«Breathe Life into your Metamodels»

Reda Bendraou
reda.bendraou{{@}}Lip6.fr
http://pagesperso-systeme.lip6.fr/Reda.Bendraou/

Supports de cours par Benoit Combemale et Jean-Marc Jézéquel
benoit.combemale@irisa.fr
jezequel@inria.fr

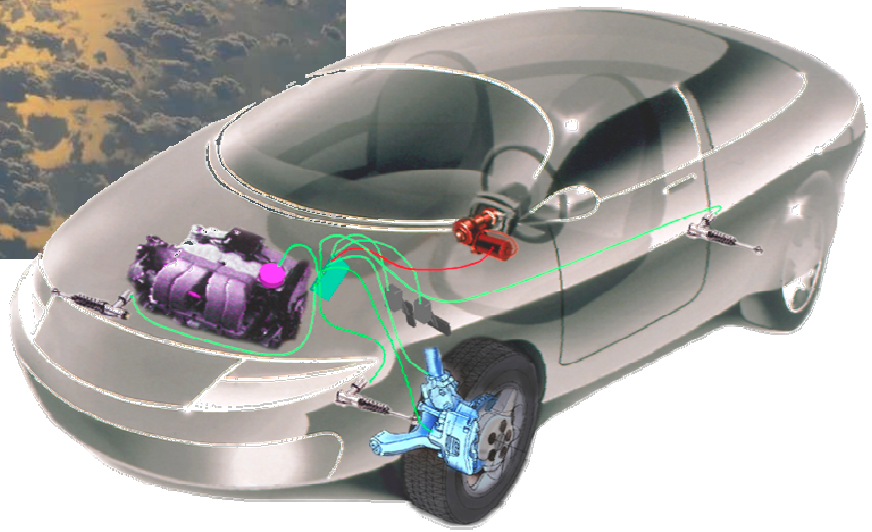
Outline

- **Introduction to MDE**
- **Defining the Operational Semantics with Kermeta**
- **The Logo Example (short reminder)**
- **Building a Simulator for Logo**
- **Building a Compiler for Logo**
- **Wrap-up and Conclusion**

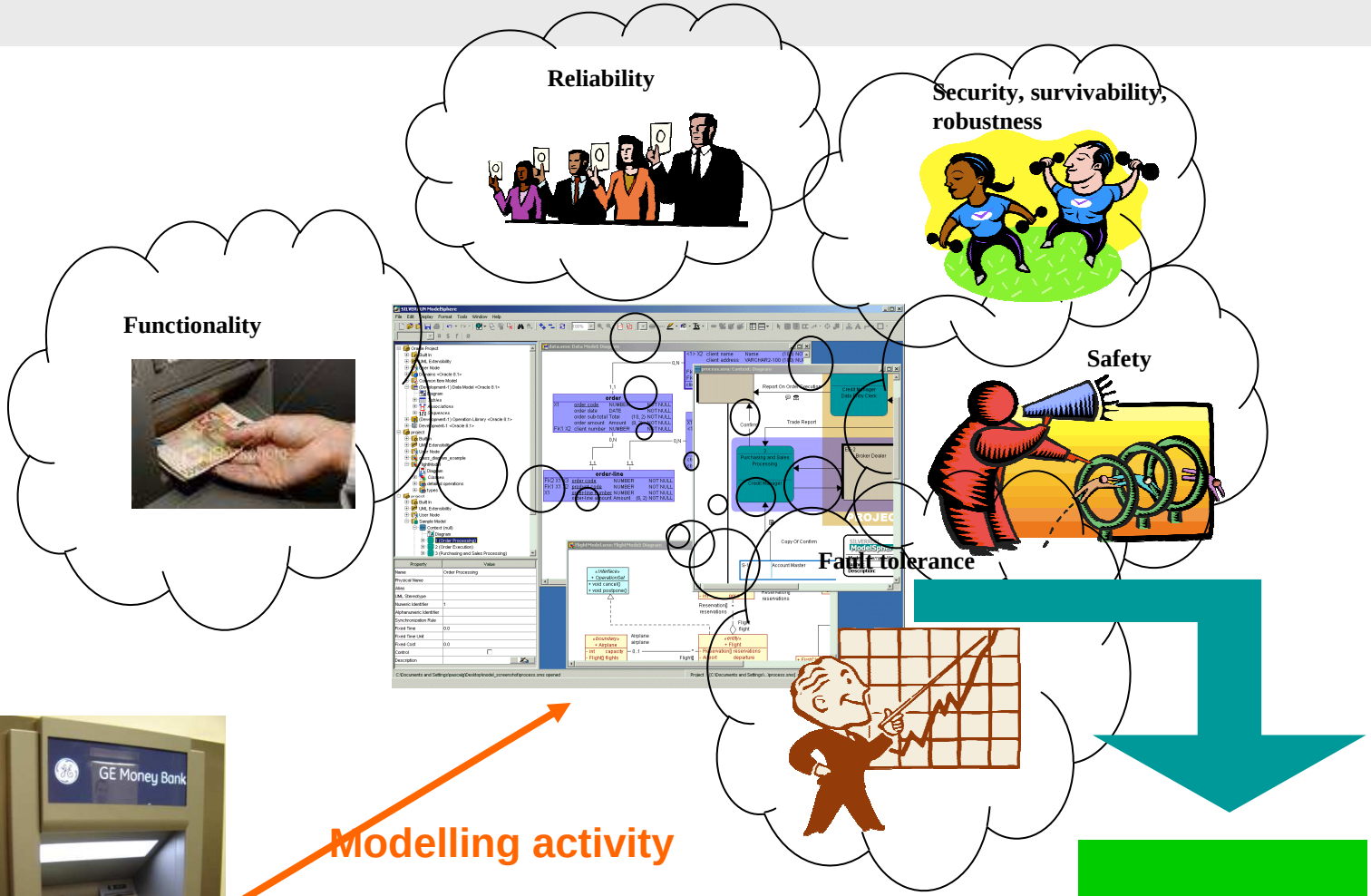
Software Complexity

Software Complexity

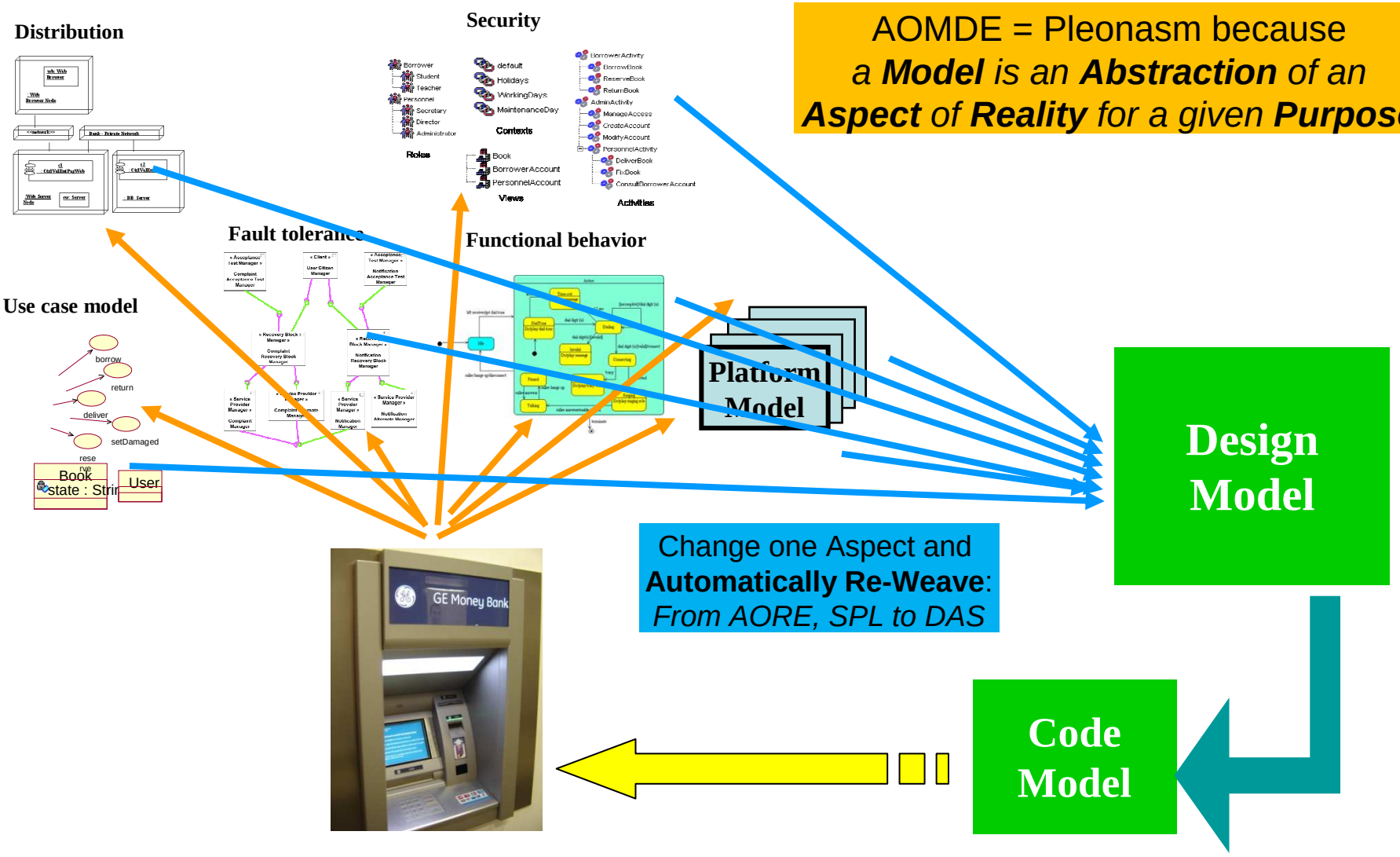
- **Critical**
- **Real-time**
- **Embedded**



Naive Model Driven Engineering



Aspect Oriented
Model Driven Engineering



AOMDE = Pleonasm because
*a **Model** is an **Abstraction** of an **Aspect** of **Reality** for a given **Purpose***

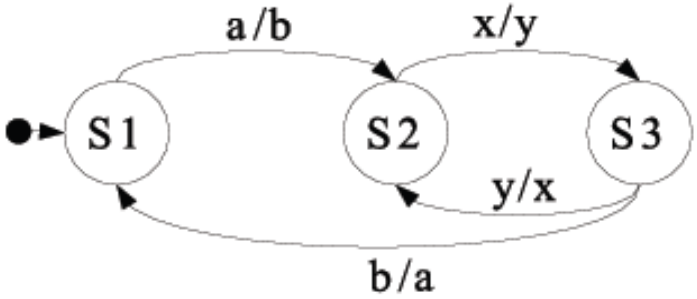
Change one Aspect and
Automatically Re-Weave:
From AORE, SPL to DAS

Outline

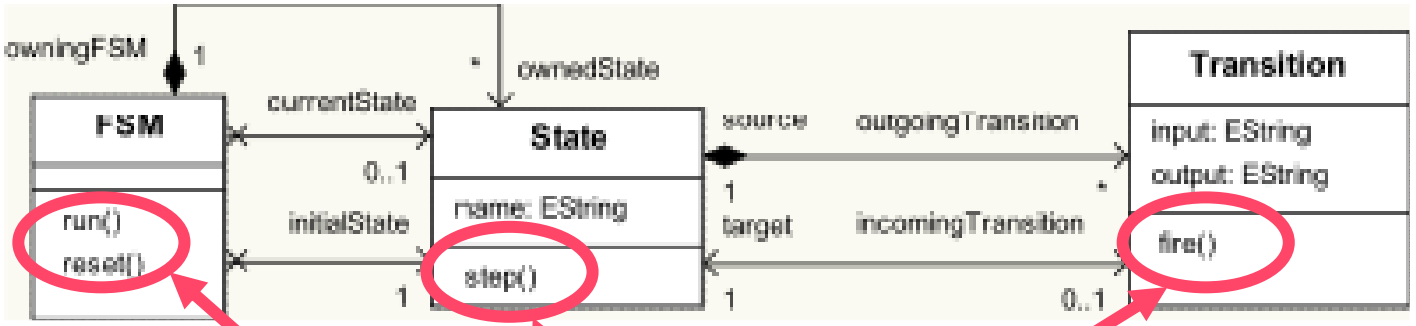
- **Introduction to MDE**
- **Defining the Operational Semantics with Kermeta**
- **The Logo Example (short reminder)**
- **Building a Simulator for Logo**
- **Building a Compiler for Logo**
- **Wrap-up and Conclusion**

Operational Semantics of State Machines

- A model

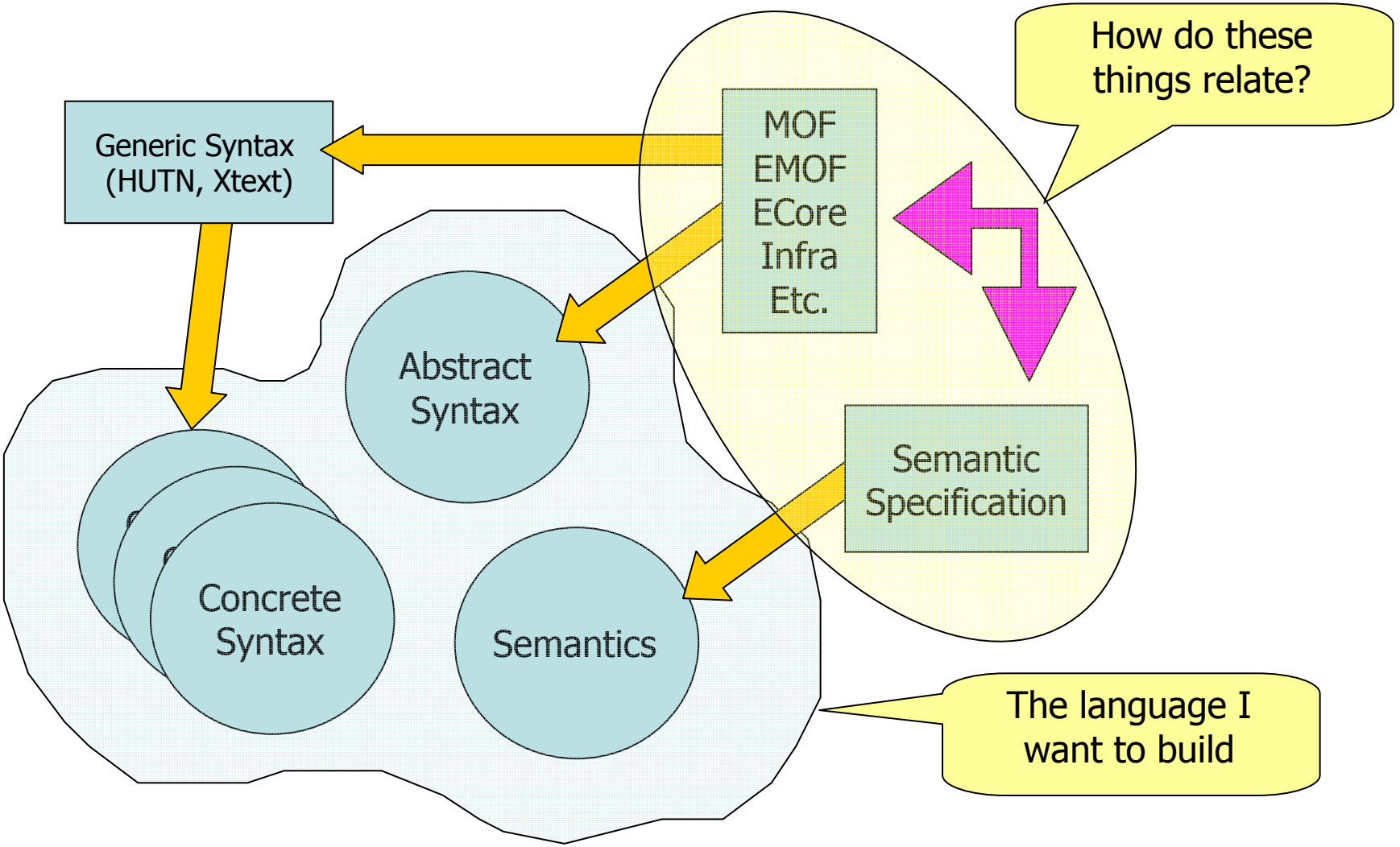


- Its metamodel



- Adding Operational Semantics to OO Metamodels

From Metamodels to Languages



Metadata languages

- (E)MOF => Only data structures
  - classes, properties, associations, ...
  - operations : only signatures

- Not sufficient to operate on models
  - Constraints
  - Actions
  - Transformations
  - ...

Typical example (excerpted from MOF spec)

- *Operation* ***isInstance(element : Element) : Boolean***

  "Returns true if the element is an instance of this type or a subclass of this type. Returns false if the element is null".

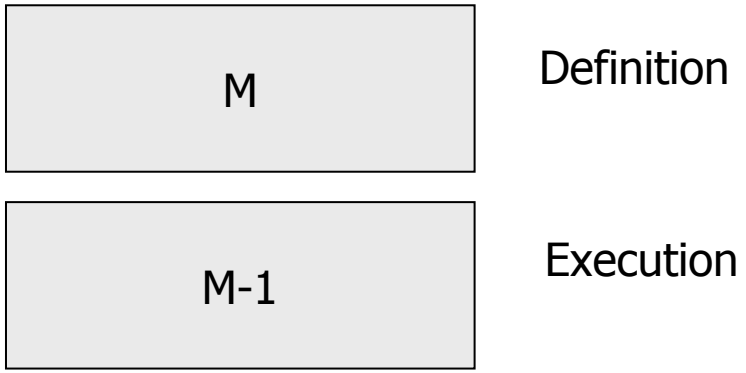
A natural language specification

```
operation isInstance (element : Element) : Boolean is do
   // false if the element is null
   if element == void then result := false
   else
      // true if the element is an instance of this type
      // or a subclass of this type
      result := element.getMetaClass == self or
          element.getMetaClass.allSuperClasses.contains(self)
   end
end
```

An operational specification

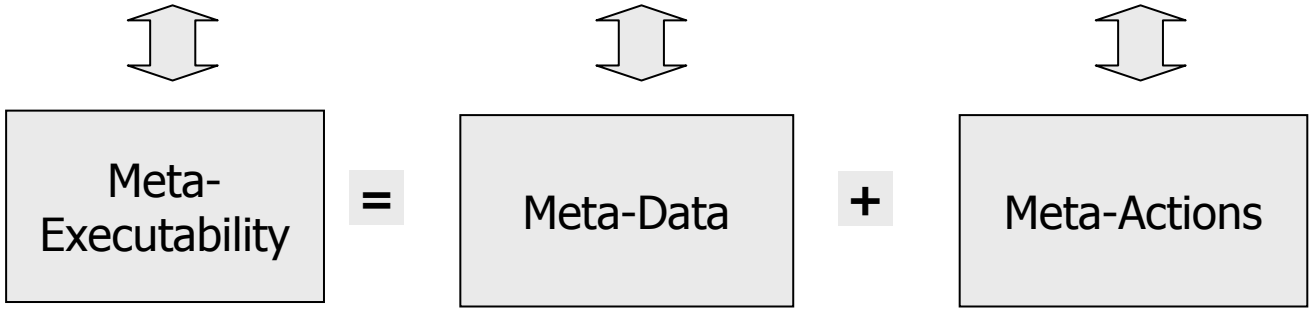
What is “meta”-executability?

- Basic CRUD Operations
- Merge, Composition…



- Simply an (object-oriented) program that manipulates model elements

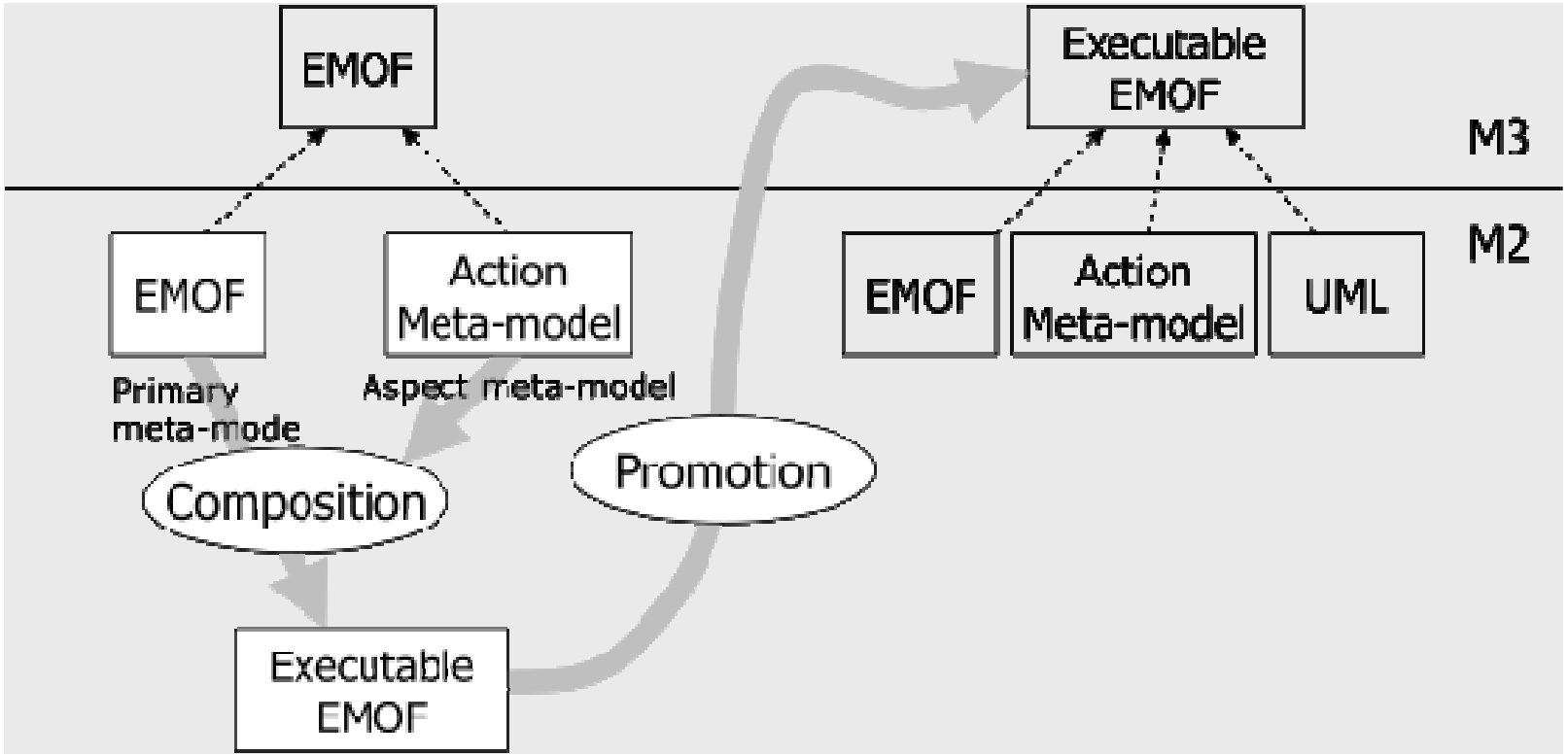
“Program = Data Structure + Algorithm”, Niklaus Wirth



Kermeta Rationale

- Model, meta-model, meta-metamodel, DSLs...
  - Meta-bla-bla too complex for the normal engineer
- On the other hand, engineers are familiars with
  - OO programming languages (Java,C#,C++,..)
  - UML (at least class diagram)
  - May have heard of *Design-by-Contract*
- Kermeta leverages this familiarity to make Meta-modeling easy for the masses

Using aspect-composition to reflectively build Kermeta

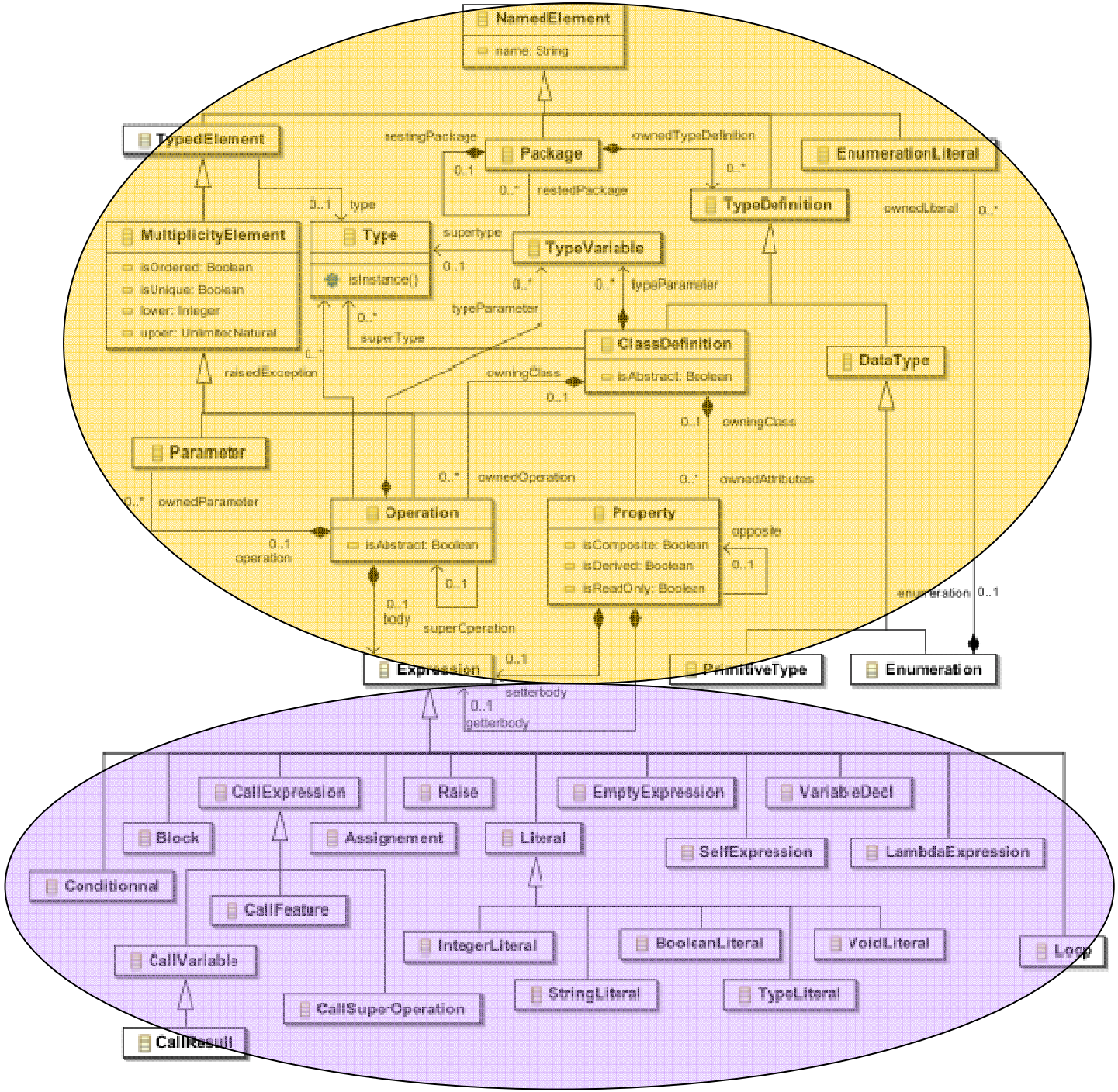


The action metamodel

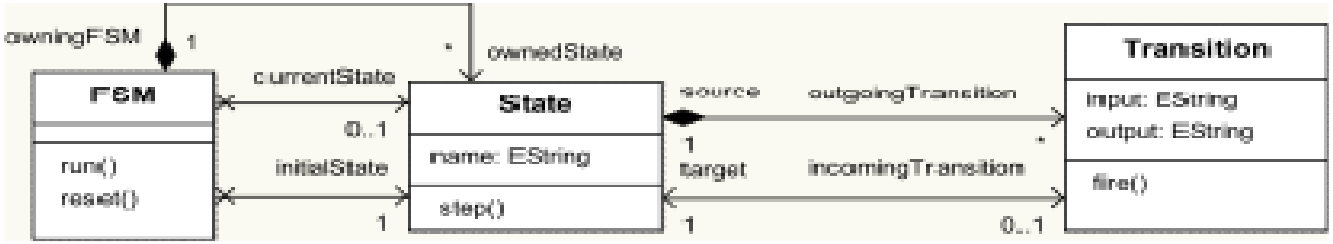
Close to the OCL

- CRUD operation
- Control structures
- Operation call
- Variables and assignment
- Exceptions handling
- Functions (OCL-like iterators)

KerMeta Metamodel



Breathing life into Meta-Models



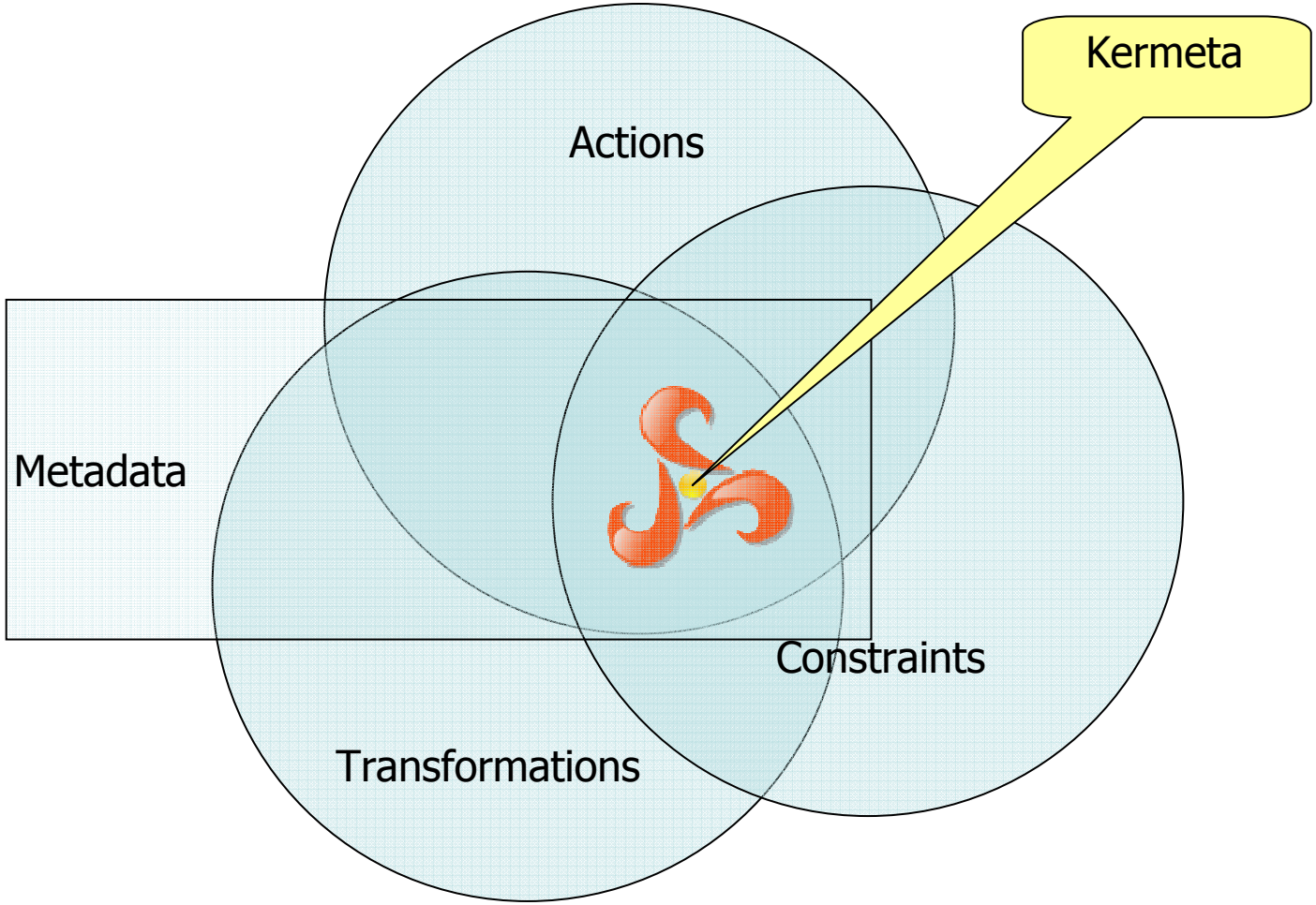
- *// MyKermetaProgram.kmt*
- *// An E-MOF metamodel is an OO program that does nothing*
  - require "StateMachine.ecore" *// to import it in Kermeta*
- *// Kermeta lets you weave in* aspects
  - *// Contracts (OCL WFR)*
  - require “StaticSemantics.ocl”
  - *// Method bodies (Dynamic semantic*
  - require “DynamicSemantics.kmt”
  - *// Transformations*

```
Context FSM
inv: ownedState->forAll(s1,s2|
s1.name=s2.name implies s1=s2)
```

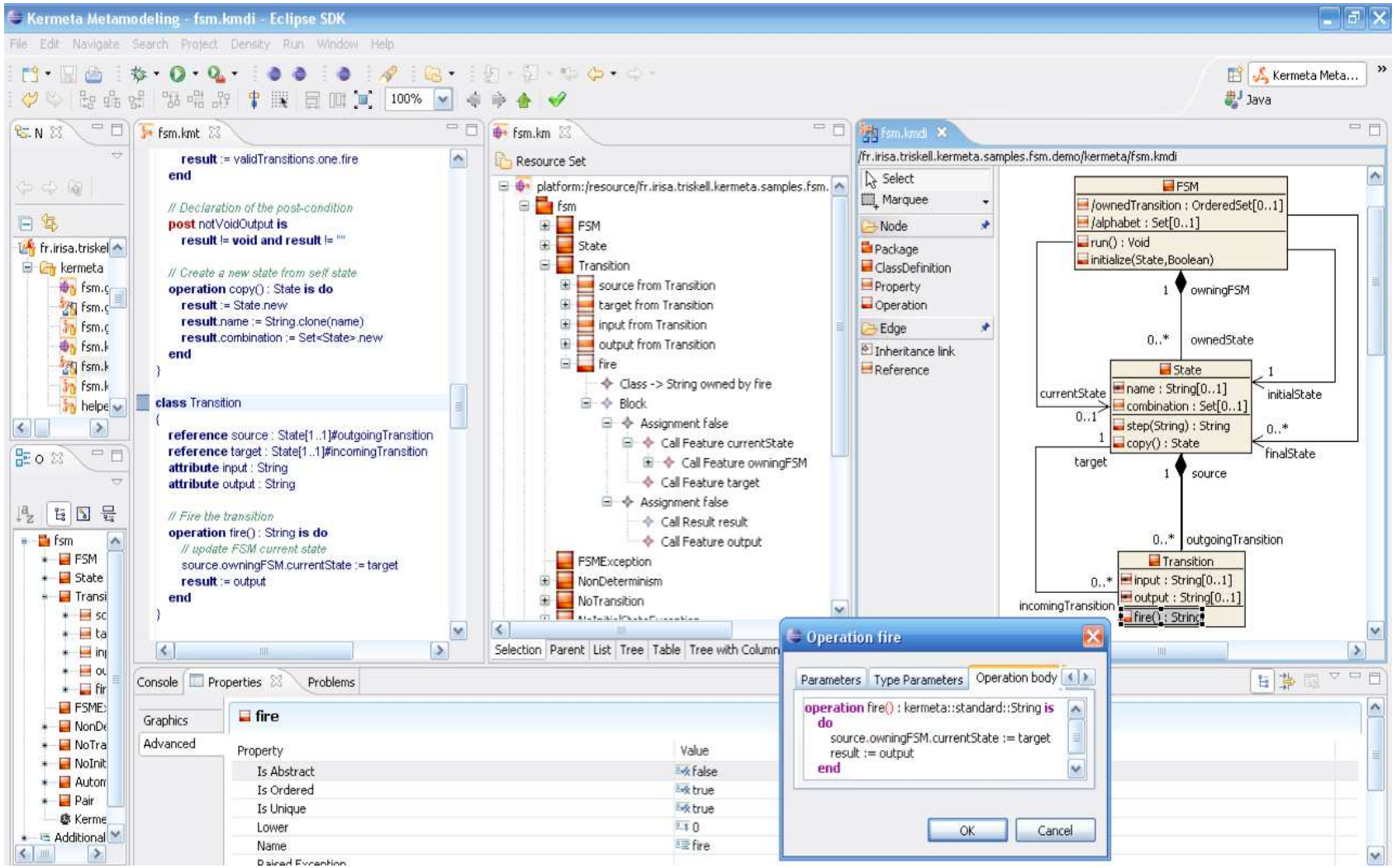
```
aspect class FSM {
   operation reset() : Void {
            currentState := initialState
}}
```

```
class Minimizer {
    operation minimize (source: FSM):FSM {…}
}
```

Kermeta, a Kernel to Meta



Kermeta workbench snapshot



Kermeta:
a Kernel metamodeling language

- Strict EMOF extension
- Statically Typed
  - Generics, Function types (for OCL-like iterators)
- Object-Oriented
  - Multiple inheritance / dynamic binding / reflection
- Model-Oriented
  - Associations / Compositions
  - Model are first class citizens, notion of model type
- Aspect-Oriented
  - Simple syntax for static introduction
  - Arbitrary complex aspect weaving as a framework
- Still “kernel” language
  - Seamless import of Java classes in Kermeta for GUI/IO etc.

Types & opérateurs usuels

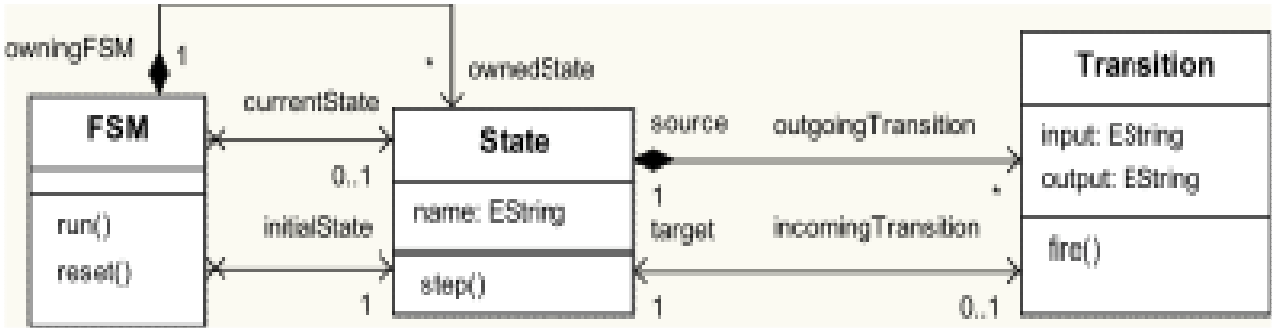
- Types scalaires très restreints
  - Integer, String, Boolean
- Opérateurs :
  - Affectation : := (naïve), ?= (cast)
  - Arithmétique : +,-,/,*
  - Comparaison : ==, !=, <, <=,>,>=
  - Logique : and, or, not
- Collections : fondées sur la définition d'OCL

Mot–clé	Classe générique	Unicité	Ordre
set	Set< T >	Oui	Non
oset	OrderedSet< T >	Oui	Oui
bag	Bag< T >	Non	Non
seq	Sequence< T >	Non	Oui

Définition de classes, opérations, méthodes

- Déclaration de classes à la Java (class C { })
  - Classes abstraites (abstract), généricité (class A<T>)
  - Héritage (inherits), multiple ou non
- Constructions : pas de constructeur ! (MyClass.new)
- Variables de classes : Attributs & Référence
  - attribute a: String => a est contenue par composition ()
  - reference r: String => r est référencée
  - self représente l'instance courante
  - Absence de visibilité : tout est public
- Méthode d'instance : Opérations & Méthodes
  - operation name(arg1: T): OutputType is do ... end
  - Redéfinition par héritage : operation → method
  - Variable locale : var tmp: String init String.new
  - Retour : pas de return ! On utilise la variable result
  - Pas de surcharge dans le langage (simplification)

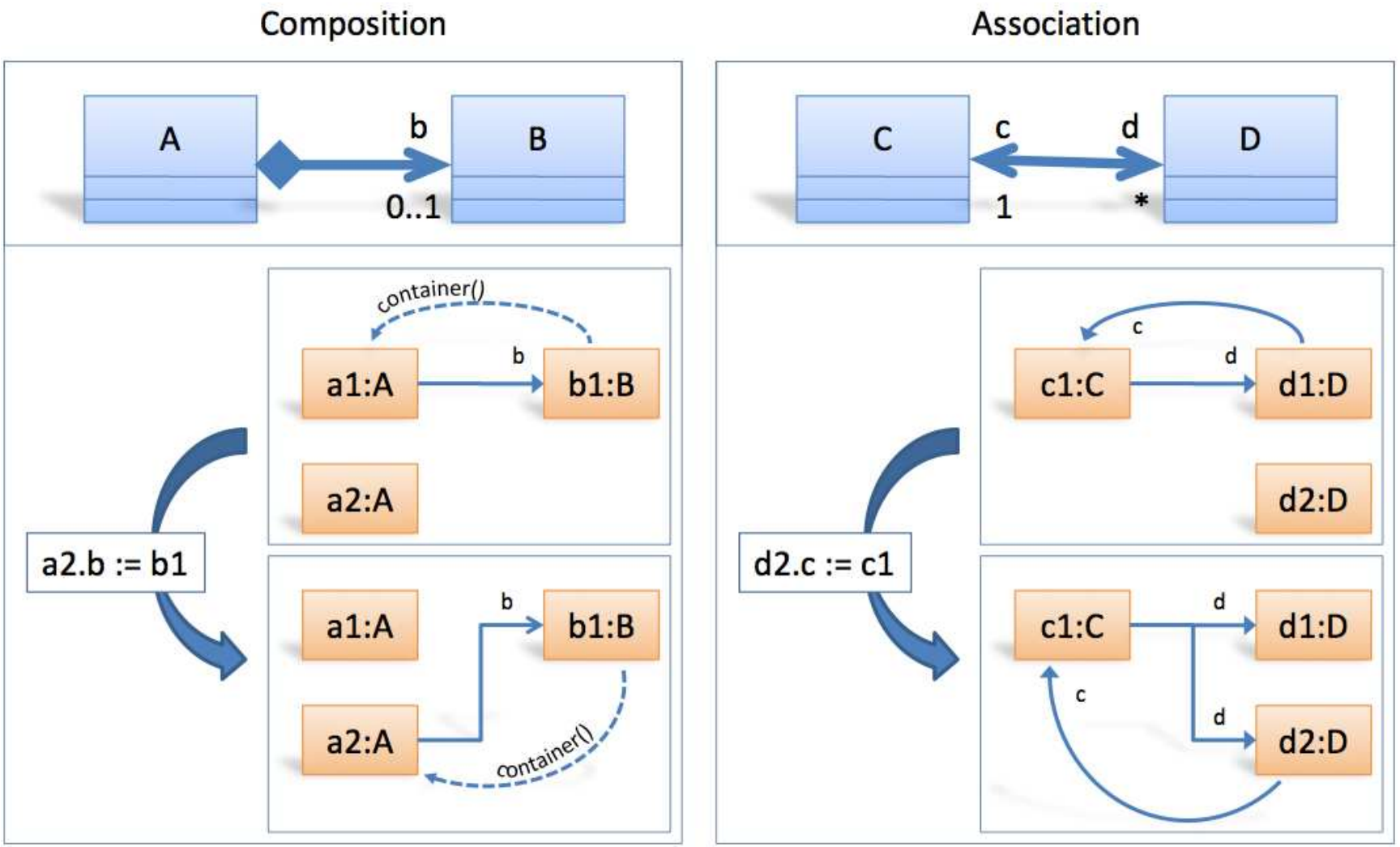
EMOF ⇔Kermeta




```
class FSM
{
attribute ownedState : State[0..*]#owningFSM
reference initialState : State[1..1]
reference currentState : State
operation run() : kermeta::standard::~Void is do
end
operation reset() : kermeta::standard::~Void is do
end
}
```

```
class State{
   reference owningFSM : FSM[1..1]#ownedState
   attribute name : String
   attribute outgoingTransition : Transition[0..*]#source
   reference incomingTransition : Transition#target
   operation step(c : String) : kermeta::standard::~Void is do
    end
}
class Transition{
   reference source : State[1..1]#outgoingTransition
   reference target : State[1..1]#incomingTransition
   attribute input : String
   attribute output : String
   operation fire() : String is do
   end
}
```

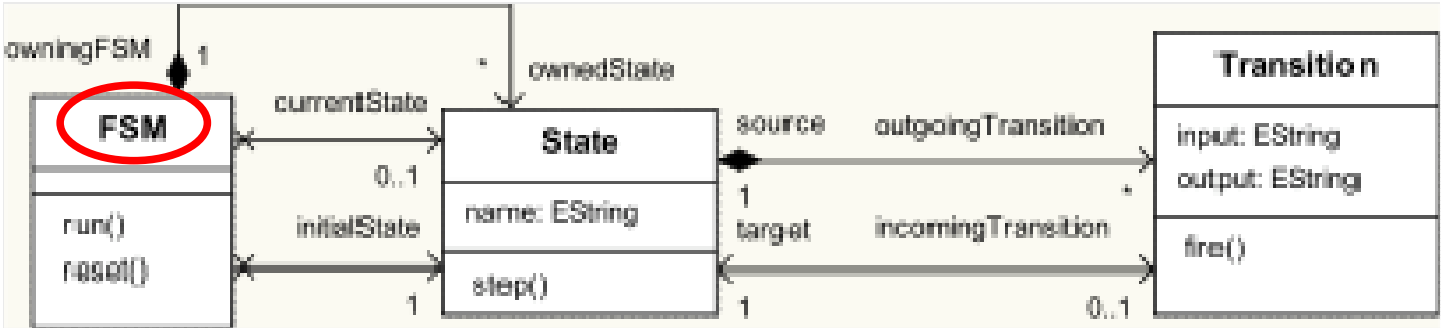
Assignment semantics



Fermetures & λ-fonctions pour l'itération

- Effectuer une action ∀ e ∈ C each
  - f.each{ n | stdio.write(n.toString + " ") }
- Vérifier une condition ∀ e ∈ C forAll
  - var b: Boolean init f.forAll{ n | n < 250 }
- Sélection d'un sous-ensemble (filter) select
  - var f2: Sequence<Integer> init f.select{n | n < 100}
- Exclusion d'un sous-ensemble reject
  - var f3: Sequence<Integer> init f.reject{n | n < 100}
- Mapping de fonction collect
  - var f4: Sequence<Integer> init fib.collect{n | n + 1}
- Détection d'un élément detect
  - var x: Integer init fib.detect{n | n > 47}
- Existence d'un élément exists
  - var b2: Boolean init fib.exists{n | n > 47 }

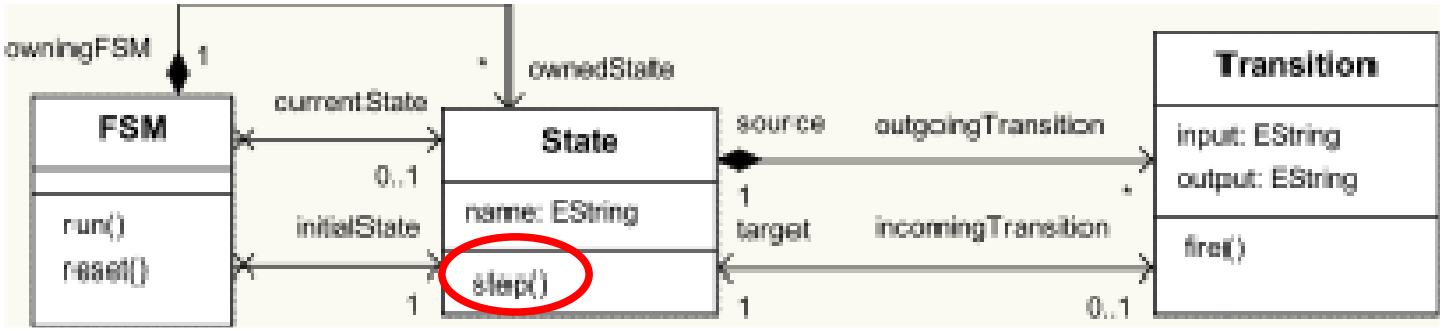
Example



operation run() : **Void**

```
from var str : String
until str == "exit"
loop
        stdio.writeln("current state is " + currentState.name)
        str := stdio.read("Enter an input string or 'exit'
                                                to exit simulation : ")
        stdio.writeln(str)
        if str != "exit" then
                do
                        stdio.writeln("Output string : " + currentState.step(str))
                rescue (ex : FSMException)
                        stdio.writeln("ERROR : " + ex.toString)
                end
        end
end
stdio.writeln("* END OF SIMULATION *")
```

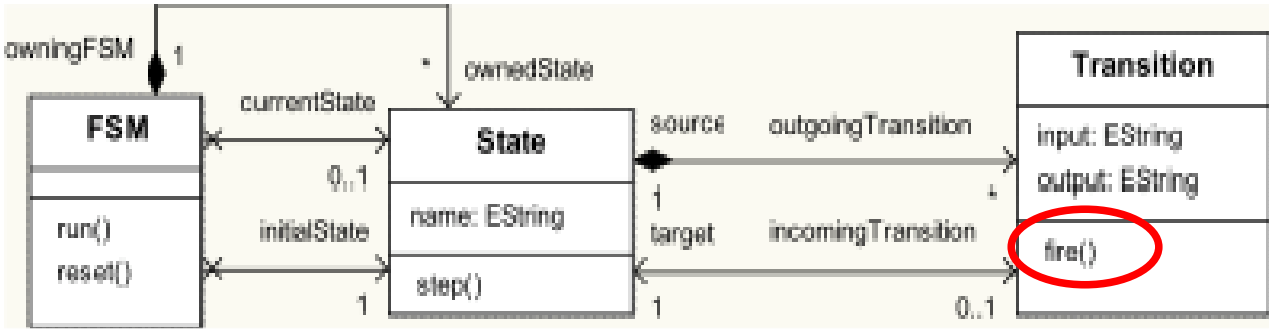
Example



operation step(c : String) : String

```
// Get the valid transitions
var validTransitions : Collection<Transition>
validTransitions :=    outgoingTransition.select { t |
          t.input.equals(c)
}
// Check if there is one and only one valid transition
if validTransitions.empty then raise NoTransition.new end
if validTransitions.size > 1 then
          raise NonDeterminism.new
end
// fire the transition
result := validTransitions.one.fire
```

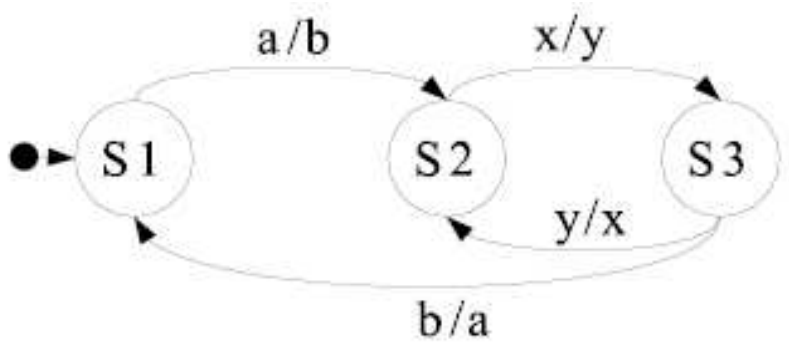
Example

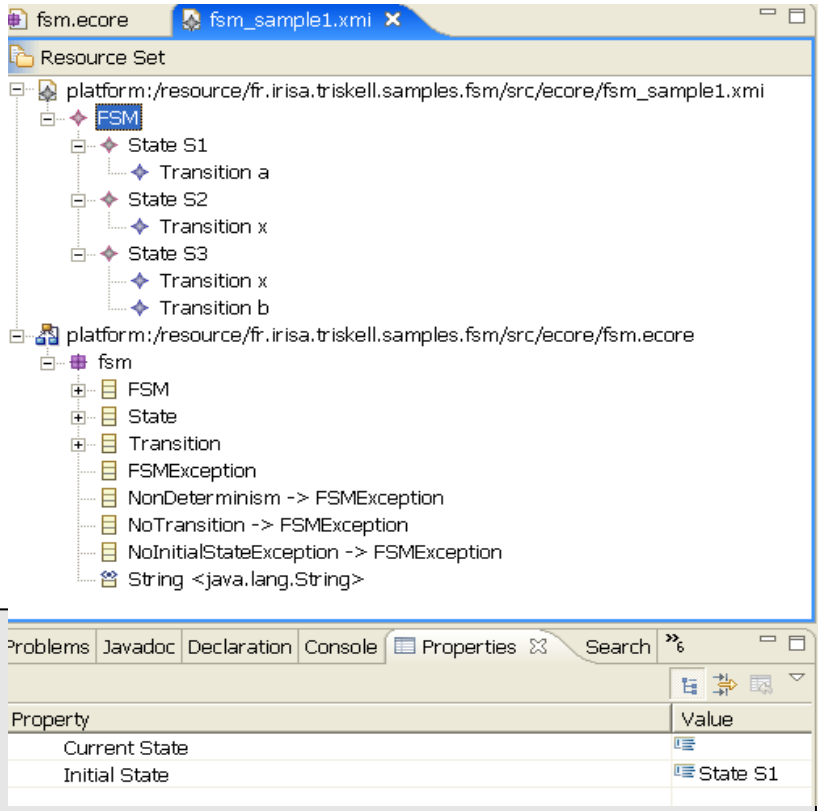


operation fire() : String

```
source.owningFSM.currentState := target
result := output
```

Example






```
/**
 * Load a sample FSM from a xmi2 file
 */
operation loadFSM() : FSM is do
  var repository : EMFRepository init EMFRepository.new
  var resource : EMFResource
  resource ?= repository.createResource("../models/fsm_sample1.xmi", "../metamodels/fsm.ecore")
  resource.load


  // Load the fsm (we get the main instance)
  result ?= resource.instances.one
end
```

Current Status

- Latest version (1.4.1)

  Parser, type checker, interpreter, debugger, Java compiler

  Eclipse plug-in: Textual Editor, Browser, Launcher

  EMF Ecore metamodel Import / Export

  EMF model Import / Export

  Seamless import of Java classes in Kermeta

  Constraints (Kermeta or OCL)

  Graphical Editor (generated with Topcased)

  Documentation and Examples

Smoothly interoperates with Eclipse/EMF
Open Source

- **Download it now!**

A statically typed object-oriented executable meta-language

- Home page
  - http://www.kermeta.org
- Development page
  - http://kermeta.gforge.inria.fr/

Outline

- **Introduction to MDE**
- **Defining the Operational Semantics with Kermeta**
- **The Logo Example (short reminder)**
- **Building a Simulator for Logo**
- **Building a Compiler for Logo**
- **Wrap-up and Conclusion**

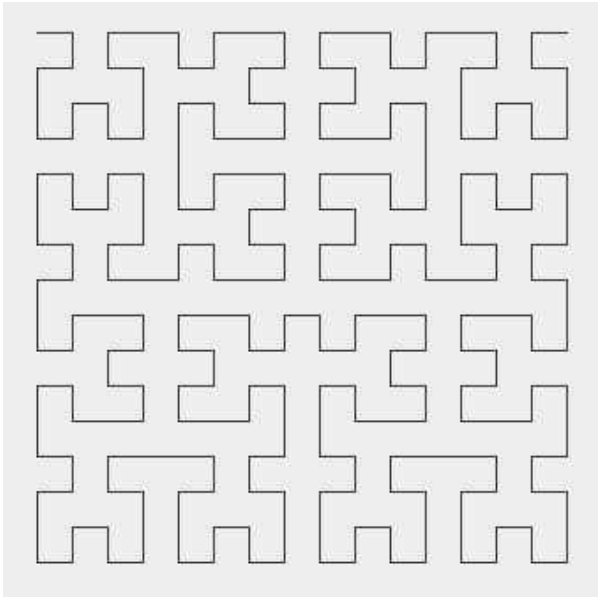
DIY with LOGO programs

- Consider LOGO programs of the form:
  - repeat 3 [pendown forward 3 penup forward 4]

  - to square :width
  - repeat 4 [forward :width right 90]
  - end
  - pendown square 10

Fractals in LOGO

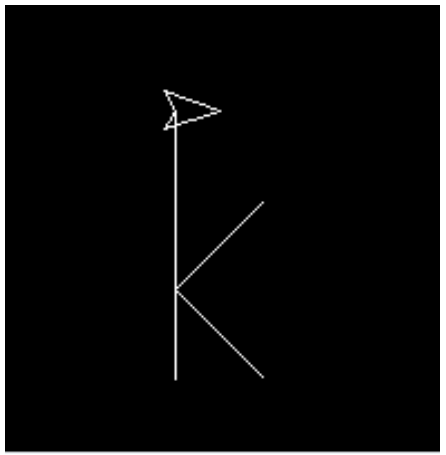
```
; lefthilbert
to lefthilbert :level :size
   if :level != 0 [
left 90
righthilbert :level-1 :size
forward :size
right 90
lefthilbert :level-1 :size
forward :size
lefthilbert :level-1 :size
right 90
forward :size
righthilbert :level-1 :size
left 90
  ]
end
```

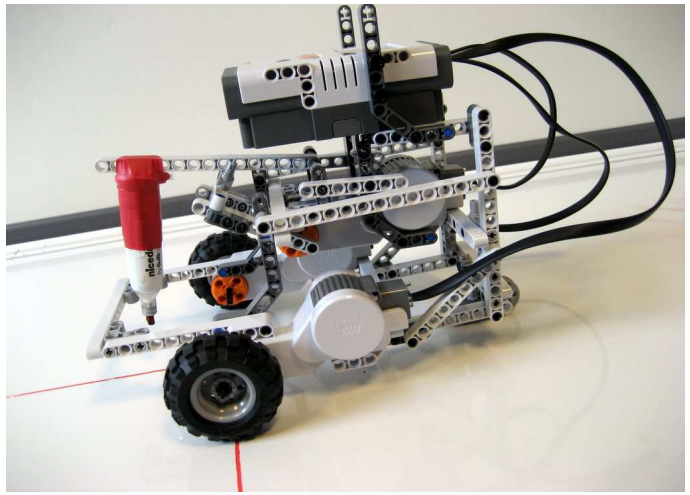


```
; righthilbert
to righthilbert :level :size
    if :level != 0 [
right 90
lefthilbert :level-1 :size
forward :size
left 90
righthilbert level-1 :size
forward :size
righthilbert :level-1 :size
left 90
forward :size
lefthilbert :level-1 :size
right 90
  ]
end
```

Case Study: Building a Programming Environment for Logo

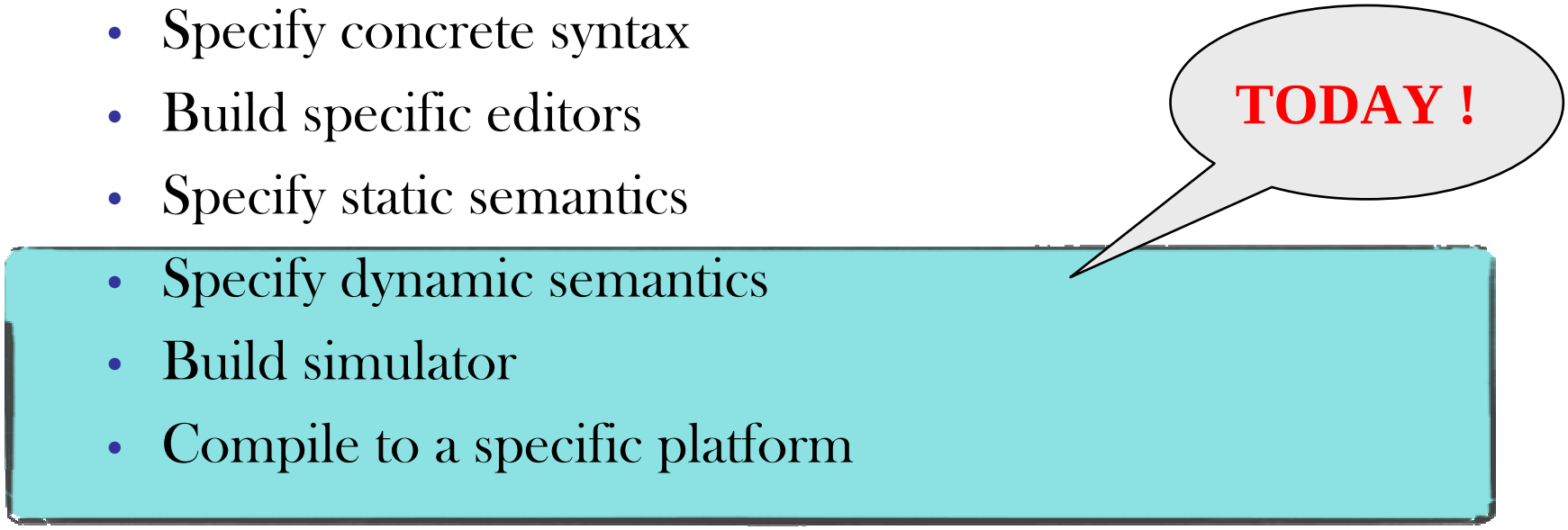
- Featuring
  - Edition in Eclipse
  - On screen simulation
  - Compilation for a Lego Mindstorms robot





Model Driven Language Engineering : the Process

- Specify abstract syntax
- Specify concrete syntax
- Build specific editors
- Specify static semantics
- Specify dynamic semantics
- Build simulator
- Compile to a specific platform



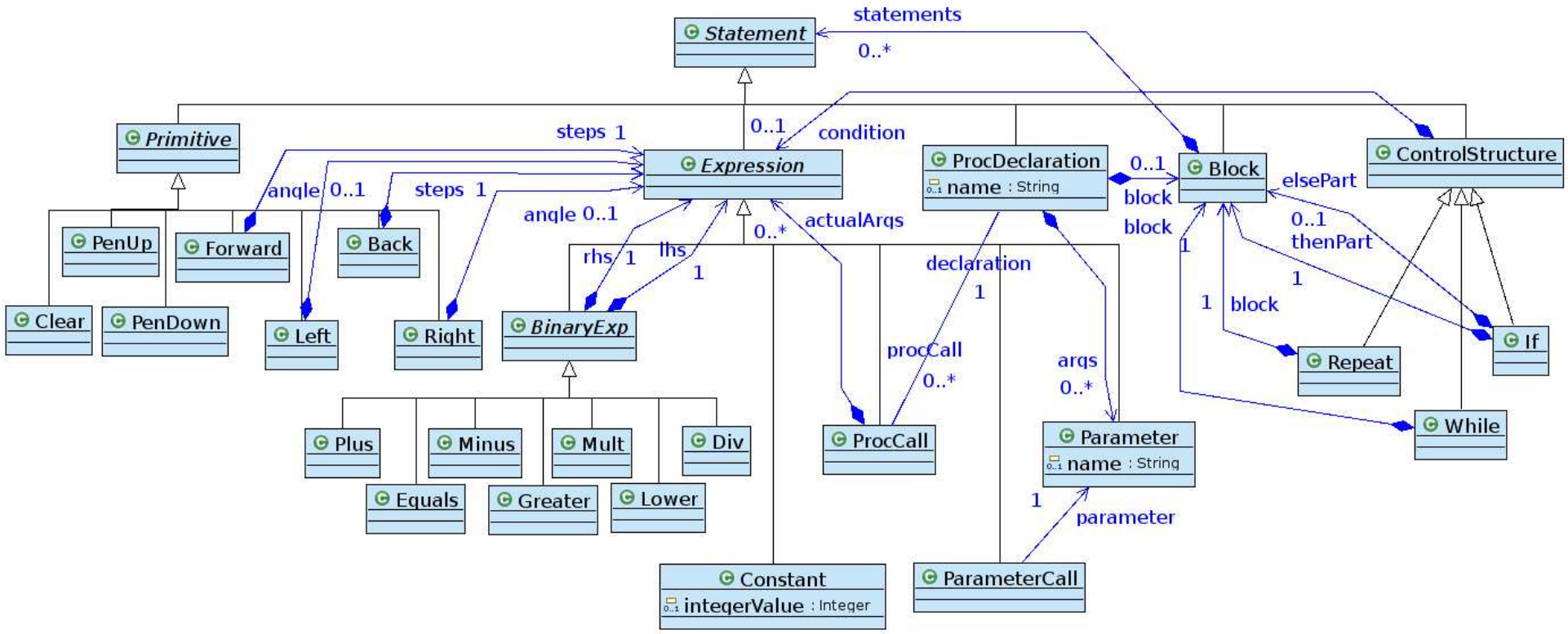
Meta-Modeling LOGO programs

- Let's build a meta-model for LOGO
  - Concentrate on the abstract syntax
  - Look for concepts: instructions, expressions...
  - Find relationships between these concepts
    - It's like UML modeling !

■Defined as an Ecore model

– Using EMF tools and editors

LOGO metamodel



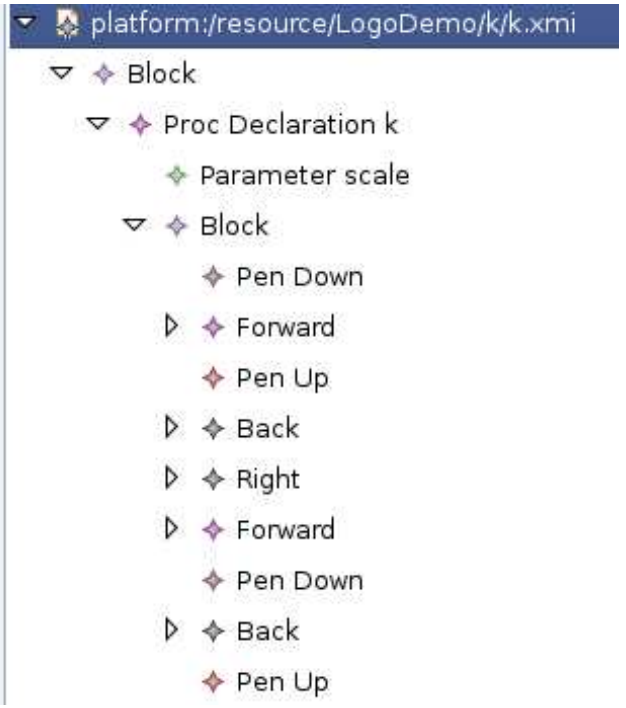
ASMLogo.ecore

Concrete syntax

- Any regular EMF based tools
- Textual using Sintaks, TMF, XText, EMFText...
- Graphical using GMF or TopCased

logo.sts

```
TO k :scale
     PENDOWN
     FORWARD *(30, :scale)
     PENUP
     BACK *(10, :scale)
     RIGHT 45
     FORWARD *(14, :scale)
     PENDOWN
     BACK *(14, :scale)
     PENUP
     RIGHT 90
     FORWARD *(14, :scale)
     PENDOWN
     BACK *(14, :scale)
     PENUP
     RIGHT 45
     FORWARD *(20, :scale)
     LEFT 180
END
```



Outline

- **Introduction to MDE**
- **Defining the Operational Semantics with Kermeta**
- **The Logo Example (short reminder)**
- **Building a Simulator for Logo**
- **Building a Compiler for Logo**
- **Wrap-up and Conclusion**

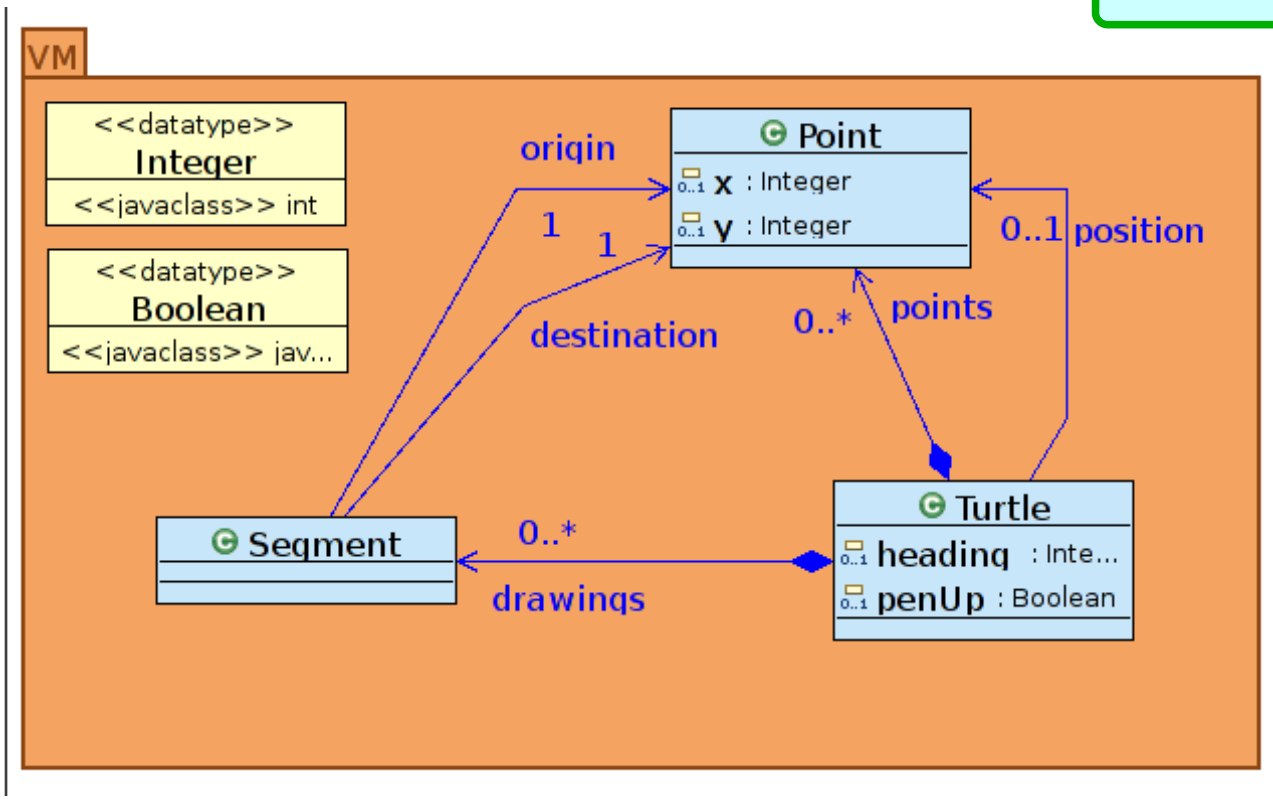
Operational Semantics for LOGO

- Expressed as a mapping from a meta-model to a virtual machine (VM)

- LOGO VM ?
  - Concept of Turtle, Lines, points...
  - Let's Model it !
  - (Defined as an Ecore meta-model)

Virtual Machine - Model

VMLogo.ecore



- Defined as an Ecore meta-model

Virtual Machine - Semantics

LogoVMSemantics.kmt

```
require "VMLogo.ecore"
require "TurtleGUI.kmt"

aspect class Point {
  method toString() : String is do
    result := "[" + x.toString + "," + y.toString + "]"
  end
}

aspect class Turtle {
  operation setPenUp(b : Boolean) is do
    penUp := b
  end
  operation rotate(angle : Integer)  is do
    heading := (heading + angle).mod(360)
  end
}
```

Map Instructions to VM Actions

- Weave an interpretation aspect into the meta-model
  - add an *eval()* method into each class of the LOGO MM

```
aspect class PenUp {
   eval (ctx: Context) {
      ctx.getTurtle().setPenUp(true)
}
…
aspect class Clear {
   eval (ctx: Context) {
      ctx.getTurtle().reset()
}
```

Handling control structures

- Block
- Conditional
- Repeat
- While

Operational semantics

LogoDynSemantics.kmt

```
require "ASMLogo.ecore"
require "LogoVMSemantics.kmt"

aspect class If {
  operation eval(context : Context) : Integer is do
    if condition.eval(context) != 0 then
      result := thenPart.eval(context)
    else result := elsePart.eval(context)
    end
  end
}

aspect class Right {
  operation eval(context : Context) : Integer is do
    context.turtle.rotate(angle.eval(context))
  end
}
```

Handling function calls

- Use a stack frame
  - Owned in the Context

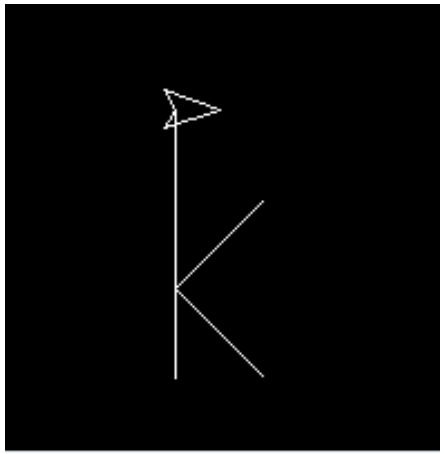
Getting an Interpreter

- Glue that is needed to load models
  - ie LOGO programs
- Vizualize the result
  - Print traces as text
  - Put an observer on the LOGO VM to graphically display the resulting figure

Simulator

- Execute the operational semantics

```
TO k :scale
    PENDOWN
    FORWARD *(30, :scale)
    PENUP
    BACK *(10, :scale)
    RIGHT 45
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 90
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 45
    FORWARD *(20, :scale)
    LEFT 180
END

CLEAR
$k(4)
```



Problems | Javadoc | Declaration | Console | Pro

KM Logo Console

```
Launching logo interpreter on file : /home/
Tortue trace vers [0,120]
Tortue se deplace en [0,80]
Tortue se deplace en [39,119]
Tortue trace vers [0,80]
Tortue se deplace en [39,41]
Tortue trace vers [0,80]
Tortue se deplace en [0,0]
Execution terminated successfully.
```

Outline

- **Introduction to MDE**
- **Defining the Operational Semantics with Kermeta**
- **The Logo Example (short reminder)**
- **Building a Simulator for Logo**
- **Building a Compiler for Logo**
- **Wrap-up and Conclusion**

Implementing a model-driven compiler

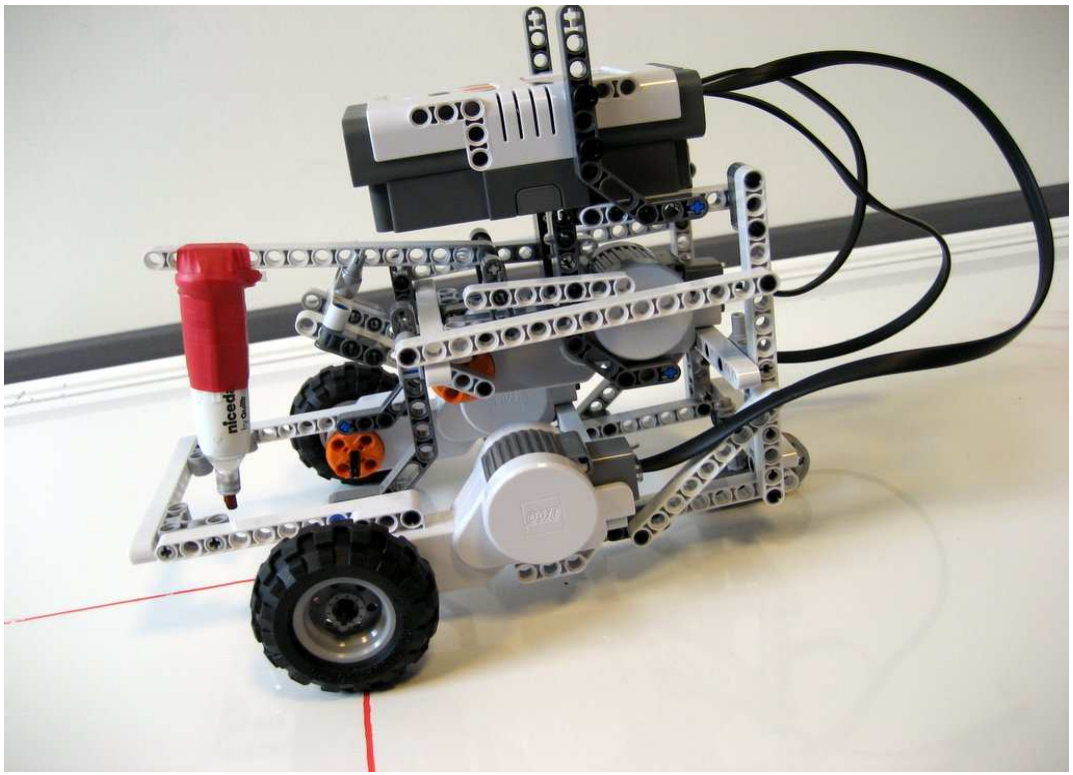
- Map a LOGO program to Lego Mindstroms
  - The LOGO program is like a PIM
  - The target program is a PSM
  - => model transformation

- Kermeta to weave a « compilation » aspect into the logo meta-model

```
aspect class PenUp {
    compile (ctx: Context) {

}
…
aspect class Clear {
}
```

Specific platform

- Lego Mindstorms Turtle Robot
  - Two motors for wheels
  - One motor to control the pen



Model-to-Text vs. Model-to-Model

- Model-to-Text Transformations
  - For generating: code, xml, html, doc.
  - Should be limited to syntactic level transcoding

- Model-to-Model Transformations
  - To handle more complex, semantic driven transformations

Model-to-Text Approaches

- For generating: code, xml, html, doc.
  - Visitor-Based Approaches:
    - Some visitor mechanisms to traverse the internal representation of a model and write code to a text stream
    - Iterators, Write ()
  - Template-Based Approaches
    - A template consists of the target text containing slices of meta-code to access information from the source and to perform text selection and iterative expansion
    - The structure of a template resembles closely the text to be generated
    - Textual templates are independent of the target language and simplify the generation of any textual artefacts

Classification of Model-to-Model Transformation Techniques

1. General purpose programming languages
   Java/C#...

2. Generic transformation tools
   Graph transformations, XSLT...

3. CASE tools scripting languages
   Objecteering, Rose...

4. Dedicated model transformation tools
   OMG QVT style

5. Meta-modeling tools
   Metacase, Xactium, Kermeta...

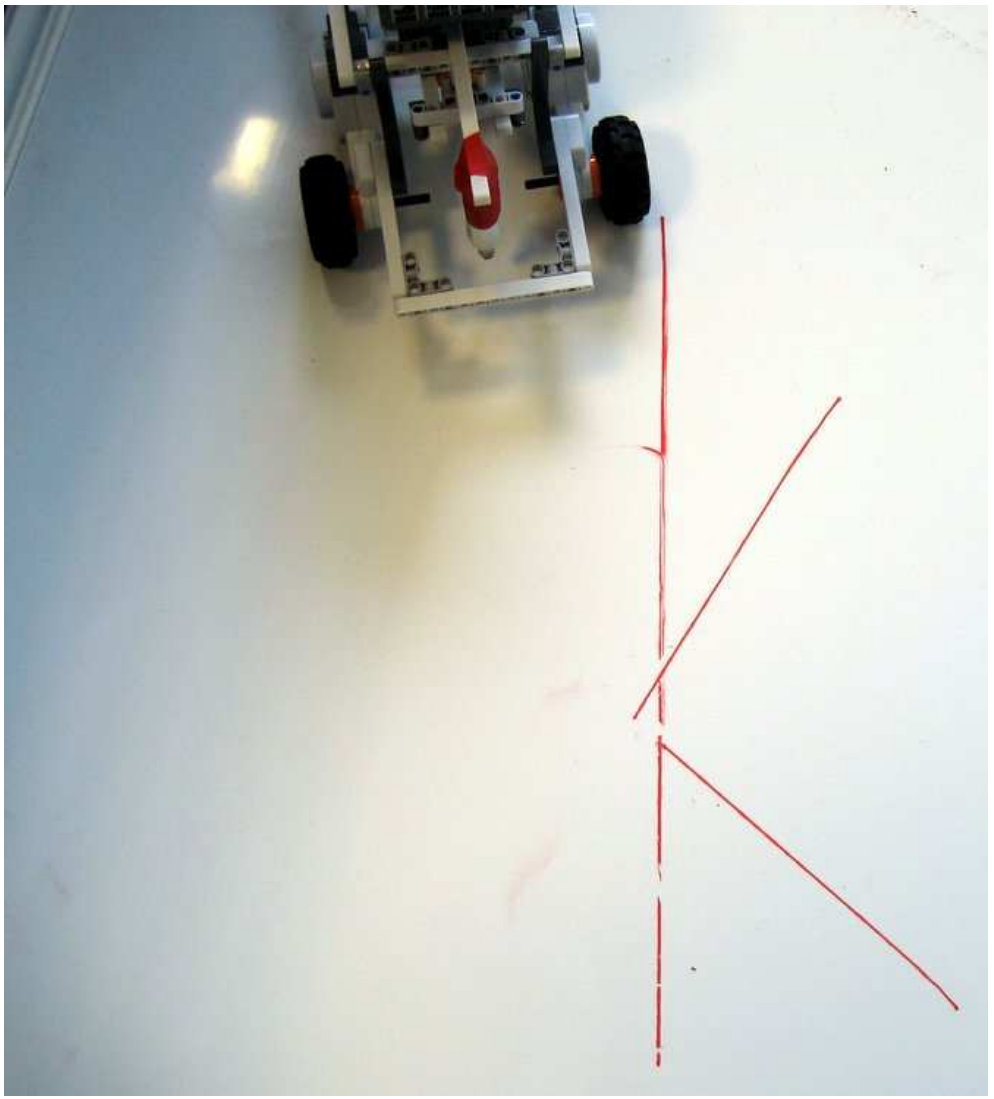
Logo to NXC Compiler

- Step 1 – Model-to-Model transformation

- Step 2 – Code generation with template

Execution

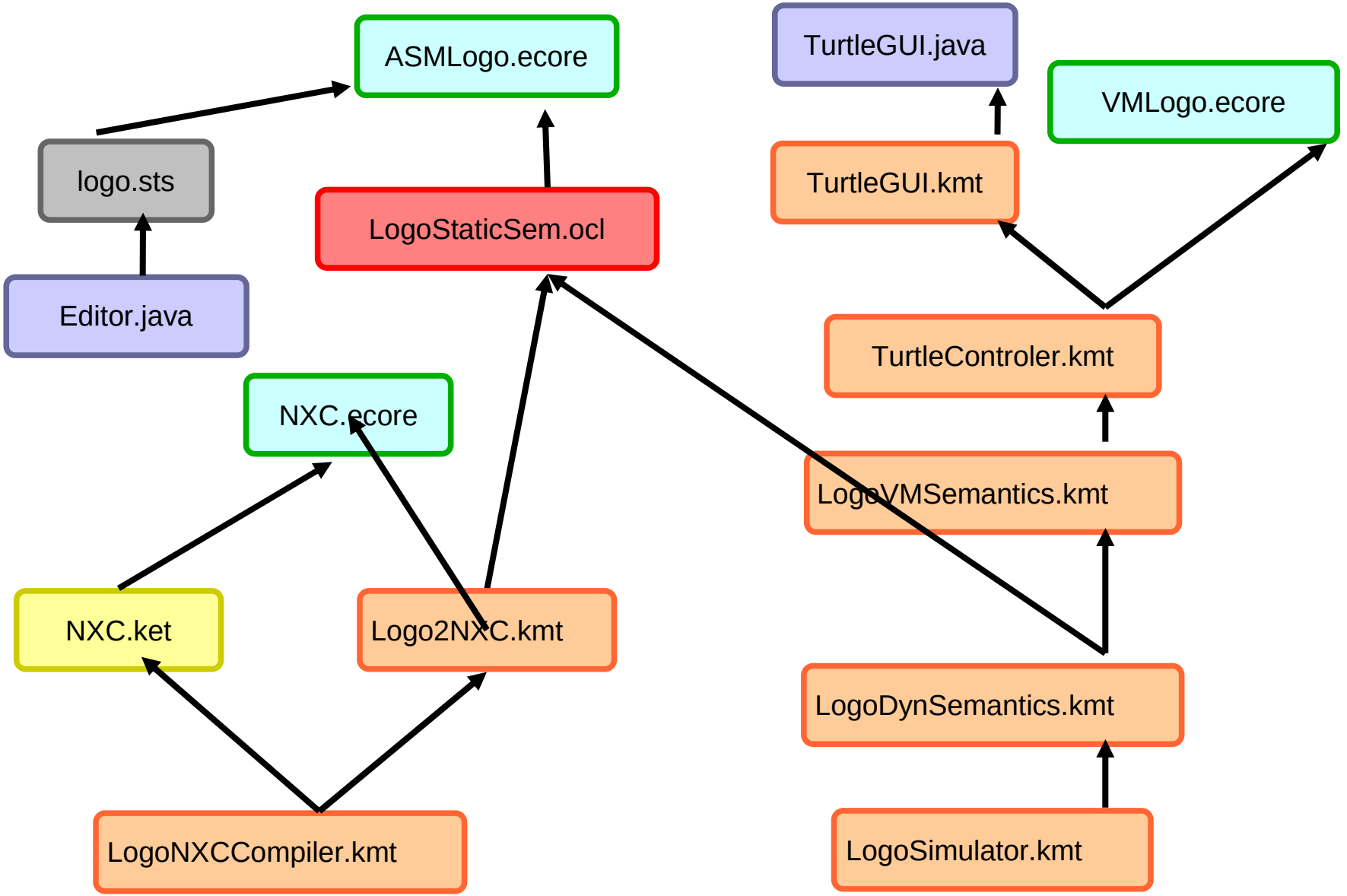
```
TO k :scale
     PENDOWN
     FORWARD *(30, :scale)
     PENUP
     BACK *(10, :scale)
     RIGHT 45
     FORWARD *(14, :scale)
     PENDOWN
     BACK *(14, :scale)
     PENUP
     RIGHT 90
     FORWARD *(14, :scale)
     PENDOWN
     BACK *(14, :scale)
     PENUP
     RIGHT 45
     FORWARD *(20, :scale)
     LEFT 180
END

CLEAR
$k(4)
```



Outline

- **Introduction to MDE**
- **Defining the Operational Semantics with Kermeta**
- **The Logo Example (short reminder)**
- **Building a Simulator for Logo**
- **Building a Compiler for Logo**
- **Wrap-up and Conclusion**

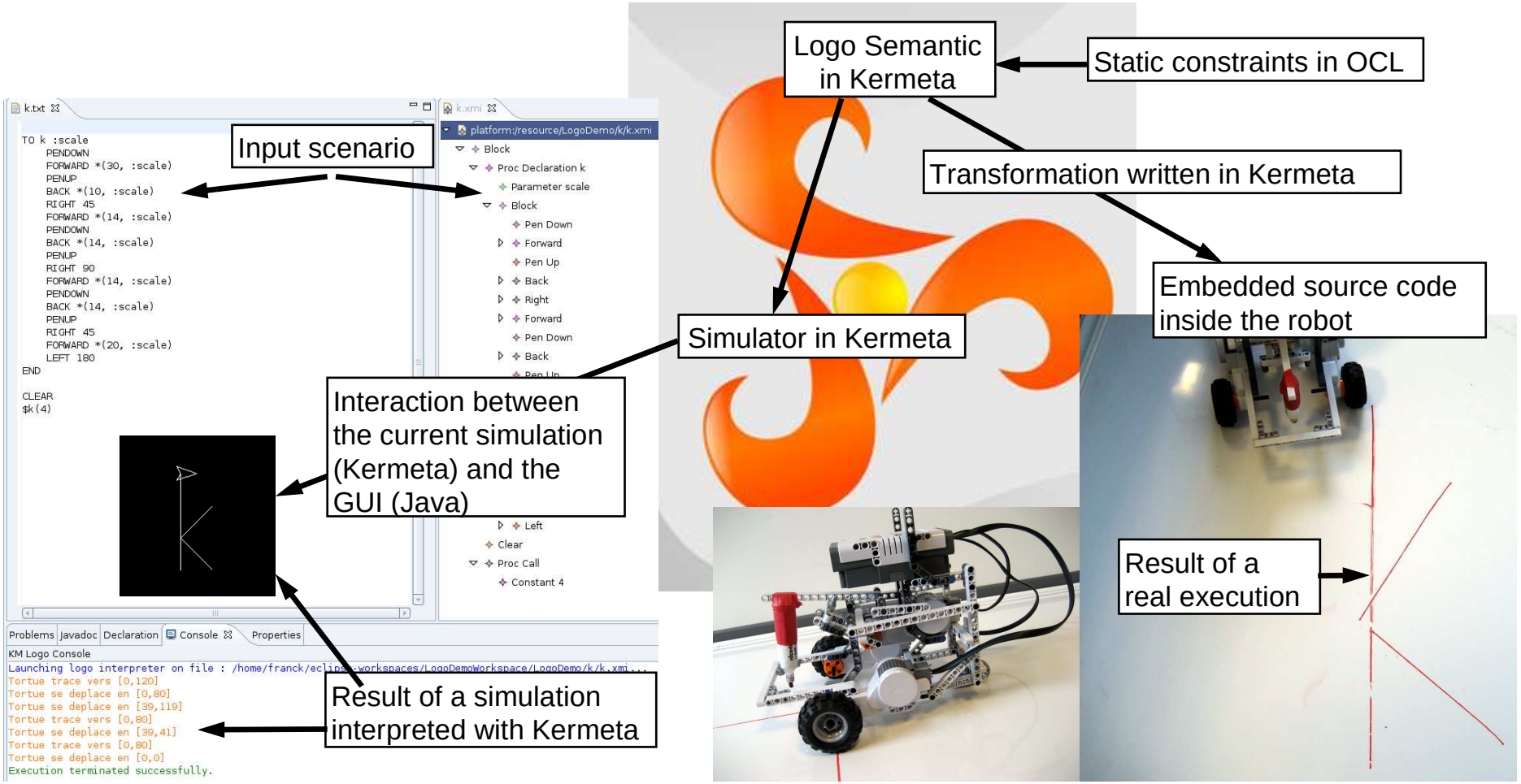
Logo Summary (1)



Logo Summary (2)

- Integrate all aspects coherently
  - syntax / semantics / tools
- Use appropriate languages
  - MOF for abstract syntax
  - OCL for static semantics
  - Kermeta for dynamic semantics
  - Java for simulation GUI
  - ...
- Keep separation between concerns
  - For maintainability and evolutions

From LOGO to Mindstorms



Kermeta in real projects

- Artist2, *the European network of excellence on real-time embedded systems*
- UsineLogicielle, *a System@tic project where Kermeta based operational semantic is associated to functional requirement for test synthesis purposes.*
- Speeds, *a European FP6 project for aspect-oriented metamodeling of avionics and automotive systems, including operational semantics aspects*
- OpenEmbedd, *A French project building a MDE platform for realtime system.*
- Mopcom, *a French project applying MDE to hardware for generating SOC and introduce dynamic reconfigurability to them.*
- Topcased, *a consortium that aims to build modelers for avionics and system engineering*
- DiVA, *a European FP7 STREP project on handling Dynamic variability in complex, adaptive systems*
- *Etc.*

Conclusion and Wrap-up

- Kermeta is an open-source initiative
  - Started January 2005
  - More than 10 active developers
- Feel free to use
  - Start with a meta-model in EMF
    - Get XML an reflective editor for free
  - Weave in static semantics in OCL
  - Weave in an interpreter,
    - connect to a simulation platform
  - Weave in one or more compilers
  - Finaly care for concrete syntax issues
- Feel free to contribute!
  - www.kermeta.org

Do It Yourself !

- Source code of the Logo demo:
  - https://gforge.inria.fr/scm/viewvc.php/trunk/kmlogo_projects/?root=kermeta
- Kermeta (http://www.kermeta.org/):
  - reference documentation: http://www.kermeta.org/documents
  - formation supports: https://gforge.inria.fr/scm/viewvc.php/integration/training_projects/?root=openembedd
- More information:
  - Eclipse Modeling (EMF, GMF...): http://www.eclipse.org/modeling/
  - OMG (UML, OCL, MOF...): http://www.omg.org/