- A mathematical formalisation of the MDE framework
- Being implemented using the COQ proof assistant
- Formal definition of the *conformsTo* predicate and *promotion* operation
- Formal definition of the EMOF Core language
- Framework validation through the expression of traditional problems

Future Works:

- Expressing operational semantics (including operations)
- Proof of behavioral equivalence (bissimulation)
- Reflecting the OCL logic into our formalism
- Formalizing common operations on models such as *merge*, *import*...

Thank you
for your attention...

Questions?