A Tutorial about Metamodeling
Using OMG Norms and Eclipse Modeling

Benoit Combemale
University of Rennes 1 (IRISA & ESIR)

http://www.combemale.fr/mde

Model Driven Engineering

De « L’ingénierie dirigée par les modèles. Au-delà du MDA. »
FAURE Jean-Marc, ESTUBLIER Jacy, BLAY-FORNARINO Mireille
Modeling and Metamodeling

Metamodeling

Metamodelling: Norms and Tools

- Norms (OMG): MOF, OCL, QVT, fUML, ALF…

- Metamodelling environments:
  - Eclipse Modeling (e.g., Ecore, OCL Tools…), Kermeta
  - MS DSL Tools
  - Vanderbilt GME
  - Etc.
Eclipse Modeling: Overview

- Eclipse Modeling is the umbrella project for all things about modeling that happen on the Eclipse platform:

  The Eclipse Modeling Project (EMP) focuses on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modeling frameworks, tooling, and standards implementations.

- Eclipse Modeling is not formally related to OMG, but implements several of their standards.

- It is fair to say that many leading edge modeling tools are hosted/developed at Eclipse Modeling.

- Everything Open Source under the Eclipse Public License

Eclipse Modeling: Overview

The answer to "What is Eclipse Modeling?" depends on who you ask!

A set of Eclipse projects dedicated to...
- ... Modeling: modeling tools
  - Model Development Tools (UML2, OCL, SysML, MARTE, BPMN2, etc.)

- ... Metamodeling: workbench for language design and implementation
  - Abstract Syntax Development (EMF)
  - Concrete Syntax Development (GMP, TMF)
  - Model Transformation (M2M, M2T)

- See http://www.eclipse.org/modeling

EMF: Overview

- What is it?
  - MetaModeling (think of UML/OCL)
  - Interoperability (think of XMI)
  - Editing tool support (think Eclipse)
  - Code generation (think of MDA)

- EMF serves as the foundation: It provides the Ecore metamodel, and frameworks and tools around it for tasks such as
  - Editing
  - Transactions
  - Validation
  - Query
  - Distribution/Persistence (CDO, Net4j, Teneo)

- See http://www.eclipse.org/modeling/emf
DIY with LOGO programs

- Consider LOGO programs of the form:

  ```logocode
  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
    forward :size
    righthilbert :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
Outline – Metamodelling Process

1. Domain modeling (abstract syntax)
2. Model edition (concrete syntax)
3. Model serialization (concrete syntax)
4. Model checker (static semantics)
5. Simulator implementation (behavioral semantics)
6. Compiler implementation (behavioral semantics)

OMG (Essential) MOF

- Provides language constructs for specifying a DSL metamodel
- mainly based on Object-Oriented constructs: package, classes, properties (attribute and reference), and (multiple) inheritance.
- specificities: composition, opposite…
- Defined as a model, called metamodel:

Ecore: a metamodel for metamodels

- Ecore is an implementation proposed by EMF, and aligned to EMOF
- Provides a language to build languages
- A metamodel is a model; and its metamodel is Ecore.
  - So a metamodel is an Ecore model!
- Ecore has concepts like:
  - Class – inheritance, have properties
  - Property – name, multiplicity, type
- Essentially this is a simplified version of class modeling in UML
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

Implementation with Java

How it works?

EMF

An Ecore model and its sources
(from EMF: Eclipse Modeling Framework 2nd)

Implementation with Java

- EMF is a software (E)framework
- Model driven…
  …but implemented using a programming language!
- Reification MDE → Java:
  - Metamodels are represented with EClasses
  - Models are represented with EObjects
Implementation with Java

EMF Toolset from 30.000 Feet

- The EMF Generator do not work on the \texttt{.ecore}
- EMF defines a \texttt{.gmodel} in parallel:
  - New/ Other/ Eclipse Modeling Framework/ EMF Model
  - We can customize the code generator!
  - The IDE takes care of maintaining the consistency (or not!)

Actions available on the metamodel:
1. \texttt{Generate Model Code}: Java Classes corresponding to the metamodel
2. \texttt{Generate Edit Code}: Plugin supporting the edition
3. \texttt{Generate Editor Code}: Plugin for a tree based model editor
4. \texttt{Generate Test Code}: Plugin for unit testing

Actions available from the \texttt{.genmodel}, and into an EMF Project.

From "Mastering Eclipse Modeling Framework", V. Bacvanski and P. Graff
EMF: open the box

- The EMF.edit separates the GUI from the business model
- To understand the EMF.edit plug-in, it is essential to understand three basic design patterns
  - Observer pattern
  - Command pattern
  - Adapter pattern

Outline – Metamodeling Process

1. Domain modeling (abstract syntax)
2. Model edition (concrete syntax)
3. Model serialization (concrete syntax)
4. Model checker (static semantics)
5. Simulator implementation (behavioral semantics)
6. Compiler implementation (behavioral semantics)

EOperation Implementation

Localization of the methods in the generated code
1. In the subpackage graph.impl
2. In the class GraphImpl
3. Scattered in the code automatically generated by EMF...

```java
/**
 * @generated NOT
 */
public int order () {
    return this.getEdges().size();
}
```

Do not forget to mark (@generated NOT) to prevent crashing!

Default EMF Editor

- EMF’s default reflective editor can be used out of the box to create instances of any meta model
  - Generic,
  - But not very usable or scalable
EMF as cornerstone for the other projects

The underlying framework for many many other tools

The GMP and TMF Eclipse Projects

- GMP (Graphical Modeling Project):
  - generative components and runtime infrastructures for developing graphical editors based on EMF and GEF

- TMF (Textual Modeling Framework):
  - tools and frameworks for developing textual syntaxes and corresponding editors based on EMF

Building Model Editors

GMP: Overview

- Mainly composed of GMF (Graphical Modeling Framework)
  - Define custom graphical editors for your metamodel
  - Use the editor to “draw” instances of the metamodel
**GMP: Overview**

- An example of a GMF graphical editor:

**TMF: Overview**

- Mainly composed of xText
- Xtext is a framework/tool for development of external textual DSLs.
- Based on an EBNF grammar, xText generates
  - ANTLR3-based parser
  - EMF-based metamodel
  - Eclipse editor with
    - syntax highlighting
    - code completion
    - customizable outline
    - code folding
    - real-time constraint checking
Outline – Metamodeling Process

1. Domain modeling (abstract syntax)
2. Model edition (concrete syntax)
3. Model serialization (concrete syntax)
4. Model checker (static semantics)
5. Simulator implementation (behavioral semantics)
6. Compiler implementation (behavioral semantics)

XMI (XML Metadata Interchange)

- XMI is XML standard for meta-data interchange
- If the metamodel can be expressed in MOF, its meta-data can be serialised as XMI
- Ecore models are stored/shared in XMI
- Ecore diagrams are in XMI
- EMF generator models are in XMI!
Static Semantics with OCL

- Complementing a metamodel with Well-Formedness Rules, aka contracts or context conditions, e.g.:
  - A procedure is called with the same number of arguments as specified in its declaration

  ⇒ Design-by-Contract applied at the meta-level

- 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 (aka. context conditions or well-formedness rules)

- 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
```
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)

```
Person 1 owner ownership owning * Car
```

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

```
Collection
  Set
    minus
    symmetricDifference
    asSequence
    asBag

Bag
  asSequence
  asSet

Sequence
  first
  last
  append
  prepend
  asBag
  asSet
```

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
**Operation select**

- 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.

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

- shortcut:

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

**Operation forAll**

- 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.

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

- shortcut:

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

**Operations on Collections**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>The number of elements in the collection</td>
</tr>
<tr>
<td>count(object)</td>
<td>The number of occurrences of object in the collection</td>
</tr>
<tr>
<td>includes(object)</td>
<td>True if the object is an element of the collection</td>
</tr>
<tr>
<td>includesAll(collection)</td>
<td>True if all elements of the parameter collection are present in the current collection</td>
</tr>
<tr>
<td>isEmpty</td>
<td>False if the collection contains no elements</td>
</tr>
<tr>
<td>notEmpty</td>
<td>True if the collection contains one or more elements</td>
</tr>
<tr>
<td>iterate(expression)</td>
<td>Expression is evaluated for every element in the collection</td>
</tr>
<tr>
<td>sort(collection)</td>
<td>The addition of all elements in the collection</td>
</tr>
<tr>
<td>exists(expression)</td>
<td>True if expression is true for at least one element in the collection</td>
</tr>
<tr>
<td>forAll(expression)</td>
<td>True if expression is true for all elements.</td>
</tr>
</tbody>
</table>
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
  ```

OCL Tools

- Il existe de nombreux outils qui permettent de vérifier la syntaxe, la sémantique et d'évaluer une expression OCL :
  - EMF MDT, [http://www.eclipse.org/modeling/mdt/?project=ocl](http://www.eclipse.org/modeling/mdt/?project=ocl)
  - Use 2.3, [http://www.db.informatik.uni-bremen.de/projects/USE](http://www.db.informatik.uni-bremen.de/projects/USE)
  - The Kent OCL library v1, [http://www.cs.kent.ac.uk/projects/ocl](http://www.cs.kent.ac.uk/projects/ocl)
  - Octopus OCL, [http://octopus.sourceforge.net](http://octopus.sourceforge.net)
  - Topcased, [http://www.topcased.org](http://www.topcased.org)
  - Kermeta, [http://www.kermeta.org](http://www.kermeta.org)
  - etc.

OCL Tools - Kermeta

- Use AOM to weave your static semantics into your metamodel!
- If the boolean statement is evaluated to false then the pre or post condition is violated and an exception
  ```
  ConstraintViolatedPre or ConstraintViolatedPost
  ```
  is raised.

Two methods to check the WFR:
- `checkInvariants`, check only the current model element
- `checkAllInvariants`, checks recursively the element being a containment link with the checked element
Outline – Metamodeling Process

1. Domain modeling *(abstract syntax)*
2. Model edition *(concrete syntax)*
3. Model serialization *(concrete syntax)*
4. Model checker *(static semantics)*
5. Simulator implementation *(behavioral semantics)*
6. Compiler implementation *(behavioral semantics)*

Metadata languages

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

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

Breathing life into metamodels

- A model
- Its metamodel
- Adding Operational Semantics to OO Metamodels

From Metamodels to Languages
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".

```plaintext
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
```

What is “meta”-executability?

- Basic CRUD Operations
- Merge, Composition...

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

"Program = Data Structure + Algorithm", Niklaus Wirth

```
Meta-Executability = Meta-Data + Meta-Actions
```

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

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

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

Kermeta workbench snapshot

Kermeta, a Kernel to Meta

Kermeta: a Kernel metamodeling language

- Seamless integration with Ecore and EMF-based tools
- Statically Typed
  - Generics, Function types (for OCL-like iterators)
- Object-Oriented
  - Multiple inheritance / dynamic binding / reflection
- Model-Oriented
  - Associations / Compositions
  - Models 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

<table>
<thead>
<tr>
<th>Mot-clé</th>
<th>Classe générique</th>
<th>Unicité</th>
<th>Ordre</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>Set&lt; T &gt;</td>
<td>Oui</td>
<td>Non</td>
</tr>
<tr>
<td>oset</td>
<td>OrderedSet&lt; T &gt;</td>
<td>Oui</td>
<td>Oui</td>
</tr>
<tr>
<td>bag</td>
<td>Bag&lt; T &gt;</td>
<td>Non</td>
<td>Non</td>
</tr>
<tr>
<td>seq</td>
<td>Sequence&lt; T &gt;</td>
<td>Non</td>
<td>Oui</td>
</tr>
</tbody>
</table>

Classes, opérations et 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 : Attributes & 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
  - Retour : pas de return ! On utilise la variable result
  - Pas de surcharge dans le langage (simplification)

EMOF ⇔ Kermeta

Assignment semantics
Fermetures & λ-fonctions: les itérateurs

- Effectuer une action $\forall e \in C$ each
  - $f$.each(n | stdio.write(n.toString + " ")
- Vérifier une condition $\forall e \in C$ 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

```java
operation fire() : String
source.owningFSM.currentState := target
result := output
```

```java
operation step(c : String) : String
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
 stddev.write("Output string : " + currentState.step(str))
  rescue (ex : FSMException)
    stddev.write(ex.toString)
  end
end
stdio.writeln("* END OF SIMULATION *")
```

```java
operation run() : Void
from var str : String until str = "exit"
loop
stdio.writeln("Enter an input string or 'exit' to exit simulation:")
stdio.writeln("current state is " + currentState.name)
str := stdio.read()
if str != "exit" then
  do
    stddev.write("Output string : " + currentState.step(str))
  rescue (ex : FSMException)
    stddev.write(ex.toString)
  end
end
stdio.writeln("* END OF SIMULATION *")
```
/**
 * Load a sample FSM from a xmi2 file
 */

operation loadFSM() : FSM is
do
var repository : EMFRepository = init EMFRepository.new
var resource : EMFResource = repository.createResource("../models/fsm_sample1.xmi", "../metamodels/fsm.ecore")
resource.load
// Load the fsm (we get the main instance)
result => resource.instances.one
end

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

⇒ Defined as an Ecore meta-model

Virtual Machine - Semantics

```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(pb : Boolean) is do penUp := pb
end operation rotate(angle : Integer) is do
turtleheading := (turtleheading + angle).mod(360)
end }
```
Map Instructions to VM Actions

• Weave an interpretation aspect into the metamodel
  • add an eval() method into each class of the LOGO MM


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

Handling control structures

• Block
• Conditional
• Repeat
• While

Operational semantics

```java
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
}

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
  - i.e. LOGO programs
- Visualize the result
  - Print traces as text
  - Put an observer on the LOGO VM to graphically display the resulting figure

Simulator

- Execute the operational semantics

Outline – Metamodeling Process

1. Domain modeling (abstract syntax)
2. Model edition (concrete syntax)
3. Model serialization (concrete syntax)
4. Model checker (static semantics)
5. Simulator implementation (behavioral semantics)
6. Compiler implementation (behavioral semantics)
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

```java
aspect class PenUp {
    compile (cxt: 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

Summary
Logo Summary (1)

- 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

Do It Yourself!

- Source code of the Logo demo:
  - [https://gforge.inria.fr/scm/viewvc.php/trunk/kmlogo_projects/?root=kermeta](https://gforge.inria.fr/scm/viewvc.php/trunk/kmlogo_projects/?root=kermeta)

- Kermeta ([http://www.kermeta.org/](http://www.kermeta.org/)):
  - reference documentation: [http://www.kermeta.org/documents](http://www.kermeta.org/documents)
  - formation supports: [https://gforge.inria.fr/scm/viewvc.php/integration/training_projects/?root=openembedd](https://gforge.inria.fr/scm/viewvc.php/integration/training_projects/?root=openembedd)

- More information:

- Webpage of this talk:
  - [http://www.combemale.fr/mde/](http://www.combemale.fr/mde/)

Logo Summary (2)

- Source code of the Logo demo:
  - [https://gforge.inria.fr/scm/viewvc.php/trunk/kmlogo_projects/?root=kermeta](https://gforge.inria.fr/scm/viewvc.php/trunk/kmlogo_projects/?root=kermeta)

- Kermeta ([http://www.kermeta.org/](http://www.kermeta.org/)):
  - reference documentation: [http://www.kermeta.org/documents](http://www.kermeta.org/documents)
  - formation supports: [https://gforge.inria.fr/scm/viewvc.php/integration/training_projects/?root=openembedd](https://gforge.inria.fr/scm/viewvc.php/integration/training_projects/?root=openembedd)

- More information:

- Webpage of this talk:
  - [http://www.combemale.fr/mde/](http://www.combemale.fr/mde/)

Based on Eclipse/EMF
Open Source

Download it now!

A statically typed object-oriented executable meta-language

- Home page
  - [http://www.kermeta.org](http://www.kermeta.org)

- Development page
  - [http://kermeta.gforge.inria.fr/](http://kermeta.gforge.inria.fr/)
References

• Eclipse Modeling website:
  • http://www.eclipse.org/modeling/

• EMF Refcardz "Essential EMF":
  • http://refcardz.dzone.com/refcardz/essential-emf

• Books:
  • "Eclipse Modeling Framework" (2nd edition), Budinsky et al, Addison Wesley
  • "Ingénierie Dirigée par les Modèles : des concepts à la pratique", Jézéquel et al., Ellipses