Model Driven Engineering
Bringing formal validation in the industrial process

Benoît Combemale\textsuperscript{1}, Bernard Berthomieu\textsuperscript{2}, Jean-Paul Bodeveix\textsuperscript{1}, Xavier Crégut\textsuperscript{1}, Mamoun Filali\textsuperscript{1}, Pierre-Loïc Garoche\textsuperscript{1}, Marc Pantel\textsuperscript{1}, Martin Strecker\textsuperscript{1}, Xavier Thiriaux\textsuperscript{1} & François Vernadat\textsuperscript{2}

\textsuperscript{1}IRIT CNRS
2, rue Charles Camichel - BP 7122
F-31071 Toulouse Cedex 7

\textsuperscript{2}LAAS CNRS
7, avenue du colonel Roche
F-31077 Toulouse Cedex 4

June 16, 2007
Plan

1. Introduction
   - Applicative domain
   - The TOPCASED Project

2. Model Driven Engineering

3. MDE technologies
   - Relating modeling and programming languages
   - Metamodeling
   - Concrete syntaxes
   - Model transformations and weaving
   - Model Validation and Feedback

4. Semantics in MDE
   - Static validation for structural properties (Axiomatic semantics)
   - Operational semantics
   - Direct style using Kermeta
   - SOS style using ATL
   - Required metamodels extensions
Model Driven Engineering for safety critical systems

- **Purpose**: Introducing formal V & V in the industrial practise
- **Constraints**: Cooperation of several partners
  - Industrial end users
  - Software engineering
  - Formal technologies
  - Tool vendors
- **Proposal**: Ease cooperation and understanding
- **Proposal**: Reduce the cooperative chain by empowering users
- **Requirement**: Ensure the correctness (tools must preserve properties)
Toolkit in OPen source for Critical Applications & SystEms Development

- Open source CASE tool for safety critical systems (aeronautic, automotive, space, ...) based on the Eclipse platform (V1 for Europa).
- Validation & Verification are essential points
- System and Context Behavior Modeling (iTV out of scope, not really)
- Domain Specific Languages & Formal Technologies
- Generative technologies for tool development
- Rely on existing tools and participate in their development
- Add tools based on users requirements

Partners: Airbus France, EADS Astrium, Rockwell Collins, Siemens-VDO, Thales Aerospace, Anyware Technologies, Atos-Origin, CS, Sodifrance, Micouin consulting, Sinters, Sodifrance, Sogeti-Hitech, Sopra Groupe, Tectosages, TNI, ENSIETA, ESEO, ESSAIM, FéRIA (IRIT/LAAS/ONERA), INRIA Rhone-alpes, INRIA Bretagne, LINA, UFSC (Brésil), ... (and rising)

More details on:
- Web site: http://www.topcased.org,
TOPCASED : Current status

- Started in Q3/2005, current version 1.0M4.1
- Graphical editors for Ecore, UML2, AADL, SAM, SysML, EAST-ADL, SPEM
- EMF, EMFT, UML, UMLT, GEF and low-level GMF based
- Graphical editor generator (metamodels & OCL constraints validation)
- Textual metamodel-sensitive OCL constraints editor
- Simple ModelBus and Eclipse deployment tools implementation
- Model importation (using ATL)
- Experiments in code, tests and documentation generation (using JET, OBE0 Acceleo, oAW, ATL, and others)
- Experiments in behavioral semantics (model execution and simulation using Java, Kermeta, ATL and AGG/Tiger)
- Experiments in model validation (model checking using ATL and TINA)
- Common Model based development process for safety critical systems
Our Approach (through case study)

SimplePDL metamodel
Our Approach (through case study)
Our Approach (through case study)
Our Approach (through case study)
TOPCASED : Toward version 1.0

- Must be an industrial strength CASE toolkit
- Synchronised with Europa (June 2007)
  - Requirement traceability (TNI ReqTify based)
  - Change management (generic web-based tool)
  - Collaborative work
  - Import-export capabilities
- Verification & Validation: Prototypes for SimplePDL (SPEM subset) and UML2/StateCharts
- Code and documentation generation: Prototypes for UML2/Class and UML2StateCharts
TOPCASED : Toward version 2.0

- Synchronised with Ganymede (June 2008)
- Contribute parts of the Graphical editor generator to GMF3
- Port the Ecore editor to GMF3 and contribute to EMP
- Merge TOPCASED-UML2 with CEA-LIST PAPYRUS using GMF3 and contribute to EMP
- Use and contribute to MDDi ModelBus
- Graphical model animator generator for V & V results
- Model checking intermediate language (FIACRE and POLYCHRONY)
- Model execution kernel (Java-based, KERMETA and others)
- Model transformation to intermediate V & V language (ATL-based)
- Assistant for generating these transformations for V & V
- Model transformation to model checking tools
- Model transformation validation technologies
- and many other things
Model driven engineering

- Nothing (really) new under the sun
- Rely on existing technologies
- But promotes an unifying point of view
- System development process centered on models
- Common generative and user friendly tools
MDE : Purpose of models

- Help in understanding the problems
- Abstraction of the real system
- Viewpoint/Projection of the real system
- Proposal : Every software-managed entity is a model
- R. Milner : Towers of models (composition and refinement tower)
- Uniform representations for software-managed entities
- Common tools for manipulating entities
- Mega-models : Coordinated models
MDE : The Y development cycle

- Separate introduction of the various system concerns
- Promotes (and enforce, thus requires) Separation of Concerns
- Leftmost branch is the main concern
- Right branches are the secondary concerns
- Each branch correspond to a coordinated model
- Defined in a Domain Specific Language
MDE : Promoting DSL

- Domain specific languages
- Specific purpose languages exhibit strong properties
- Specific purpose languages have (quite) simple semantics
- One DSL for each concern of a system
- One DSL for each semantics hard-point
- Coordinating DSLs
- UML = A family of (ill) coordinated DSL
- UML profile = DSL hidden in a General Purpose Language
- Problem = Explaining and managing restricted UML
Semantics and programming languages

Concrete Syntax
Semantics and programming languages

Parsing

Concrete Syntax

Abstract Syntax Tree
Semantics and programming languages

Parsing

Concrete Syntax

Abstract Syntax Tree

Translational

Code Generation

Target Languages
Semantics and programming languages

Abstract Syntax Tree

Intermediate Languages

Target Languages

Code Generation

Translational

Pantel et al. (IRIT & LAAS)

June 16, 2007 13 / 61
Semantics and programming languages

Concrete Syntax → Parsing → Abstract Syntax Tree → Code Generation → Target Languages

Pretty Printing

Reverse Engineering
Semantics and programming languages

Concrete Syntax

Abstract Syntax Tree

Static analysis

Target Languages

Parsing

Code Generation

Translational
Introduction
Model Driven Engineering
MDE technologies
Semantics in MDE

Relating modeling and programming languages
Metamodeling
Concrete syntaxes
Model transformations and weaving
Model Validation and Feedback

Semantics and programming languages

Concrete Syntax

Parsing

Abstract Syntax Tree

Static analysis

Translational
Axiomatic

Proof

Code Generation

Target Languages
Semantics and programming languages

Concrete Syntax

Abstract Syntax Tree

Parsing

Static analysis

Translation

Axiomatic

Operational

Proof

Reduction

Semantic context

Abstract Syntax Tree (Value)

Semantic context
Semantics and models
Semantics and models

Reference Model

Model Transformation

Concrete Models

Target Models
Semantics and models
Semantics and models
Metamodelling technologies

- Tools to define DSL
- A DSL definition is a coordinated set of models
- Models are Graphes = Attributed nodes and edges
- Metamodel : (kind of) Type of the model
- Model nodes and edges are instances of reference model nodes
- Coordinated DSL for design DSL
- OMG/MOF, EMF/ECORE, Graph Grammars
- Metamodel : DSL afor bstract syntax : concepts and their relationship
**SimplePDL, a Process Modelling Language**

- stems from:
  - SPEM (OMG)
  - UMA (EPF)
- could easily be extended:
  - roles
  - products
  - ...
Modelling a complex process

MACAO process
What do we want to check?

- resource constraints
  - computers
  - manpower
- timing constraints
  - minimum achievement time
  - maximum achievement time
- causality constraints
  - startToStart
  - startToFInish
  - finishToStart
  - finishToFInish
- . . .
- for some execution
- or for all executions
What do we want to check?

- resource constraints
  - computers
  - manpower
- timing constraints
  - minimum achievement time
  - maximum achievement time
- causality constraints
  - startToStart
  - startToFinish
  - finishToStart
  - finishToFinish
- ...
- for some execution
- or for all executions
A sample property

- checking the global *min_time* constraint
Some SimplePDL-expert properties

For all executions
- every WD must start and then finish
- once a WD is finished, it remains so
- resource and causality constraints must hold

For some execution
- every WD must take between $\text{min}$ and $\text{max}$ time units to complete
- the overall process is able to finish
The Temporal Object Contraint Language

TOCL (Gogolla & al., 2002) embeds
- the Object Constraint Language for spatial relations
- the Linear Temporal Logic for time relations

TOCL is used
- to express fine behavioral spec (next, existsNext, always, sometime, . . .)
- about some execution or all executions

Some properties of WD alone
- \( \forall w, (w.state = \text{notStarted} \land \text{sometime } w.state = \text{inProgress}) \)
- \( \forall w, \text{always } (w.state = \text{inProgress} \Rightarrow \text{sometime } w.state \in \{\text{finishedOk, tooEarly, tooLate}\}) \)
- \( \forall w, \text{always } (w.state = \text{finishedOk} \Rightarrow \text{always } w.state = \text{finishedOk}) \)
- \( \neg \exists w, \text{always } w.state \neq \text{finishedOk} \)
Extending the metamodel

```
Process
max_time: EInt
min_time : EInt
workDefinitions
workSequences
0..* predecessor
0..* linkToSuccessor
0..* linkToPredecessor

WorkDefinition
name : EString
max_time: EInt
min_time : EInt
workDefinition

Guidance
description: EString

Parameter

Ressource
name: EString
OccurencesNb : EInt

Guidances

workDefinitions

WorkSequence
name: EString
linkType : WorkSequenceType
predecessor
successor

<<enumeration>>
WorkSequenceType
- startToStart
- startToFinish
- finishToStart
- finishToFinish

Pantel et al. (IRIT & LAAS)

SIMPLEPDL & TOCL Case Study
june 16, 2007 22 / 61
Extending the metamodel

```
<<enumeration>>
StateType
- notStarted
- InProgress
- finishedOk
- tooLate
- tooEarly
```

```
Process
max_time: EInt
min_time : EInt

WorkDefinition
name : EString
state : StateType
max_time: EInt
min_time : EInt

Guidance
description: EString

Parameter

Ressource
name: EString
OccurencesNb : EInt

WorkSequence
name: EString
linkType : WorkSequenceType

<<enumeration>>
WorkSequenceType
- startToStart
- startToFinish
- finishToStart
- finishToFinish
```

Pantel et al. (IRIT & LAAS)

SimplePDL & TOCL Case Study
A sample run
Illustrating operational semantics

- $t = 0$: WDs are notStarted
A sample run
Illustrating operational semantics

- $t = 0$: WDs are notStarted
- $t = 1$: A starts

\[
egin{align*}
  t = 0 : & \text{ WDs are notStarted} \\
  t = 1 : & \text{ A starts} \\
  t = 3 : & \text{ B starts} \\
  t = 4 : & \text{ A completes} \\
  t = 5 : & \text{ C starts} \\
  t = 7 : & \text{ B completes} \\
  t = 8 : & \text{ C completes}
\end{align*}
\]

\begin{itemize}
  \item \text{min} \text{time} = 5 \\
  \text{max} \text{time} = 11
\end{itemize}

\begin{itemize}
  \item \text{state} = \text{inProgress} \\
  \text{min} \text{time} = 2 \\
  \text{max} \text{time} = 4
\end{itemize}

\begin{itemize}
  \item \text{state} = \text{notStarted} \\
  \text{min} \text{time} = 1 \\
  \text{max} \text{time} = 4
\end{itemize}

\begin{itemize}
  \item \text{state} = \text{notStarted} \\
  \text{min} \text{time} = 2 \\
  \text{max} \text{time} = 3
\end{itemize}
A sample run
Illustrating operational semantics

- $t = 0$: WDs are notStarted
- $t = 1$: A starts
- $t = 3$: B starts
A sample run
Illustrating operational semantics

- $t = 0$: WDs are notStarted
- $t = 1$: A starts
- $t = 3$: B starts
- $t = 4$: A completes
A sample run
Illustrating operational semantics

- $t = 0: WDs$ are notStarted
- $t = 1: A$ starts
- $t = 3: B$ starts
- $t = 4: A$ completes
- $t = 5: C$ starts

$$
\begin{align*}
\text{\textless\textgreater WorkDefinition} & \text{\textgreater} A \\
\text{state = finishedOk} & \\
\text{min\_time = 2} & \\
\text{max\_time = 4} & \\
\text{\textless\textgreater WorkDefinition} & \text{\textgreater} B \\
\text{state = inProgress} & \\
\text{min\_time = 2} & \\
\text{max\_time = 3} & \\
\text{\textless\textgreater WorkDefinition} & \text{\textgreater} C \\
\text{state = inProgress} & \\
\text{min\_time = 1} & \\
\text{max\_time = 4} & \\
\text{\textless\textgreater Process} & \text{\textgreater} P \\
\text{min\_time = 5} & \\
\text{max\_time = 11} & \\
\end{align*}
$$
A sample run
Illustrating operational semantics

- $t = 0$: WDs are notStarted
- $t = 1$: A starts
- $t = 3$: B starts
- $t = 4$: A completes
- $t = 5$: C starts
- $t = 7$: B completes

Diagram:

- <<Process>> P
  - min_time = 5
  - max_time = 11

- <<WorkDefinition>> A
  - state = finishedOk
  - min_time = 2
  - max_time = 4

- <<WorkDefinition>> B
  - state = finishedOk
  - min_time = 2
  - max_time = 3

- <<WorkDefinition>> C
  - state = inProgress
  - min_time = 1
  - max_time = 4
A sample run

Illustrating operational semantics

- \( t = 0 \): WDs are notStarted
- \( t = 1 \): A starts
- \( t = 3 \): B starts
- \( t = 4 \): A completes
- \( t = 5 \): C starts
- \( t = 7 \): B completes
- \( t = 8 \): C completes
Definition of concrete syntaxes

- Means for a user to define parts of a model
- Can be any kind: Textual, Graphical, whatever
- DSL for providing concrete syntax related to metamodel
- Textual: TCS (AMMA), Syntaks (Kermeta), ...
- Graphical: GMF (Eclipse), Topcased, ...
Concrete syntax in Topcased

- Add graphical editors to DSL
  ⇒ To ease the building of metamodel-conforming models
- Graphical editor generator (metamodels & OCL constraints)
  - Pragmatic bottom-up approach: Basic generator extended when common requirements occur
  - EMF, EMFT, GEF and low-level GMF based
  - Hierarchical graph (Node and Edge) kind of diagrams
  - Customized-EMF properties generator
  - Customized-EMF outline generator
  - Editor documentation generator
  - Hand-written customisation (40% for UML2)
Concrete graphical syntax definition

Abstract syntax (Ecore metamodel)

Concrete graphical syntax

Process

Activity

Guidance

Precedes

MyProcess

MyGuidance

A1

A2

A3

A4

A5
Définition d’une syntaxe concrète
Concrete graphical syntax definition
Concrete graphical syntax definition

Graphical editor

Pantel et al. (IRIT & LAAS)
Model transformations and weaving

- Express the relations between models
- Functional (model transformation):
  - Build a new model from existing models
  - Modify an existing models (imperative)
- Bijective functional (reverse model transformation)
  - Either defined separatly (proof of bijection required)
  - Either generated from an «orientable» relation
- Non-orientable relation: Checks the consistency of both models
- OMG QVT specification: Several layers (relational, operational, ...)
- Declarative tools: ATL, Graph-grammar based (add/remove, Triple Graph Grammar)
- Operational tools: Kermeta
Expressing WorkDefinition Semantics through Petri Nets

A first net to encode simple states:
Expressing WorkDefinition Semantics through Petri Nets

Another net to encode time constraints:

![Petri Net Diagram]

Pantel et al. (IRIT & LAAS)

SimplePDL & TOCL Case Study

June 16, 2007
Expressing WorkDefinition Semantics through Petri Nets

Then, we add resource constraints:

```
<<Resource>>
Machine
occurrenceNb = 4

<<WorkDefinition>>
Design
state = finishedOk
min_time = 5
max_time = 11
```

![Petri Net Diagram](image)
PetriNet, temporal Petri nets
Expressing WorkDefinition Semantics through Petri Nets

Finally, we add causality constraints:

```
<<Process>> P
--------------
min_time = 5
max_time = 11

<<WorkDefinition>> A
---------------------
state = notStarted
min_time = 2
max_time = 4

<<WorkDefinition>> B
---------------------
state = notStarted
min_time = 2
max_time = 3

<<WorkDefinition>> C
---------------------
state = notStarted
min_time = 1
max_time = 4

startToStart
finishToStart
finishToFinish
```

Pantel et al. (IRIT & LAAS)
A sample run
Translation into Petri nets

A WD with \( \text{min}_\text{time} = 5 \) and \( \text{max}_\text{time} = 11 \) time units

- \( t = 0 : \text{WD is notStarted} \)
A sample run
Translation into Petri nets

A WD with \( \text{min\_time} = 5 \) and \( \text{max\_time} = 11 \) time units

- \( t = 0 \): WD is notStarted
- \( t = 1 \): WD starts
A sample run
Translation into Petri nets

A WD with $min\_time = 5$ and $max\_time = 11$ time units

- $t = 0$ : WD is notStarted
- $t = 1$ : WD starts
- $t = 6$ : WD is now on time
A sample run
Translation into Petri nets

A WD with $min\_time = 5$ and $max\_time = 11$ time units

- $t = 0$: WD is notStarted
- $t = 1$: WD starts
- $t = 6$: WD is now on time
- $t = 7$: WD completes on time
SimplePDL2PetriNet
A Simple ATL Transformation

- One rule by concept in the source language (i.e. Process, WorkDefinition et WorkSequence)
  - Use of the ATL language
  - Declarative Definition of the transformation
    - Use of the resolveTemp operator!

```
rule Process2PetriNet {
  from p: SimplePDL!Process
  to pn: PetriNet!PetriNet (nodes ← ..., arcs ← ...)
}

rule WorkDefinition2PetriNet {
  from wd: SimplePDL!WorkDefinition
  to
    --- création des places, transitions et arcs ...
  p_notStarted : PetriNet!Place (name ← wd.name + ’_notStarted’, nbJetons ← 1)
    , ...
}
```
**SimplePDL2PetriNet**

A Simple ATL Transformation

```plaintext
rule WorkSequence2PetriNet {
    from ws: SimplePDL!WorkSequence
to
    a_ws: PetriNet!Arc (kind <- #read_arc, tokensNb <- 1,
    source <-
        if ((ws.linkType = #finishToStart) or (ws.linkType = #finishToFinish))
            then thisModule.resolveTemp(ws.predecessor, ' p_finished ')
            else thisModule.resolveTemp(ws.predecessor, ' p_started ')
        endif,
    target <-
        if ((ws.linkType = #finishToStart) or (ws.linkType = #startToStart))
            then thisModule.resolveTemp(ws.successor, ' t_start ')
            else thisModule.resolveTemp(ws.successor, ' t_finish ')
        endif
}
```

Pantel et al. (IRIT & LAAS)
Some features of our translation

**Nice properties**

- functional pattern-matching ATL program
- structural (a WD is a net & a WD.state is a marking)
- modular (a constraint is also a net)
- incremental (a constraint may be plugged in & out)
- traceable

**Target language comes equipped**: [http://www.laas.fr/tina/](http://www.laas.fr/tina/)

- **nd** (*NetDraw*) : editor and simulator of temporal Petri nets
- **tina** : scanner of temporal Petri nets state spaces
- **selt** : model-checker for the temporal logic $SE-LTL$ (State/Event $LTL$), with counter-example generation
Experiment conclusion
Conclusion & Future Works

What was achieved
- definition of an execution semantics and spec. for SimplePDL
- well-behaved translation into Petri net tools
- automatic validation of process properties
- with a feedback (traceability)
- implementation and integration as a TOPCASED service

What remains to be done
- generalisation to other source DSLs & target languages
- proof & methodology to establish equivalence between semantics (operational vs. denotational)
- automatic traceability
Context

Motivation

- to define a generic framework able to describe various execution semantics
  - axiomatic semantics (as pre/post/inv in the OCL)
  - operational semantics (as in Kermeta)
  - denotational semantics (model transformation)
- to extend existing frameworks, such as EMF or GME
- in order to simulate or validate models within

Our approach: a property-driven methodology

1. expertise in the specific domain
2. expertise in some validation technology
Semantics and meta-models

*Axiomatic semantics*

- **Purpose:**
  
  *Express properties (mainly well-formedness) at the models and meta-models level and apply static analysis on models.*

- **Available approaches:**
  
  ✓ OCL (+ verification tools)
  ✓ ATL (transformations toward error models)

- **Questions:** What kind of properties could be ensured by graph grammars generation of models?

- **Confluence and finite...**
Semantics and meta-models

*Operational semantics*

**Purpose:**

*Express the behavioural semantics of meta-models elements in order to provide executability to models.*

**Available approaches:**

- ✓ Kermeta (Triskell), xOCL (Xactium)
- ✓ Action languages (AS-UML, AS-MOF, etc.)
- ✓ Endomorphic model transformations (ATL, GReaT, KMTL, MOLA, TIGER, VIATRA2, etc.).
Context

- **Goals**: Defining structural and behavioral semantics for MDD\(^a\).
- **Why?**
  - Former modeling approaches focuses on structural safety properties.
  - Requires safety and liveness properties for behavioral models.
    - Ensuring models conformity, coherence, . . .
    - Ensuring models safety, liveness, . . .
    - Provide execution power to models (simulation, validation correctness, . . .).
  - Allows rigorous bridging between technological spaces.
- **Our approach**: Applying to modeling languages the well-known principles used in programming languages.

\(^a\)This work is supported by the TOPCASED project.
OMG provides OCL to describe structural safety properties on Models

⇒ Allow axiomatic semantics rules definition.

In metamodeling, rules extend the abstract syntax of the DSL.

⇒ Reduce the number of valid models.

TOPCASED provides an OCL editor and checker based on EMFT\(^a\).

\(^a\)Eclipse Modeling Framework Technologies.
Current needs for TOPCASED

- Currently, TOPCASED doesn’t allow the definition and verification of behavioral properties
  ⇒ Require a complete and accurate semantics (safety & liveness).
- From programming language theory, we would like to apply:
  - operational semantics,
  - denotational semantics.
- Written by Domain expert...
Operational semantics

Principles

- From Programming languages point of view
  - Describe what should be the **execution** of language features,
  - Use **rewriting rules** over the abstract syntax of the language (Structural Operational Semantics).

- MDE needs
  - Give a precise definition of model elements behavior
    - Allow to execute dynamic models

- Methods
  - Use **metaprogramming languages** (Kermeta, xOCL, etc.)
    - Define behavior of models by programming it
    - Behavior is "hard coded" in the bodies of metaclass operations
      - Abstract syntax modifications (merging Ecore metamodel)
  - Use **model transformations** (ATL, AGG, etc.)
    - Closer to former programming language approach (SOS)
    - The semantics is expressed by transformation rules
Case study : SimplePDL Execution Model

- What does it mean to **execute** a SimplePDL model?

- **The enactment** of a **Process**:
  - Starts with the **starting** of an **activity**
  - Is **repeatedly launching** activities which can be started
    - Whether an activity is **startable** depends on the state of activities that precede it and the kind of precedence.
  - Records the progress of started activities
  - Is **completed** when all the contained activities are **finished**

- This particular behavior **implies** several metamodel **modifications**
  - For example to represent different activities states (not started, started, finished) : **progress** attribute.

*SimplePDL example described with Ecore Editor of TOPCASED.*
The Kermeta toolkit
A quick presentation

- Developed by the Triskell team of the IRISA laboratory.
- Defined as an "Object oriented metaprogramming language"
- Allows to:
  - Describe modeling language abstract syntax
  - Program behavior of metaclass in their operations bodies
- Provides means to manipulate model objects
  - Roles navigation, creation, deletion, etc.
- Implemented as an Eclipse plugin
  - Based on the Ecore metametamodel
Execution model implementation in Kermeta

```kermeta
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))
    else
        result := start_ok or (previous.size() == 0)
    end
    result := false
end
```

Pantel et al. (IRIT & LAAS)
Programing language Structural Operational Semantics

- **From the syntax**
  - Syntactic values (Integer, ...)
  - Executable constructs (expressions, instructions, ...)

- **Define the semantic**
  - Environment
    - Input/Output
    - Memory
    - Variable binding
    - State
  - Build *semantic values* from syntactic ones by rewriting
  - SOS or Natural semantics define *reduction rules*
    \[ \text{Env} + \text{Executable construct} \rightarrow^* \text{Env'} + \text{Values} \]

- Toward MDE: SimplePDL using model transformation
  Use *transformation rules* as *reduction rules*. 
Metamodel Extensions

Capture execution state
Model transformation

monProcess:Process
name = "MonProcess"

activite1:Activity
name = "Activité 1"
progress = -1

activite2:Activity
name = "Activité 2"
progress = -1

precA1A2:Precedes
kind = pk_start_start
Model transformation

ATL transformation

monProcess:Process
name = "MonProcess"

activite1:Activity
name = "Activité 1"
progress = -1

activite2:Activity
name = "Activité 2"
progress = -1

precA1A2:Precedes
kind = pk_start_start

monProcess:Process
name = "MonProcess"

activite1:Activity
name = "Activité 1"
progress = 0

activite2:Activity
name = "Activité 2"
progress = -1

precA1A2:Precedes
kind = pk_start_start
Model transformation

monProcess:Process
  name = "MonProcess"
activite1:Activity
  name = "Activité 1"
  progress = -1
activite2:Activity
  name = "Activité 2"
  progress = -1
precA1A2:Precedes
  kind = pk_start_start

monProcess:Process
  name = "MonProcess"
activite1:Activity
  name = "Activité 1"
  progress = 0
activite2:Activity
  name = "Activité 2"
  progress = -1
precA1A2:Precedes
  kind = pk_start_start

monProcess:Process
  name = "MonProcess"
activite1:Activity
  name = "Activité 1"
  progress = 10
activite2:Activity
  name = "Activité 2"
  progress = 0
precA1A2:Precedes
  kind = pk_start_start

ATL transformation

Pantel et al. (IRIT & LAAS)
ATL : ATLAS Transformation Language
A quick presentation

- Developed by ATLAS INRIA and LINA research group
- **Hybrid model transformation** language
  - Possess both **declarative** and **imperative** structures.
- **ATL transformations** are composed of declarative **rules**
- **Rules** define the transformation from a **source model element** to a **target model element**
  - Can call **Helpers** to ease the processing of target model element properties.
Metamodel Extensions
Capture execution state

helper context simplepdl! Activity
    def : startable () : Boolean = ( 
        self . progress < 0  −− not started 
        and 
        self . previous −>select(p | p . kind = # pk_start_start ) 
        −>collect(p | p . before) −− precedence kind start/start 
        −>forAll(a | a . progress >= 0) −− started 
        and 
        self . previous −>select(p | p . kind = # pk_finish_start ) 
        −>collect(p | p . before) −− precedence kind finish/start 
        −>forAll(a | a . progress = 100) −− stoped 
    );
Metamodel Extensions
Capture execution state

```java
rule progressActivity {
    from
        a_in : simplepdl ! Activity
    to
        a_out : simplepdl ! Activity ( 
            -- Properties copy
            name <- a_in.name,
            process <- a_in.process,
            previous <- a_in.previous,
            next <- a_in.next,
            -- Progression
            progress <-
                if a_in.startable ()
                then 0
                else
                    if a_in.progress >= 0 and a_in.progress < 90
                        then a_in.newProgress()
                    else
                        if a_in.finishable ()
                            then 100
                        else
                            a_in.progress
                endif
            endif,
        )
}```
Feedbacks

- **Kermeta**:  
  - Common user oriented technologies  
  - Much easier than Java/EMF  
  - **Correctness**  
    - Model type checking  
    - Hoare triple approach  
    - Proven refinements?

- **ATL**:  
  - Dedicated user oriented technologies  
  - Very efficient in some cases  
  - Painful in others (requires multi-models capacities)  
  - **Correctness**  
    - Pure declarative/functional aspects  
    - Structural induction approach  
    - May require co-induction

- **Next point**: Graph grammar/rewriting technologies
Metamodel Extensions
Capture execution state

- `ExecutionContext`
  - `Process` (name: EString)
  - `Activity` (name: EString, progress: EInt)
  - `Guidance` (detail: EString)
  - `Precedes` (kind: PrecedenceKind)

- Process name: EString
- Guidance detail: EString
- Activity name: EString, progress: EInt
- Precedes kind: PrecedenceKind

Pantel et al. (IRIT & LAAS)
Metamodel Extensions

Scenario model definition

- **Execution Context**
  - **Process**
    - **Event**
      - **Activity**
        - **Precedes**
          - **Guidance**

- **Start**
- **Stop**
- **Progress**
  - **Progress**
    - **Target**

Pantel et al. (IRIT & LAAS)

June 16, 2007

SimplePDL & TOCL Case Study
Metamodel Extensions

Scenario model definition

- **Scenario**
  - **Event**: 0..* events
  - **Start**, **Stop**, **Progress**: 1
  - **Activity**: 0..* activities
  - **Guidance**: value: EInt, detail: EString
  - **Precedes**: kind: PrecedenceKind

- **Process**
  - name: EString

- **ExecutionContext**: 1 1

---

Pantel et al. (IRIT & LAAS)

**SimplePDL & TOCL Case Study**

June 16, 2007
Thank you for your attention...