

Model-Driven Language Engineering

Jean-Marc Jézéquel

e-mail : jezequel@irisa.fr
<http://www.irisa.fr/prive/jezequel>

Franck Fleurey

e-mail : Franck.Fleurey@sintef.no
<http://www.fleurey.com>



with the support of



Outline



- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up

Once upon a time... software development looked simple

- From the object as the *only* one concept
 - As e.g. in Smalltalk
- To a multitude of concepts

Middleware (middle war)

It's difficult -- in fact, next to impossible -- for a large enterprise to standardize on a single middleware platform. (R. Spley)

© J.-M. Jézéquel

Why modeling: master complexity

- Modeling, in the broadest sense, is the *cost-effective use of something in place of something else for some cognitive purpose*. It allows us to use something that is *simpler*, *safer* or *cheaper* than reality instead of reality for some purpose.
- A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.

Jeff Rothenberg.

© J.-M. Jézéquel, 2008

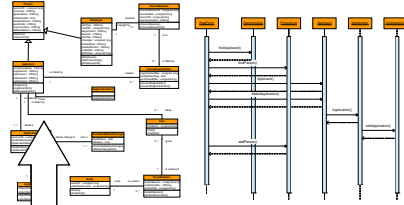
Modeling in Science & Engineering

- A Model is a *simplified* representation of an *aspect* of the World for a specific *purpose*

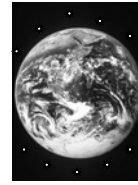
Specificity of Engineering:
Model something not yet
existing (in order to build it)

M_1
(modeling
space)

Is represented by



M_0
(the world)

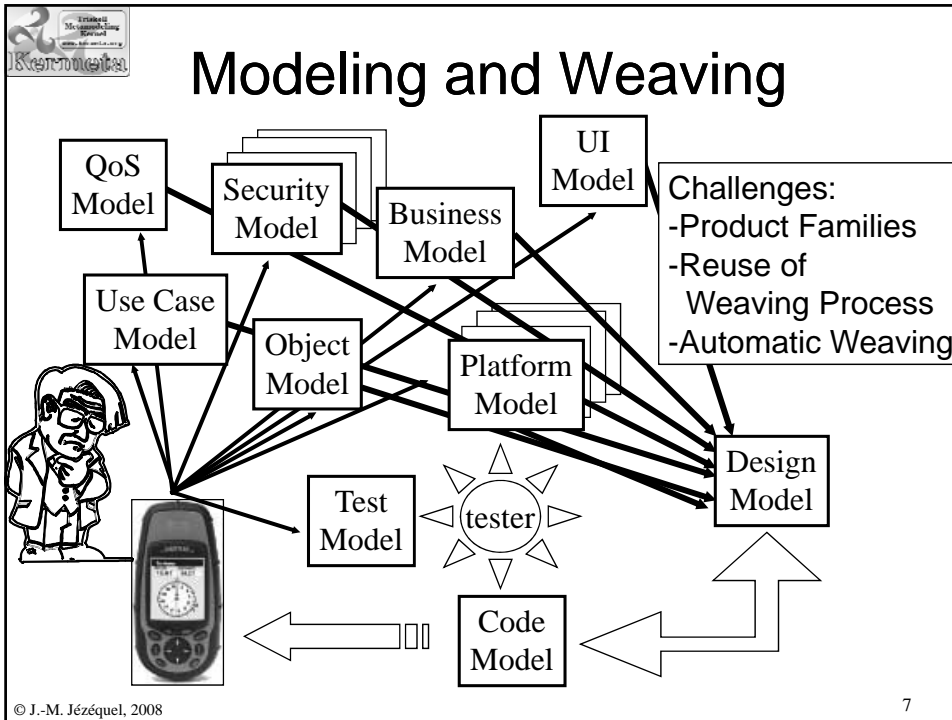


Model and Reality in Software

- Sun Tse: *Do not take the map for the reality*
- Magritte



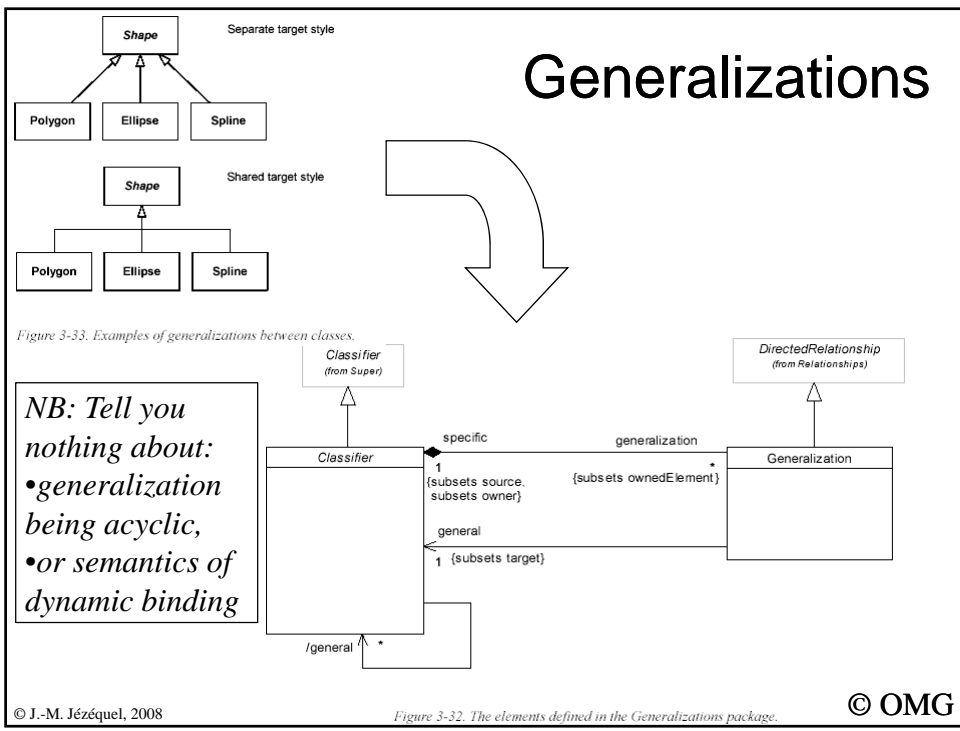
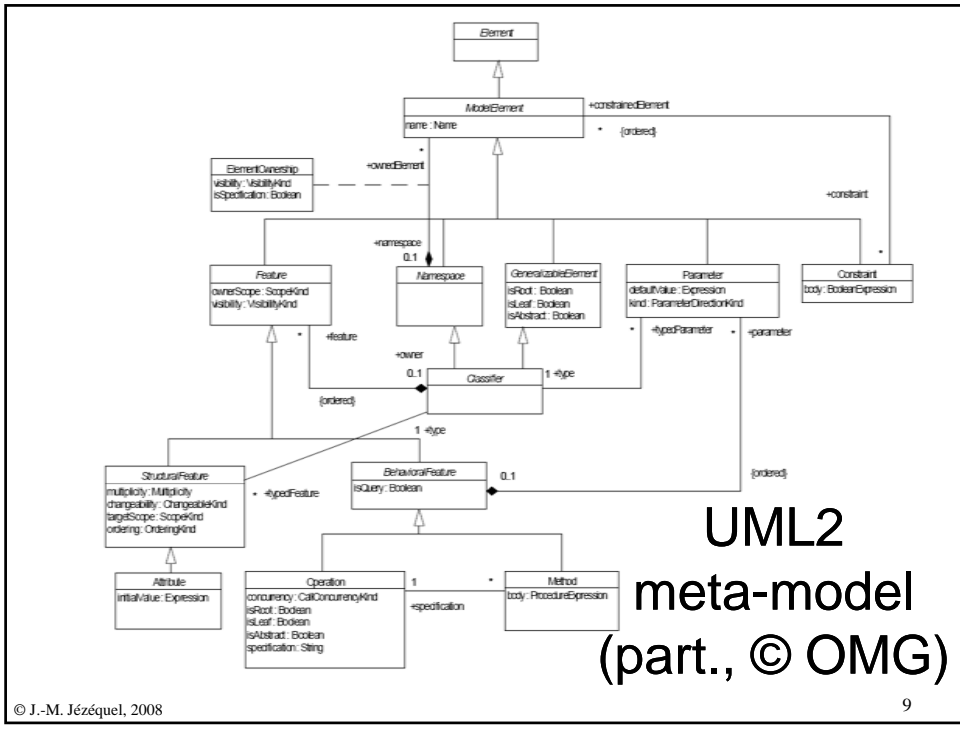
- Software Models: from contemplative to productive



Assigning Meaning to Models

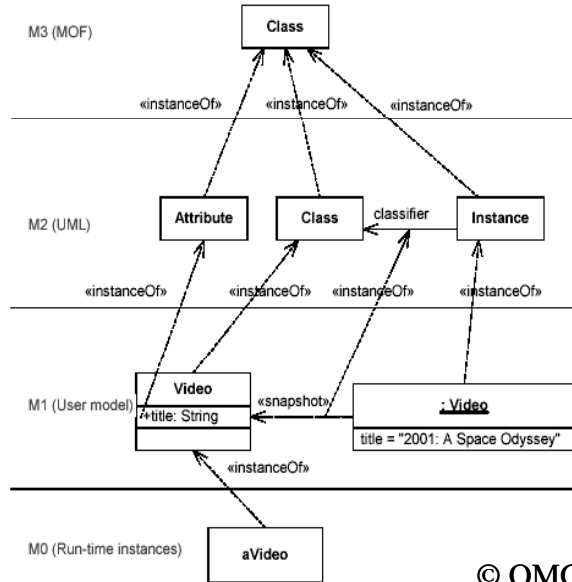
- If a model *is no longer* just
 - fancy pictures to decorate your room
 - a graphical syntax for C++/Java/C#/Eiffel...
- Then tools must be able to manipulate models
 - Let's make a model of what a model is!
 - => **meta-modeling**
 - » & meta-meta-modeling.
 - » Use Meta-Object Facility (MOF) to avoid infinite Meta-recursion

© J.-M. Jézéquel, 2008





The 4 layers in practice



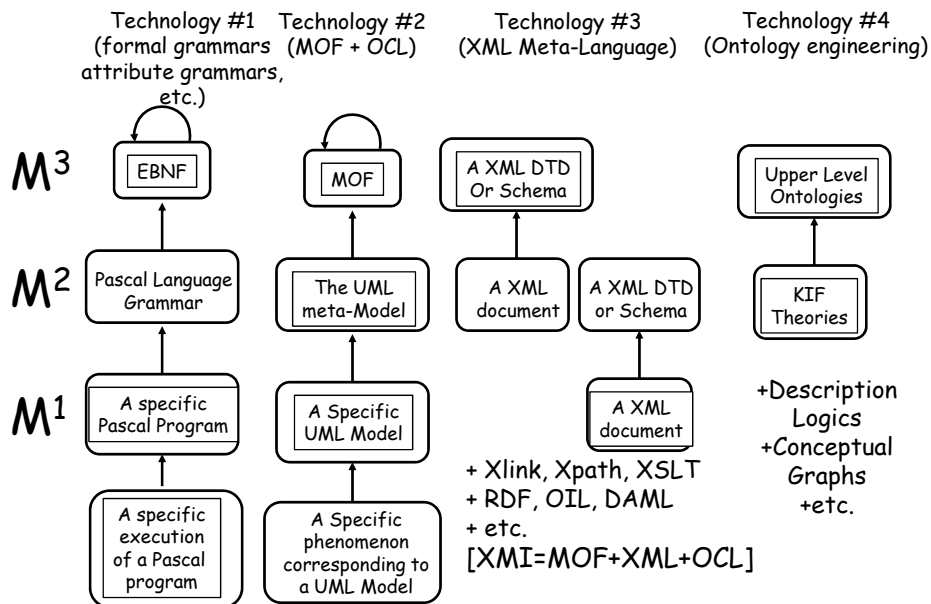
© J.-M. Jézéquel, 2008

© OMG

11



Comparing Abstract Syntax Systems



© J.-M. Jézéquel, 2008

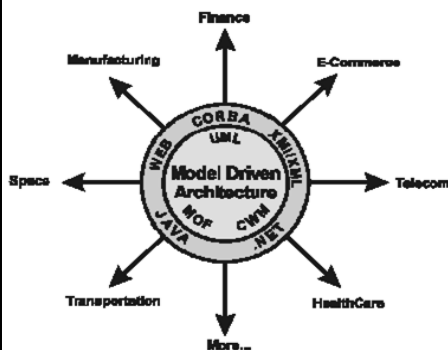
(From J. Bézivin)

12



MDA: the OMG vision

"OMG is in the ideal position to provide the model-based standards that are necessary to extend integration beyond the middleware approach... Now is the time to put this plan into effect. Now is the time for the Model Driven Architecture."

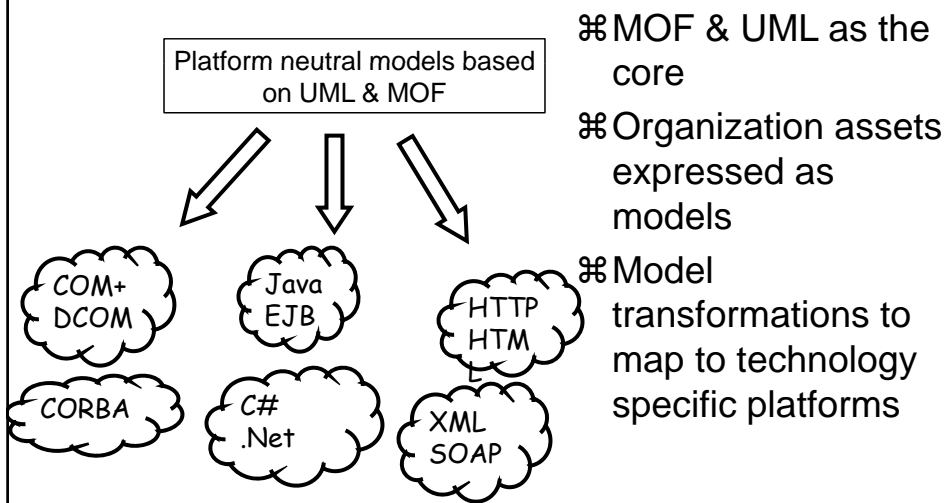


*Richard Soley & OMG staff,
MDA Whitepaper Draft 3.2
November 27, 2000*

13



Mappings to multiple and evolving platforms



⌘ MOF & UML as the core

⌘ Organization assets expressed as models

⌘ Model transformations to map to technology specific platforms

© J.-M. Jézéquel, 2008

14



The core idea of MDA: PIMs & PSMs

- MDA models
 - PIM: Platform Independent Model
 - » Business Model of a system abstracting away the deployment details of a system
 - » Example: the UML model of the GPS system
 - PSM: Platform Specific Model
 - » Operational model including platform specific aspects
 - » Example: the UML model of the GPS system on .NET
 - Possibly expressed with a UML profile (.NET profile for UML)
 - Not so clear about platform models
 - » Reusable model at various levels of abstraction
 - CCM, C#, EJB, EDOC, ...



Model Driven Engineering : Summary



- Modeling to master complexity
 - Multi-dimensional and aspect oriented by definition
- Models: from contemplative to productive
 - Meta-modeling tools, meta-models used to define languages
- Model Driven Engineering
 - Weaving aspects into a design model
 - » E.g. Platform Specificities
- Model Driven Architecture (PIM / PSM): just a special case of Aspect Oriented Design
- Related: Generative Prog, Software Factories



Outline



- Introduction to Model Driven Engineering

- Designing Meta-models: the LOGO example

- Static Semantics with OCL

- Operational Semantics with Kermeta

- Building a Compiler: Model transformations

- Conclusion and Wrap-up



Meta-Models as Shared Knowledge

- Definition of an Abstract Syntax in E-MOF

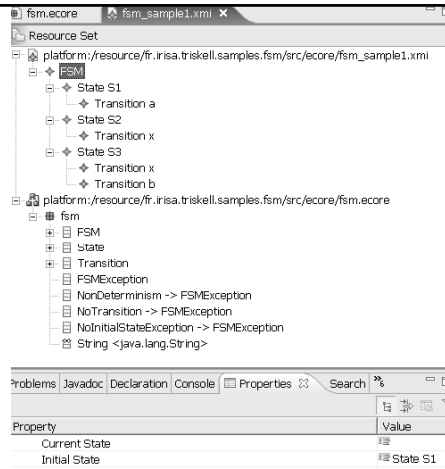
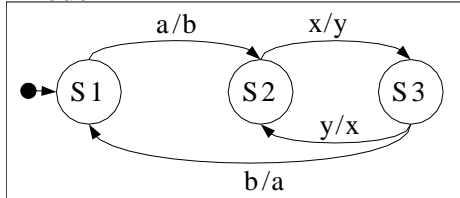
- Repository of models with EMF
- Reflexive Editor in Eclipse
- JMI for accessing models from Java
- XML serialization for model exchanges

- Applied in more and more projects

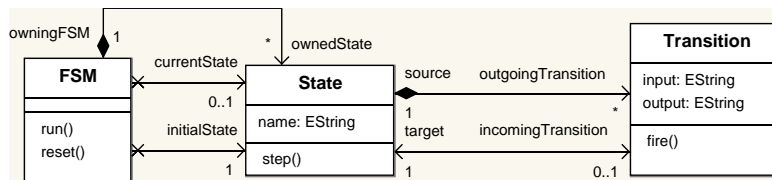
- SPEEDS, OpenEmbedd, DiVA...

Example with StateMachines

Model



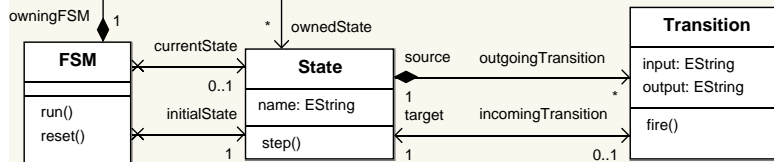
Meta-Model



© J.-M. Jézéquel, 2008

19

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.oc1"
// Method bodies (Dynamic semantics)
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 {...}
}
  
```

© J.-M. Jézéquel,

20



DIY with LOGO programs

- Consider LOGO programs of the form:

```
repeat 3 [ pendown forward 3 penup forward 4 ]
```

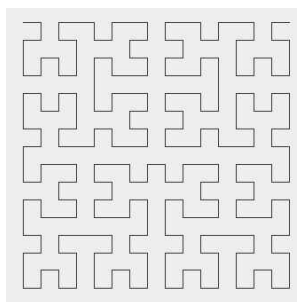


```
to square :width
  repeat 4 [ forward :width right ]
end
pendown square 10 *10
```



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
```

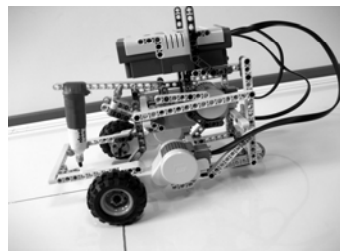
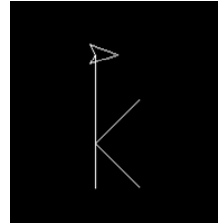


```
; righthilbert
to righthilbert :level :size
  if :level != 0 [
    right 90
    lefthilbert :level-1 :size
    forward :size
    left 90
    righthilbert :level-1 :size
    forward :size
    righthilbert :level-1 :size
    left 90
    forward :size
    lefthilbert :level-1 :size
    right 90
  ]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



Case Study: Building a Programming Environment for Logo

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



© J.-M. Jézéquel, 2008

23



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

© J.-M. Jézéquel, 2008

24



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

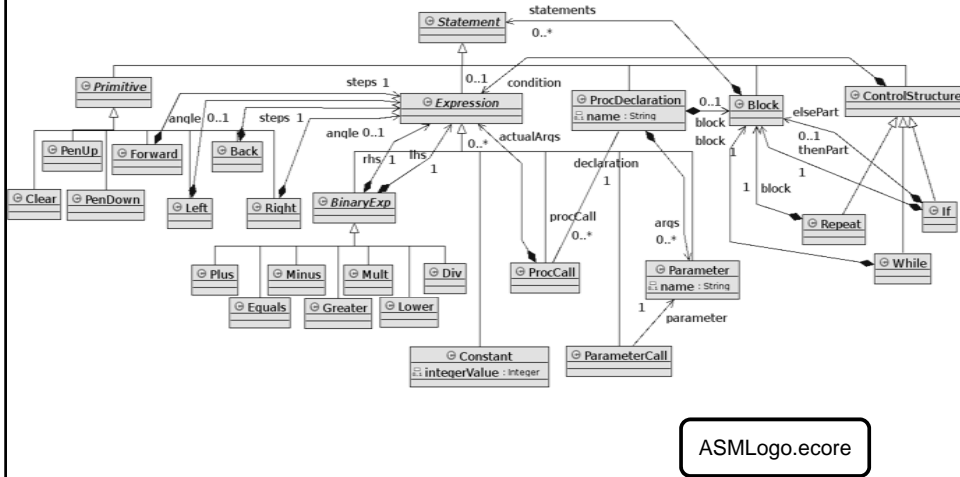


LOGO metamodel

ASMLogo.ecore



LOGO metamodel



ASMLogo.ecore



Concrete syntax

- Any regular EMF based tools
- Textual using Sintaks logo.sts
- Graphical using GMF or TopCased

```

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
  
```

```

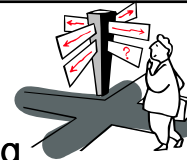
platform:/resource/LogoDemo/k/k.xmi
  Block
    Proc Declaration k
      Parameter scale
      Block
        Pen Down
        Forward
        Pen Up
        Back
        Right
        Forward
        Pen Down
        Back
        Pen Up
  
```

Do It Yourself

- **Within Eclipse**
 - Load/Edit/Save Models
 - » Conforming to the LOGO meta-model
 - » ie LOGO programs

- **Install & Run the MDLE4LOGO Bundle**
 - On your own PC
 - Or follow the beamed demo

Outline



- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up



Static Semantics with OCL

- Complementing a meta-model with Well-Formedness Rules, aka *Contracts* e.g.;
 - A procedure is called with the same number of arguments as specified in its declaration
- Expressed with the OCL (Object Constraint Language)
 - The OCL is a language of typed expressions.
 - A constraint is a valid OCL expression of type Boolean.
 - A constraint is a restriction on one or more values of (part of) an object-oriented model or system.



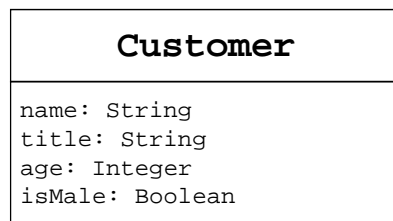
Contracts in OO languages

- Inspired by the notion of Abstract Data Type
- Specification = Signature +
 - Preconditions
 - Postconditions
 - Class Invariants
- Behavioral contracts are inherited in subclasses

OCL

- Can be used at both
 - M1 level (constraints on Models)
 - » aka *Design-by-Contract* (Meyer)
 - M2 level (constraints on Meta-Models)
 - » aka Static semantics
- Let's overview it with M1 level exemples

Simple constraints

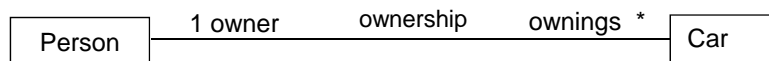


```
title = if isMale then 'Mr.' else 'Ms.' endif  
age >= 18 and age < 66  
name.size < 100
```



Non-local contracts: navigating associations

- Each association is a navigation path
 - The context of an OCL expression is the starting point
 - Role names are used to select which association is to be traversed (or target class name if only one)



Context Car inv:
self.owner.age >= 18



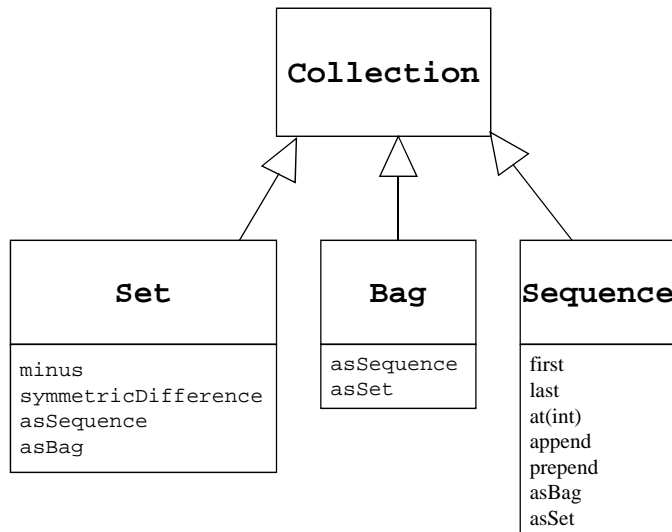
Navigation of 0..* associations

- Through navigation, we no longer get a scalar but a *collection* of objects
- OCL defines 3 sub-types of collection
 - **Set** : when navigation of a 0..* association
 - » Context Person inv: ownings return a Set[Car]
 - » Each element is in the Set at most once
 - **Bag** : if more than one navigation step
 - » An element can be present more than once in the Bag
 - **Sequence** : navigation of an association {ordered}
 - » It is an ordered Bag
- Many predefined operations on type *collection*

Syntax::
Collection->operation



Collection hierarchy



© J.-M. Jézéquel, 2008

37



Basic operations on collections

- ***isEmpty***
 - *true* if collection has no element
- ***notEmpty***
 - *true* if collection has at least one element
- ***size***
 - Number of elements in the collection
- ***count (elem)***
 - Number of occurrences of element *elem* in the collection

Context Person inv:
age < 18 implies ownings->isEmpty

© J.-M. Jézéquel, 2008

38



select Operation

- possible syntax

- collection->select(elem:T | expr)
- collection->select(elem | expr)
- collection->select(expr)

- Selects the subset of *collection* for which property *expr* holds

- e.g.

```
context Person inv:  
ownings->select(v: Car | v.mileage<100000)->notEmpty
```

- shortcut:

```
context Person inv:  
ownings->select(mileage<100000)->notEmpty
```



forAll Operation

- possible syntax

- collection->forall(elem:T | expr)
- collection->forall(elem | expr)
- collection->forall(expr)

- True iff *expr* holds for each element of the *collection*

- e.g.

```
context Person inv:  
ownings->forall(v: Car | v.mileage<100000)
```

- shortcut:

```
context Person inv:  
ownings->forall(mileage<100000)
```



Operations on Collections

Operation	Description
size	The number of elements in the collection
count(object)	The number of occurrences of object in the collection.
includes(object)	True if the object is an element of the collection.
includesAll(collection)	True if all elements of the parameter collection are present in the current collection.
isEmpty	True if the collection contains no elements.
notEmpty	True if the collection contains one or more elements.
iterate(expression)	Expression is evaluated for every element in the collection.
sum(collection)	The addition of all elements in the collection.
exists(expression)	True if expression is true for at least one element in the collection.
forAll(expression)	True if expression is true for all elements.



Static Semantics for LOGO

- No two formal parameters of a procedure may have the same name:

- A procedure is called with the same number of arguments as specified in its declaration:

Static Semantics for LOGO

- No two formal parameters of a procedure may have the same name:

context ProcDeclaration

inv unique_names_for_formal_arguments :

args -> forAll (a1 , a2 | a1.name = a2.name

implies a1 = a2)

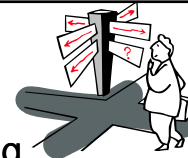
- A procedure is called with the same number of arguments as specified in its declaration:

context ProcCall

inv same_number_of_formals_and_actuais :

actualArgs -> size = declaration.args -> size

Outline



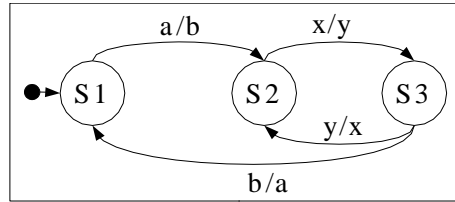
- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up



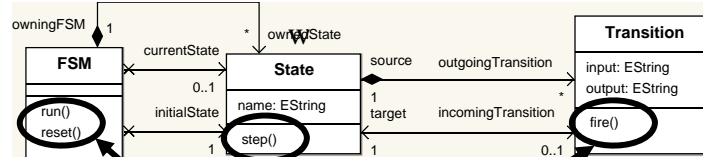


Operational Semantics of State Machines

- A model



- Its metamodel



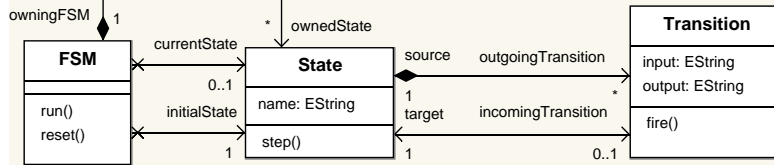
- Adding Operational Semantics to OO Metamodels



Kermeta Rationale

- Model, meta-model, meta-metamodel, DSLs...
 - Meta-bla-bla too complex for the normal engineer
- On the other hand, engineers are familiar 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

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 semantics)
require "DynamicSemantics.kmt"
// Transformations
```

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

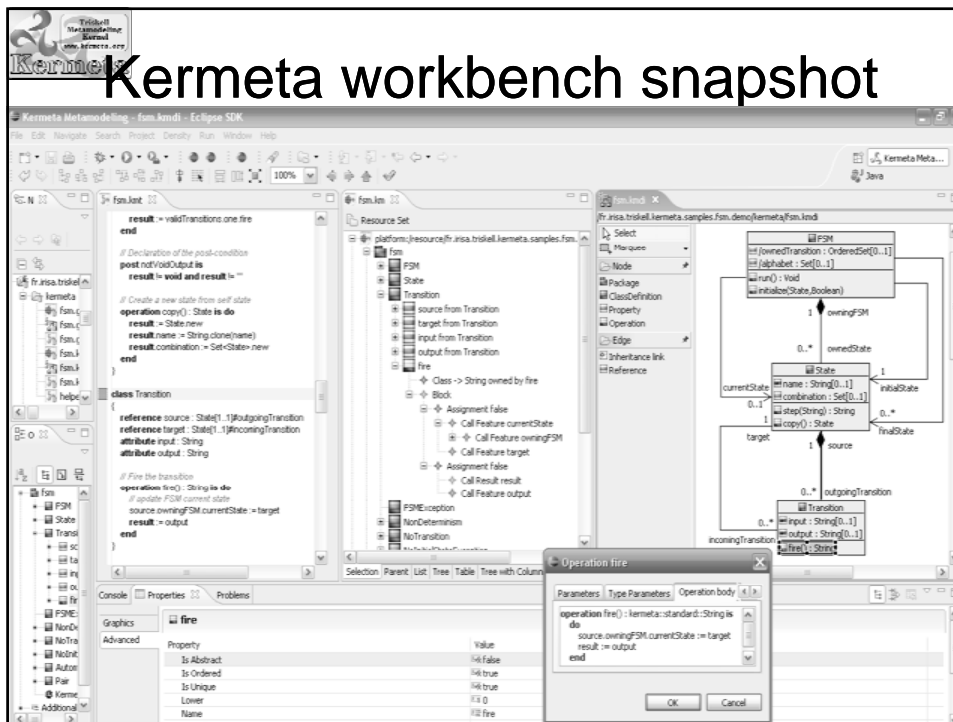
```
aspect class FSM {
operation reset(): Void {
currentState := initialState
```

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

© J.-M. Jézéquel,

47

Kermeta workbench snapshot





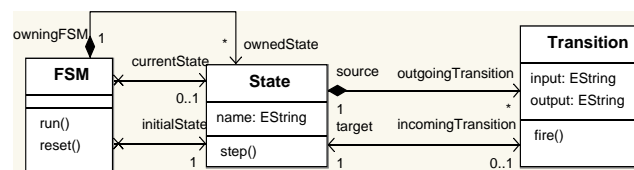
Kermeta:

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.



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
}
  
```

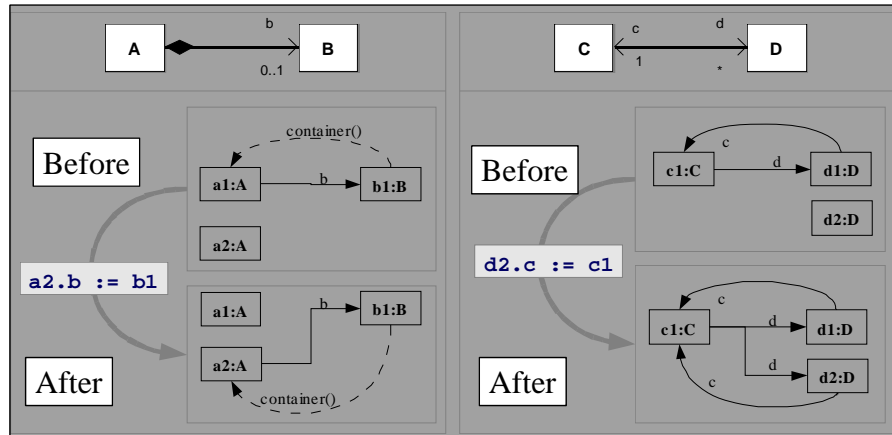
```

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
}
  
```

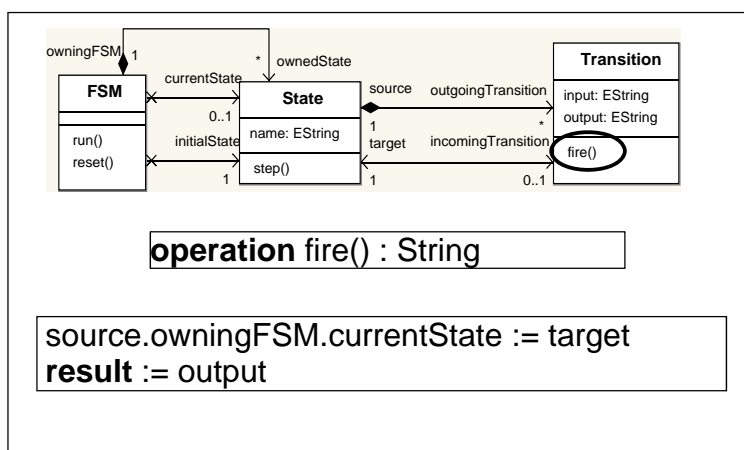
Assignment semantics

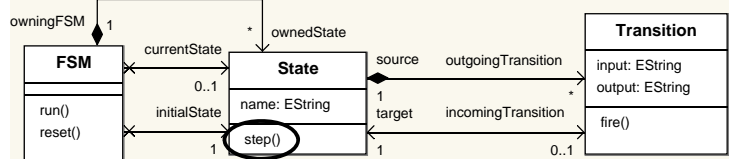
Composition

Association



Example

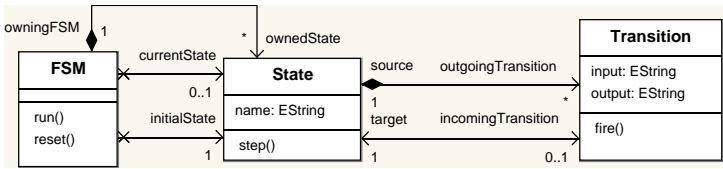




operation step(c : String) : String

```

// 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
  
```



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
end
stdio.writeln("** END OF SIMULATION **")
  
```

The image shows an Eclipse IDE window with three main components:

- Diagram:** A state machine with three states: S1 (start), S2, and S3. Transitions are: S1 to S2 (a/b), S2 to S1 (b/a), S2 to S3 (x/y), S3 to S2 (y/x).
- Model Explorer:** Shows the EMF model structure for 'fsm.ecore' and 'fsm_sample1.xmi'. The 'fsm' package contains classes for State (S1, S2, S3), Transition (a, x, y), and FSM, along with various exceptions.
- Code Editor:** Contains the following code snippet:


```

/**
 * 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
      
```

© J.-M. Jézéquel, 2008 55

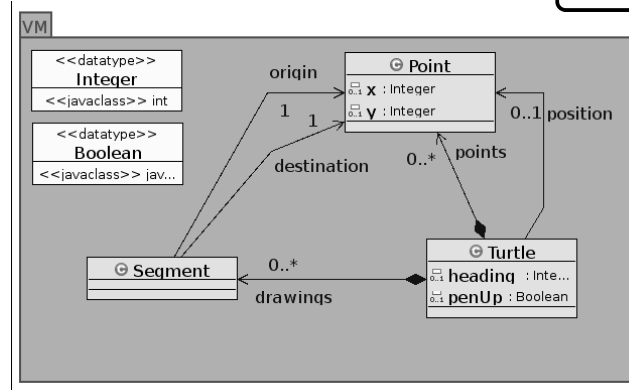
Operational Semantics for LOGO

- 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)

© J.-M. Jézéquel, 2008 56

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()
    }
}
```



Meta-level Anchoring

- Simple approach using the Kermeta VM to « ground » the semantics of basic operations
- Or reify it into the LOGO VM
 - Using eg a stack-based machine
 - Ultimately grounding it in kermeta though

```
...
aspect class Add {
    eval (ctx: Context) : Integer {
        result = lhs.eval(ctx)
        + rhs.eval(ctx)
    }
}
```

```
...
aspect class Add {
    eval (ctx: Context) {
        lhs.eval(ctx) // put result
        // on top of ctx stack
        rhs.eval(ctx) // idem
        ctx.getMachine().add()
    }
}
```



Handling control structures

- Block
- Conditional
- Repeat
- While



Operational semantics

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

LogoDynSemantics.kmt



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

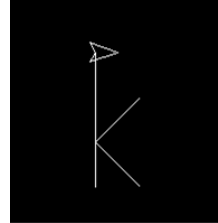
Simulator

- 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 Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up



Implementing a model-driven compiler

- Map a LOGO program to Lego Mindstorms
 - 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 {  
    }  
}
```

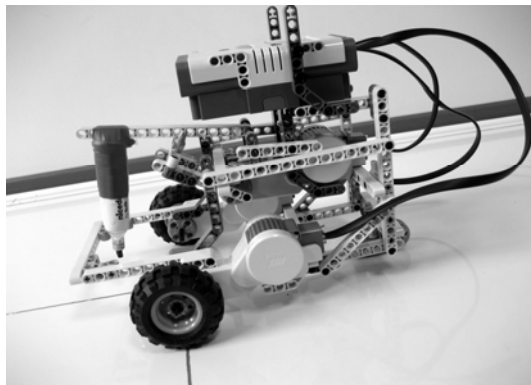
© J.-M. Jézéquel, 2008

67



Specific platform

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



© J.-M. Jézéquel, 2008

68



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...

© J.-M. Jézéquel, 2008

71



Logo to NXC Compiler

■ Step 1 – Model-to-Model transformation



■ Step 2 – Code generation with template



© J.-M. Jézéquel, 2008

72



Step 1a: The NXC Meta-Model



Step 1b: Model-to-Model

```
aspect class PenUp {  
    compile (ctx: Context) {  
  
    }  
    ...  
aspect class Clear {  
  
  
    }
```



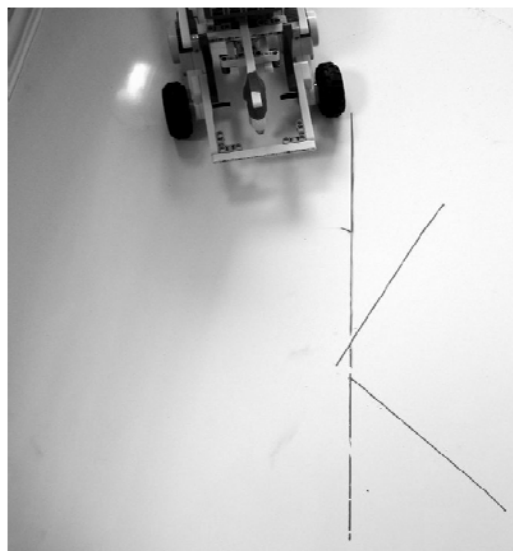
Step 2: Kermeta Emitter Template



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)
```





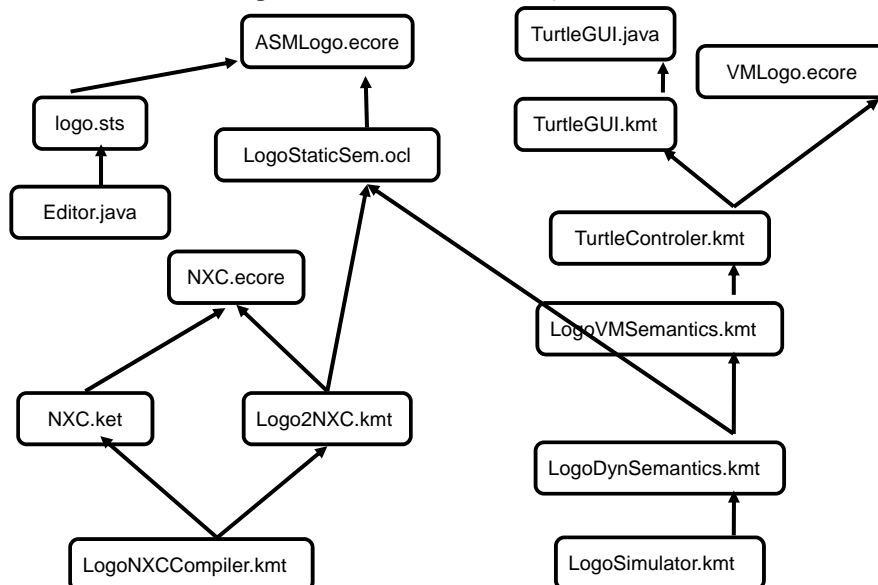
Outline



- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up



Logo Summary (1)





Logo Summary (2)

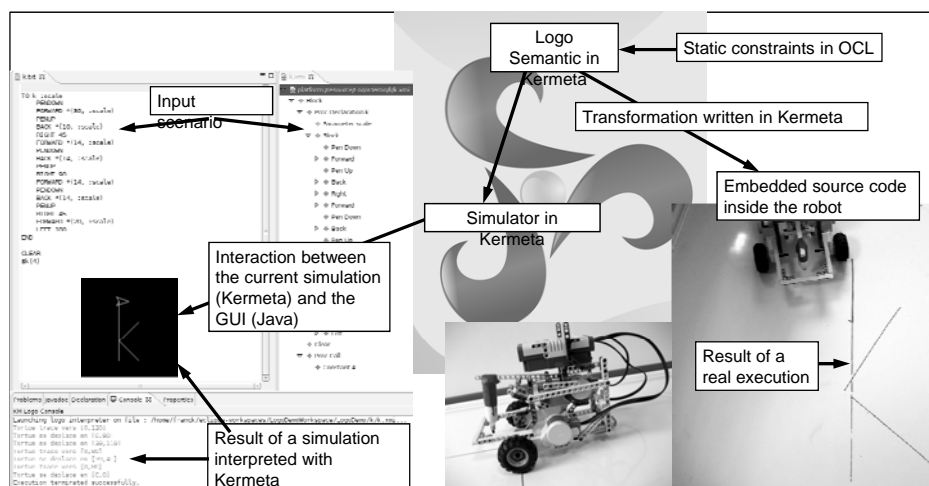
- Integrate all aspects coherently
 - syntax / semantics / tools
- Use appropriate languages
 - MOF for abstract syntax
 - OCL for static semantics
 - Kermeta for dynamic semantics
 - Java for simulation GUI
 - ...
- Keep separation between concerns
 - For maintainability and evolutions

© J.-M. Jézéquel, 2008

79



From LOGO to Mindstorms



© J.-M. Jézéquel, 2008

80



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

© J.-M. Jézéquel, 2008

81



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
 - Finally care for concrete syntax issues
- Feel free to contribute!
 - www.kermeta.org



© J.-M. Jézéquel, 2008

82



Thank you !

- Questions ?

