Model Driven Engineering

Bringing formal validation in the industrial process

Benoît Combemale[1], Bernard Berthomieu[2], Jean-Paul Bodeveix[1], Xavier Crégut[1], Mamoun Filali[1], Pierre-Loïc Garoche[1], Marc Pantel[1], Martin Strecker[1], Xavier Thirioux[1] & François Vernadat[2]

[1]IRIT CNRS
2, rue Charles Camichel - BP 7122
F-31071 Toulouse Cedex 7

[2]LAAS CNRS
7, avenue du colonel Roche
F-31077 Toulouse Cedex 4

june 16, 2007

Plan

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 techologies 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)

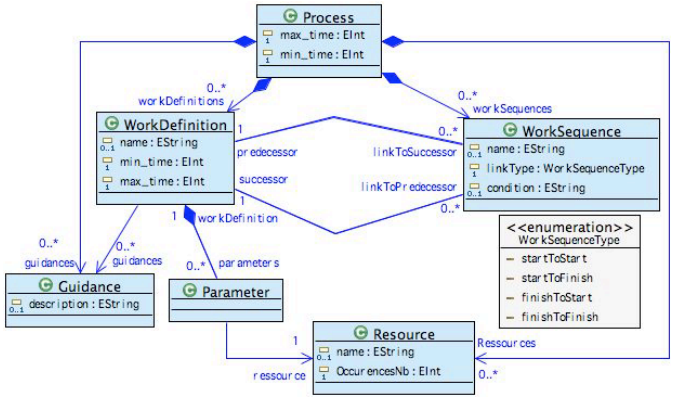
More details on :

- Web site : `http://www.topcased.org`,
- Develoment forge : `http://gforge.enseeiht.fr`

TOPCASED : Current status

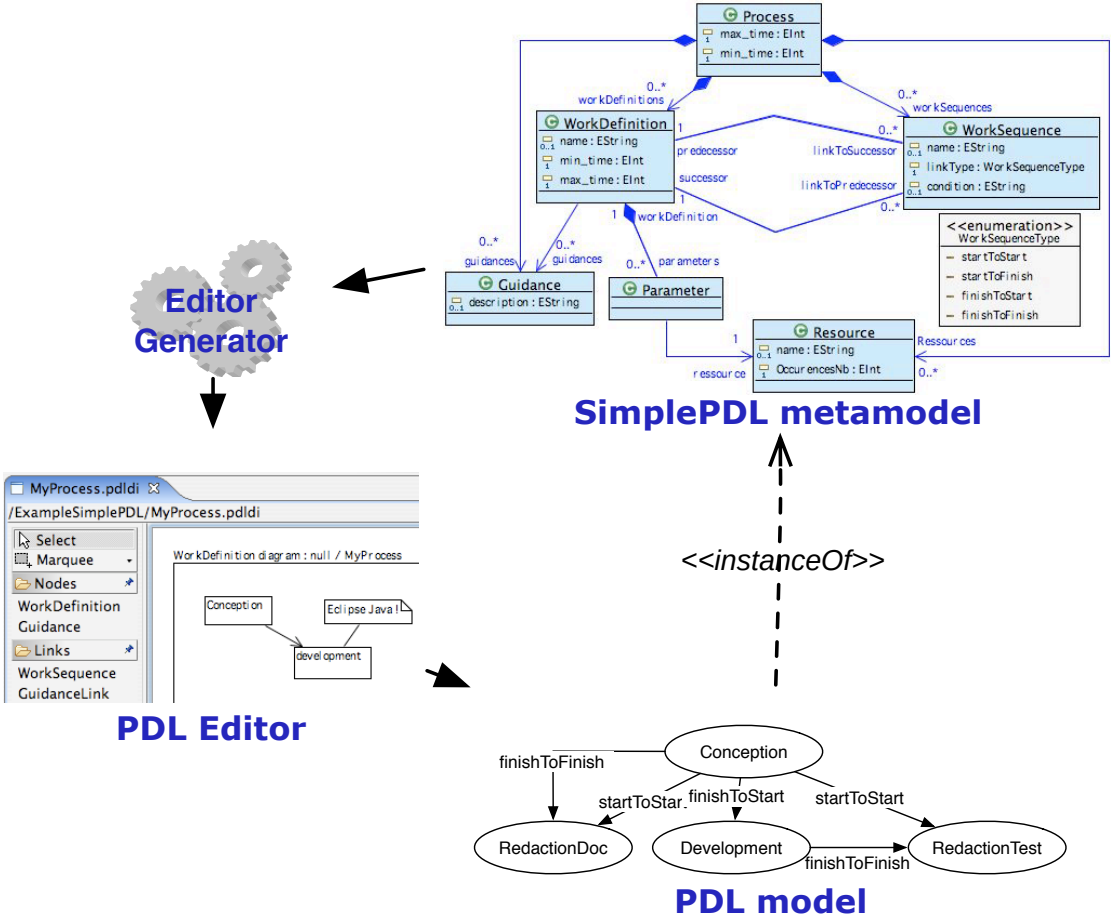
- Started in Q3/2005, current version 1.0M4.1
- Graphical editors for Ecore, UML2, AADL, SAM, SysML, EAST-ADL, SPEM
- EMF, EMFT, UML, UMLT, GEF and low-level GMF based
- Graphical editor generator (metamodels & OCL constraints validation)
- Textual metamodel-sensitive OCL constraints editor
- Simple ModelBus and Eclipse deployment tools implementation
- Model importation (using ATL)
- Experiments in code, tests and documentation generation (using JET, OBEO 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)

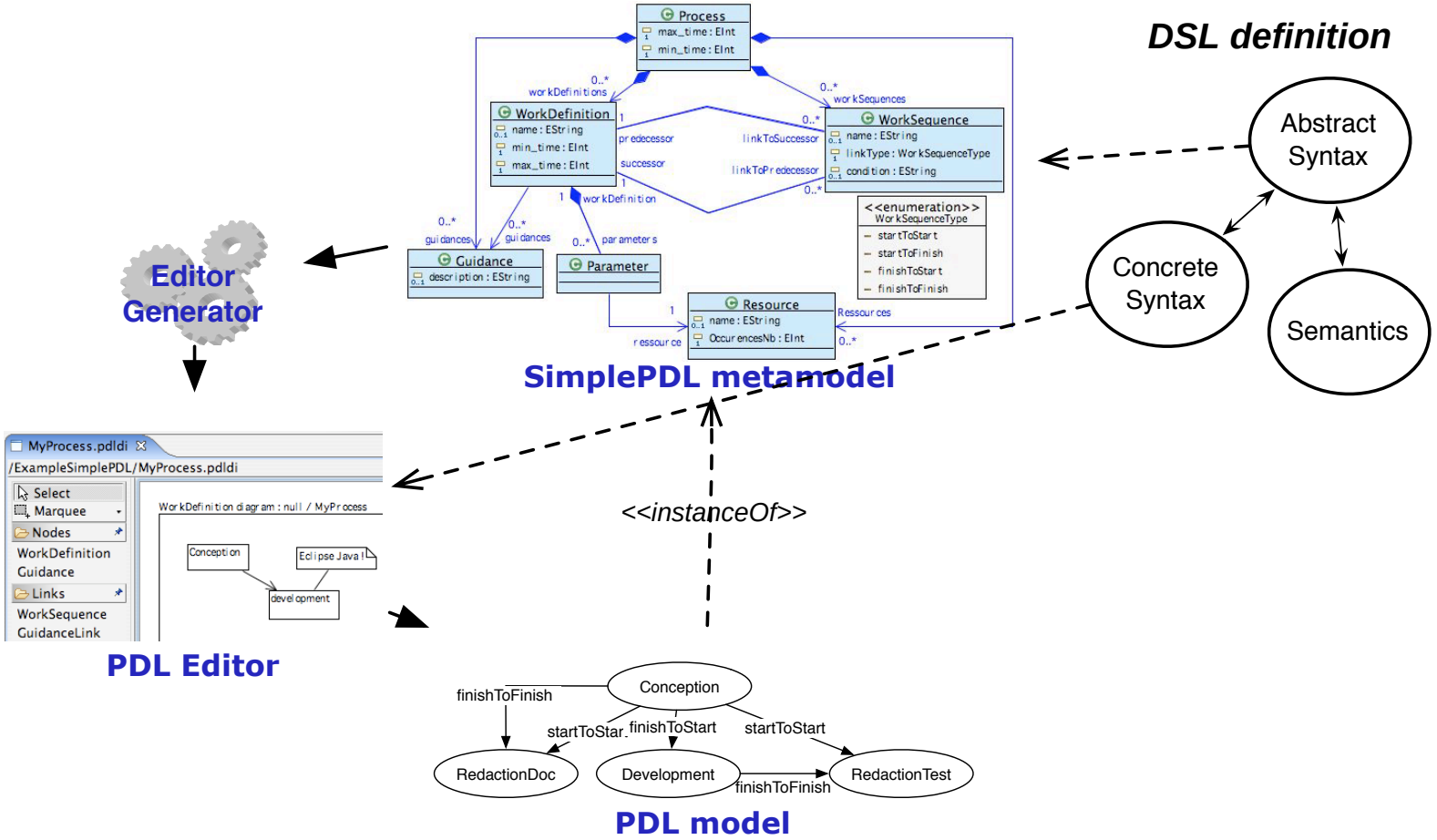


SimplePDL metamodel

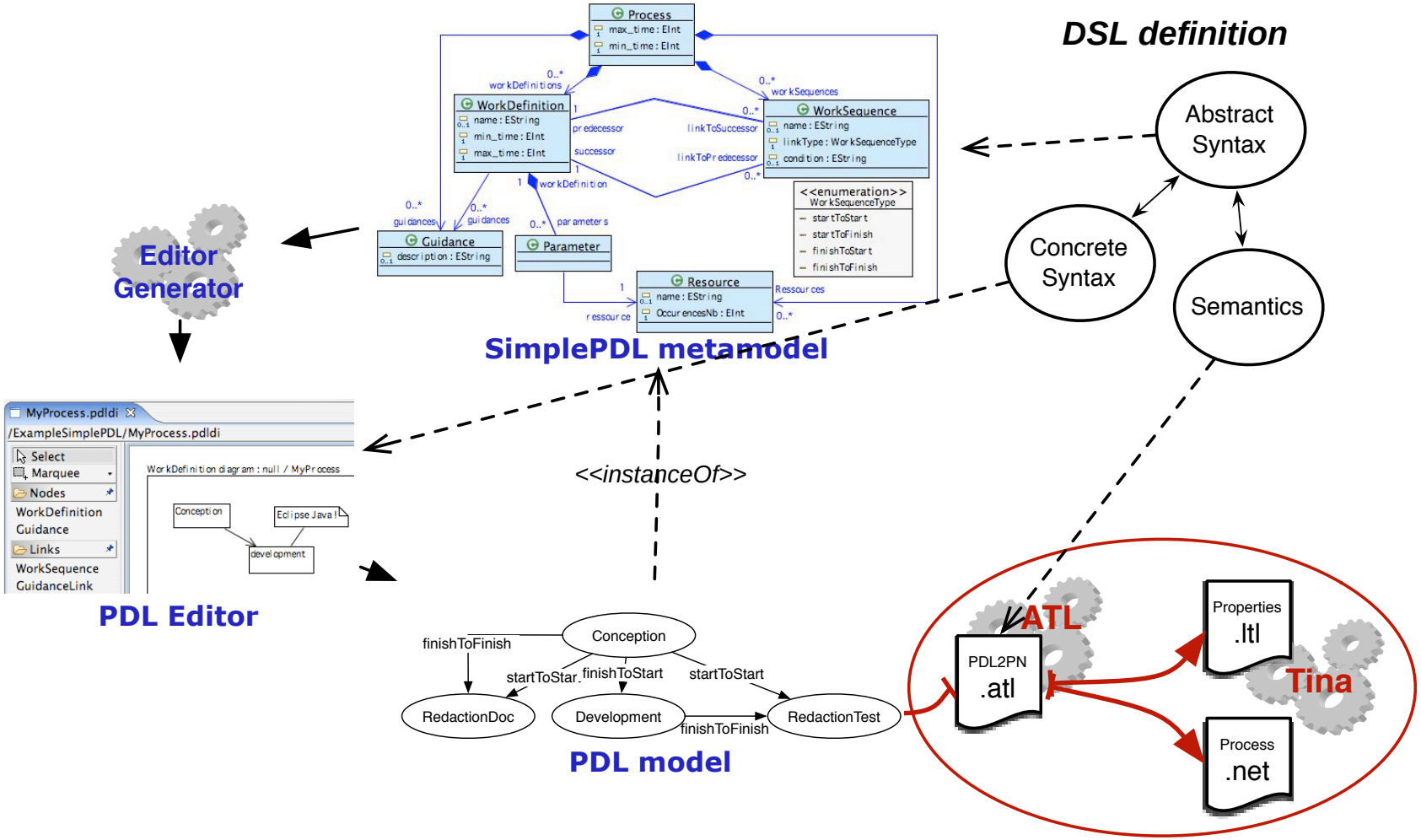
Our Approach (through case study)



Our Approach (through case study)



Our Approach (through case study)



TOPCASED : Toward version 1.0

- Must be an industrial strenght 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

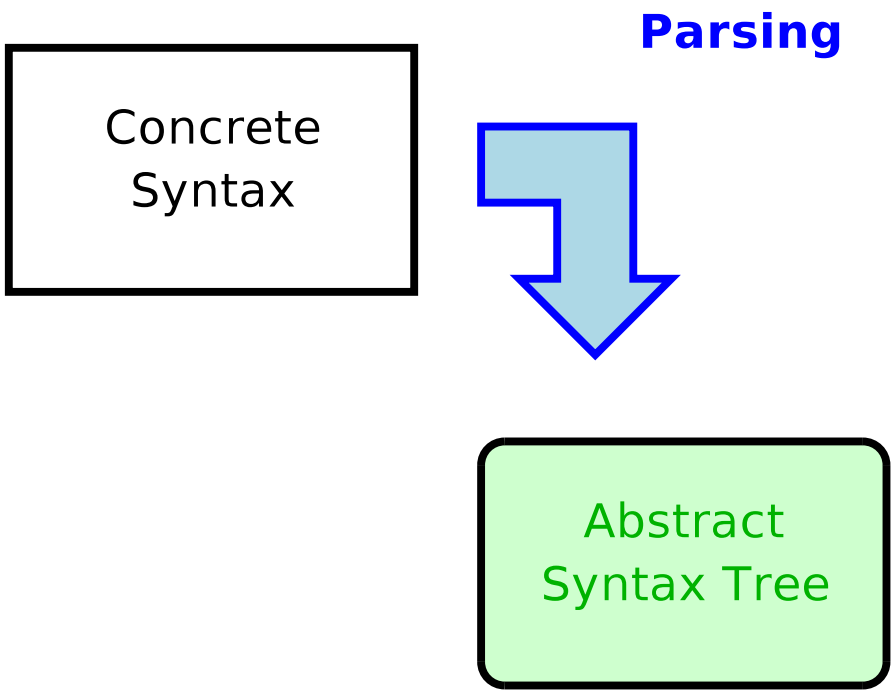
MDE : Promoting DSL

- 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

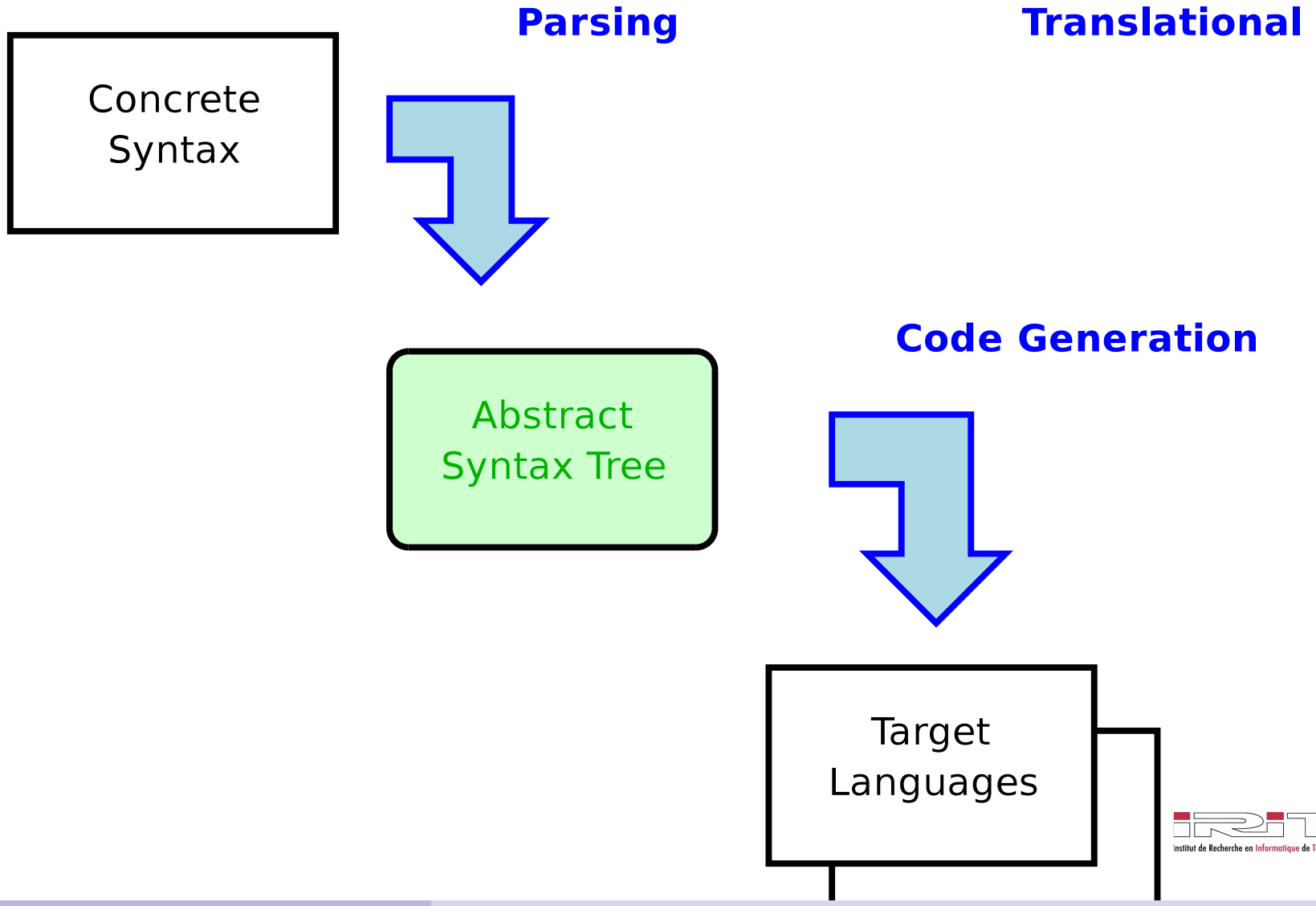
Semantics and programming languages

Concrete Syntax

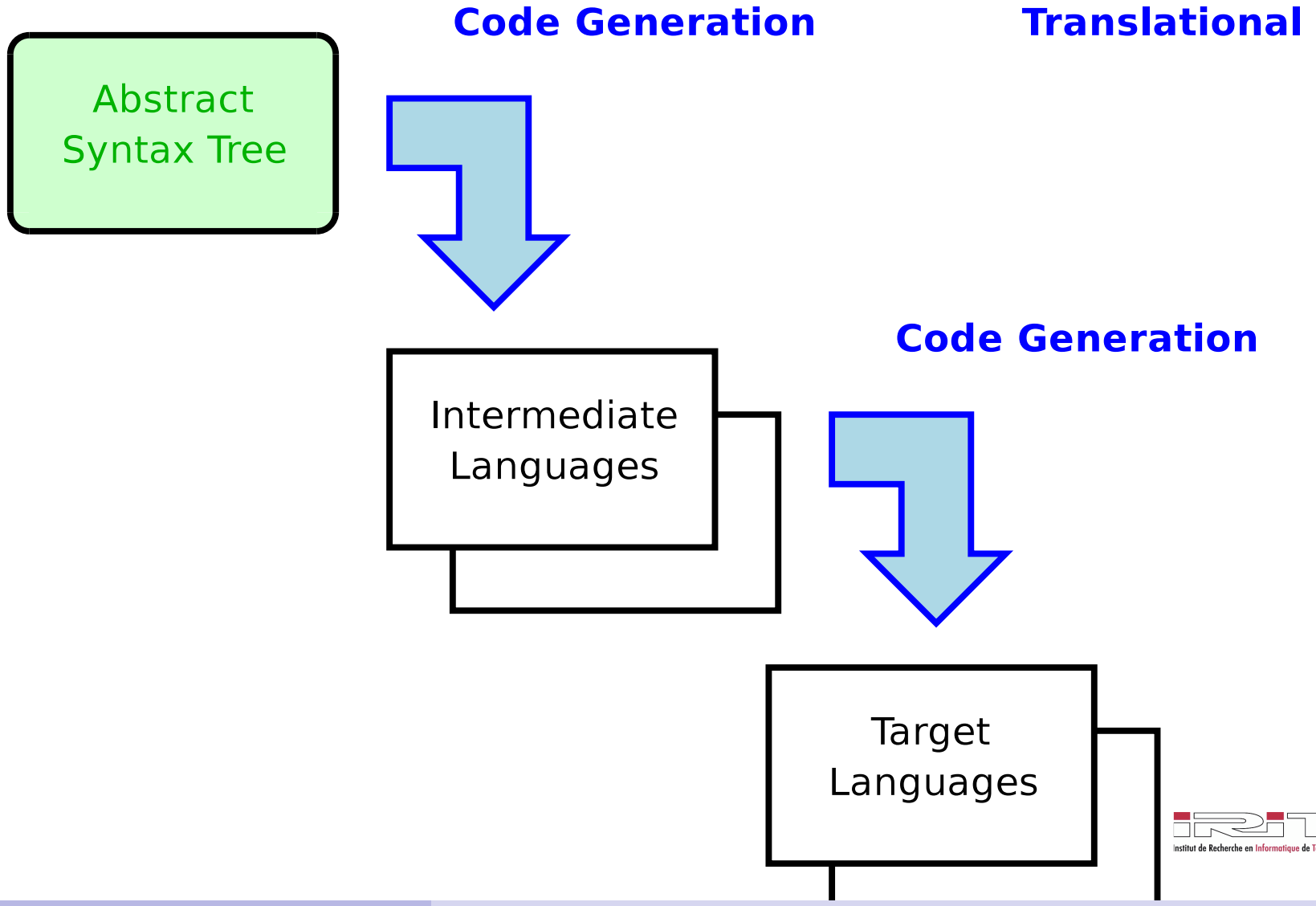
Semantics and programming languages



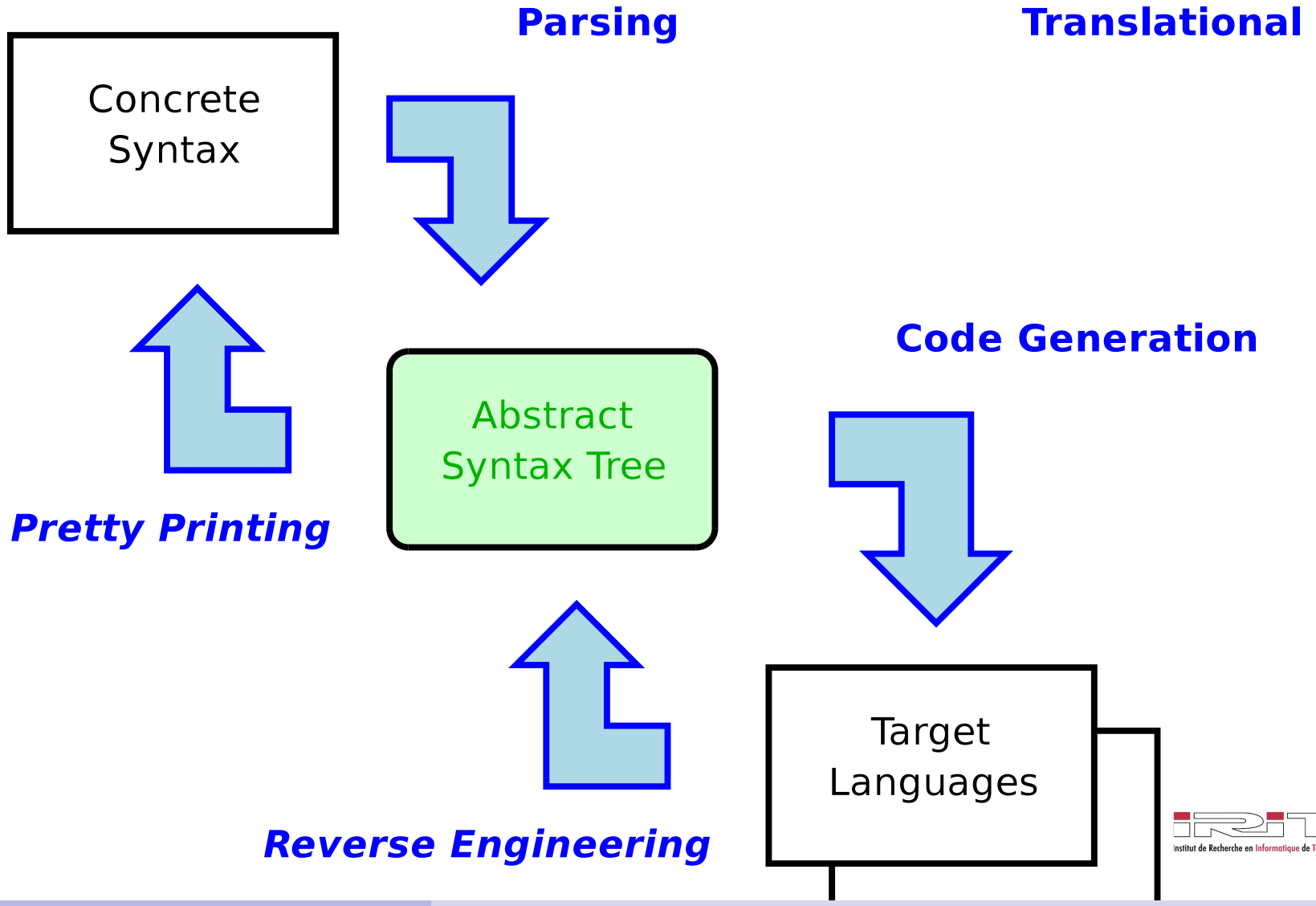
Semantics and programming languages



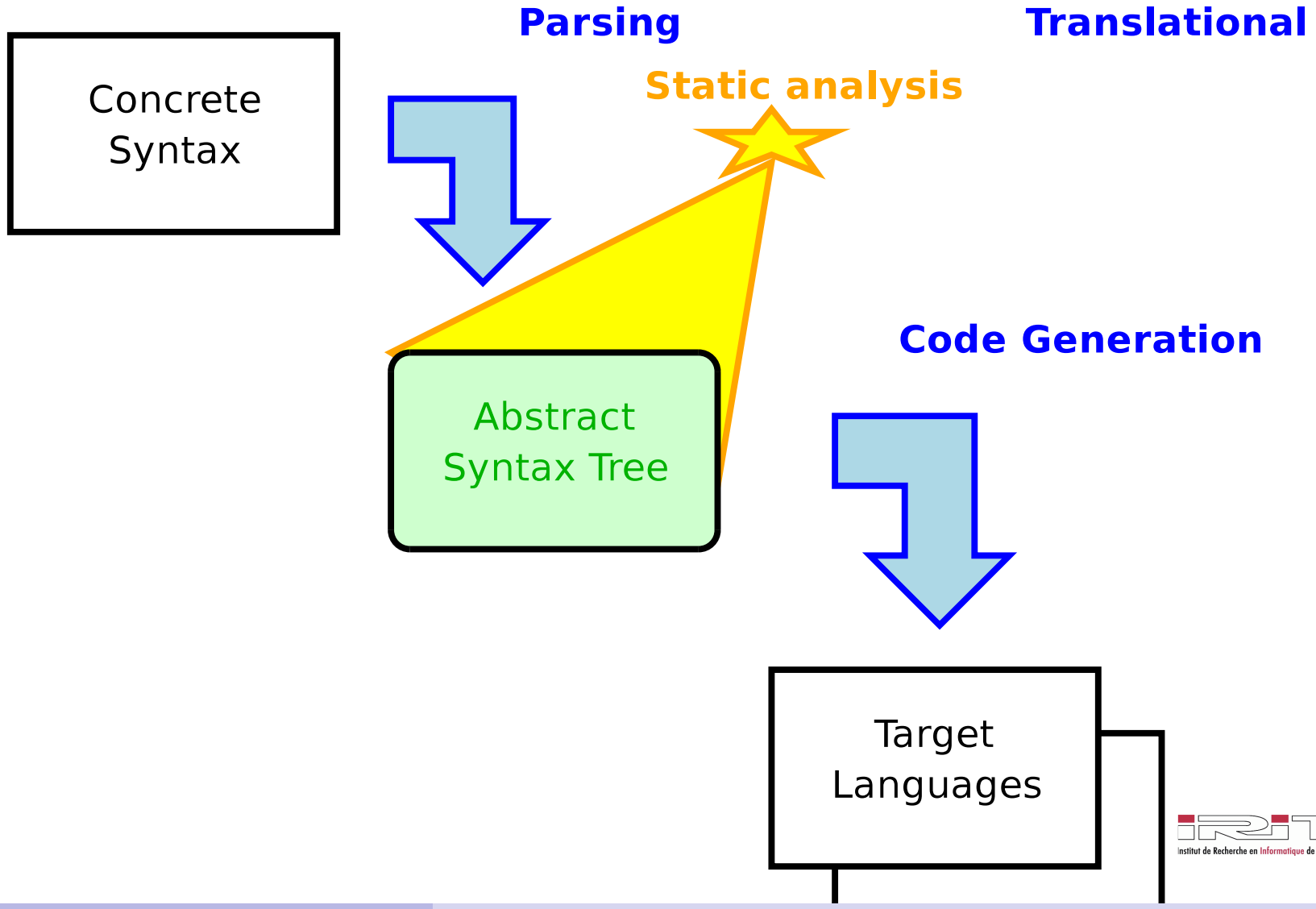
Semantics and programming languages



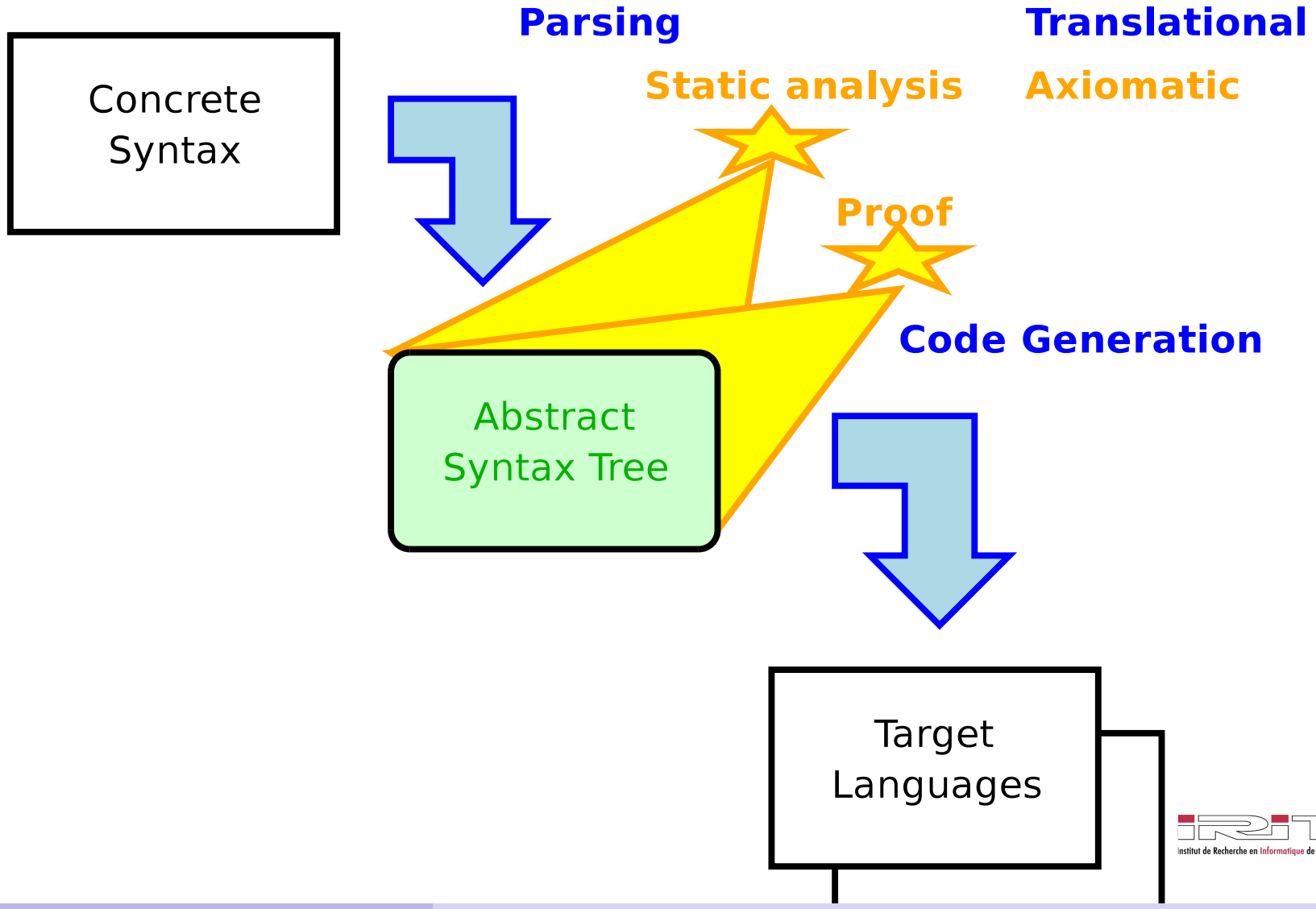
Semantics and programming languages



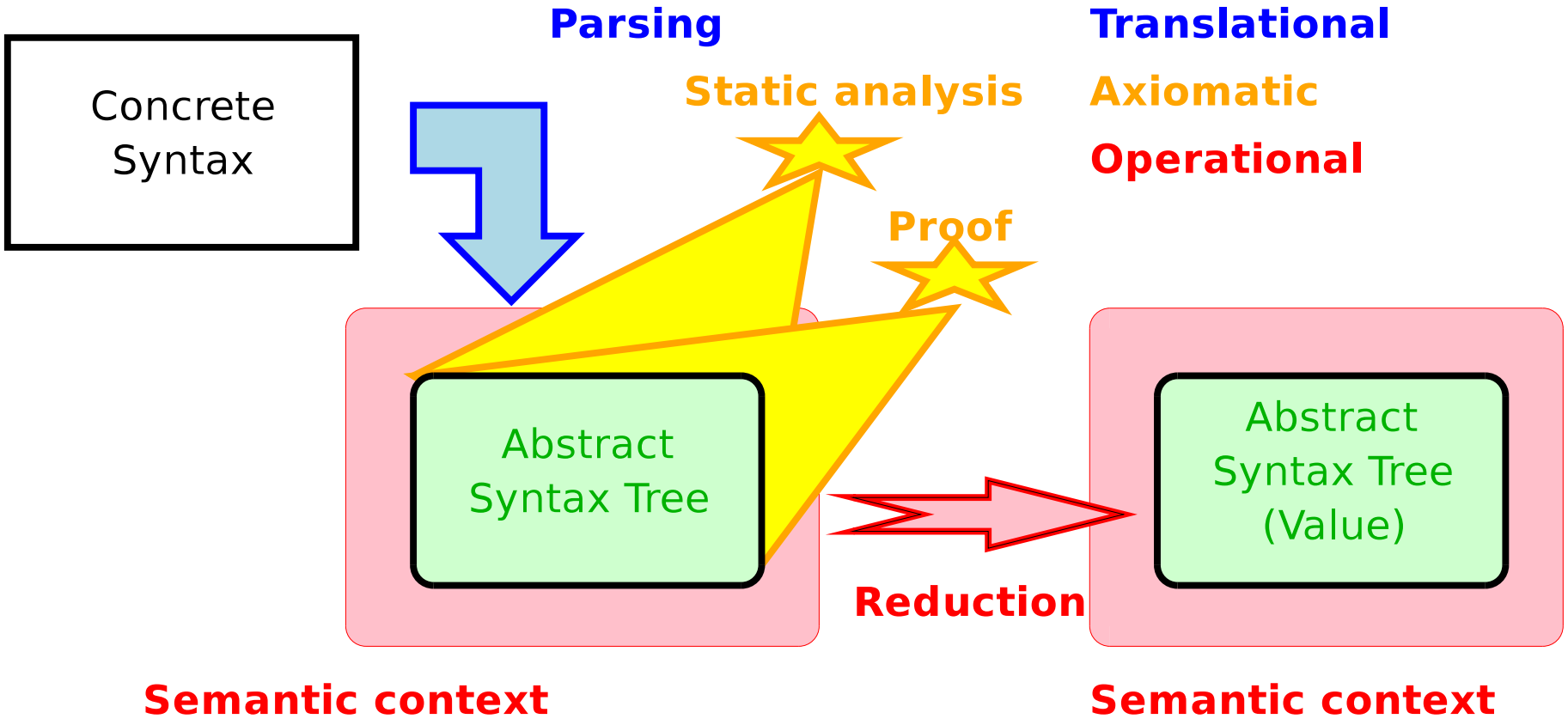
Semantics and programming languages



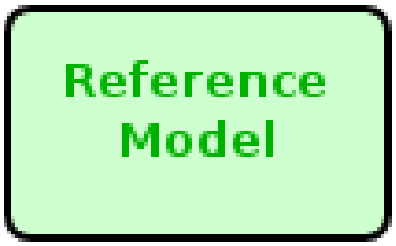
Semantics and programming languages



Semantics and programming languages

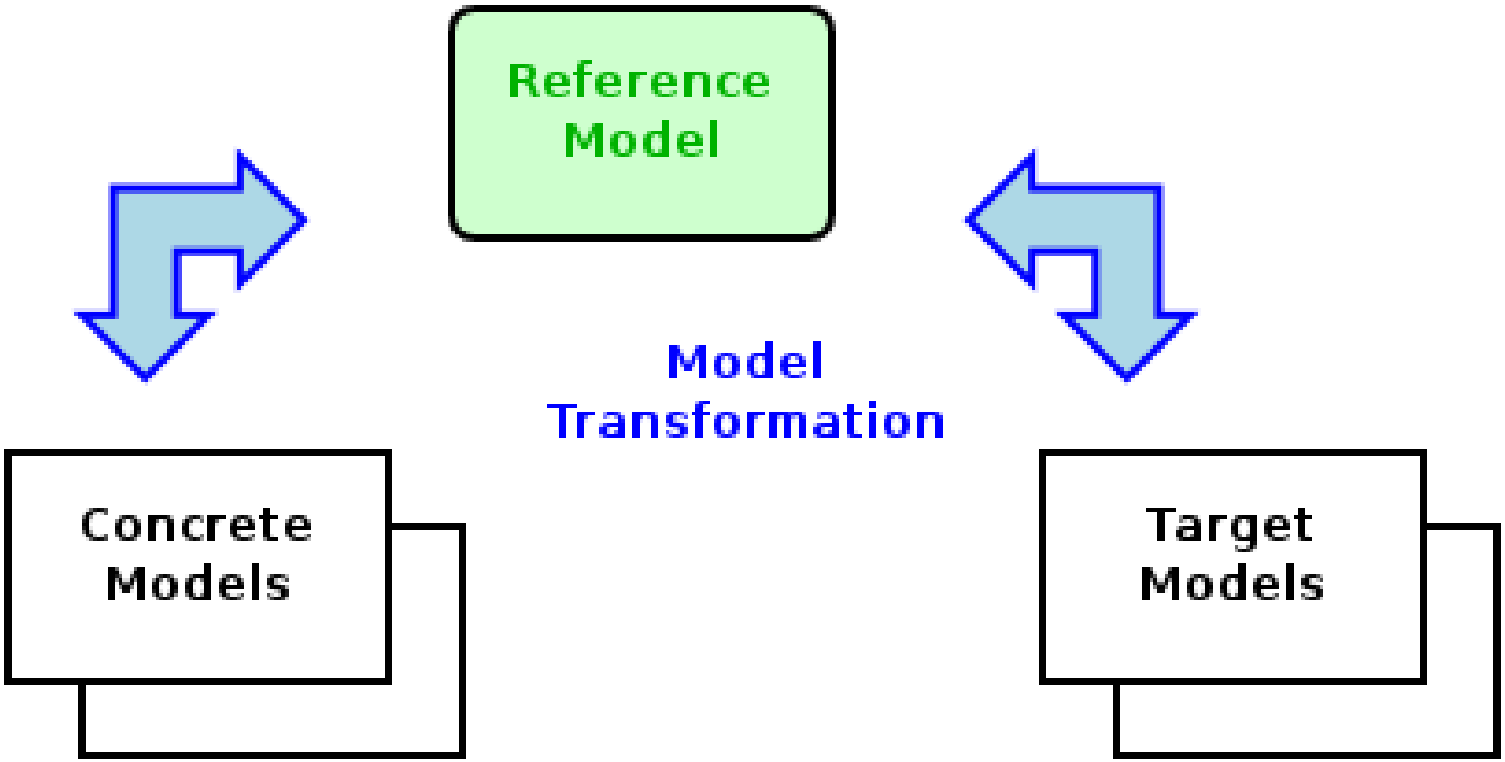


Semantics and models

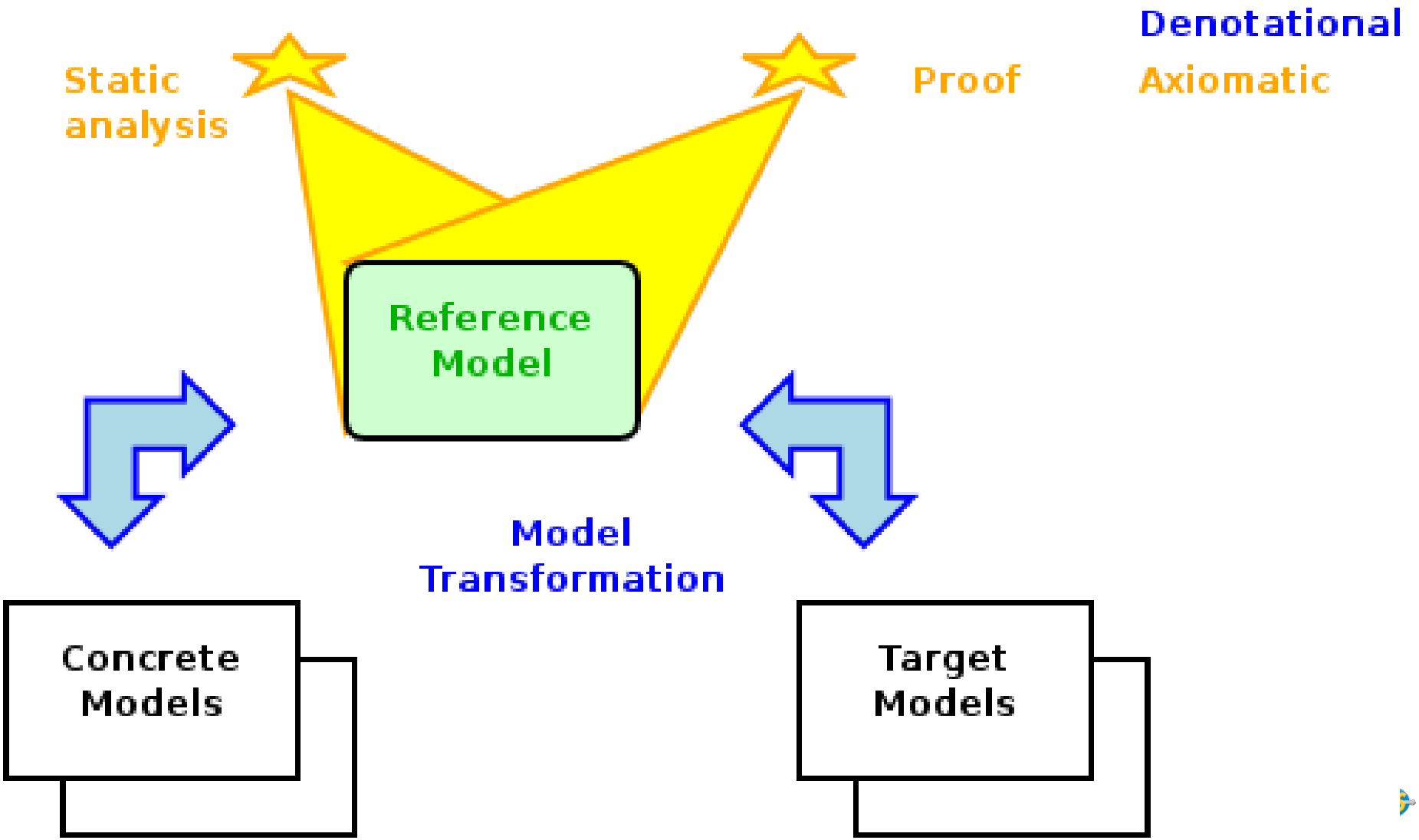


Semantics and models

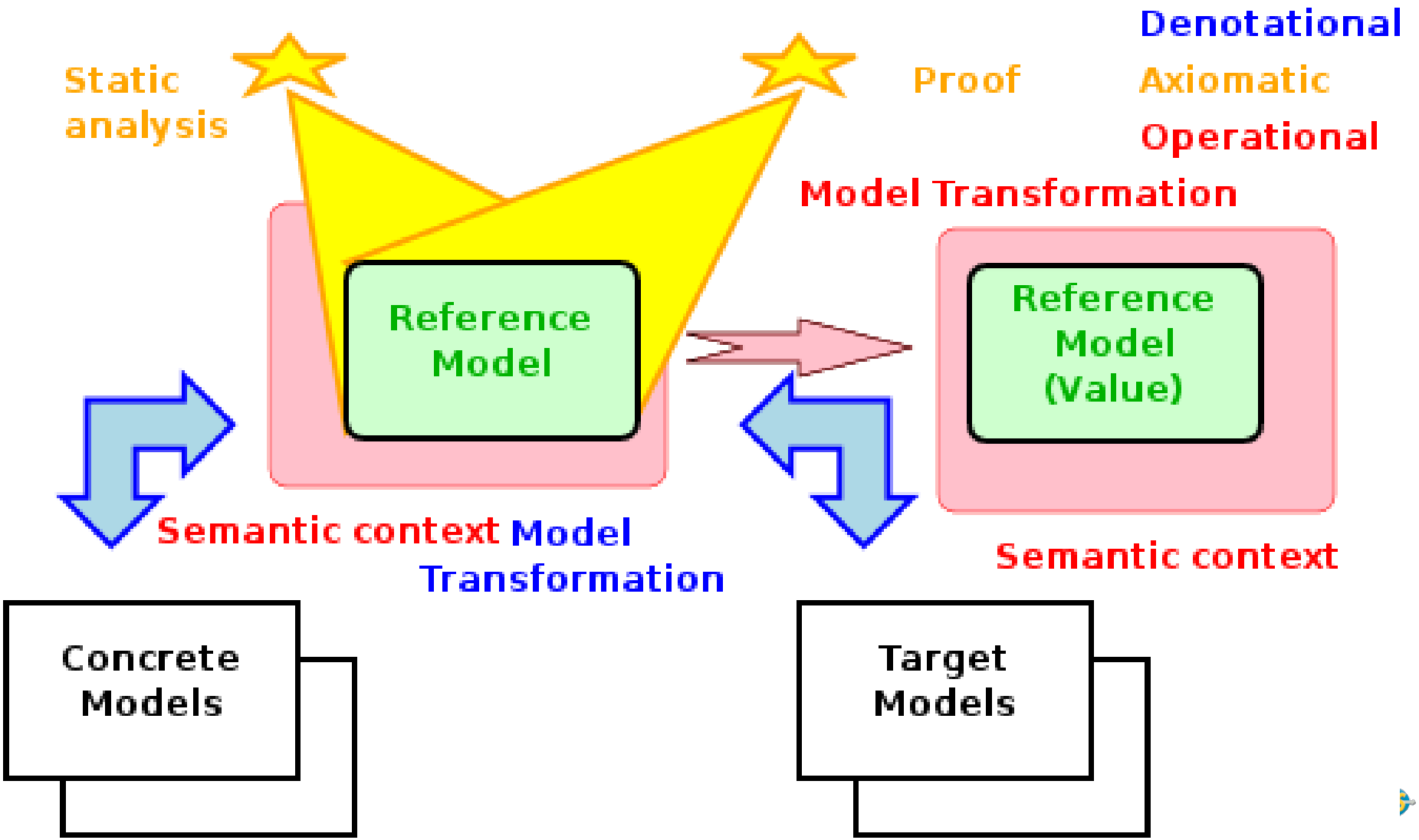
Denotational



Semantics and models



Semantics and models

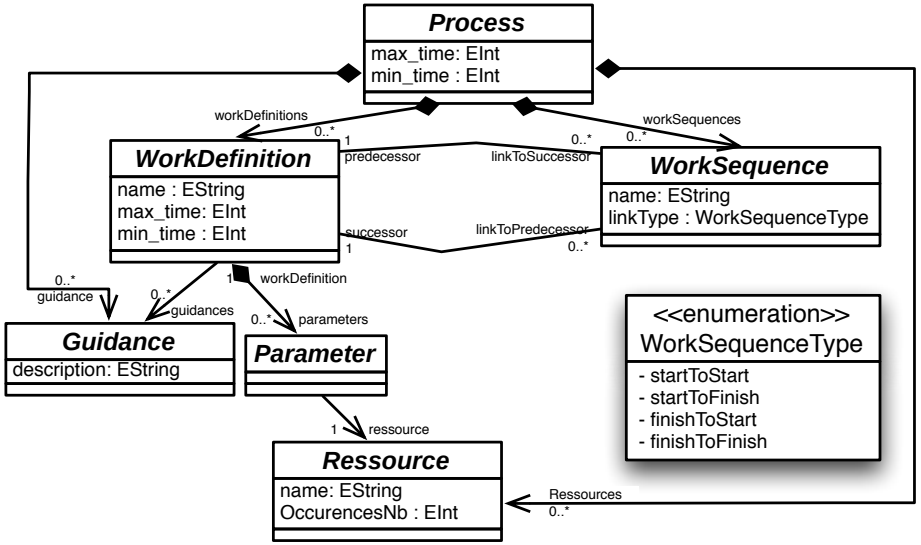


Metamodeling technologies

- 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

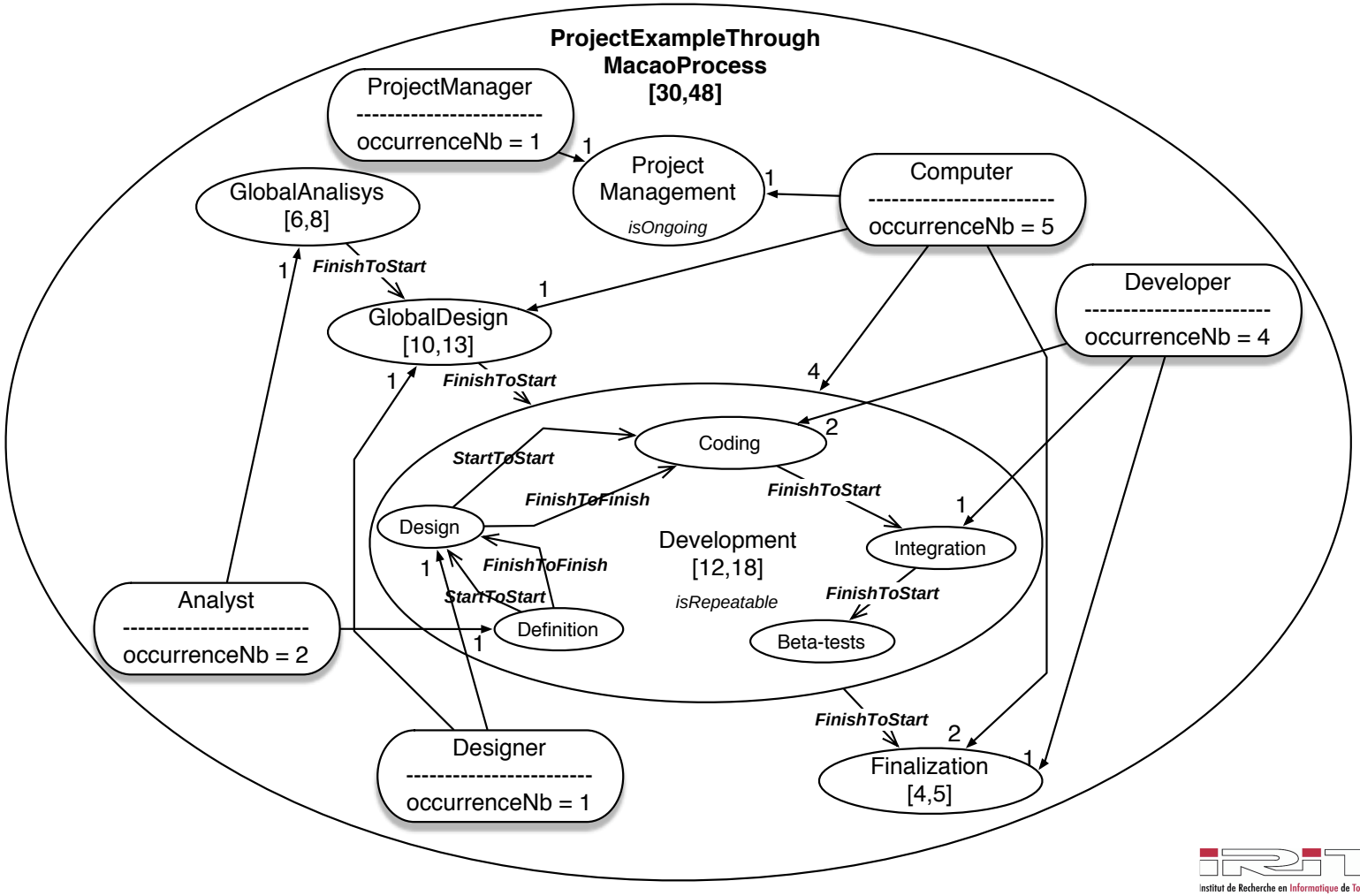
- stems from :
  - SPEM (OMG)
  - UMA (EPF)
- could easily be extended :
  - roles
  - products
  - ...



SimplePDL metamodel

Modelling a complex process

MACAO process



What do we want to check ?

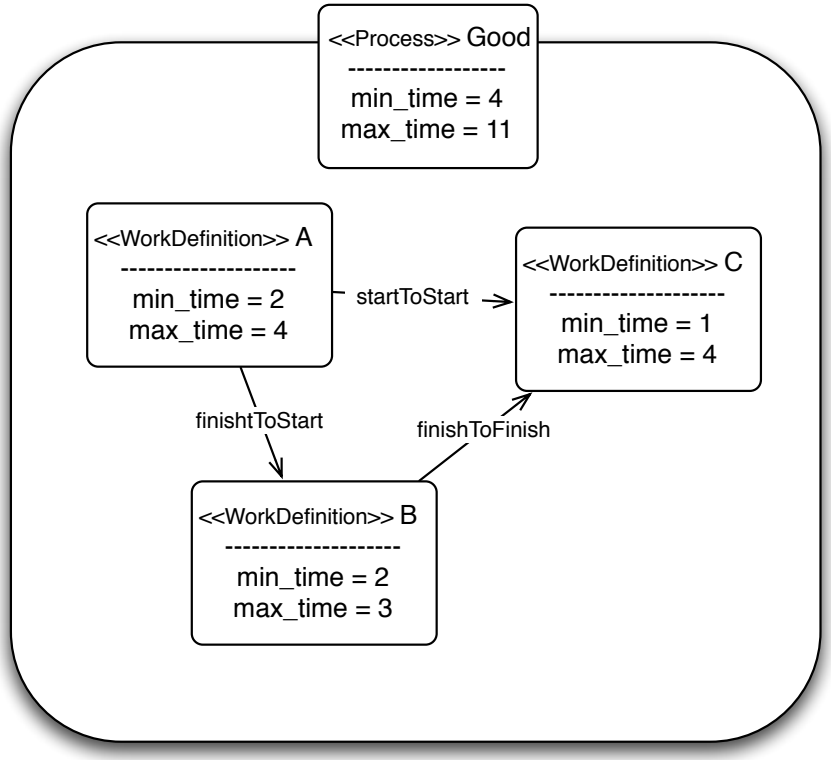
- resource constraints
  - computers
  - manpower
- timing constraints
  - minimum achievement time
  - maximum achievement time
- causality constraints
  - startToStart
  - startToFinish
  - finishToStart
  - finishToFinish
- ...
- for some execution
- or for all executions

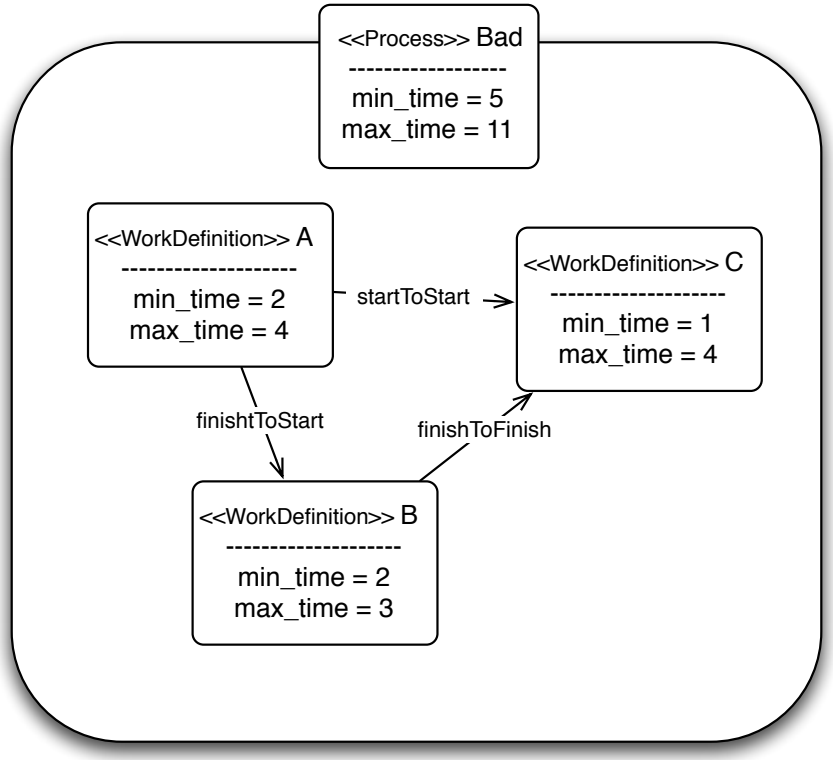
What do we want to check ?

- 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 *min* and *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

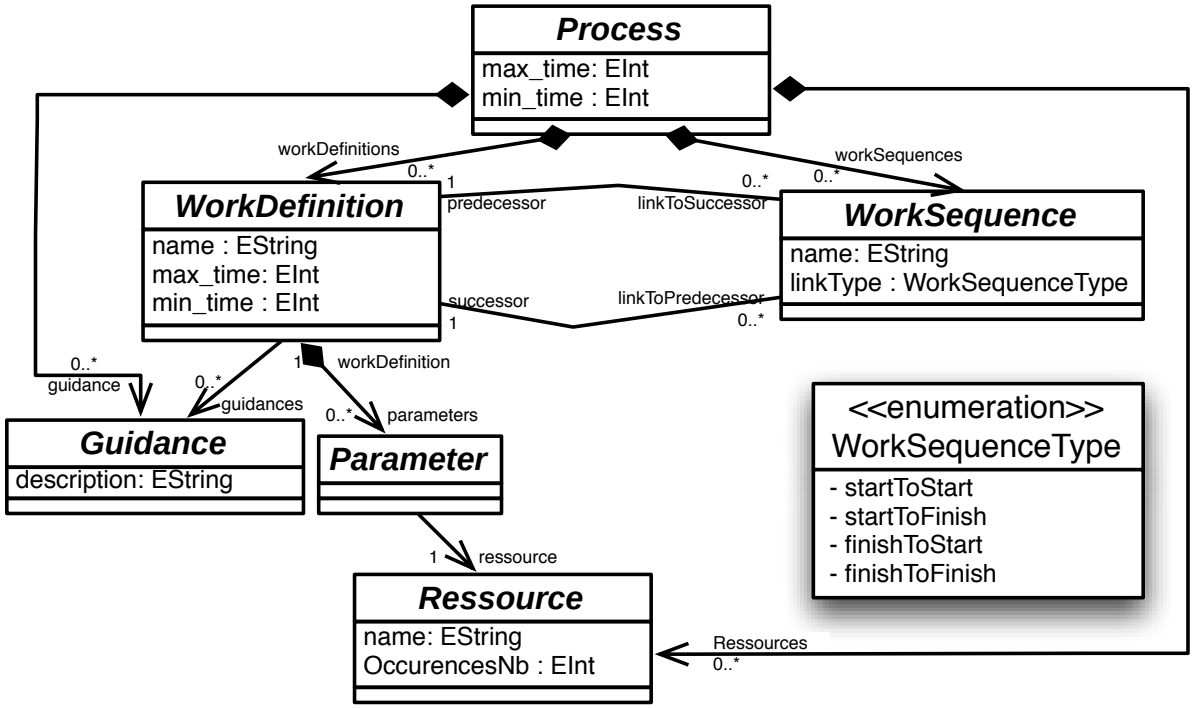
TOCL is used

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

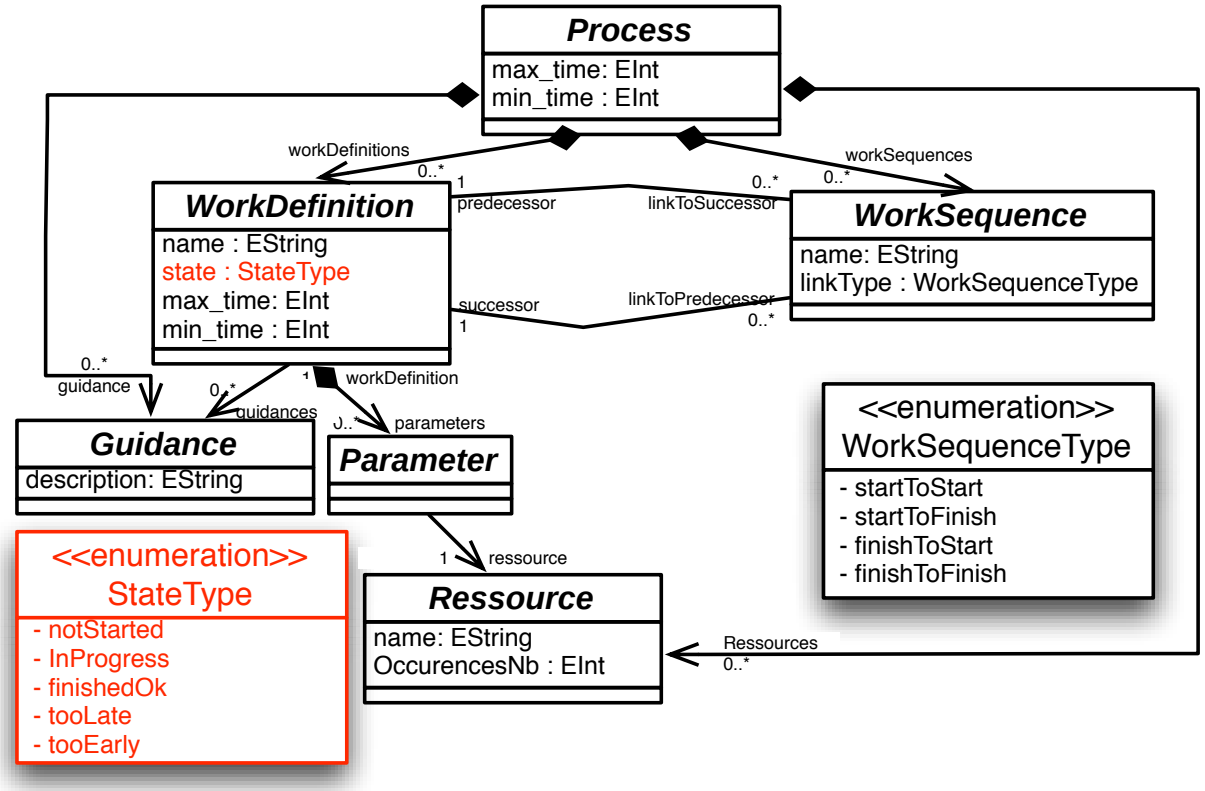
Some properties of WD alone

- $\forall w, (w.state = \texttt{notStarted} \ \wedge \ sometime\ w.state = \texttt{inProgress})$
- $\forall w, always\ (w.state = \texttt{inProgress} \Rightarrow \ sometime\ w.state \in \{\texttt{finishedOk}, \texttt{tooEarly}, \texttt{tooLate}\})$
- $\forall w, always\ (w.state = \texttt{finishedOk} \Rightarrow always\ w.state = \texttt{finishedOk})$
- $\neg\exists w, always\ w.state \neq \texttt{finishedOk}$

Extending the metamodel



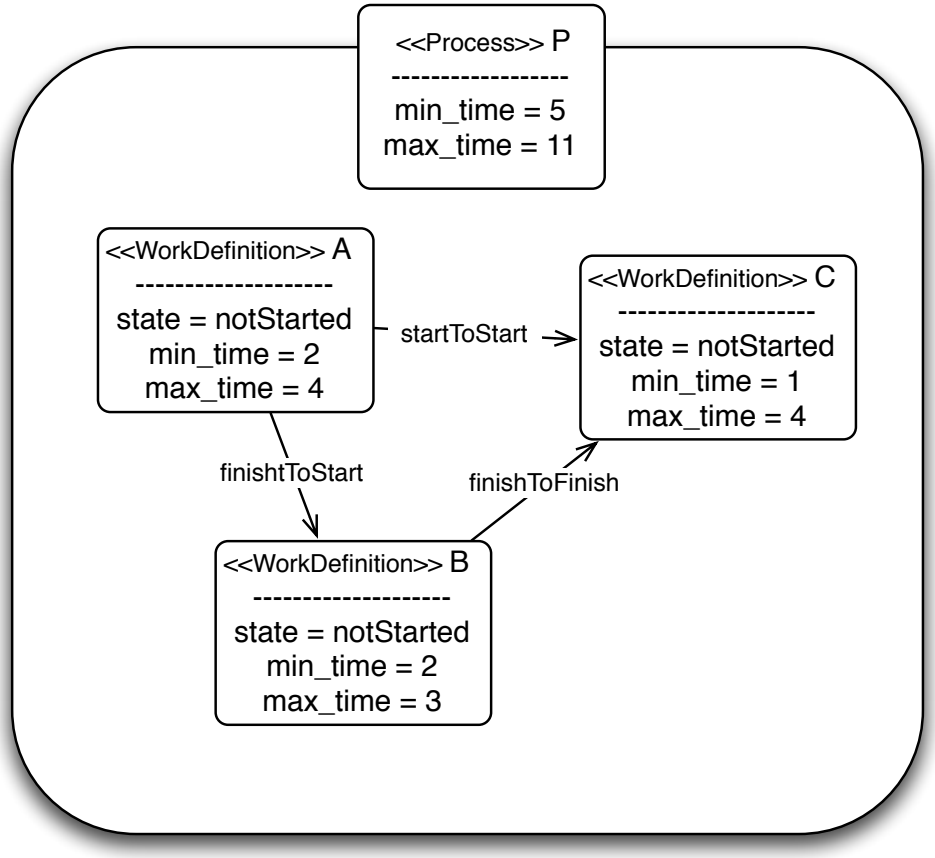
Extending the metamodel



A sample run

Illustrating operational semantics

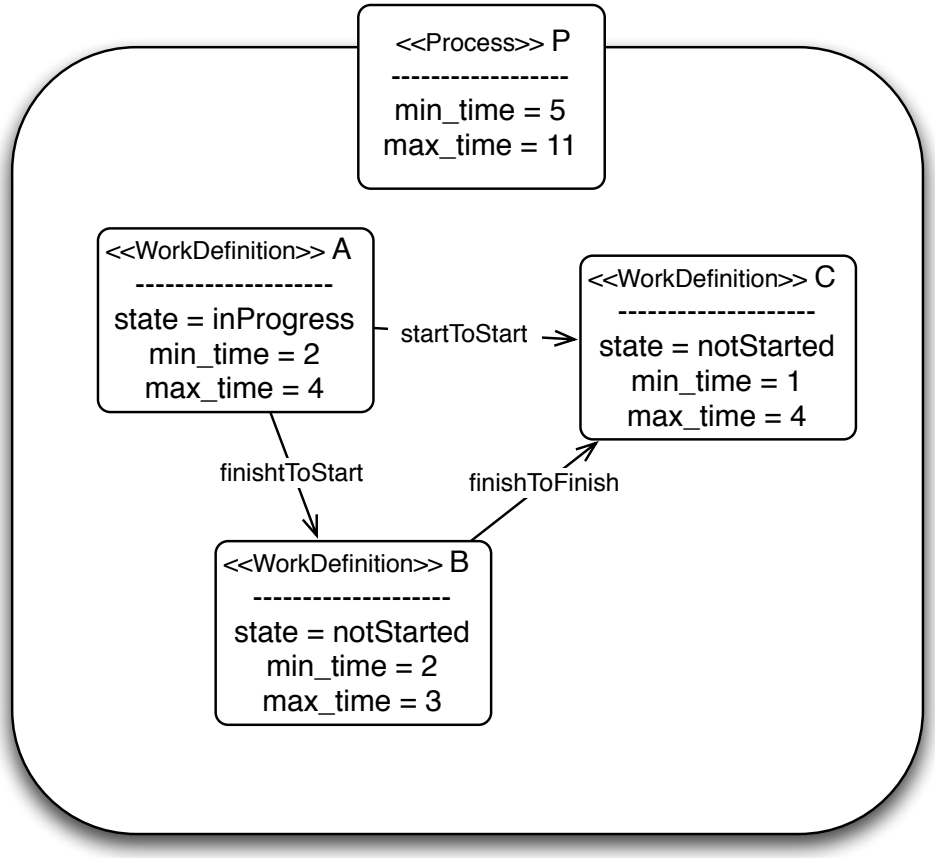
- $t = 0$: *WD*s are `notStarted`



A sample run

Illustrating operational semantics

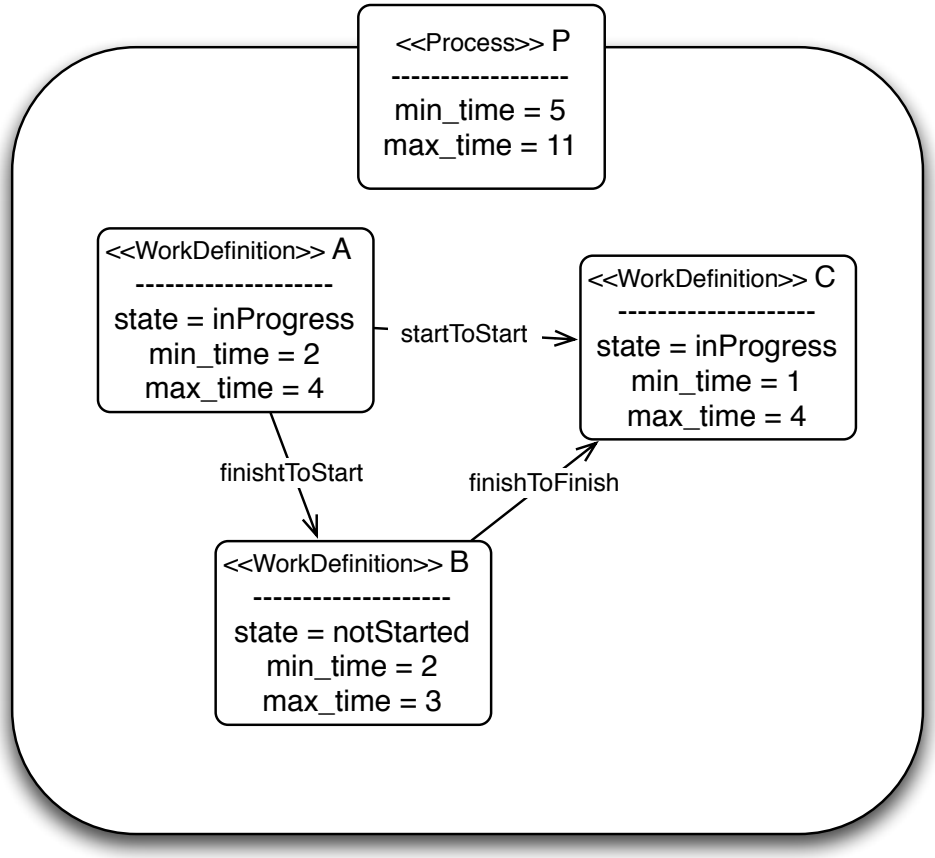
- $t = 0$: *WD*s are `notStarted`
- $t = 1$: *A* starts



A sample run

Illustrating operational semantics

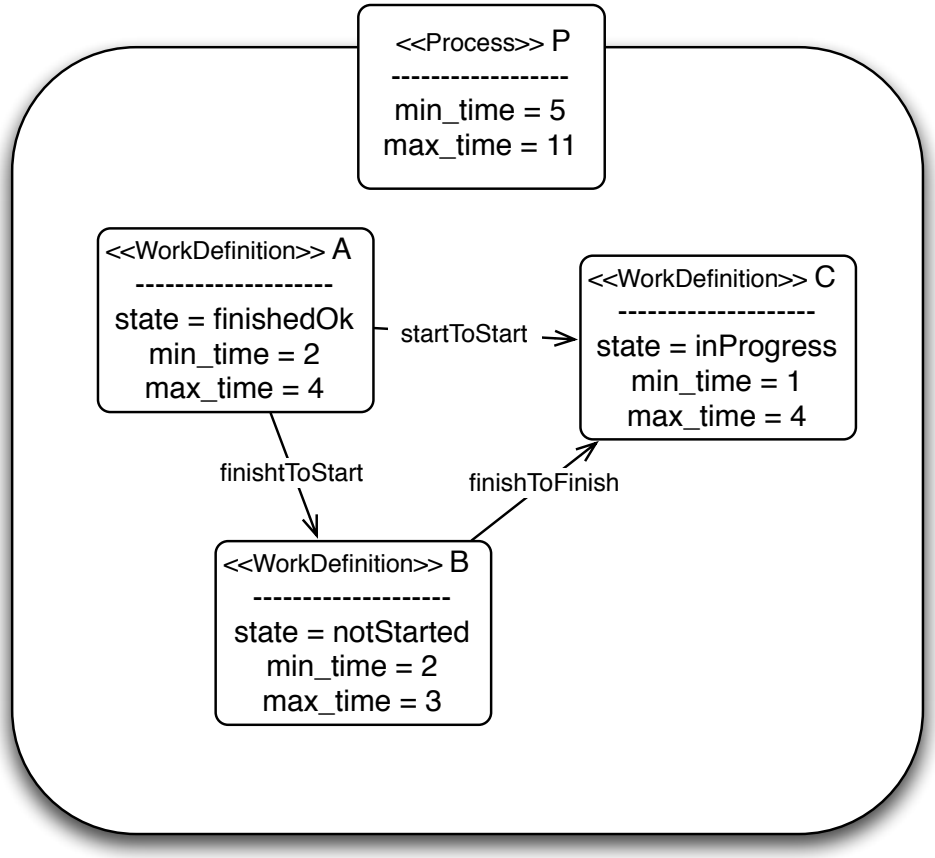
- $t = 0$: *WD*s are `notStarted`
- $t = 1$: *A* starts
- $t = 3$: *B* starts



A sample run

Illustrating operational semantics

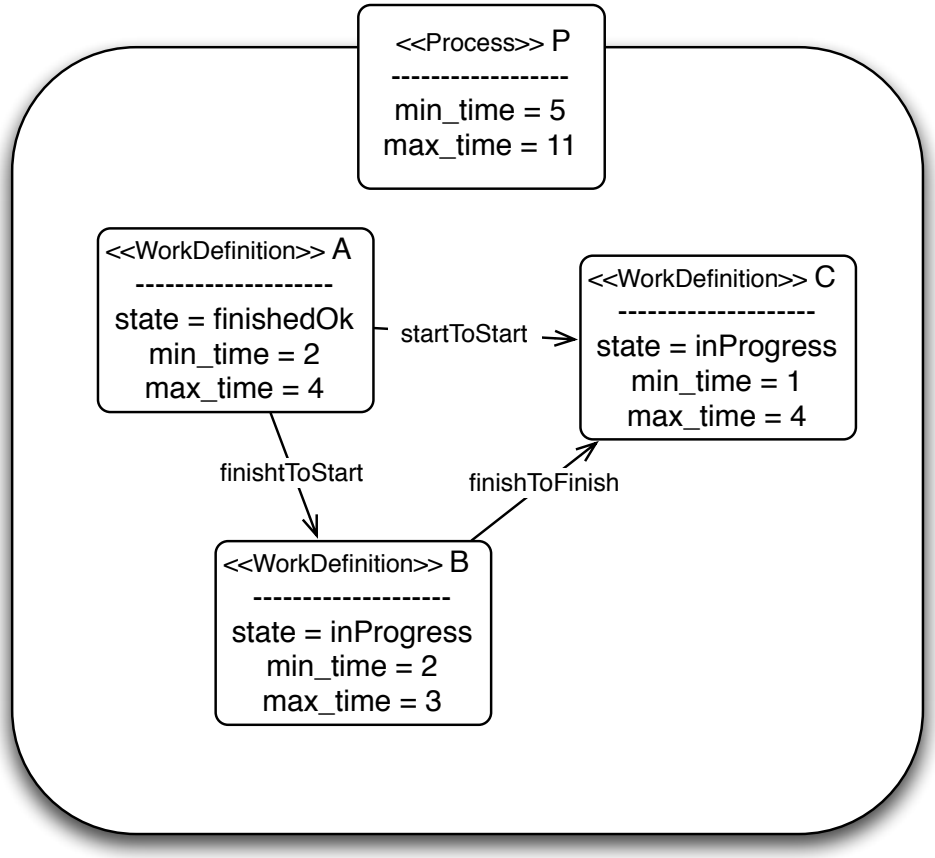
- $t = 0$: *WD*s are `notStarted`
- $t = 1$: *A* starts
- $t = 3$: *B* starts
- $t = 4$: *A* completes



A sample run

Illustrating operational semantics

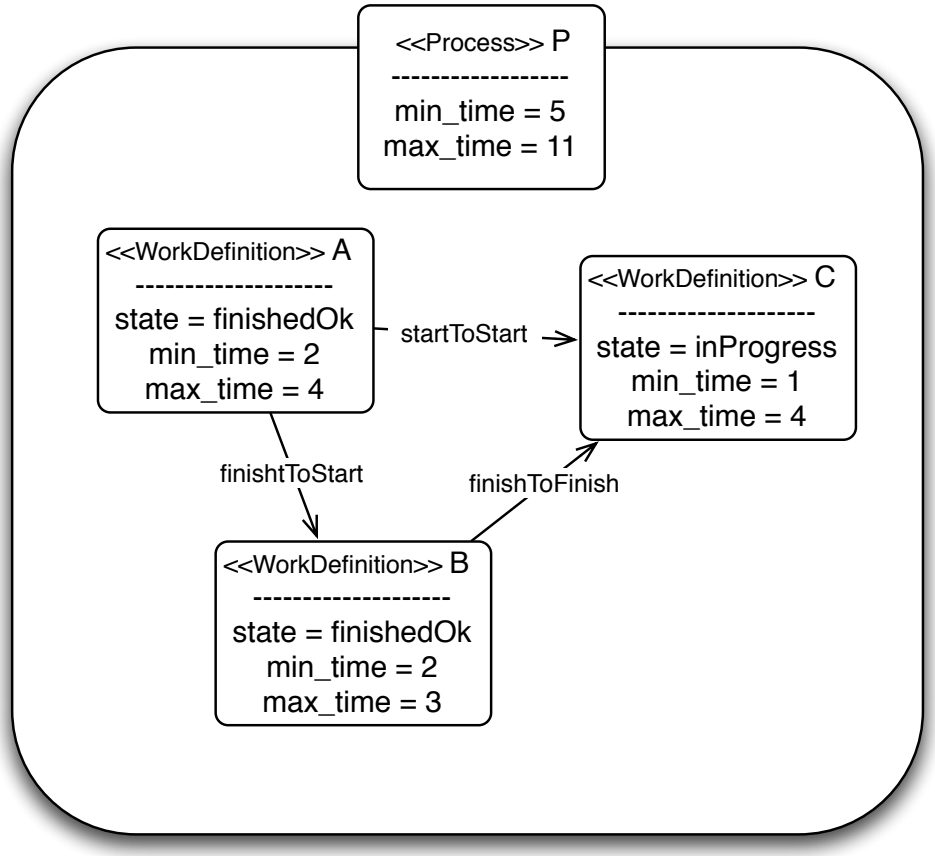
- $t = 0$: *WD*s are `notStarted`
- $t = 1$: *A* starts
- $t = 3$: *B* starts
- $t = 4$: *A* completes
- $t = 5$: *C* starts



A sample run

Illustrating operational semantics

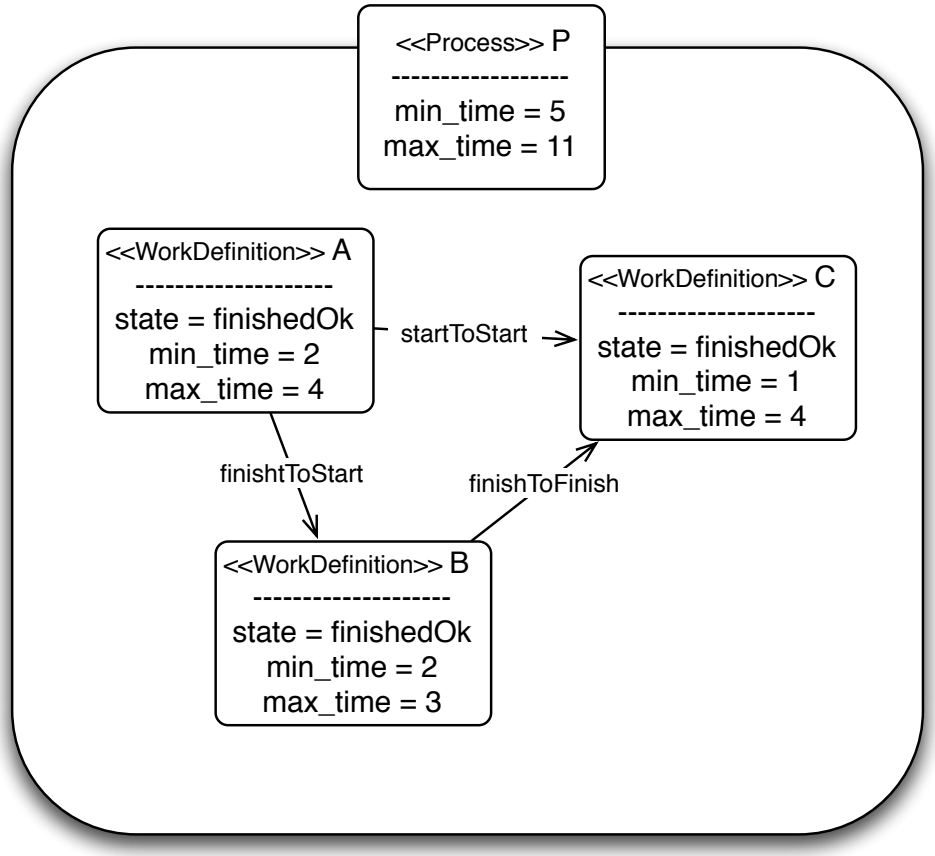
- $t = 0$: *WD*s are `notStarted`
- $t = 1$: *A* starts
- $t = 3$: *B* starts
- $t = 4$: *A* completes
- $t = 5$: *C* starts
- $t = 7$: *B* completes



A sample run

Illustrating operational semantics

- $t = 0$: *WD*s 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

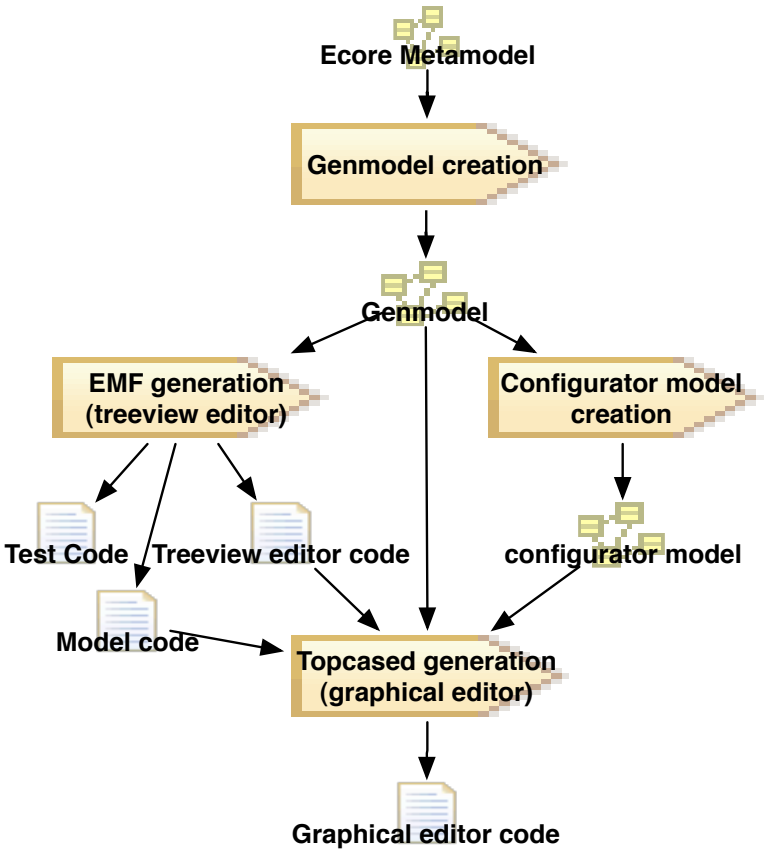


Definition of concrete syntaxes

- 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, ...

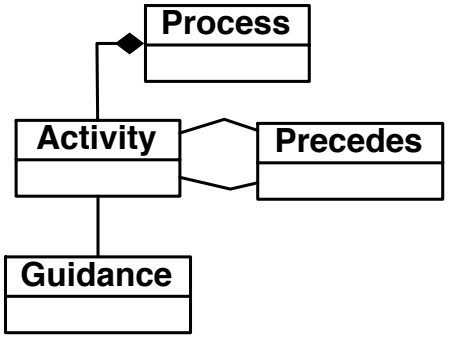
Concrete syntax in Topcased

- 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 resquirements occurs
  - 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)

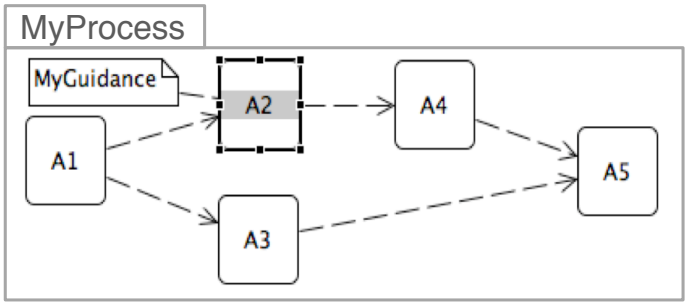


graphical editor generation process

Concrete graphical syntax definition

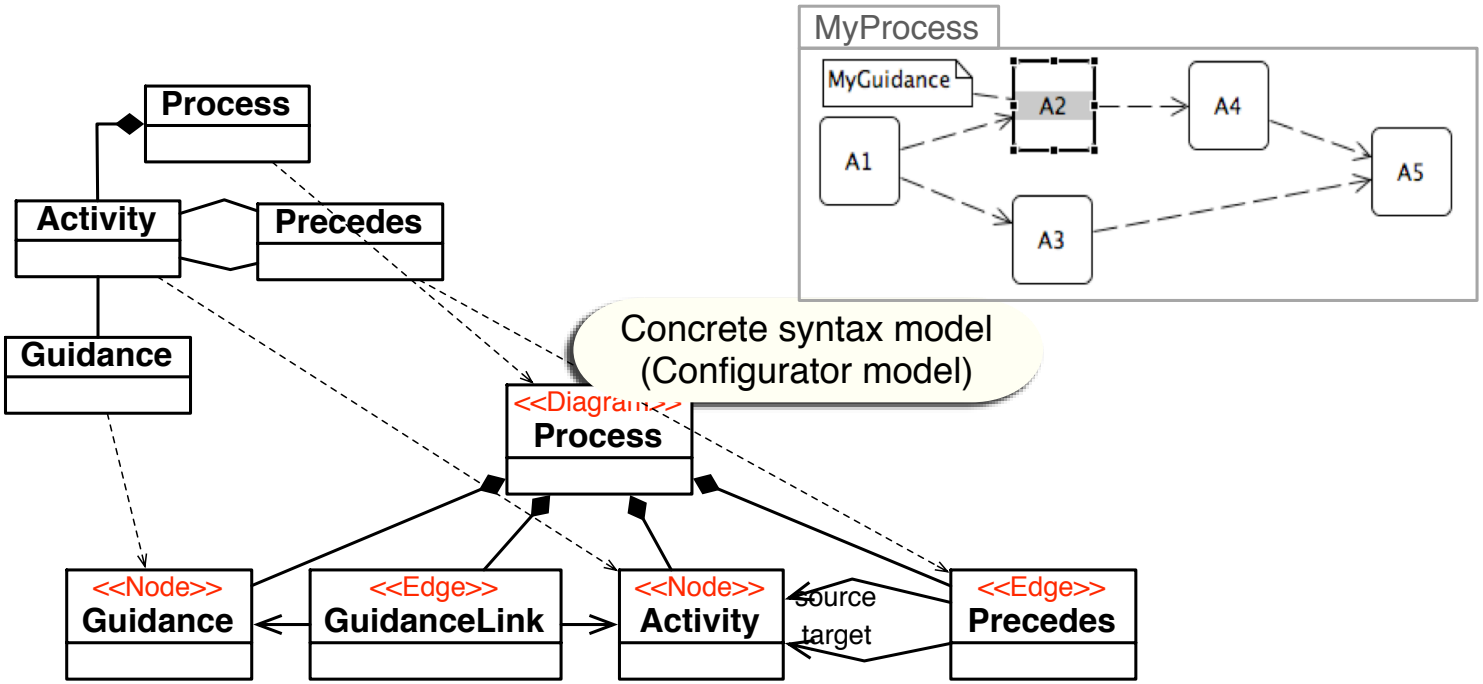


Abstract syntax
(Ecore metamodel)

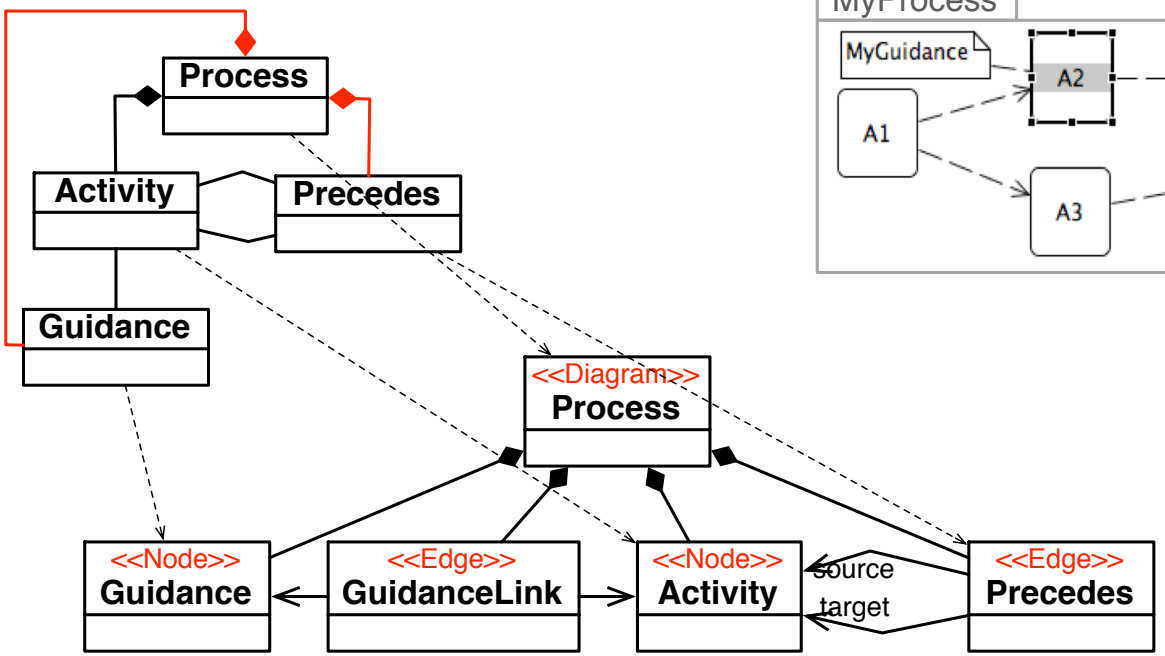


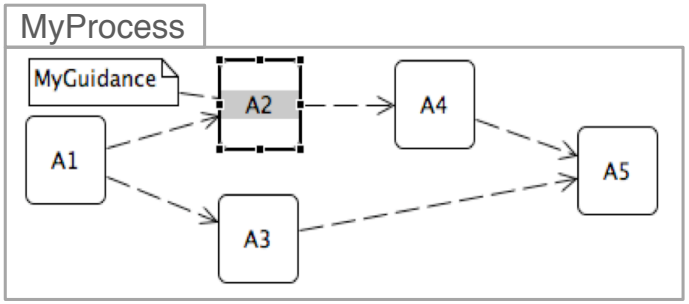
Concrete
graphical syntax

Définition d'une syntaxe concrète

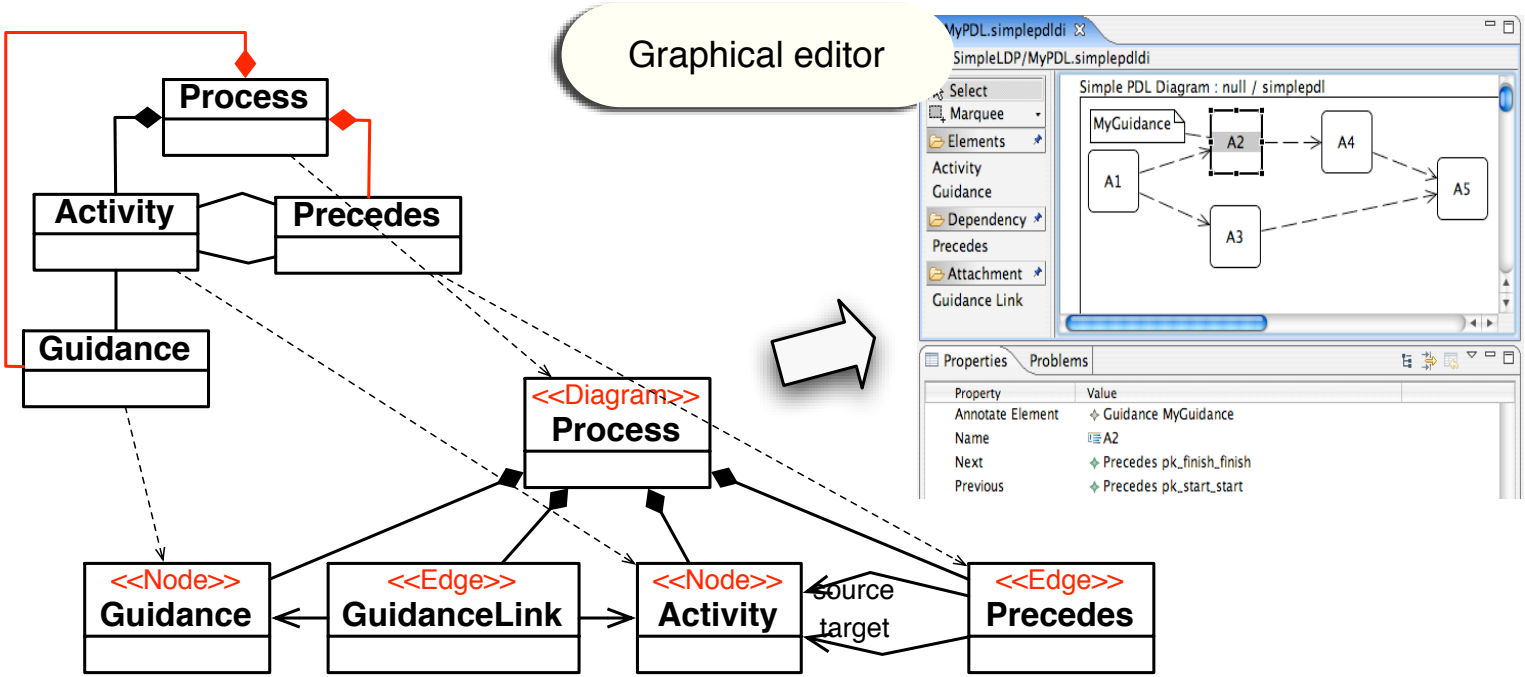


Concrete graphical syntax definition





Concrete graphical syntax definition

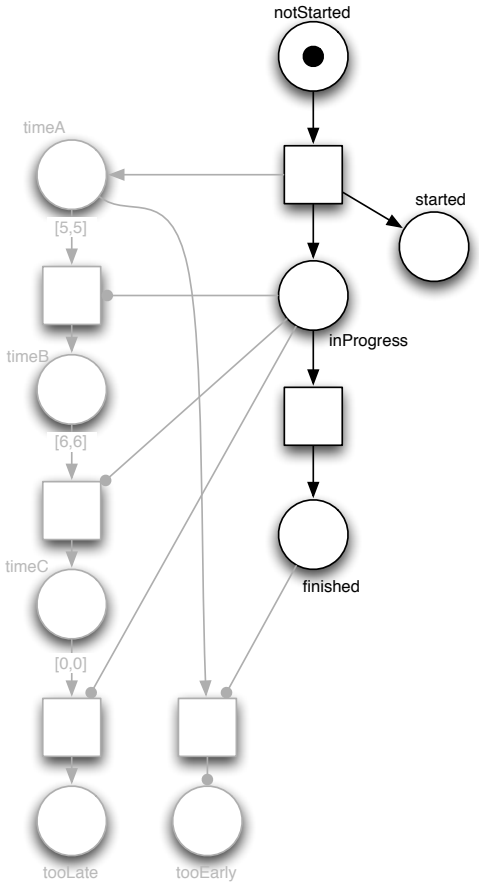


Model transformations and weaving

- 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

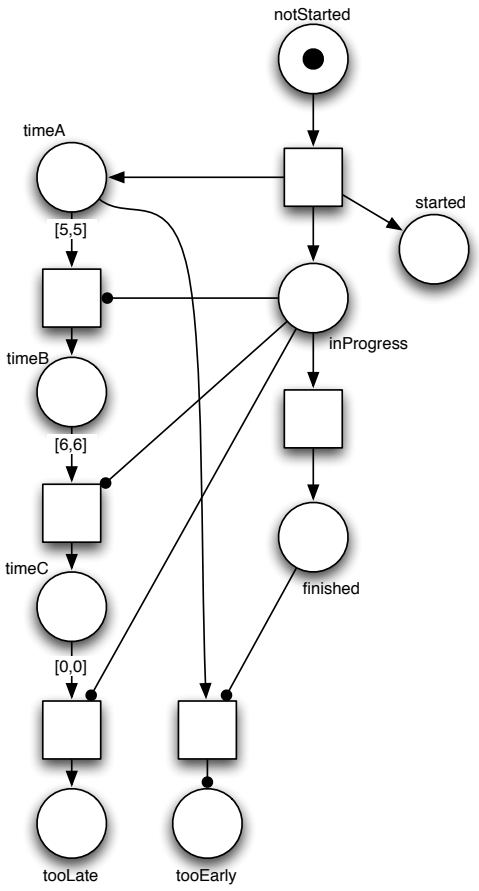
Expressing WorkDefinition Semantics through Petri Nets

A first net to encode simple states :



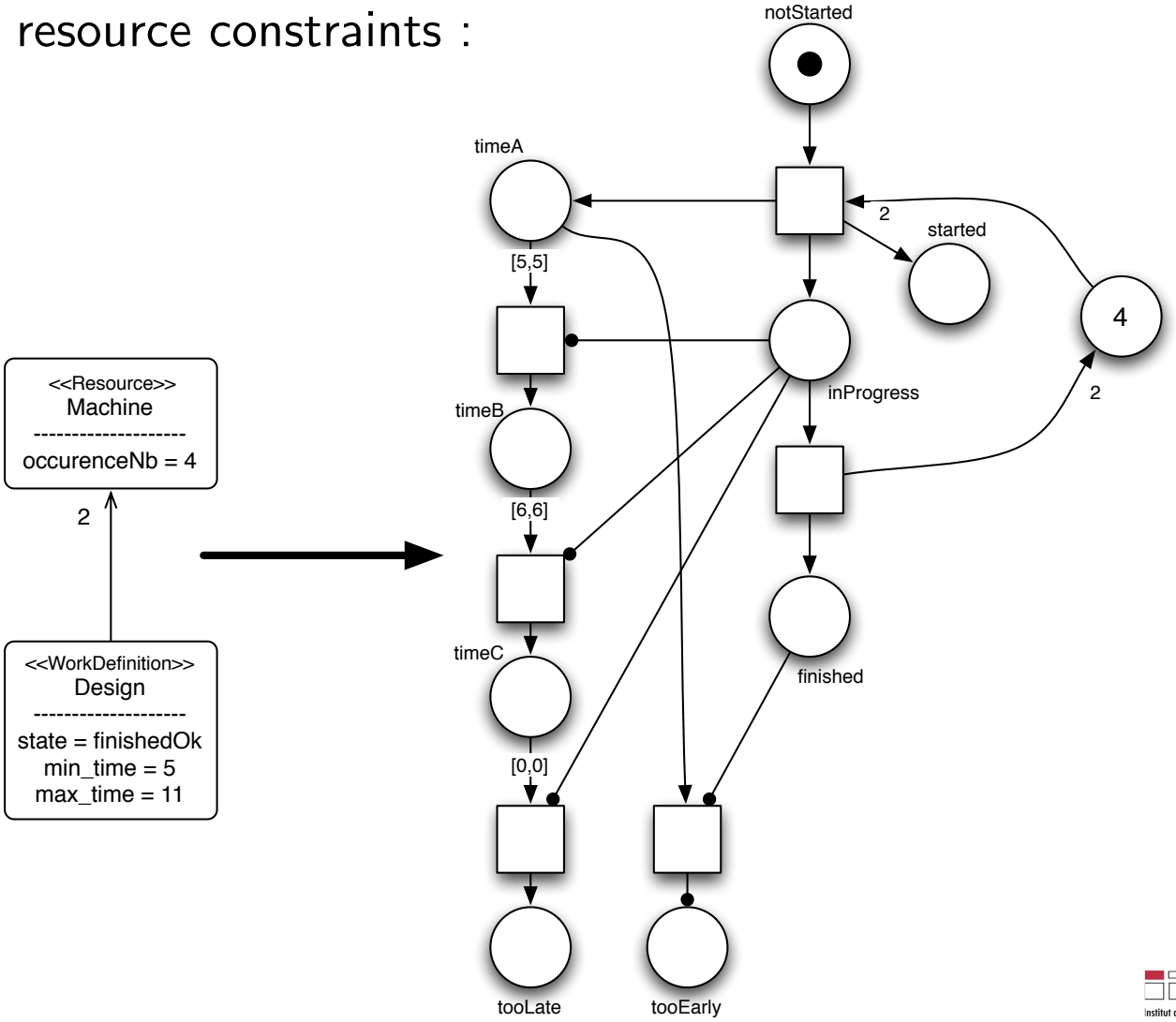
Expressing WorkDefinition Semantics through Petri Nets

Another net to encode time constraints :

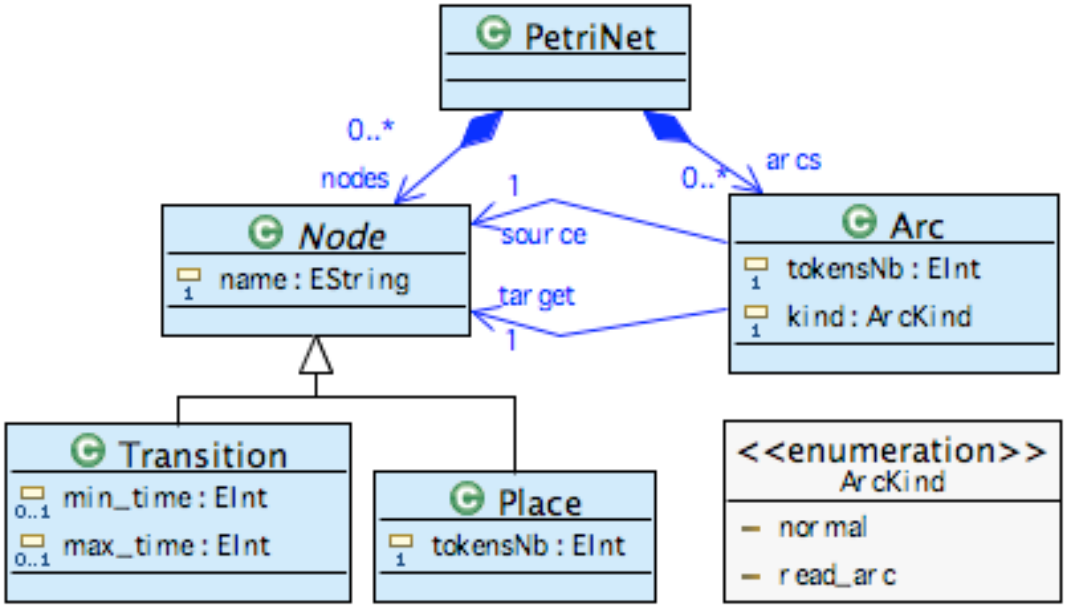


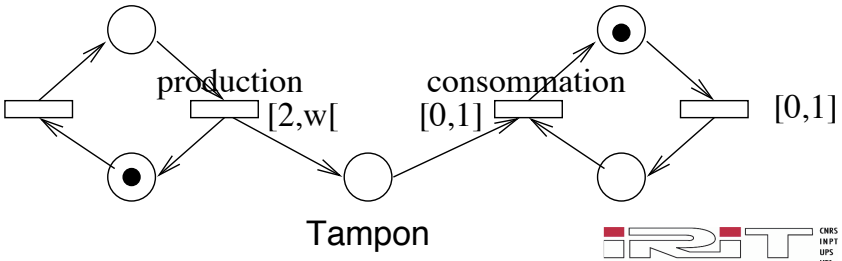
Expressing WorkDefinition Semantics through Petri Nets

Then, we add resource constraints :



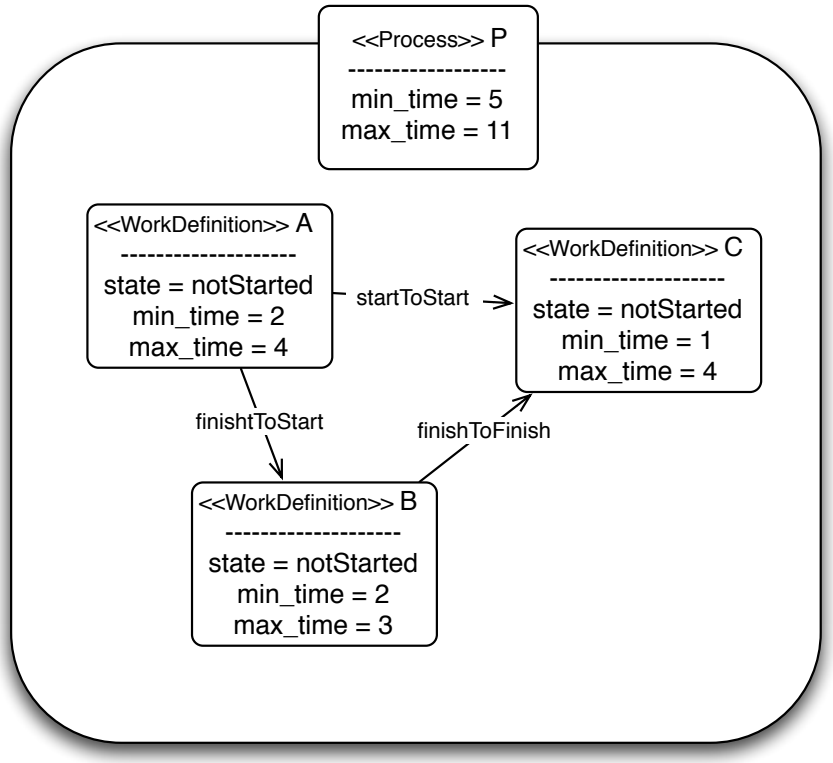
PetriNet, temporal Petri nets

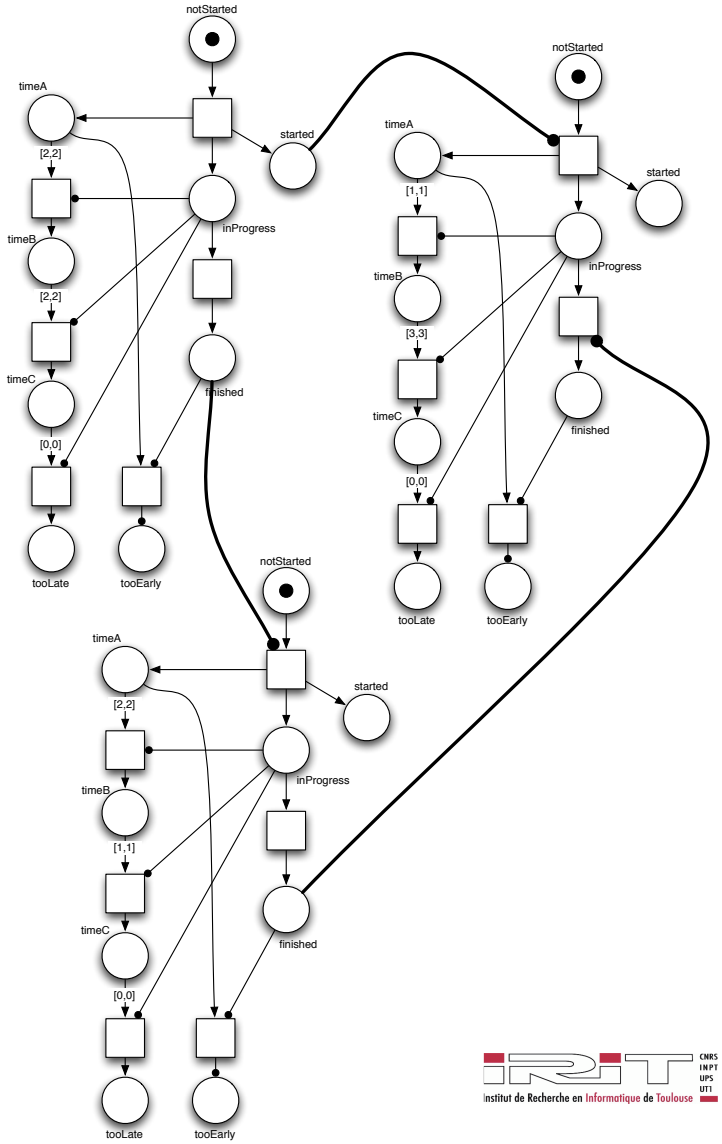




Expressing WorkDefinition Semantics through Petri Nets

Finally, we add causality constraints :



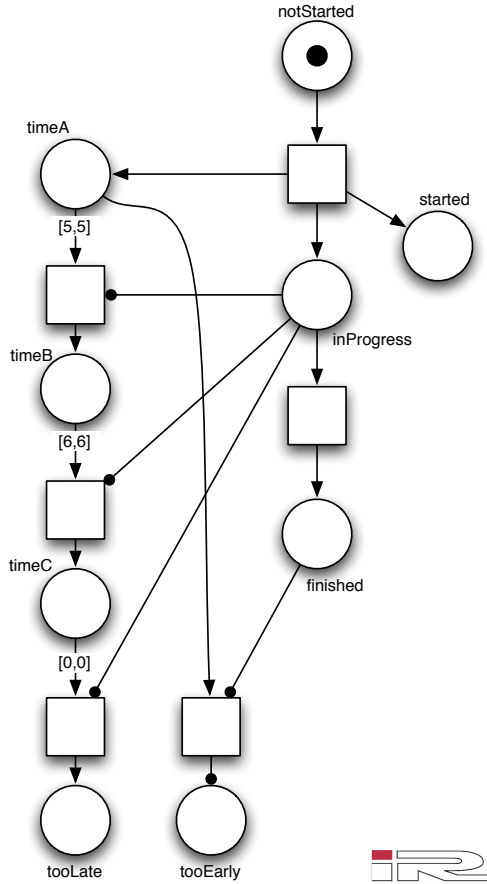


A sample run

Translation into Petri nets

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

- $t = 0$: WD is `notStarted`

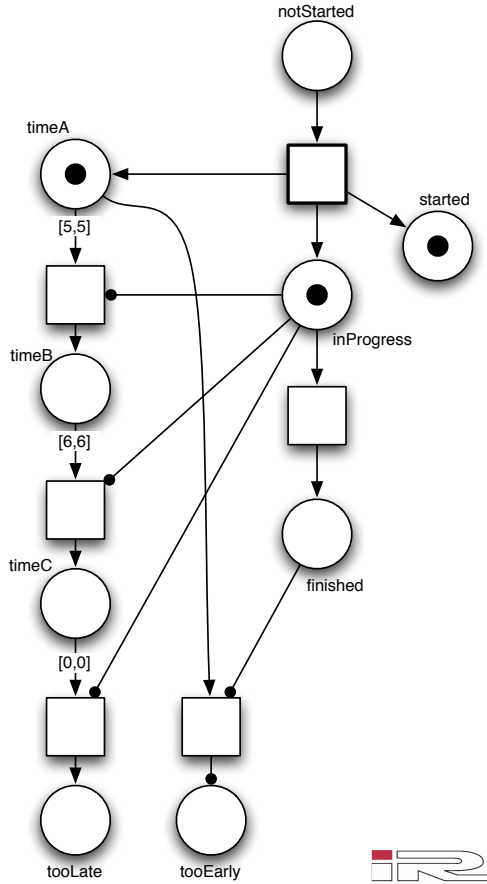


A sample run

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

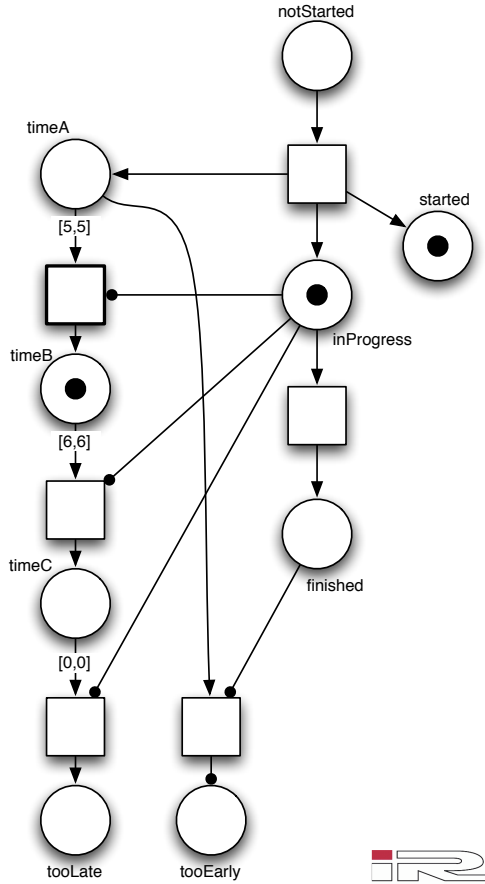


A sample run

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

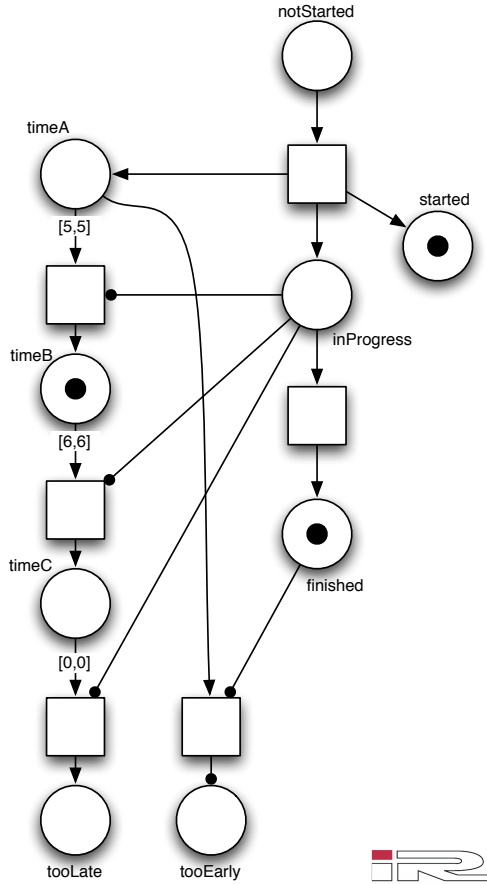


A sample run

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

```
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 )
}
```

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/`

- `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 :

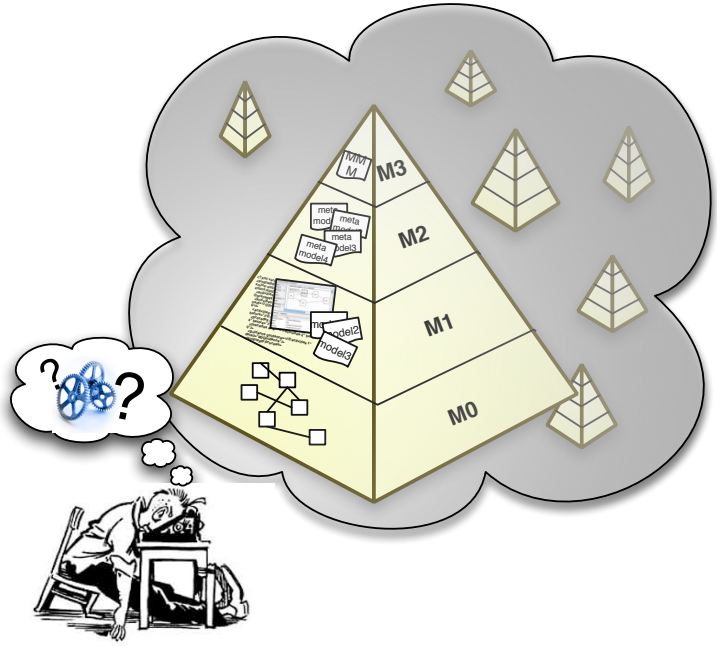
  Express the behavioural semantics of meta-models elements in order to provide executability to models.

- 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.

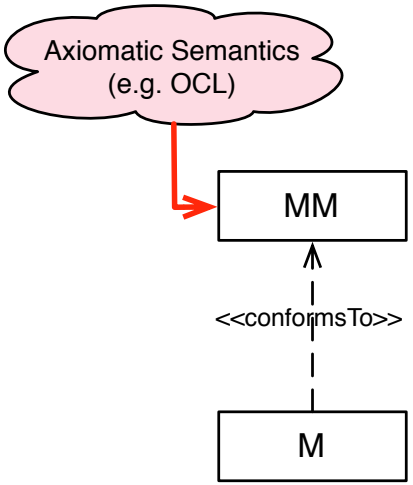
  ⇒ Requires safety and liveness properties for behavioral models.
    - Ensuring models conformity, coherence, ...
    - Ensuring models safety, liveness, ...
    - Provide execution power to models (simulation, validation correctness, ...).
  - Allows rigorous bridging between technological spaces.
- **Our approach** : Applying to modeling languages the well-known principles used in programming languages.



[a]This work is supported by the TOPCASED project.

Static validation for structural properties (Axiomatic semantics)

- 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].



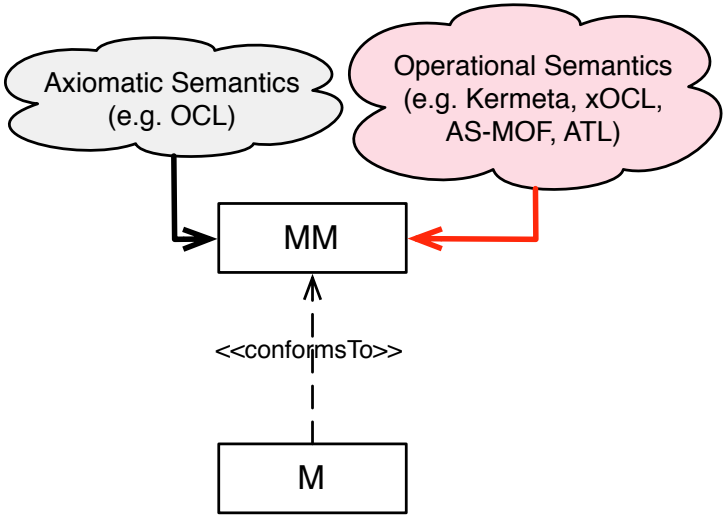
" A process must not contain two activities with the same name " :

context Process inv :
self.activities→ forAll(a1, a2 : Activity |
a1 <> a2 implies a1.name <> a2.name)

[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...



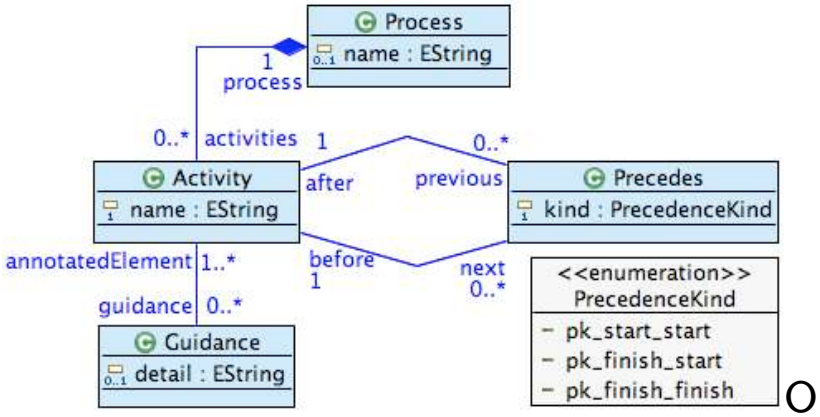
Operational semantics

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
    - Wether 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.



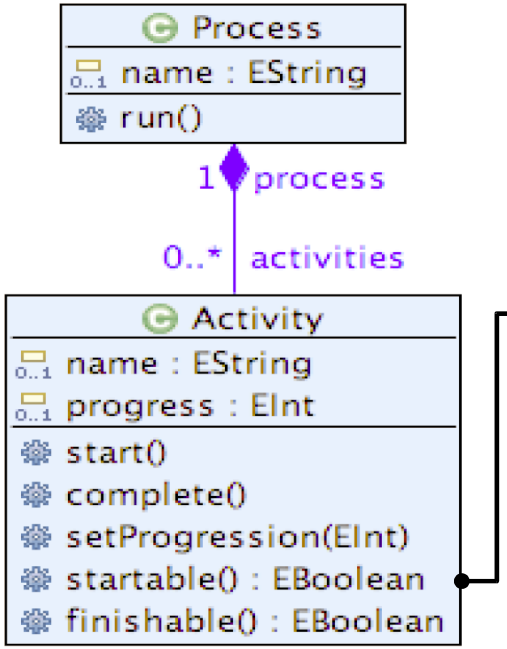
SimplePDL example described with Ecore Editor of TOPCASED

The Kermeta toolkit

A quick presentation

- Developped 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




```
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
```

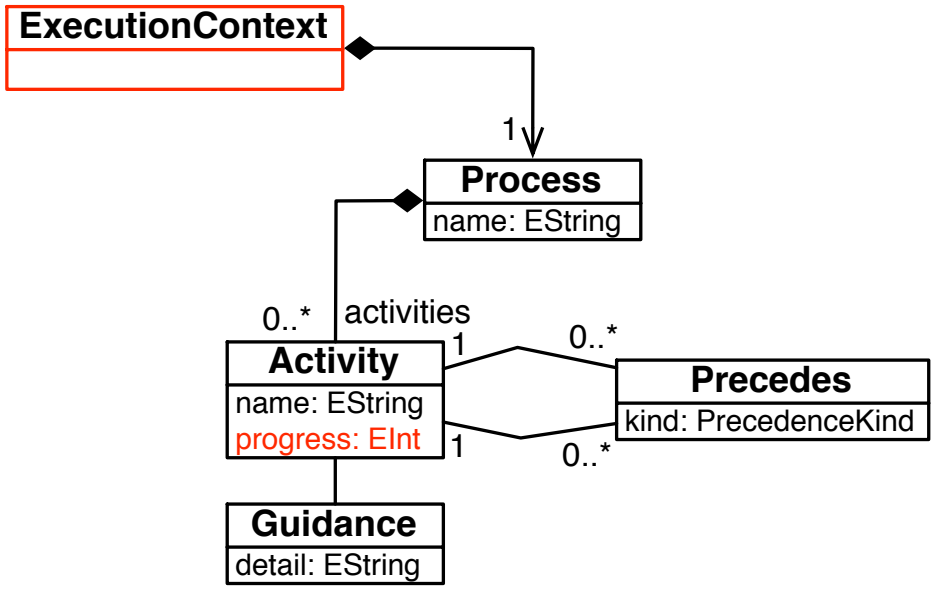
startable() Operation : Kermeta code

Programing language Structural Operational Semantics

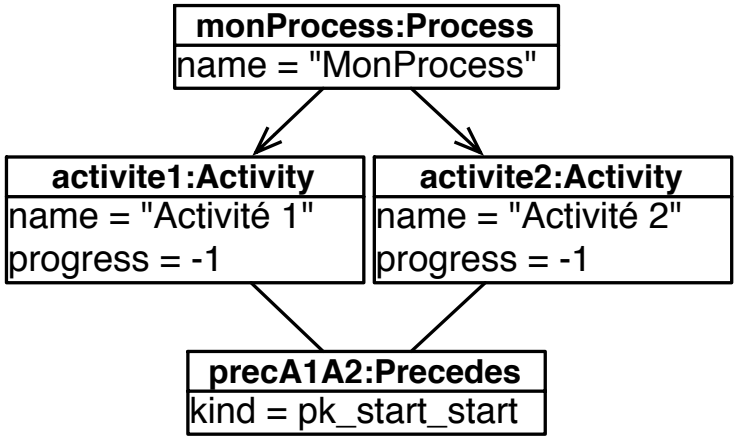
- **From the syntax**
  - ⇒ Syntactic values (Integer, ...)
  - ⇒ Executable constructs (expressions,instructions, ...)
- **Define the semantic**
  - ⇒ Environment
    - Input/Output
    - Memory
    - Variable binding
    - State
  - ⇒ Build *semantic values* from syntactic ones by rewriting
  - ⇒ SOS or Natural semantics define **reduction rules**
    Env + Executable construct →* Env' + Values
- Toward MDE : SimplePDL using model transformation
  Use *transformation rules* as *reduction rules*.

Metamodel Extensions

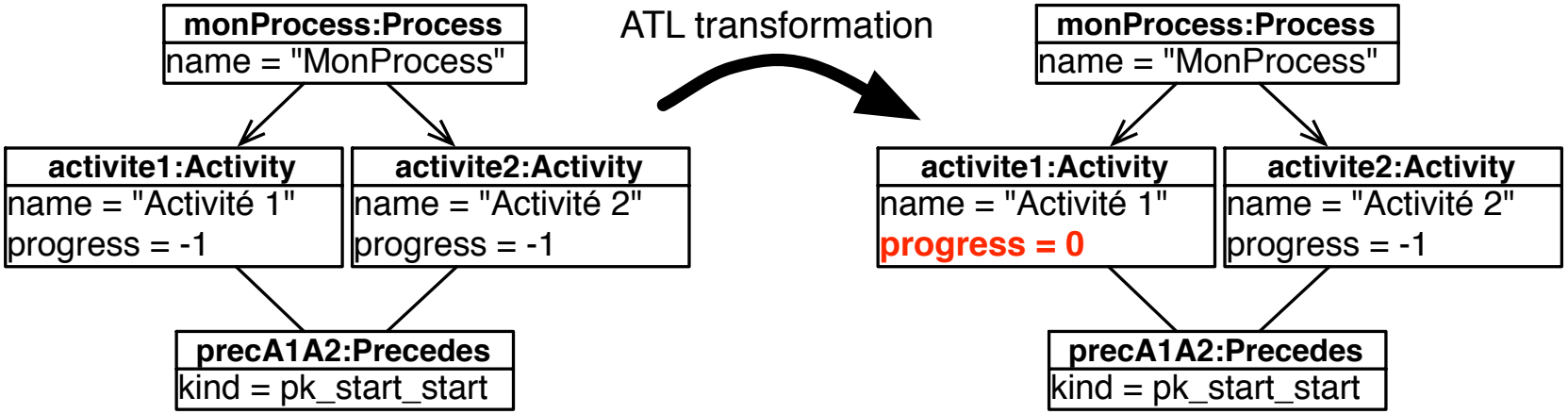
Capture execution state



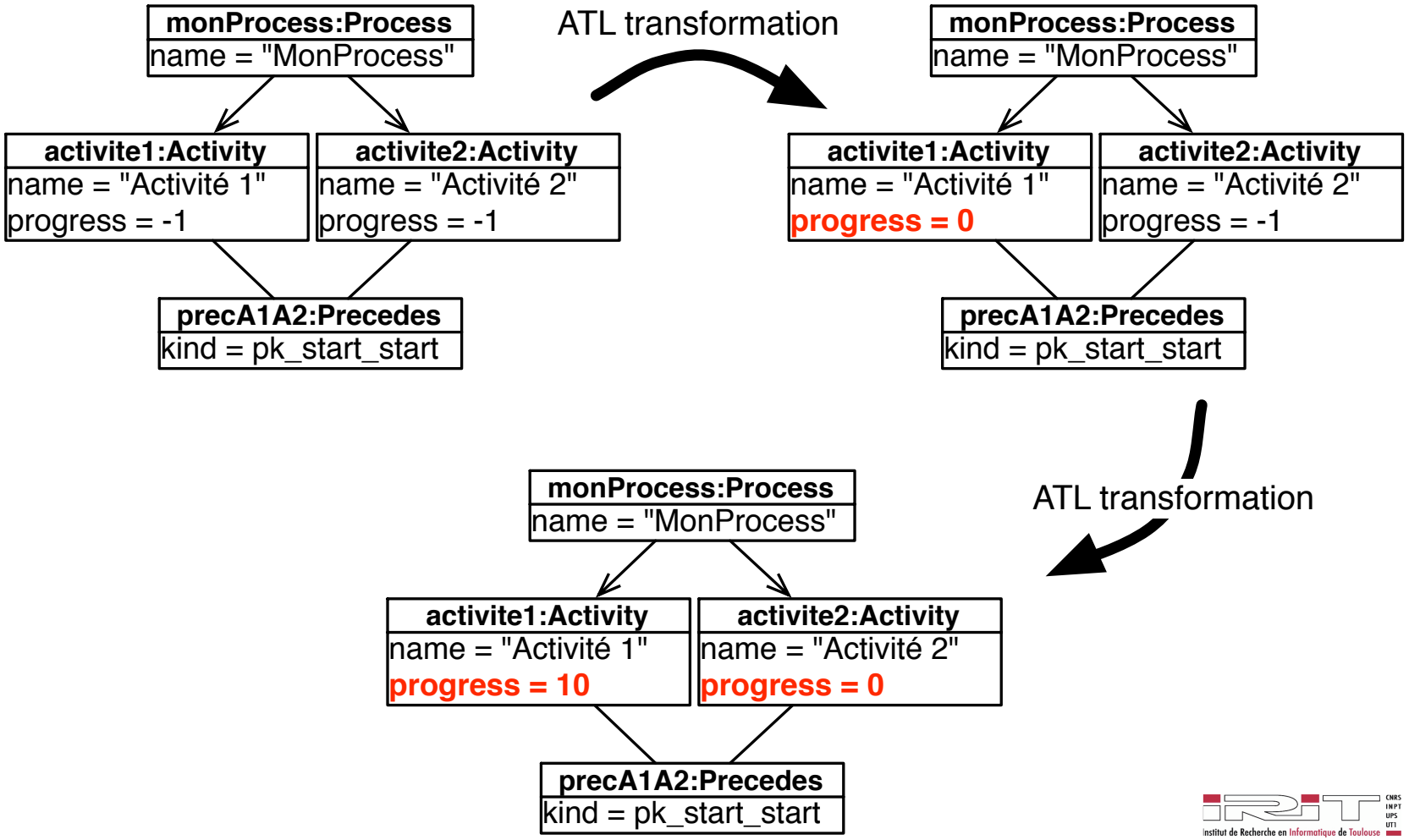
Model transformation



Model transformation



Model transformation



ATL : ATLAS Transformation Language

A quick presentation

- Developped 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

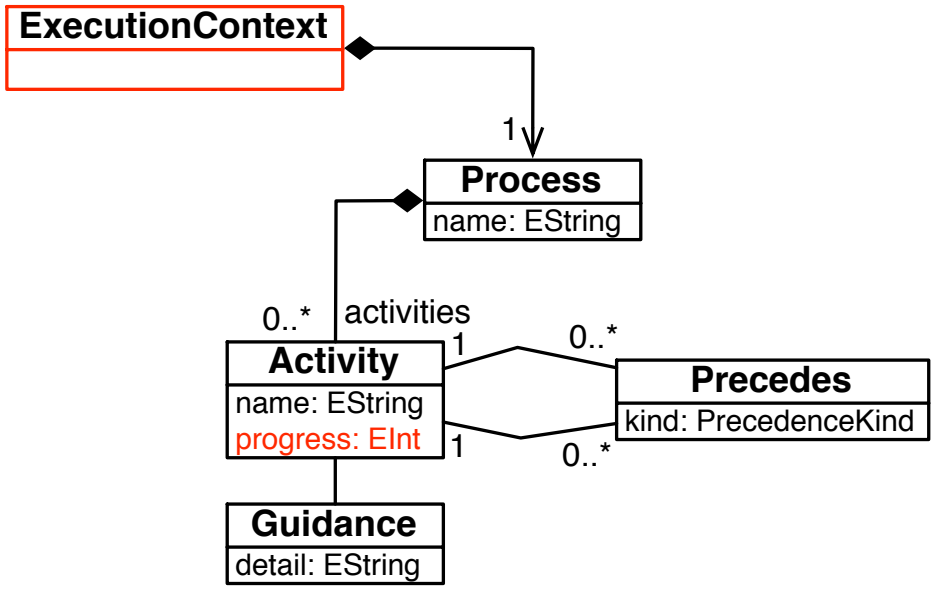
```
rule progressActivity {
  from
    a_in : simplepdl!Activity
  to
    a_out : simplepdl!Activity (
      -- Properties copy
      name <- a_in.name,
              process <- a_in.process,
              previous <- a_in.previous,
              next <- a_in.next,
      -- Progression
      progress <-
              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
              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
  - Painfull in others (requires multi-models capacities)
  - **Correctness**
    - Pure declarative/functional aspects
    - Structural induction approach
    - **May require co-induction**
- **Next point : Graph grammar/rewriting technologies**

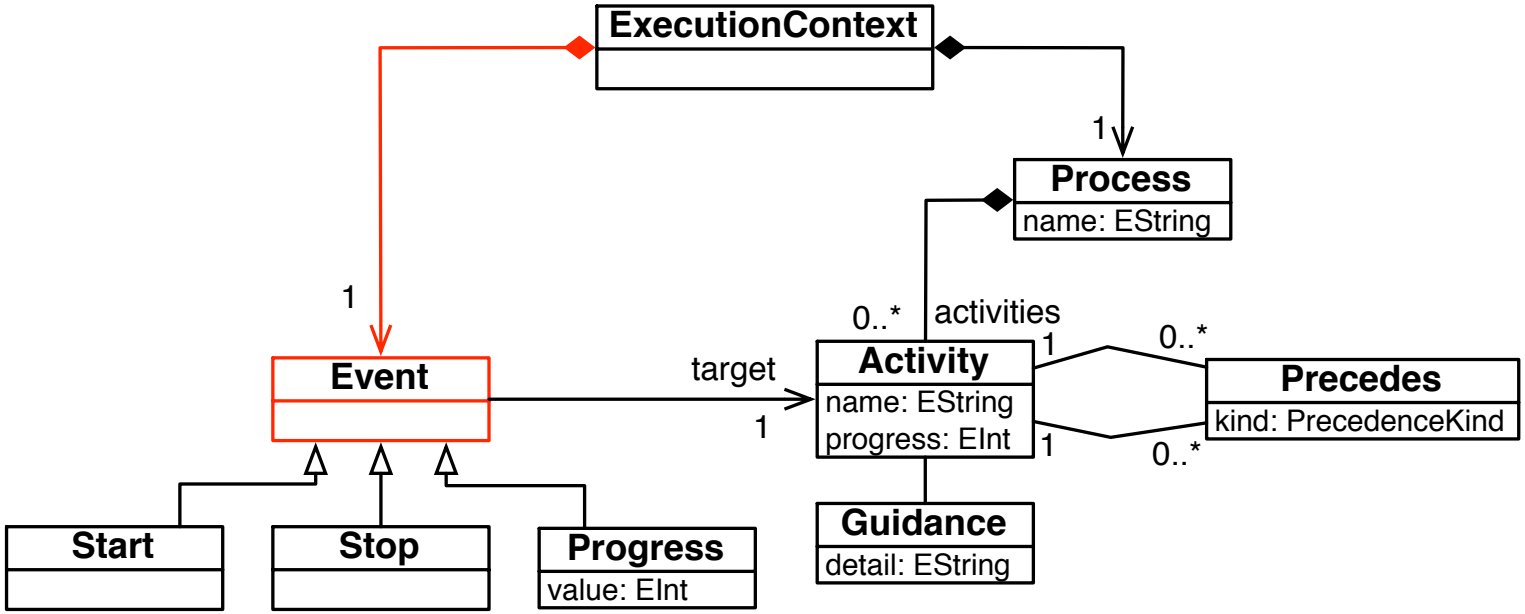
Metamodel Extensions

Capture execution state



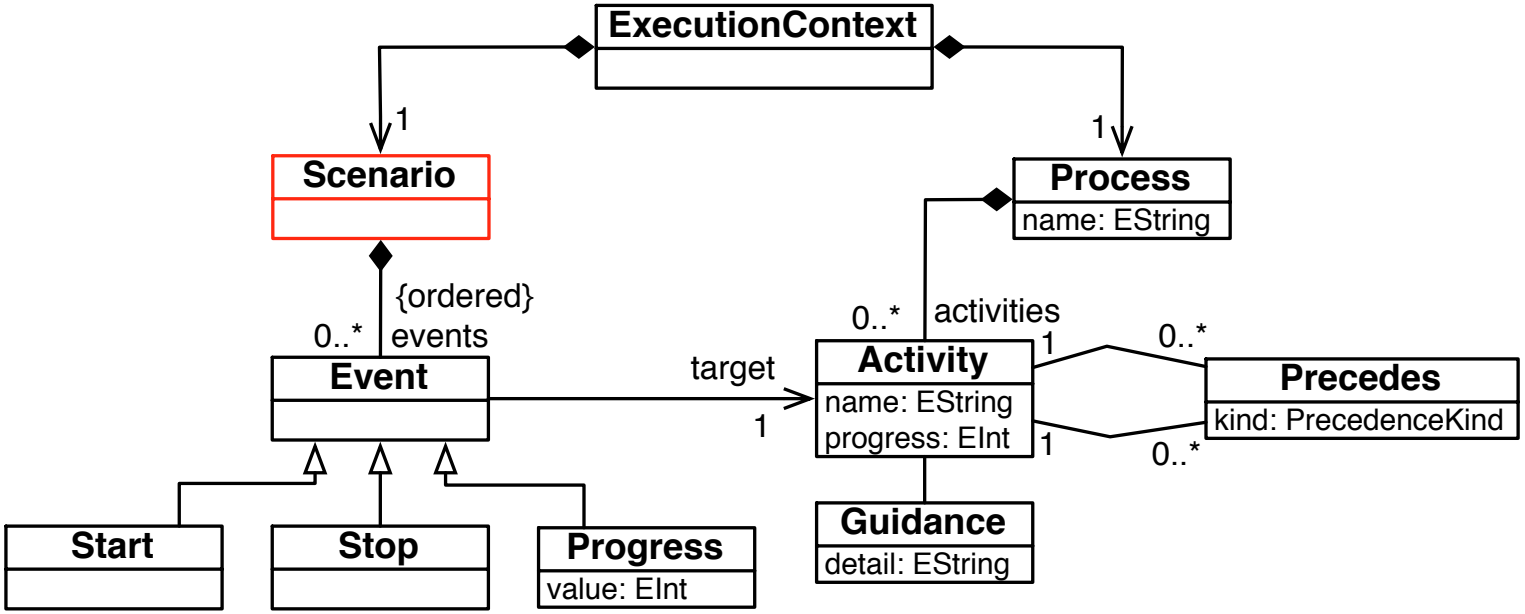
Metamodel Extensions

Scenario model definition



Metamodel Extensions

Scenario model definition



Thank you
for your attention...