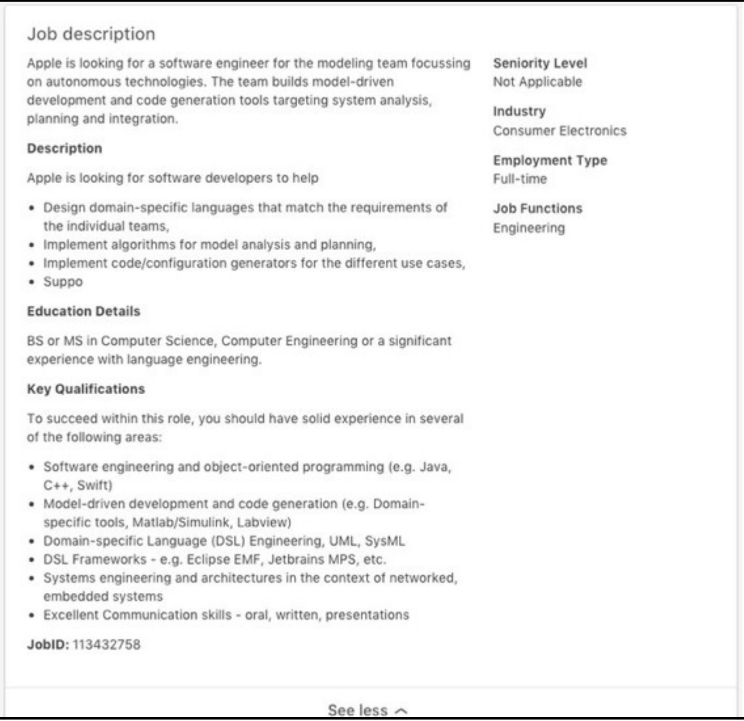
Model-Driven Engineering

(or: Why I'd like write programs that write programs rather than write programs)

Jean-Marc Jézéquel
e-mail : jezequel@irisa.fr
http://people.irisa.fr/Jean-Marc.Jezequel
Twitter @jmjezequel



Job description

Apple is looking for a software engineer for the modeling team focussing on autonomous technologies. The team builds model-driven development and code generation tools targeting system analysis, planning and integration.

Description

Apple is looking for software developers to help

- Design domain-specific languages that match the requirements of the individual teams,
- Implement algorithms for model analysis and planning,
- Implement code/configuration generators for the different use cases,
- Suppo

Education Details

BS or MS in Computer Science, Computer Engineering or a significant experience with language engineering.

Key Qualifications

To succeed within this role, you should have solid experience in several of the following areas:

- Software engineering and object-oriented programming (e.g. Java, C++, Swift)
- Model-driven development and code generation (e.g. Domain-specific tools, Matlab/Simulink, Labview)
- Domain-specific Language (DSL) Engineering, UML, SysML
- DSL Frameworks - e.g. Eclipse EMF, Jetbrains MPS, etc.
- Systems engineering and architectures in the context of networked, embedded systems
- Excellent Communication skills - oral, written, presentations

JobID: 113432758

Seniority Level
Not Applicable

Industry
Consumer Electronics

Employment Type
Full-time

Job Functions
Engineering

See less

Airbus

- **Junior Model Based Systems Development Team Member**
  - As part of the Model Based Systems Engineering Development Team "MBSD", you will support fulfilling the ADS global engineering model based development vision by participating to the extended organization, and contribute motivating project teams to achieve a high level of performance and quality in delivering model based development projects that provide exceptional business value to users. You will contribute to several concurrent high visibility development projects using advanced modeling methods in a fast-paced environment that may cross multiple business lines.
- **Required skills**
  - Undergraduate or graduate degree in a technical field, and first experience in MBSE field.
  - First experience with meta-modeling and model transformation between domains and/or other state of the art techniques.
  - First experience with systems engineering tools and representations (e.g., NoMagic, SysML, UML, or similar).
    - …

Unity: Senior Modeling Language Engineer

Unity is the world's leading platform for creating and operating real-time 3D (RT3D) content

- **Role description**
  - You are a language or tools developer with a passion for creating great user experiences…In this role you will use your expertise in building tools or **domain specific modeling languages** to shape the future of game design and development…
- **Responsibilities**
  - **Design and build modeling languages** and editors that empower game designers in ways never seen before
  - Work as a part of a cross-discipline team to build rapid prototypes that you can transform quickly into production-ready features
- **Requirements**
  - Strong understanding of data structures and algorithms
  - Fluent in C# or another statically typed language
  - Knows how to translate user needs into product features
- **Bonus points**
  - Expertise in **developing domain-specific languages and editors** before
  - …

UNIVERSITÉ DE RENNES 1

Software: Low code & No code approaches

- Shortage of programmers even for simple applications
  - Mobile phone apps, web, etc.
- Build programs that write programs
  - Put problem-solving capabilities into the hands of non-IT professionals
    - Users can more quickly and easily create business apps that help them do their jobs
- Industrial platforms
  - Google AppSheet, Mendix, Microsoft PowerApps, OutSystems, Robocoder Rintagi, Salesforce Lightning, Wix Editor X, etc.
  - Analysts at Gartner estimate that the low-code market grew 23% in 2020 to reach $11.3 billion, and will grow to $13.8 billion in 2021 and almost $30 billion by 2025

UNIVERSITÉ DE RENNES 1

Copied from Microsoft web site

Example: Microsoft PowerApps 1/2

- Quickly create apps that work on any device using a Microsoft Office-like experience, templates to get started quickly and a visual designer to automate workflows.
- Use built-in connections, or ones built by your company, to connect PowerApps to cloud services
- Build additional data connections and APIs to any existing business systems, thus empowering any users in your organization to create the apps they need.
- Data security and privacy controls are respected by PowerApps, so you can manage data access and maintain corporate policies
  - *Hum hum!*

Copied from Microsoft web site

Example: Microsoft PowerApps 2/2

- Model-driven apps
  - start with your data model – building up from the shape of your core business data and processes in the Dataverse to model forms, views, and other components.
  - automatically generate great UI that's responsive across devices.
- When you create a model-driven app, you can use all the power of the Dataverse to rapidly configure your forms, business rules, and process flows
- Dataverse is a data platform that allows you to store and model business data
  - securely store and manage data within a set of standard and custom tables, and you can add columns to those tables when you need them.

Goal of this module

- Learn the principles to build a use lowcode/nocode approach
  - Not to use lowcode/nocode!
- Use it when you identify a niche market where
  - the technical aspect of applications is always the same (or can be configured among a small set of options)
  - The functional aspect is simple enough that it can be described through a formalism close to natural language
    - Eg workflow

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

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.

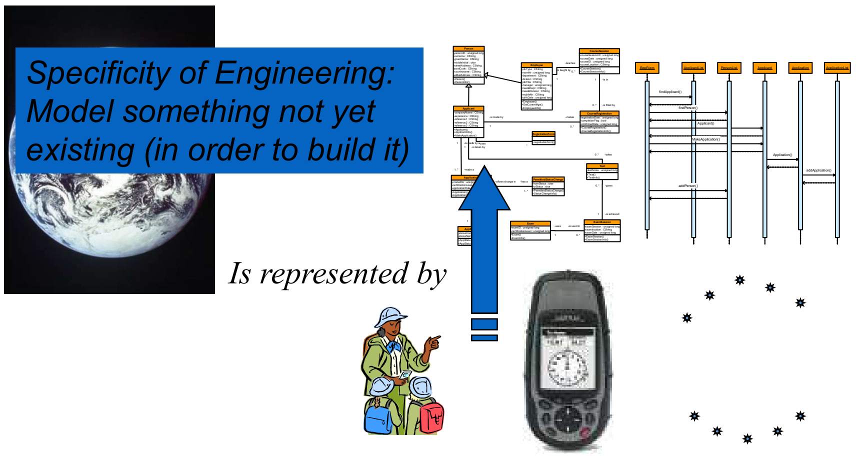
Jeff Rothenberg.

Modeling in Science & Engineering

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

M_1 (modeling space)

M_0 (the world)


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

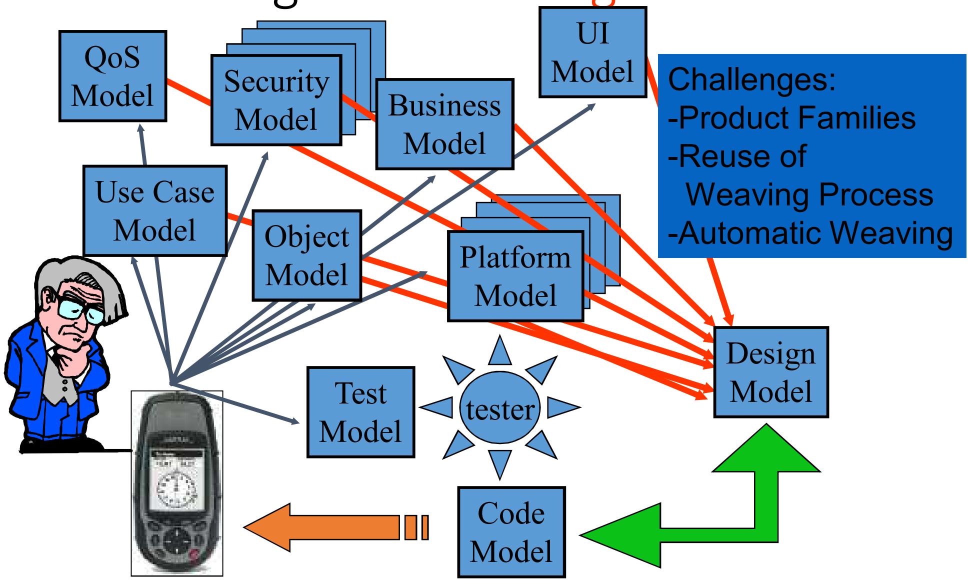
Model and Reality in Software

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

Ceci n'est pas une pipe.

- Software Models: from contemplative to productive

Modeling and Weaving



Challenges:
-Product Families
-Reuse of Weaving Process
-Automatic Weaving

Complex Software Intensive Systems

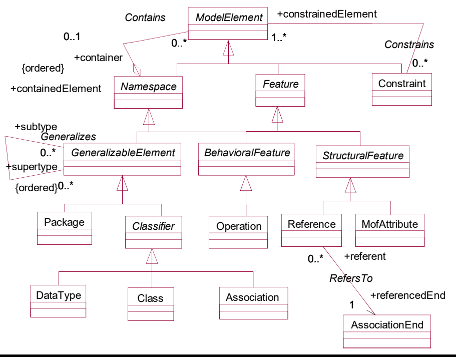
- Multiple concerns
- Multiple viewpoints & stakeholders
- Multiple domains of expertise
- => Need languages to express them!
  - In a meaningful way for experts
  - With tool support (analysis, code gen., V&V..)
    - Which is still costly to build
  - At some point, all these concerns must be integrated

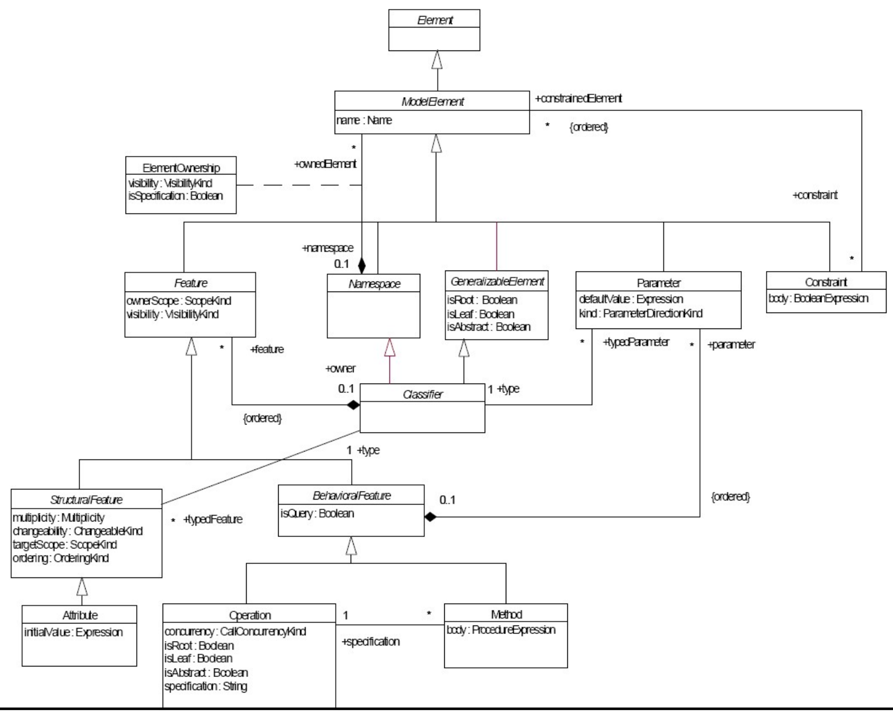
Modeling Languages

- General Purpose Modeling Languages
  - UML and its profiles (MARTE for RT…)
- Domain Specific Modeling Languages
  - Airbus, automotive industry…
  - Matlab/Simulink
  - Lowcode/nocode
- General Purpose Programming Languages
  - With restrictions (not everything allowed)
    - GWT (Google Web Toolkit)
- Annotations, aspects…
- *In any case, Need for Language Processors*

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





Generalizations

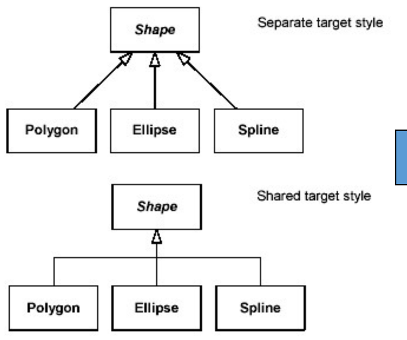


Figure 3-33. Examples of generalizations between classes.

NB: Tell you nothing about:
•generalization being acyclic,
•or semantics of dynamic binding

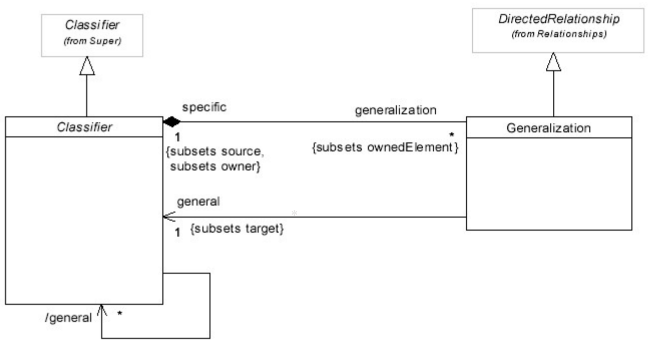
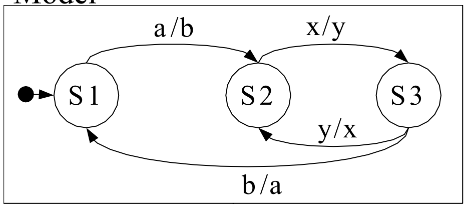


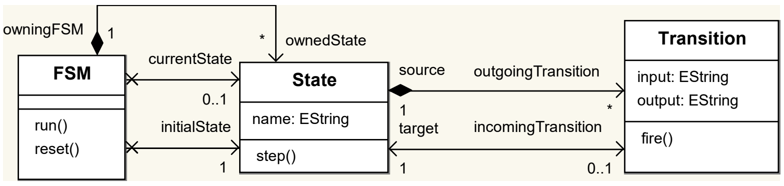
Figure 3-32. The elements defined in the Generalizations package.

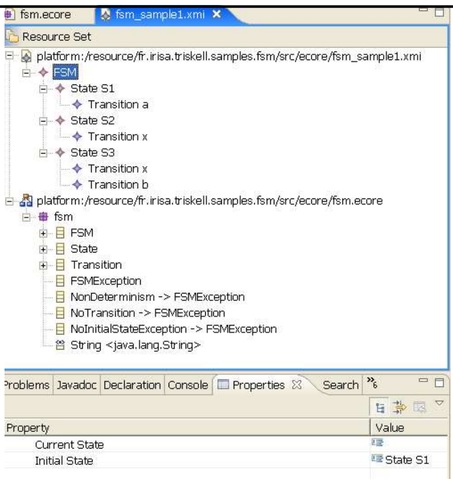
Example with StateMachines

Model



Meta-Model





```
fsm.ecore    fsm_sample1.xmi
Resource Set
platform:/resource/fr.irisa.triskell.samples.fsm/src/ecore/fsm_sample1.xmi
  FSM
    State S1
      Transition a
    State S2
      Transition x
    State S3
      Transition x
      Transition b
platform:/resource/fr.irisa.triskell.samples.fsm/src/ecore/fsm.ecore
  fsm
    FSM
    State
    Transition
    FSMException
    NonDeterminism -> FSMException
    NoTransition -> FSMException
    NoInitialStateException -> FSMException
    String <java.lang.String>
```

Problems | Javadoc | Declaration | Console | Properties | Search

Property	Value
Current State	
Initial State	State S1

The 4 layers in practice

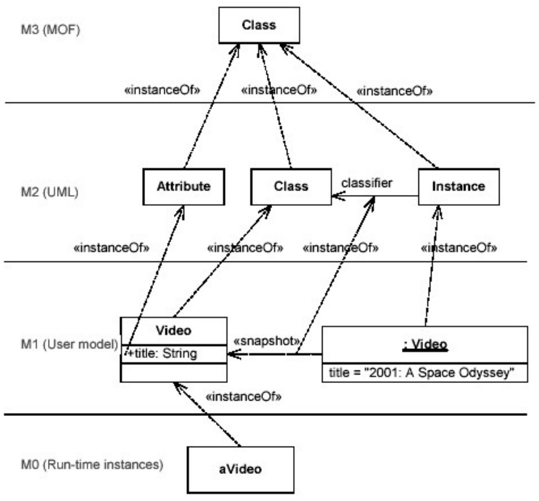
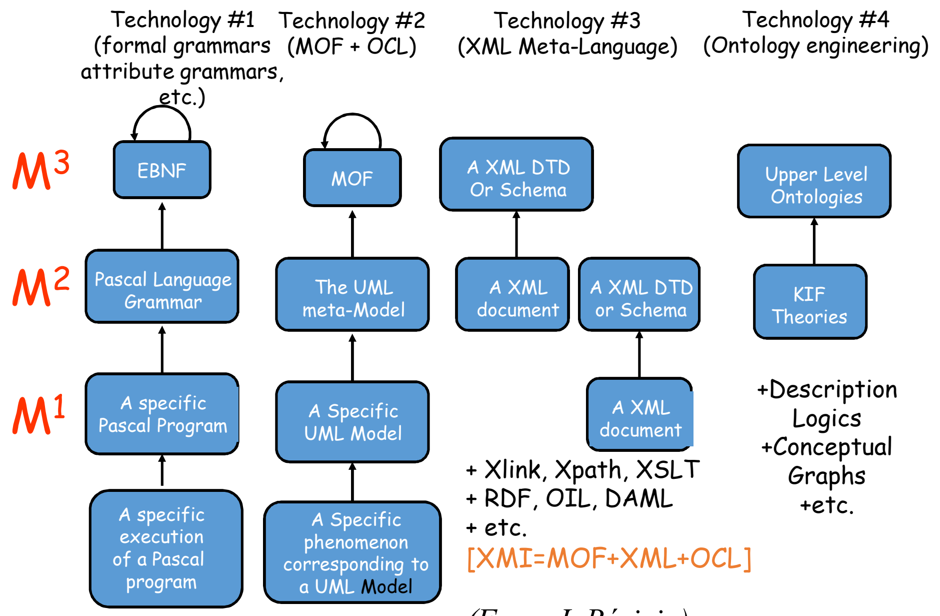


Figure 1-8. An example of the four-layer metamodel hierarchy.

Comparing Abstract Syntax Systems



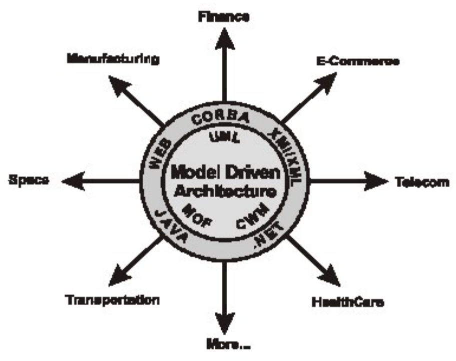
Technology #1 (formal grammars attribute grammars, etc.)
Technology #2 (MOF + OCL)
Technology #3 (XML Meta-Language)
Technology #4 (Ontology engineering)
M3
M2
M1
EBNF
Pascal Language Grammar
A specific Pascal Program
A specific execution of a Pascal program
MOF
The UML meta-Model
A Specific UML Model
A Specific phenomenon corresponding to a UML Model
A XML DTD Or Schema
A XML document
A XML DTD or Schema
A XML document
Upper Level Ontologies
KIF Theories
+Description Logics
+Conceptual Graphs
+etc.

+ Xlink, Xpath, XSLT
+ RDF, OIL, DAML
+ etc.
[XMI=MOF+XML+OCL]

(From J. Bézivin)

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."

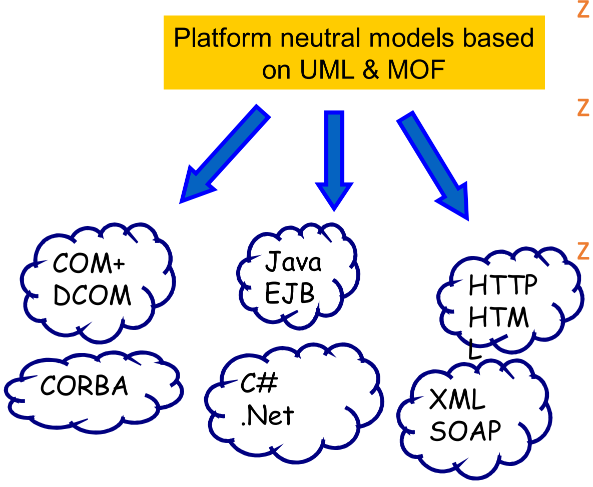


Richard Soley & OMG staff,

MDA Whitepaper Draft 3.2

November 27, 2000

Mappings to multiple and evolving platforms



- MOF & UML as the core
- Organization assets expressed as models
- Model transformations to map to technology specific platforms

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, …

UNIVERSITÉ DE RENNES 1

Model Driven Engineering : Summary

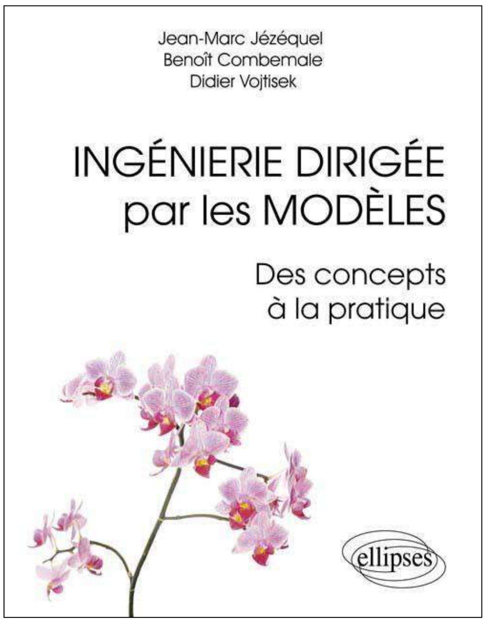
- Modeling to master complexity
  - Multi-dimensional and aspect oriented by definition
- Models: from contemplative to productive
  - Meta-modeling tools, meta-models used to define languages
- Model Driven Engineering
  - Weaving aspects into a design model
    - E.g. Platform Specificities
- Model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

© J.-M. Jézéquel, 2012-2020

32

UNIVERSITÉ DE RENNES 1

To learn more…

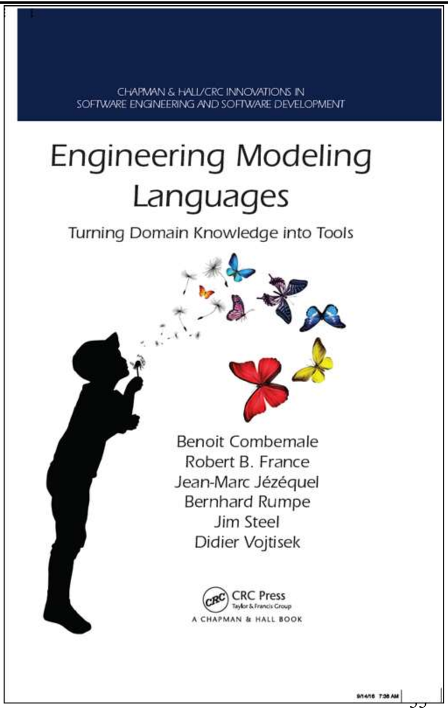


Jean-Marc Jézéquel
Benoît Combemale
Didier Vojtisek

INGÉNIERIE DIRIGÉE par les MODÈLES

Des concepts à la pratique

ellipses



CHAPMAN & HALL/CRC INNOVATIONS IN SOFTWARE ENGINEERING AND SOFTWARE DEVELOPMENT

Engineering Modeling Languages

Turning Domain Knowledge into Tools

Benoit Combemale
Robert B. France
Jean-Marc Jézéquel
Bernhard Rumpe
Jim Steel
Didier Vojtisek

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

© J.-M. Jézéquel, 2012-2020

Outline

34

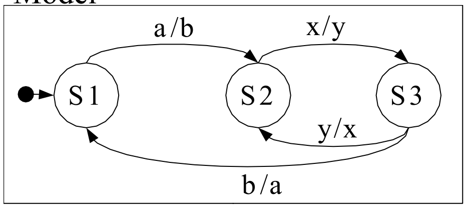
Meta-Models as Shared Knowledge

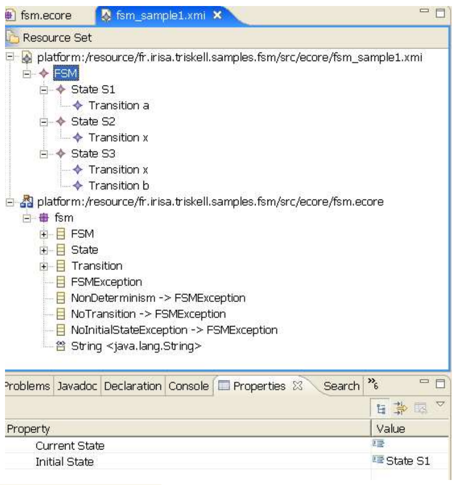
- Definition of an Abstract Syntax in E-MOF
  - Repository of models with EMF
  - Reflexive Editor in Eclipse
  - JMI for accessing models from Java
  - XML serialization for model exchanges
- Applied in more and more projects
  - SPEEDS, OpenEmbedd,DiVA…

35

Example with StateMachines

Model



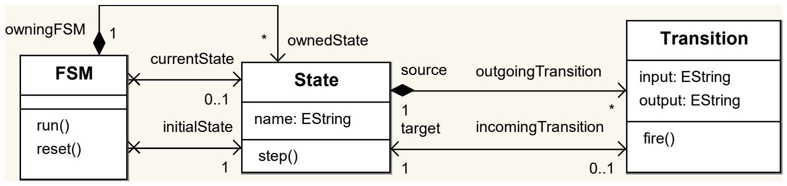

fsm.ecore | fsm_sample1.xmi

Resource Set
- platform:/resource/fr.irisa.triskell.samples.fsm/src/ecore/fsm_sample1.xmi
  - FSM
    - State S1
      - Transition a
    - State S2
      - Transition x
    - State S3
      - Transition x
      - Transition b
- platform:/resource/fr.irisa.triskell.samples.fsm/src/ecore/fsm.ecore
  - fsm
    - FSM
    - State
    - Transition
    - FSMException
    - NonDeterminism -> FSMException
    - NoTransition -> FSMException
    - NoInitialStateException -> FSMException
    - String <java.lang.String>

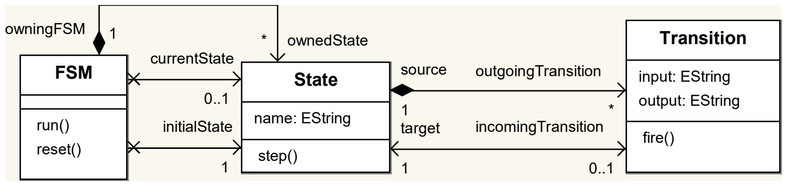
Problems | Javadoc | Declaration | Console | Properties | Search

Property	Value
Current State	
Initial State	State S1

Meta-Model



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.xtend”

// Transformations

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

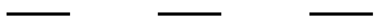
```
class FSM {
  public def void reset()  {
       currentState = initialState
```

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

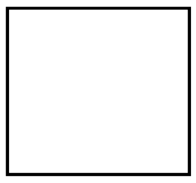
DIY with LOGO programs

- Consider LOGO programs of the form:

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

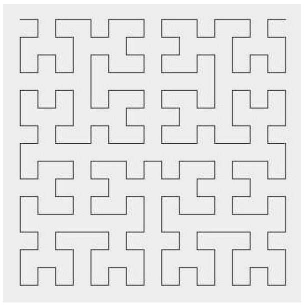


```
to square :width
  repeat 4  [ forward :width right  90]
end
pendown square 10 *10
```



Fractals in LOGO

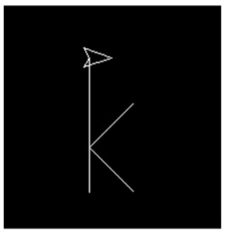
```
; lefthilbert
to lefthilbert :level :size
   if :level != 0 [
      left 90
      righthilbert :level-1 :size
      forward :size
      right 90
      lefthilbert :level-1 :size
      forward :size
      lefthilbert :level-1 :size
      right 90
      forward :size
      righthilbert :level-1 :size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

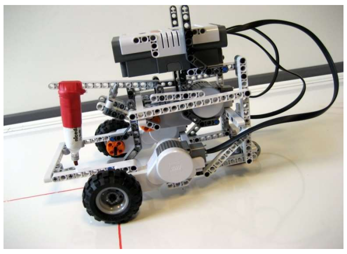


```
; righthilbert
to righthilbert :level :size
    if :level != 0 [
      right 90
      lefthilbert :level-1 :size
      forward :size
      left 90
      righthilbert level-1 :size
      forward :size
      righthilbert :level-1 :size
      left 90
      forward :size
      lefthilbert :level-1 :size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

UNIVERSITÉ DE RENNES 1

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

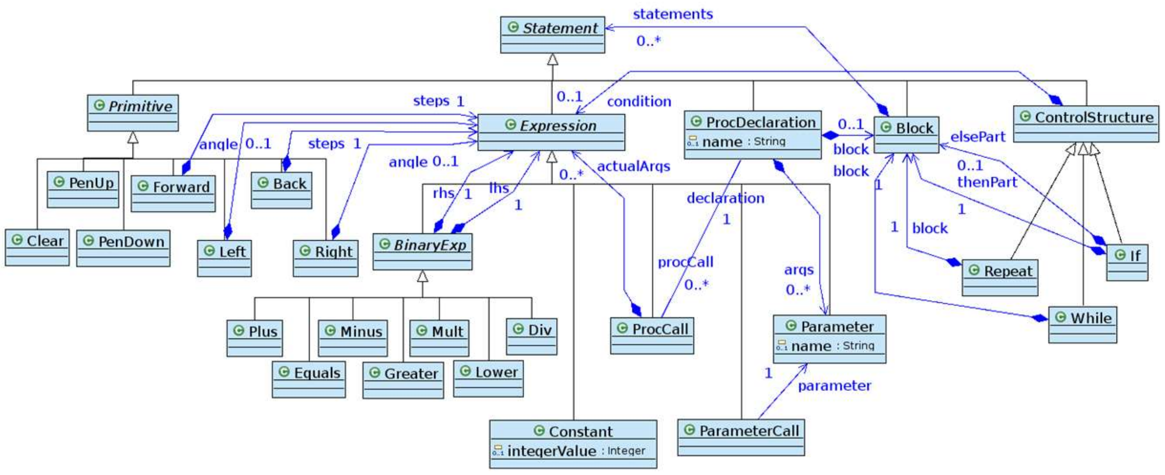
UNIVERSITÉ DE RENNES 1

LOGO metamodel : find the program concepts

```
; 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
  ]
```

- *Comment (?)*
- *ProcDeclaration, Parameter (formal)*
- *If*
- *ParameterCall, NotEqual, Constant (BinaryExp, Expression)*
- *Block*
- *Left (Primitive Instruction), Constant*
- *ProcedureCall, Expression, Expression*
- *…*

LOGO metamodel

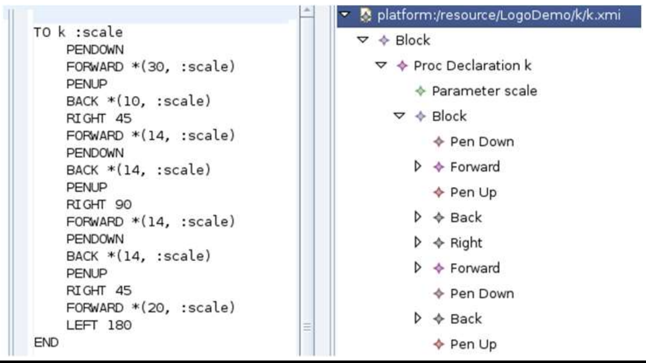


ASMLogo.ecore

Concrete syntax

- Any regular EMF based tools
- Textual using Sintaks
- 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
```

```
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
```

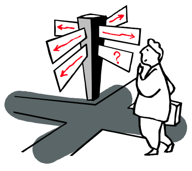
UNIVERSITÉ DE RENNES 1

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

UNIVERSITÉ DE RENNES 1

Outline



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

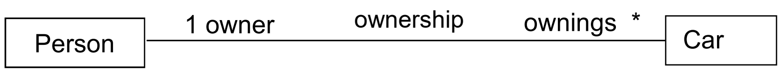
```
Customer
--------------------
name: String
title: String
age: Integer
isMale: Boolean
```

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

UNIVERSITÉ DE RENNES 1

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

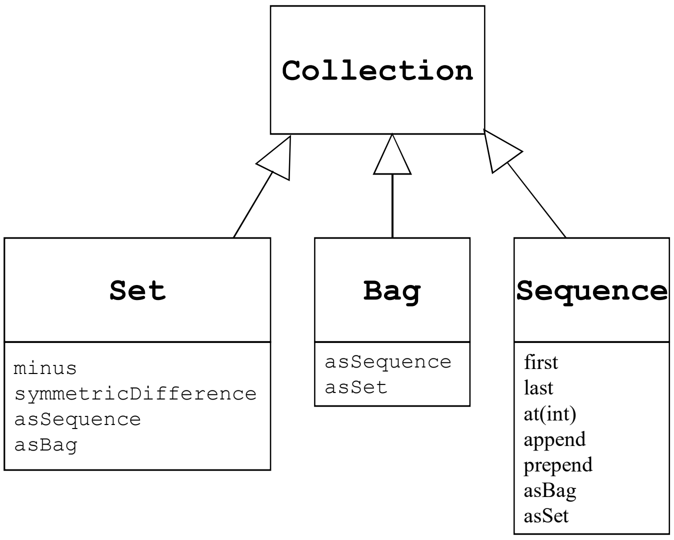
UNIVERSITÉ DE RENNES 1

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



Basic operations on collections

- *isEmpty*
  - *true* if collection has no element

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

- *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

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 occurences 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.

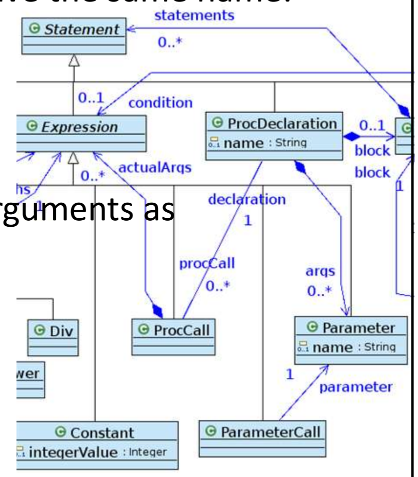
Static Semantics for LOGO

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

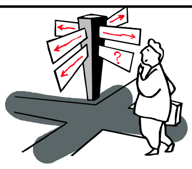
  context ProcDeclaration
    inv unique_names_for_formal_arguments :
      args -> forAll (a1 , a2 | a1. name = a2.name
          implies a1 = a2)

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

  context ProcCall
    inv same_number_of_formals_and_actuals :
      actualArgs -> size = declaration .args -> size



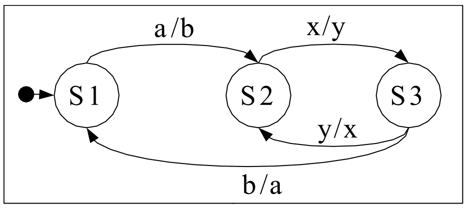
Outline



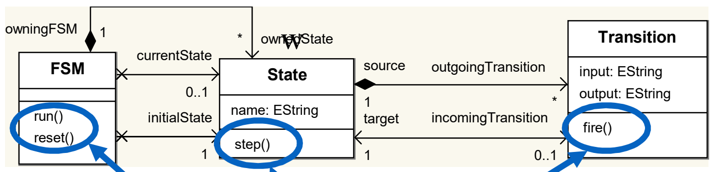
- Introduction to Model Driven Engineering
- Designing Meta-models: the LOGO example
- Static Semantics with OCL
- Operational Semantics with Kermeta
- Building a Compiler: Model transformations
- Conclusion and Wrap-up

Operational Semantics of State Machines

- A model



- Its metamodel



- Adding Operational Semantics to OO Metamodels

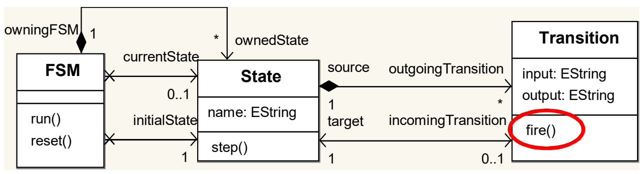
UNIVERSITÉ DE RENNES 1

Kermeta Action Language: XTEND

- Xtend
  - flexible and expressive dialect of Java
  - compiles into readable Java 5 compatible source code
  - can use any existing Java library seamlessly
- Among features on top of Java:
  - Extension methods
    - enhance closed types with new functionality
  - Lambda Expressions
    - concise syntax for anonymous function literals (like in OCL)
  - ActiveAnnotations
    - annotation processing on steroids
  - Properties
    - shorthands for accessing & defining getters and setter (like EMF)

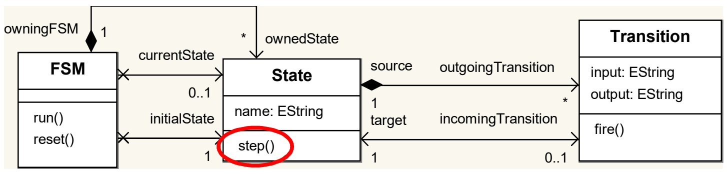
UNIVERSITÉ DE RENNES 1

Example with Xtend



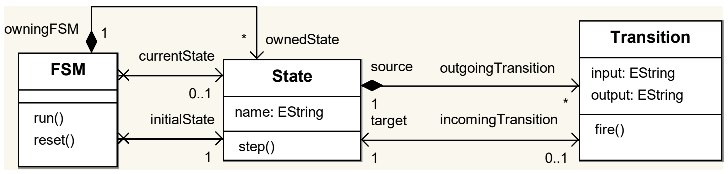
public def String fire()

```
_self.source.owningFSM.currentState = _self.target
return _self.output
```



```
def String step(String c) {

  // Get the valid transitions
  var validTransitions =
        _self.outgoingTransition.filter[t|t.input.equals(c)]

  // Check if there is one and only one valid transition
  if(validTransitions.empty) throw new NoTransition
  if(validTransitions.size > 1) throw new NonDeterminism

  // Fire the transition
  return validTransitions.get(0).fire()
}
```

© J.-M. Jézéquel, 2012-2020



```
def void run() {
// reset if there is no current state
if (_self.currentState == null) _self.currentState = _self.initialState
var str = ""
while (str != "quit") {
  println("Current state : " + _self.currentState.name)
  str = Console.instance.readLine("give me a letter : ")
  try {
      var textRes = _self.currentState.step(str)
      if (textRes == void || textRes == "") textRes = "NC"
      println("string produced : " + textRes)
    } catch (NonDeterminism err) {
      println(err.toString)
      str = "quit"
   } catch (NoTransition err) {
      println(err.toString)
      str = "quit"
  }
}
```

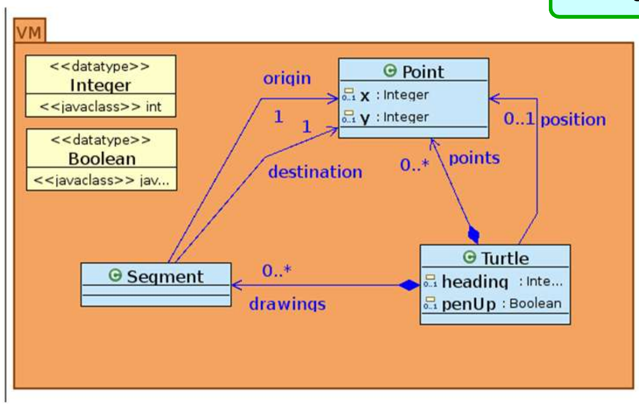
© J.-M. Jézéquel, 2012-2020

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 {
 def String toString() {
  return "[" + x.toString + "," + y.toString + "]"
 }
}

aspect class Turtle {
 def void setPenUp(b : Boolean) {
  penUp = b
 }
 def void rotate(angle : Integer)  {
  heading = (heading + angle).mod(360)
 }
}
```

© J.-M. Jézéquel, 2012-2020

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 {
        def int eval (ctx: Context) {

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

© J.-M. Jézéquel, 2012-2020

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 {
   def int eval (ctx: Context)  {
           return lhs.eval(ctx)
          + rhs.eval(ctx)
}
```

```
…
aspect class Add {
   def void 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

LogoDynSemantics.kmt

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

aspect class If {
 def int eval(context : Context) {
  if (condition.eval(context) != 0)
   return thenPart.eval(context)
  else return elsePart.eval(context)
 }
}

aspect class Right {
 def int eval(context : Context) {
  return context.turtle.rotate(angle.eval(context))
 }
}
```

Handling function calls

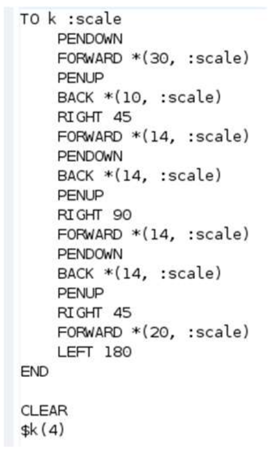
- Use a stack frame
  - Owned in the Context
- Bind formal parameters to actual
- Push stack frame
- Execute method body
- Pop stack frame

Getting an Interpreter

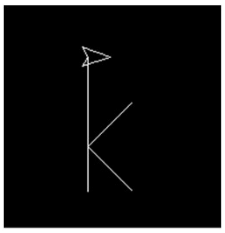
- Glue that is needed to load models
  - ie LOGO programs
- Vizualize the result
  - Print traces as text
  - Put an observer on the LOGO VM to graphically display the resulting figure

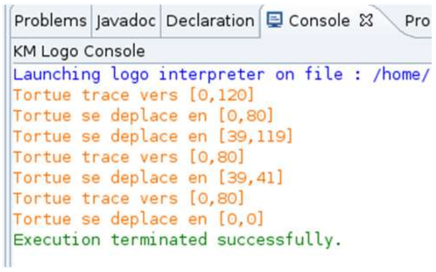
Simulator

- 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)
```





```
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

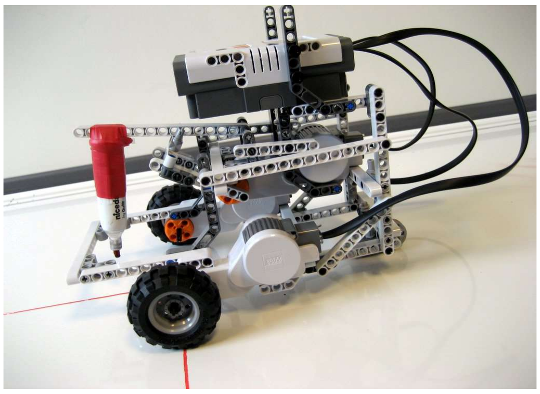
Implementing a model-driven compiler

- Map a LOGO program to Lego 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 {
        def void 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
    - PIM to PSM a la OMG MDA
    - Refining models
    - Reverse engineering (code to models)
    - Generating new views
    - Applying design patterns
    - Refactoring models
    - Deriving products in a product line
    - … any model engineering activity that can be automated…

Model-to-Text Approaches

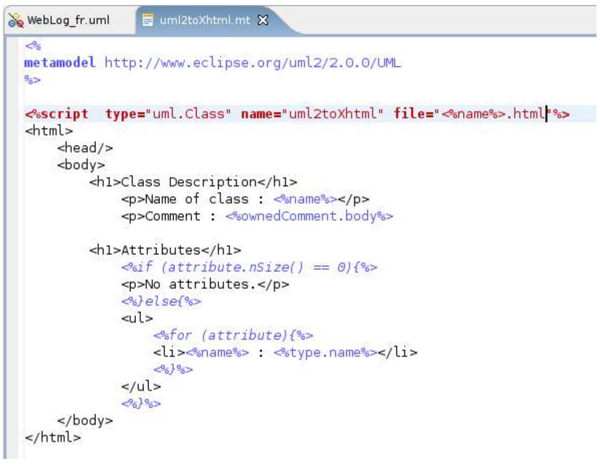
- For generating: code, xml, html, doc.
  - Visitor-Based Approaches:
    - Some visitor mechanisms to traverse the internal representation of a model and write code to a text stream
    - Iterators, Write ()
  - Template-Based Approaches
    - A template consists of the target text containing slices 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

Model to Text in practice

- For simple cases, use the template mecanism of Xtend
  - Output = ``` template expression’’’
- Many template generators for MDE do exist
  - E.g. Acceleo (from Obeo) is quite popular in industry
    - a pragmatic implementation of the Object Management Group (OMG) MOF Model to Text Language (MTL) standard
    - http://www.eclipse.org/acceleo/

Example with Acceleo

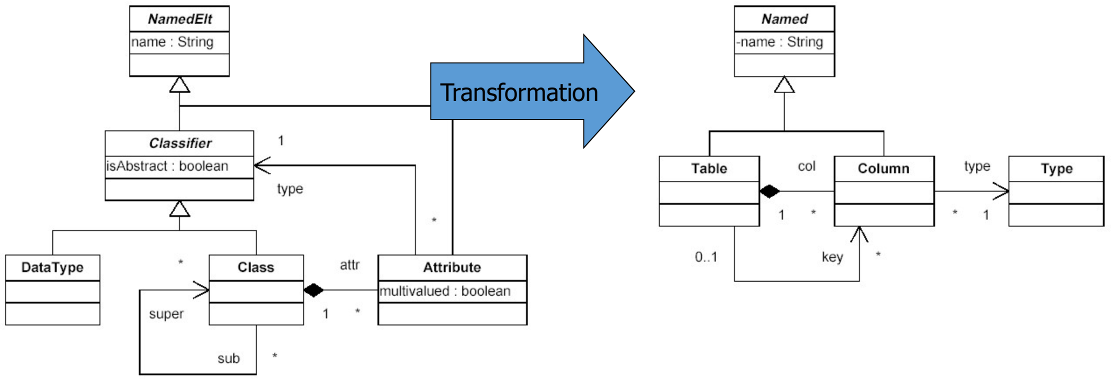
- A template that prints the class name, its comments and attributes



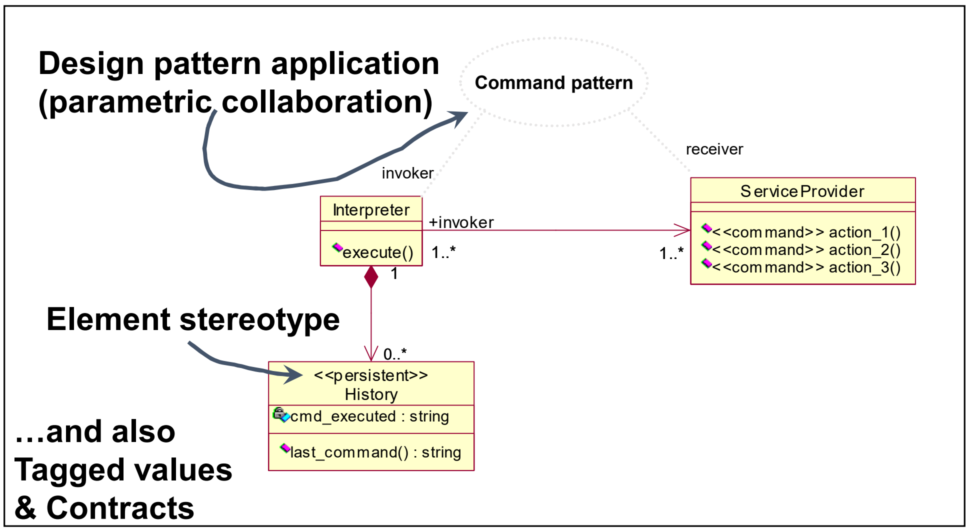
```
<%
metamodel http://www.eclipse.org/uml2/2.0.0/UML
%>

<%script  type="uml.Class" name="uml2toXhtml" file="<%name%>.html"%>
<html>
    <head/>
    <body>
        <h1>Class Description</h1>
            <p>Name of class : <%name%></p>
            <p>Comment : <%ownedComment.body%>

        <h1>Attributes</h1>
            <%if (attribute.nSize() == 0){%>
            <p>No attributes.</p>
            <%}else{%>
            <ul>
                <%for (attribute){%>
                <li><%name%> : <%type.name%></li>
                <%}%>
            </ul>
            <%}%>
    </body>
</html>
```

Model-to-Model: Typical Example

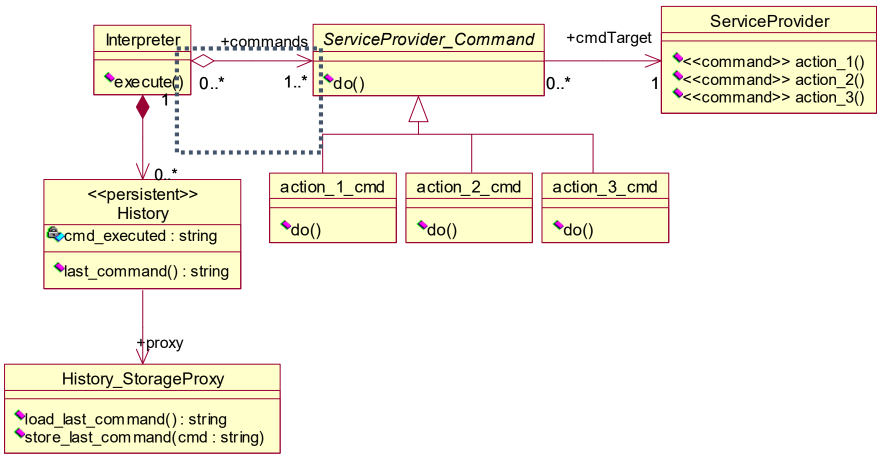
From UML to RDBMS



M2M: Reuse Engineering Know-How (Design/Test/…)



The result we want : design patterns application



UNIVERSITÉ DE RENNES 1

Classification of Model Transformation Tools

- Several approaches
  - Graph-transformation-based approaches
  - Relational approaches (aka Logic Programming)
  - Structure-Driven (OO) approaches
  - Hybrid approaches
- Rich ecosystem of tools, e.g.
  - ATL : a transformation language developed by Inria
  - GReAT : a transformation language available in the GME
  - Epsilon: for model-to-model, model-to-text, update-in-place, migration and model merging transformations.
  - Henshin: a model transformation language for EMF, based on graph transformation concepts, providing state space exploration capabilities
  - Kermeta : a general purpose modeling and programming language, also able to perform transformations
  - Mia-TL : a transformation language developed by Mia-Software
  - QVT : the OMG has defined a standard for expressing M2M transformations, called **MOF/QVT** or in short QVT.
  - SiTra: a pragmatic transformation approach based on using a standard programming language, e.g. Java, C#
  - Stratego/XT : a transformation language based on rewriting with programmable strategies
  - Tefkat : a transformation language and a model transformation engine
  - UML-RSDS [9] : a model transformation and MDD approach using UML and OCL
  - VIATRA : a framework for transformation-based verification and validation environment
  - …

UNIVERSITÉ DE RENNES 1

Model to Models in Practice

- **M2M Transformations as OO Programs**
  - **Kermeta/Xtend used to be a good choice**
  - **But now with modern Java, can be in plain Java using JMI**

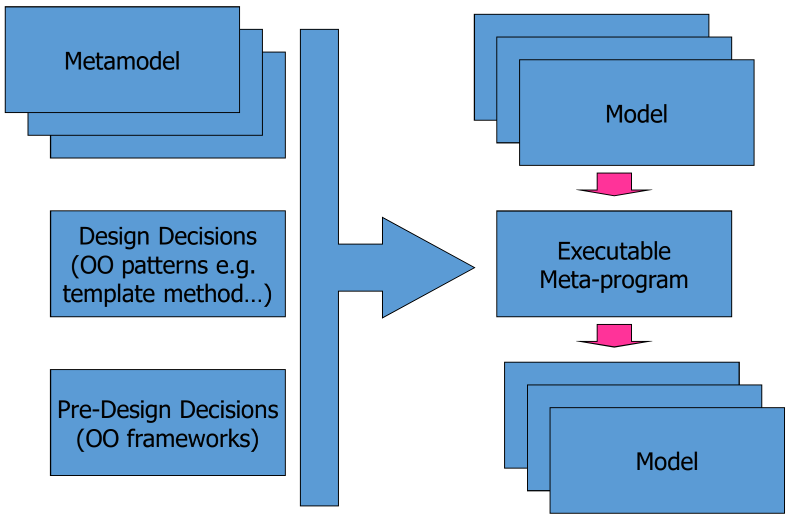
```
package javax.jmi.model;
import javax.jmi.reflect.*;
public interface Attribute extends StructuralFeature {
    public boolean isDerived();
    public void setDerived(boolean newValue);
}
```

Attributes

```
package javax.jmi.model;
import javax.jmi.reflect.*;
public interface Operation extends BehavioralFeature {
    public boolean isQuery();
    public void setQuery(boolean newValue);
    public java.util.List getExceptions();
}
```

Operations

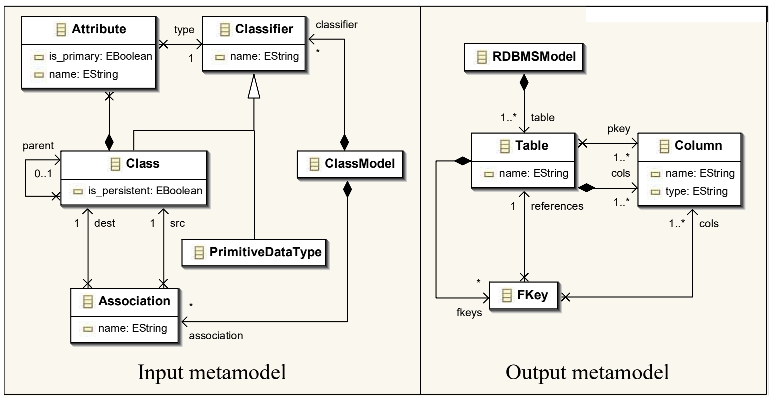
General scheme


Metamodel
Design Decisions (OO patterns e.g. template method…)
Pre-Design Decisions (OO frameworks)
Model
Executable Meta-program
Model

"Programming style" Issues

- The transformation is simply an object-oriented program that manipulates model elements
  - Uses the OO structure of the meta-model to cleanly modularize the transformation
- OO techniques
  - Customizability through inheritance/dyn. binding
  - Pervasive use of GoF like Design Patterns

Defining the metamodels



UML2RDBMS template method

- Create tables
  - Tables are created from classes marked as persistent in the input model
- Create columns
  - For each persistent class process all attributes and outgoing associations to create corresponding columns. The foreign keys are created but the *cols* property cannot be filled and the corresponding columns cannot be created because primary keys of *references* table cannot be known before it has been processed.
- Update foreign-keys
  - The foreign-key columns are created in the table that contains the foreign-key and the property *cols* of foreign-keys is updated.

=> Handle details/variability into subclasses

UNIVERSITÉ DE RENNES 1

Writing the transformation

Loading ECore and Kermeta metamodels

```
package Class2RDBMS;
require kermeta        // The kermerta standard library
require "trace.kmt     // The trace framework
require "../metamodels/ClassMM.ecore"  // Input metamodel in ecore
require "../metamodels/RDBMSMM.kmt" // Output metamodel in kermeta
[…]
class Class2RDBMS
{
  /** The trace of the transformation */
  reference class2table : Trace<Class, Table>

  /** Set of keys of the output model */
  reference fkeys : Collection<FKey>
[…]
```

UNIVERSITÉ DE RENNES 1

```
def RDBMSModel transform(inputModel : ClassModel) {
// Initialize the trace
class2table = new Trace<Class, Table>()
fkeys = new Set<FKey>()
result = new RDBMSModel()
// Create tables
getAllClasses(inputModel).select{ c | c.is_persistent }.each{ c |
  var Table table = new Table()
  table.name = c.name
  class2table.storeTrace(c, table)
  result.table.add(table)
}
// Create columns
getAllClasses(inputModel).select{ c | c.is_persistent }.each{ c |
  createColumns(class2table.getTargetElem(c), c, "")
}
// Create foreign keys
fkeys.each{ k | k.createFKeyColumns }
```

Trace Initialization

Create Tables

Create Columns

Update Foreign Keys

Object-orientation

- Classes and relations, multiple inheritance, late binding, static typing, class genericity, exception, typed function objects
- OO techniques such as patterns, may be applied to model transformations
  - Template method as above
  - Command, undo-redo
    - Refactorings example

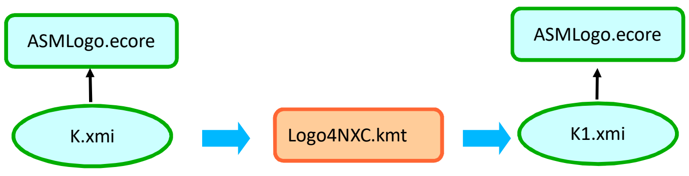
```
abstract class RefactoringCommand
{
    operation check() : Boolean is abstract
    operation transform() : Void is abstract
    operation revert() : Void is abstract
}
```

Software Engineering Concerns

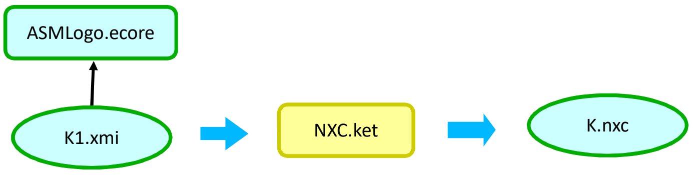
- Modularity in the small and the large
  - classes & packages
- Reliability
  - static typing, typed function objects and exception handling
- Extensibility and reuse
  - inheritance, late binding and genericity
- V & V
  - test cases

Logo to NXC Compiler

- Step 1 – Model-to-Model transformation
-



- Step 2 – Code generation with template



Step 1: Model-to-Model

- Goal: prepare a LOGO model so that code generation is a simple traversal
  - => *Model-to-Model transformation*
- Example: local2global
  - In the LOGO meta-model, functions can be declared anywhere, including (deeply) nested, without any impact on the operational semantics
  - for NXC code generation, all functions must be declared in a “flat” way at the beginning of the outermost block.
  - => implement this model transformation as a *local-to-global* aspect woven into the LOGO MM

UNIVERSITÉ DE RENNES 1

Step 1: Model-to-Model example

```
// aspect local-to-global
aspect class Statement {
 def void local2global(rootBlock: Block) {
  }
}
aspect class ProcDeclaration
 def void local2global(rootBlock: Block) {
       …
  }
}
aspect class Block
   def void local2global(rootBlock: Block) {
       …
  }
}
…
```

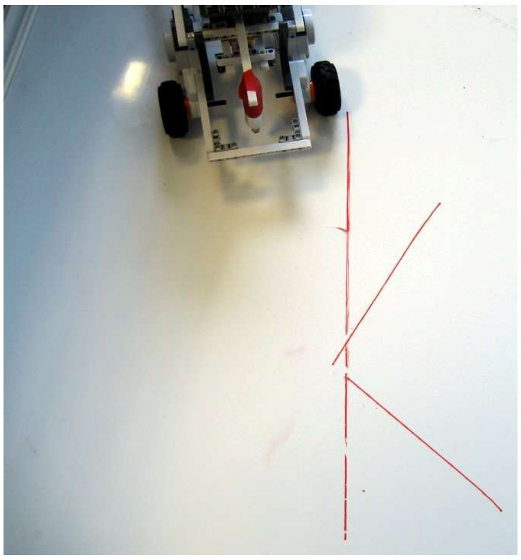
UNIVERSITÉ DE RENNES 1

Step 2: Model to text

- NXC Code generation using a template
  - Left as an exercise

Execution

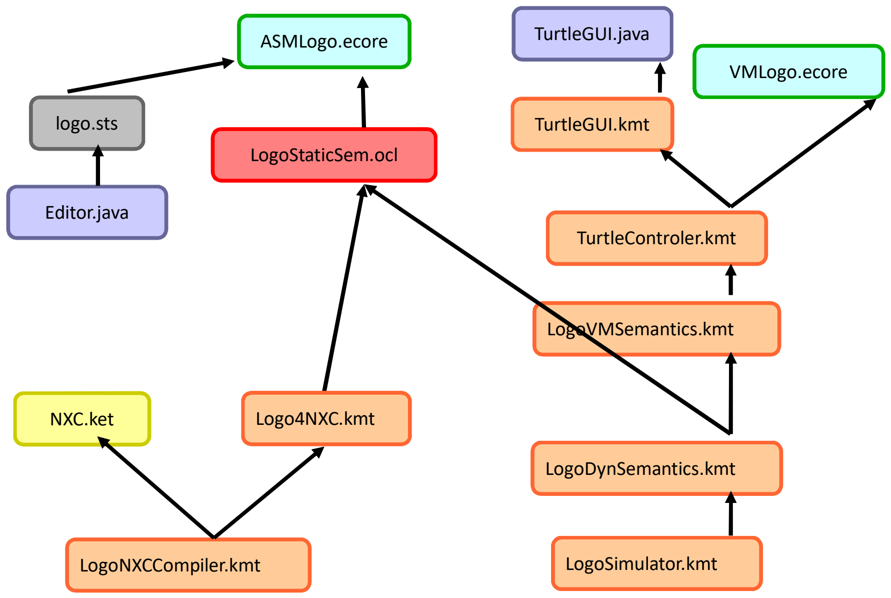
```
TO k :scale
    PENDOWN
    FORWARD *(30, :scale)
    PENUP
    BACK *(10, :scale)
    RIGHT 45
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 90
    FORWARD *(14, :scale)
    PENDOWN
    BACK *(14, :scale)
    PENUP
    RIGHT 45
    FORWARD *(20, :scale)
    LEFT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

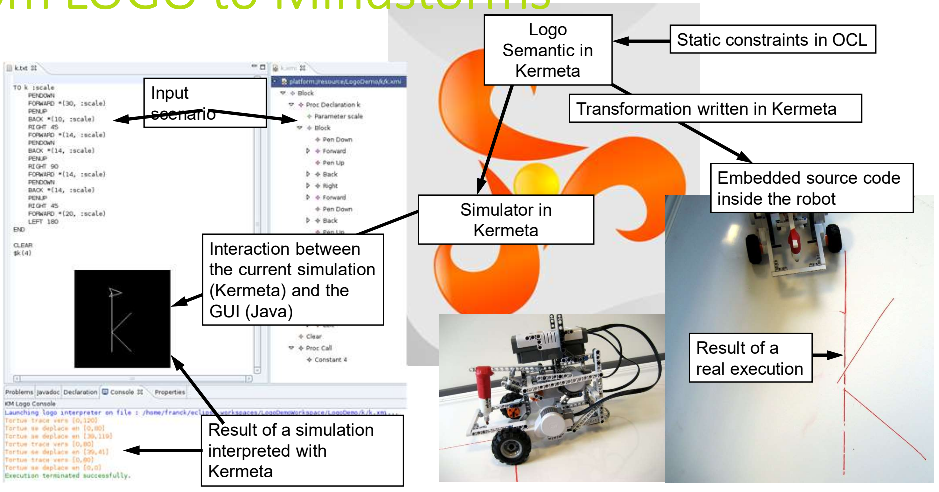


Logo Summary (2)

- 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

UNIVERSITÉ DE RENNES 1

From LOGO to Mindstorms

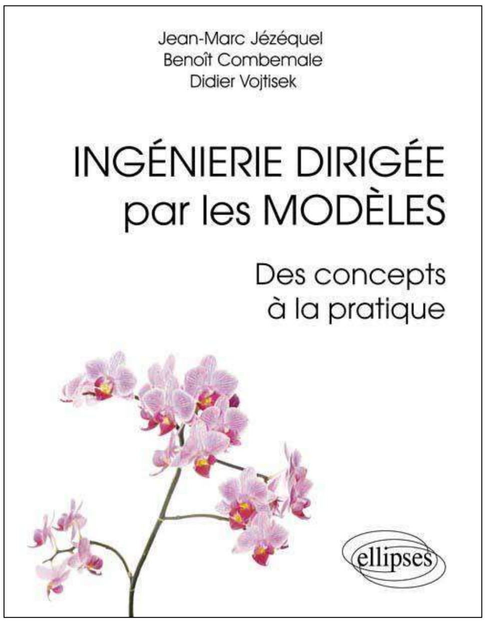

Logo Semantic in Kermeta
Static constraints in OCL
Transformation written in Kermeta
Input scenario
Embedded source code inside the robot
Simulator in Kermeta
Interaction between the current simulation (Kermeta) and the GUI (Java)
Result of a real execution
Result of a simulation interpreted with Kermeta
k.txt
Problems
Javadoc
Declaration
Console
Properties
KM Logo Console

```
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)
```

```
Launching logo interpreter on file : /home/franck/eclipse-workspaces/LogoDemoWorkspace/LogoDemo/k/k.xmi...
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.
```

UNIVERSITÉ DE RENNES 1

To learn more…

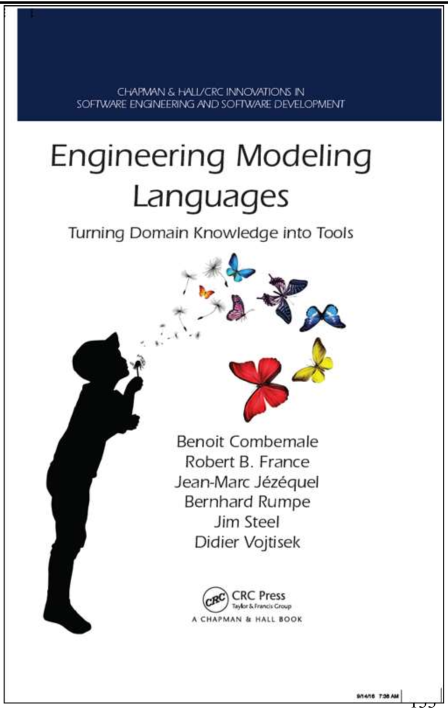


Jean-Marc Jézéquel
Benoît Combemale
Didier Vojtisek

INGÉNIERIE DIRIGÉE par les MODÈLES

Des concepts à la pratique

ellipses



CHAPMAN & HALL/CRC INNOVATIONS IN SOFTWARE ENGINEERING AND SOFTWARE DEVELOPMENT

Engineering Modeling Languages

Turning Domain Knowledge into Tools

Benoit Combemale
Robert B. France
Jean-Marc Jézéquel
Bernhard Rumpe
Jim Steel
Didier Vojtisek

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK