Introduction à l'ingénierie dirigée par les modèles

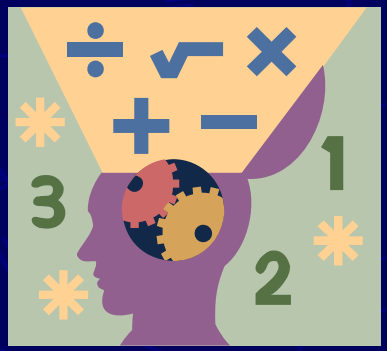
pierre-alain.muller@irisa.fr

Au menu

- Introduction à la metamodélisation
- Modélisation des langages
- Le langage Kermeta
- Le TP : metamodélisation d’un automate

Les choses et leurs représentations

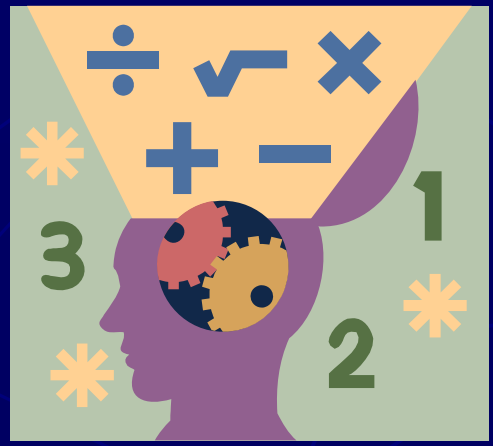
- Les choses
  - Réelles, virtuelles
  - Rares, chères, fragiles, dangereuses, inaccessibles, lointaines, trop nombreuses...
- Les concepts pour penser les choses
  - Plus facile, moins cher, moins dangereux...



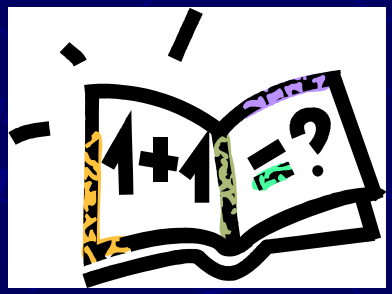
Représentation

Représentation mentale vs. physique

Représentation



Représentation



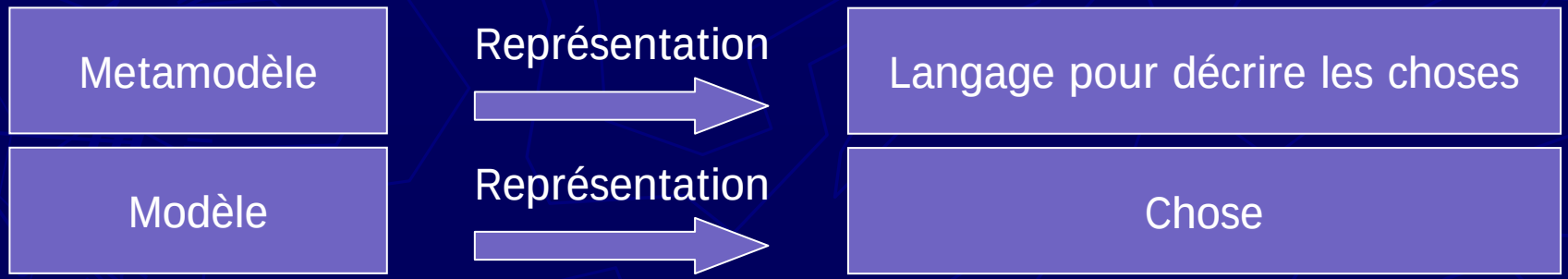
Modèle

Représentation

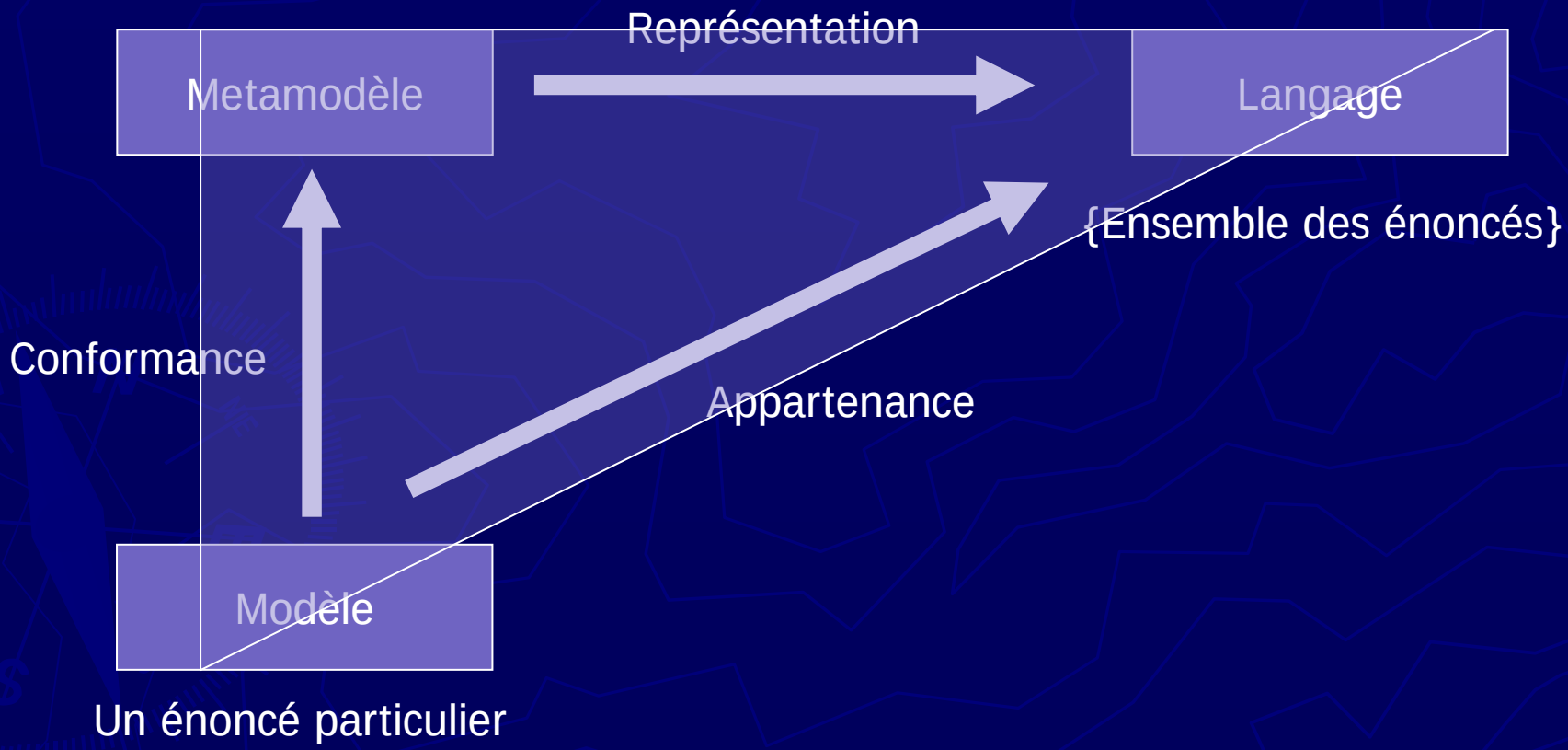
Chose

Le langage

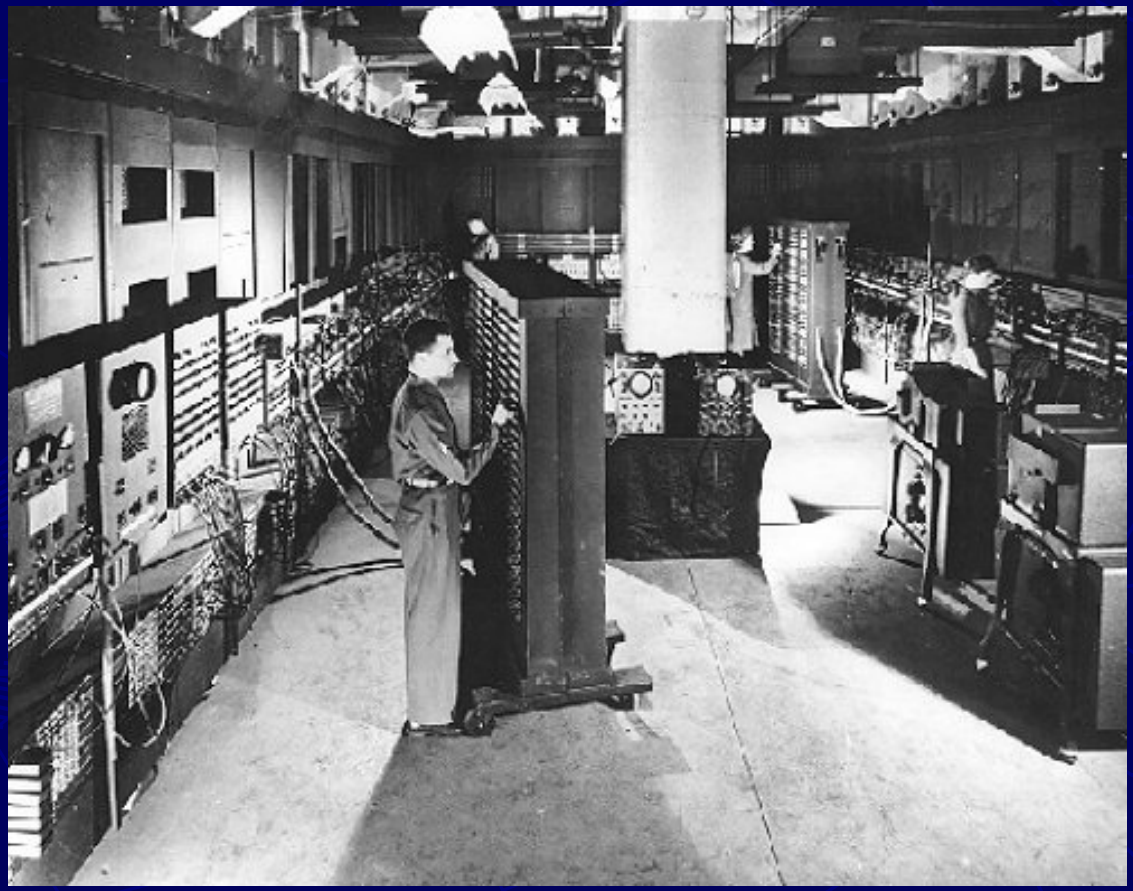
- Les mots pour dire les choses
  - Outil pour la manifestation externe de la pensée
  - Plus ou moins généraliste



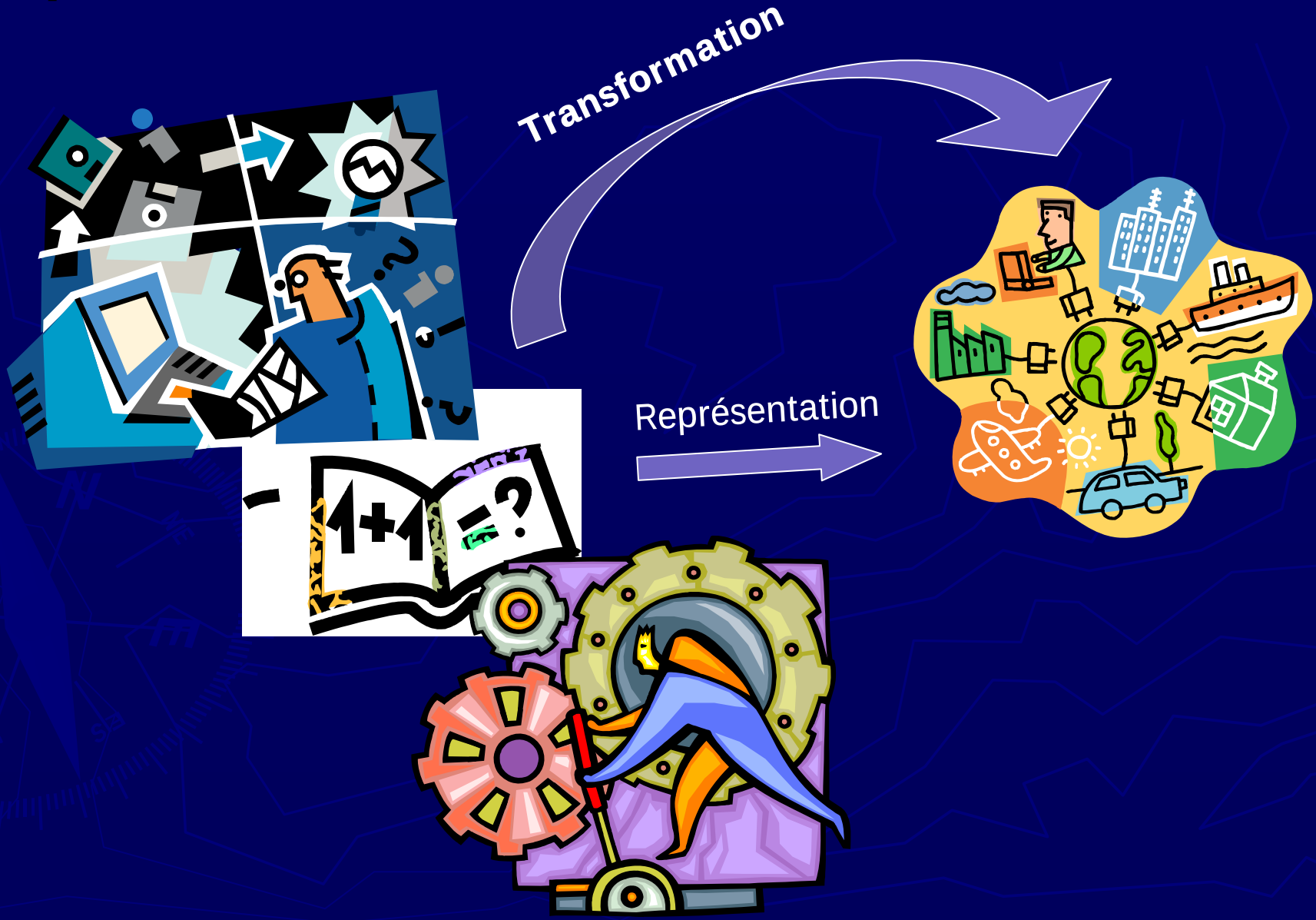
Relations modèles/metamodèles/langages



Arrivée de l’ordinateur

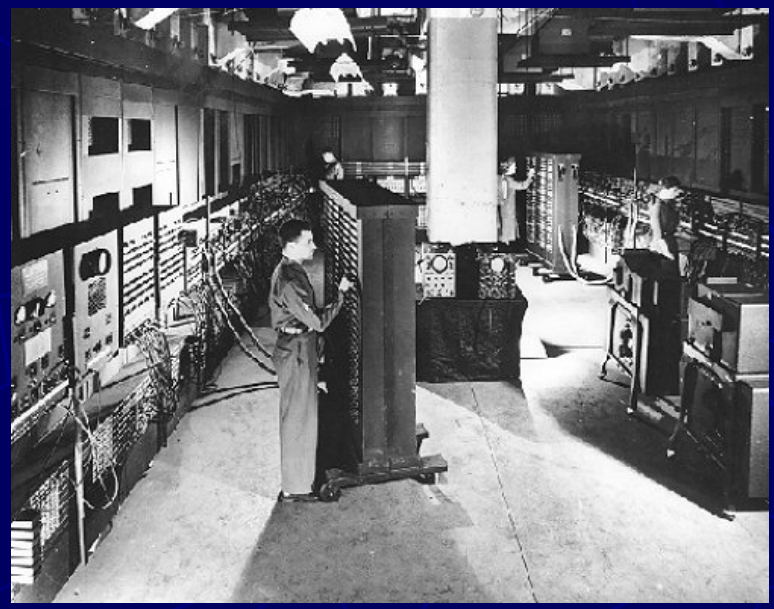


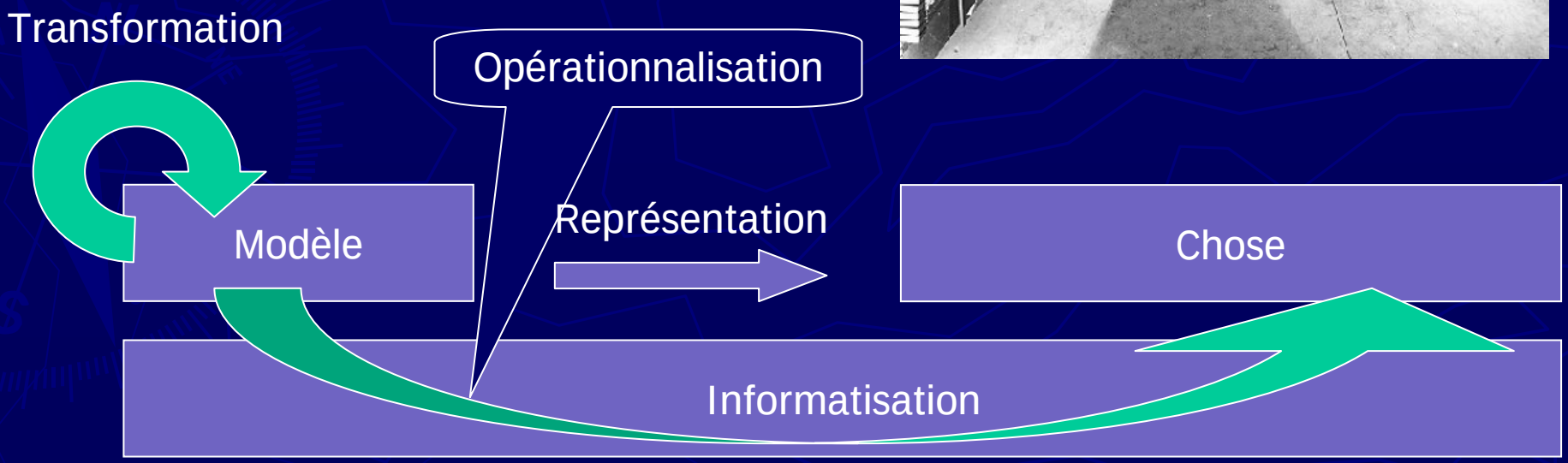
Opérationnalisation des modèles



L'ordinateur : une chose « spéciale »

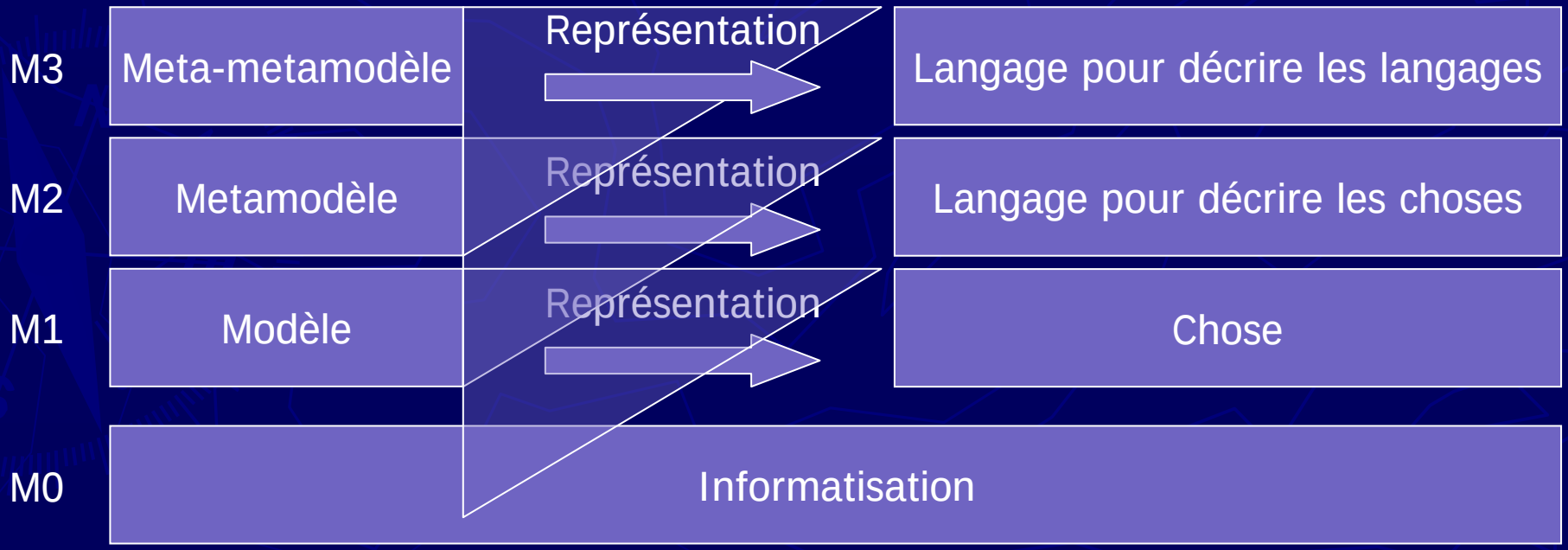
- Support
  - des choses (en partie)
  - des modèles
  - des transformations





Architecture de metamodélisation

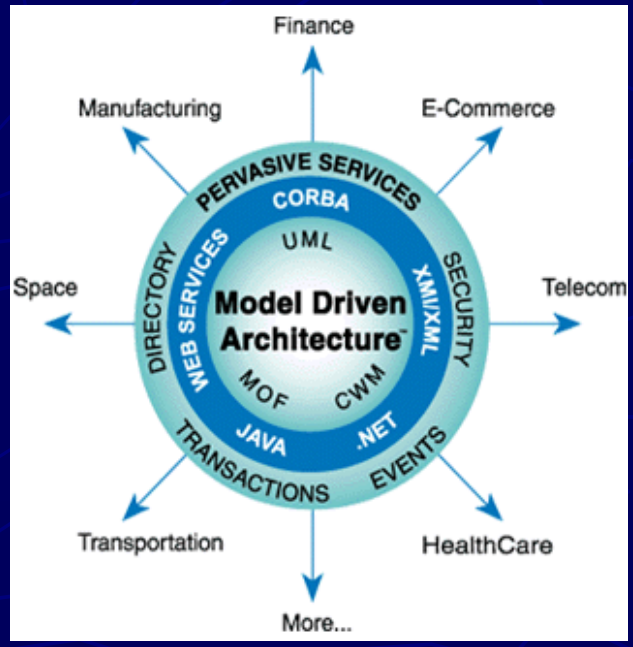
- MDA (Model-Driven Architecture)
  - 2000
  - www.omg.org/mda



Technologies du moment

M3	Meta-metamodèle	MOF Meta-Object Facility	EMOF	Ecore	Infra	
M2	Metamodèle	UML	Java	XMI	CWM	DSL ...

- Sphère MDA
  - Normalisation MOF et UML
- Sphère Eclipse
  - EMF/Ecore + Java

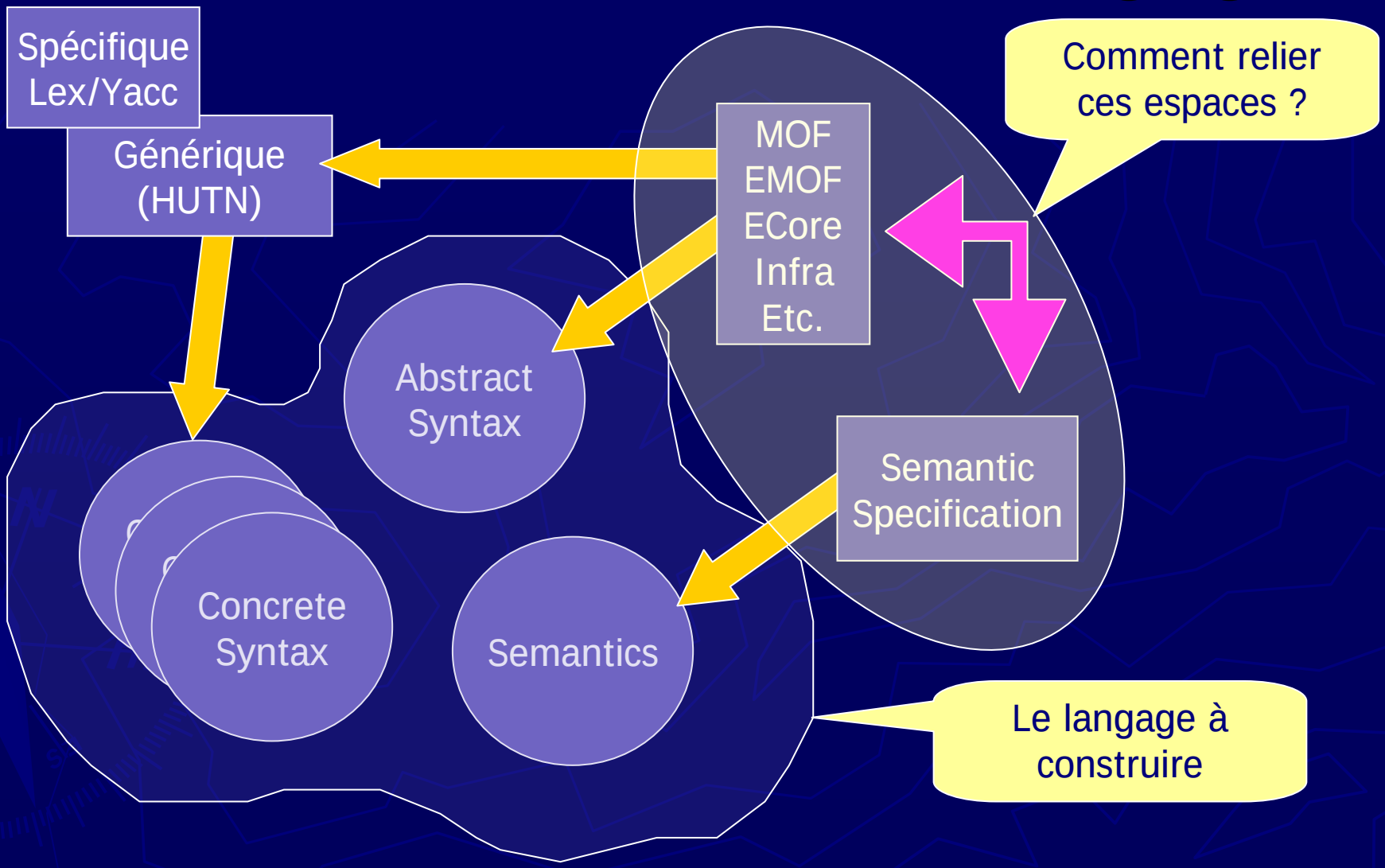


www.omg.org

Prolifération

- De metamodèles
- De langages spécifiques à l'IDM
  - Transformations: QVT, ATL, MTL...
  - Contraintes: OCL
  - Actions: AS, Xion...
- De langages connexes
  - XSLT, XMI...

Des metamodèles aux langages



Languages de metadonnées

- (E)MOF => Aspects structurels
  - classes, propriétés, références...
  - operations mais pas les méthodes
- Pas de support pour agir sur les modèles
  - Contraintes
  - Actions (CRUD sur des éléments)
  - Transformations
  - ...

Exemple typique (extrait de la spec du MOF)

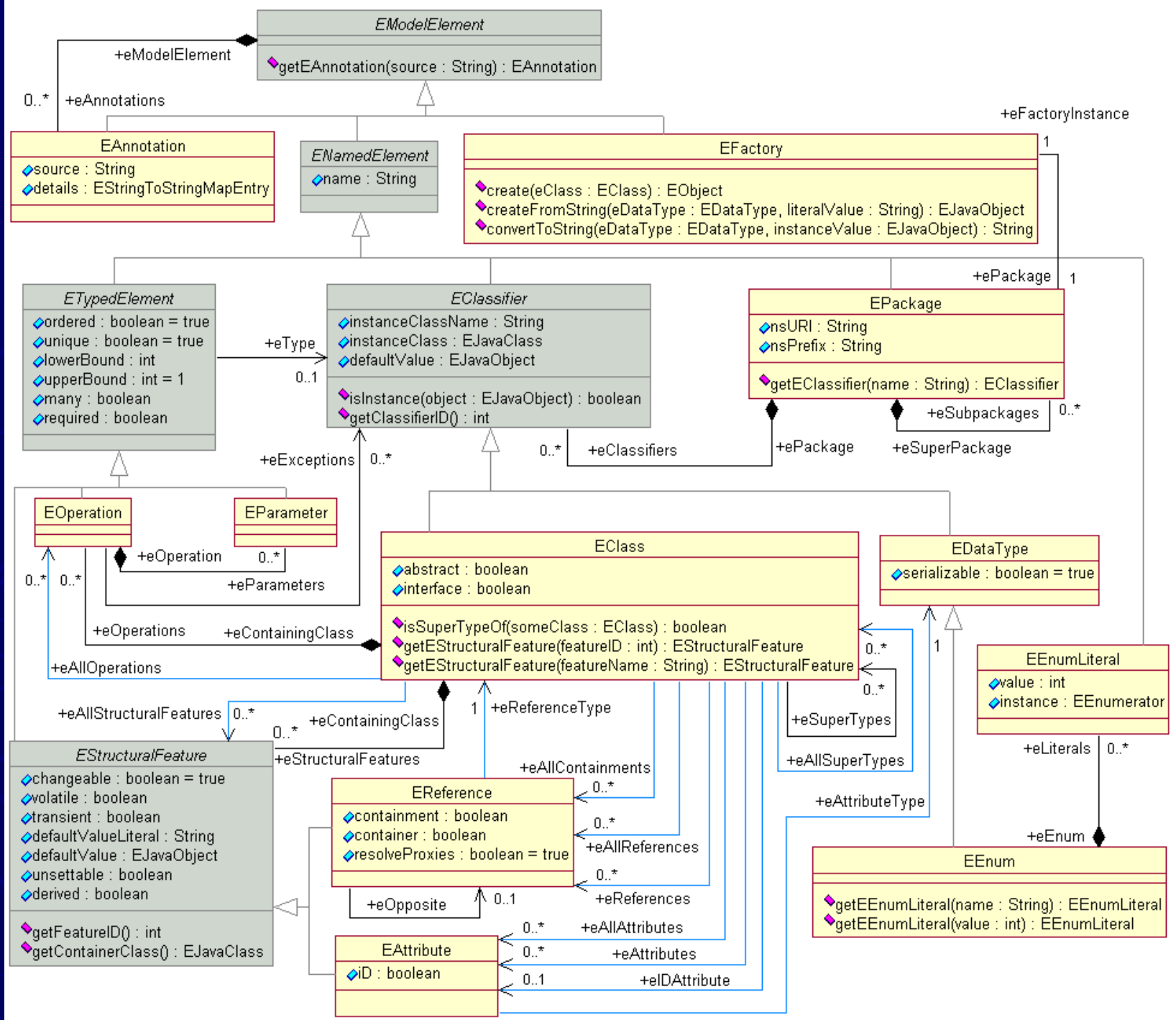
- *Operation* ***isInstance(element : Element) : Boolean***
  - "*Returns true if the element is an instance of this type or a subclass of this type. Returns false if the element is null*".

Spécification en langage naturel

```
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
```

Spécification opérationnelle

Metamodèle d'Ecore



Problèmes

- Multiplicité de langages différents
  - Différentes approches, niveaux de détail, systèmes de types
  - Interopérabilité difficile
- Metamodèles structurels
  - Pas de comportement en natif
  - Projections vers autres langages

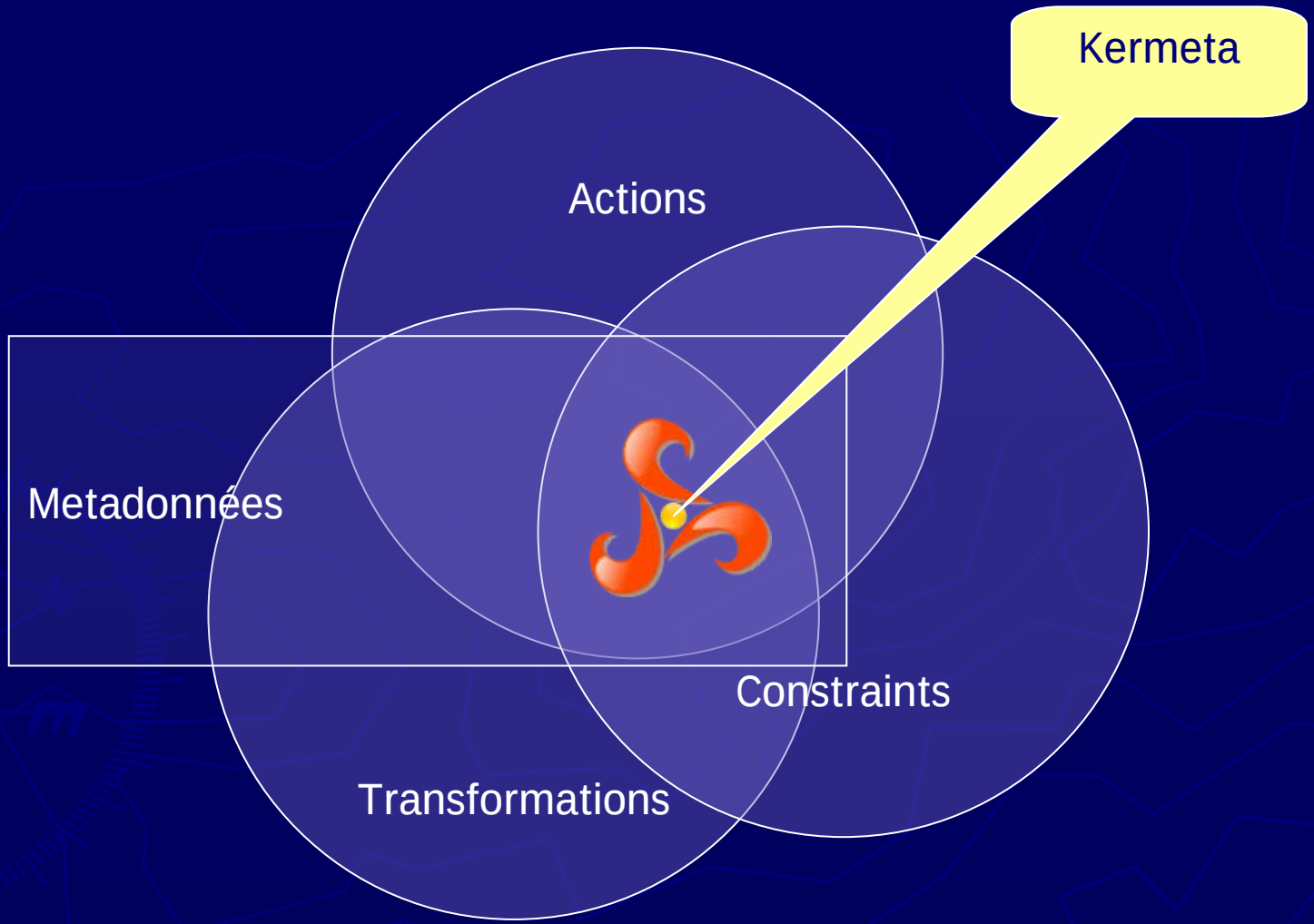
Vers la “meta”-exécutabilité ?

- Opérations de base CRUD
- Merge, Composition...

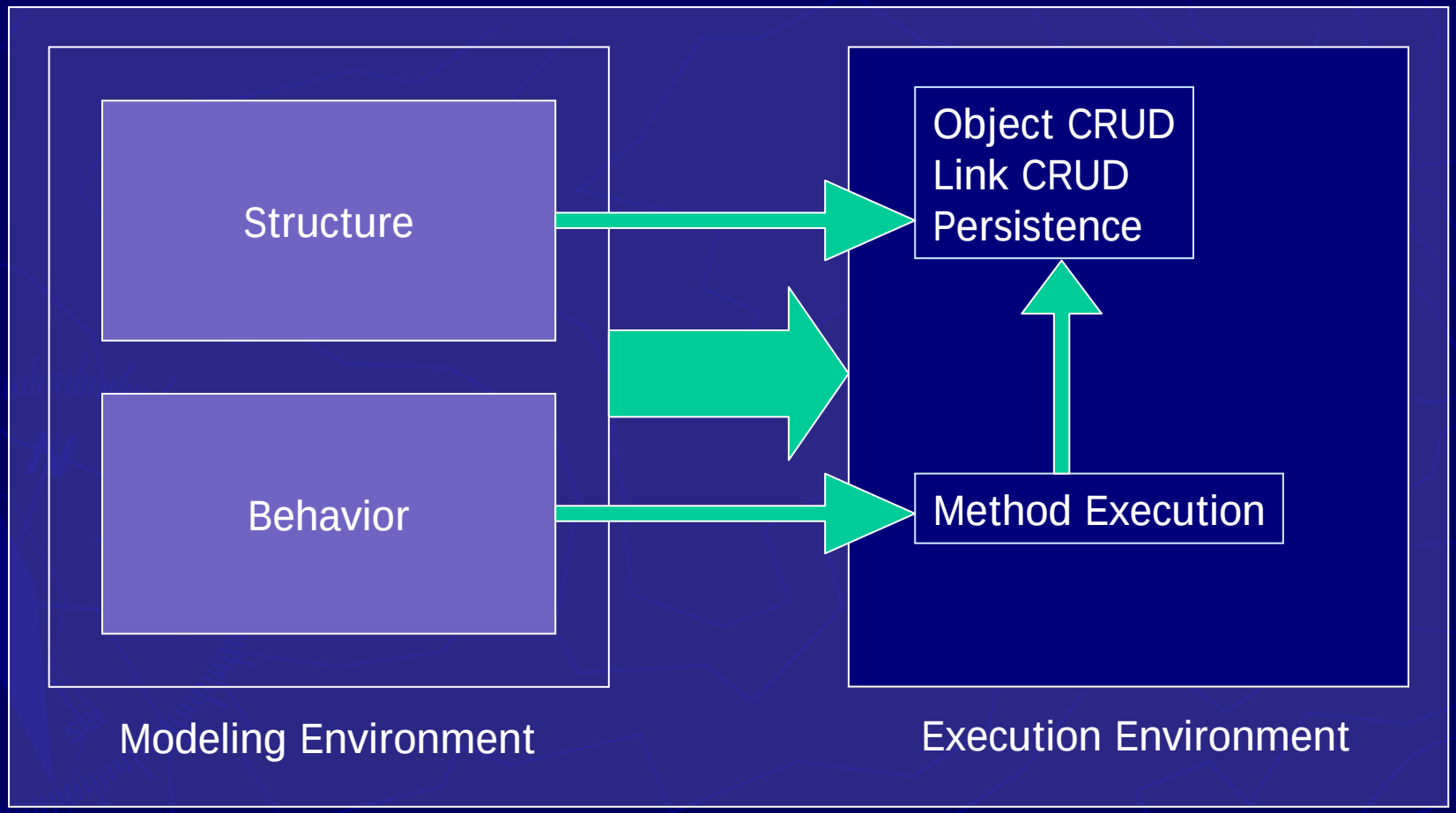
►Un programme qui manipule des éléments de modélisation

“***Program = Data Structure + Algorithm***”, **Niklaus Wirth**

Kermeta, un noyau Meta



Environment d'exécution



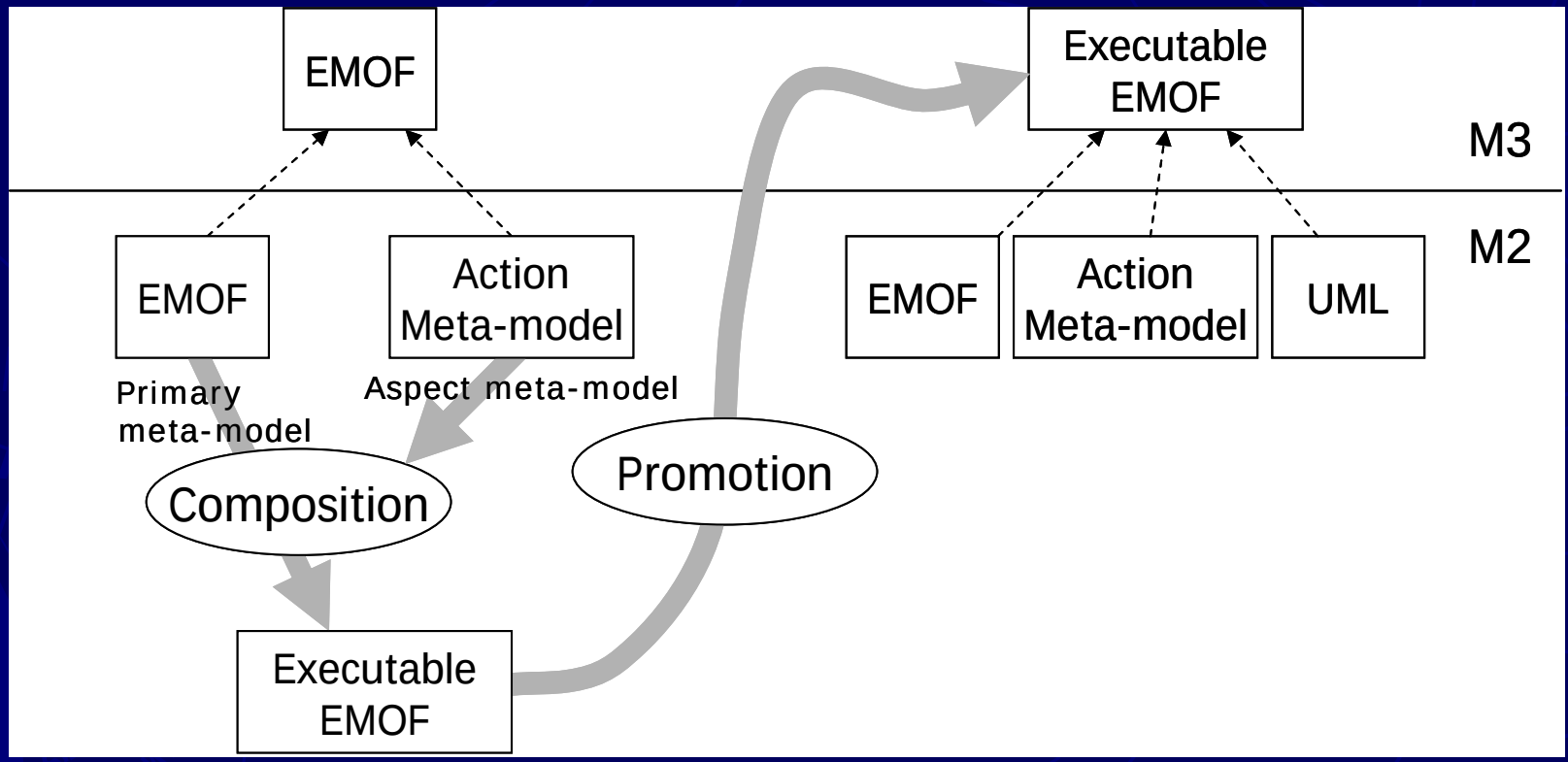
Comment construire Kermeta?

- Metadonnées + projections en Java
  - Solution de facilité, ou solution optimal ?
- Definir un nouveau language
  - *Éventuellement un Pivot*
- Etendre un language de metadonnées
  - Ecore++, EMOF++...

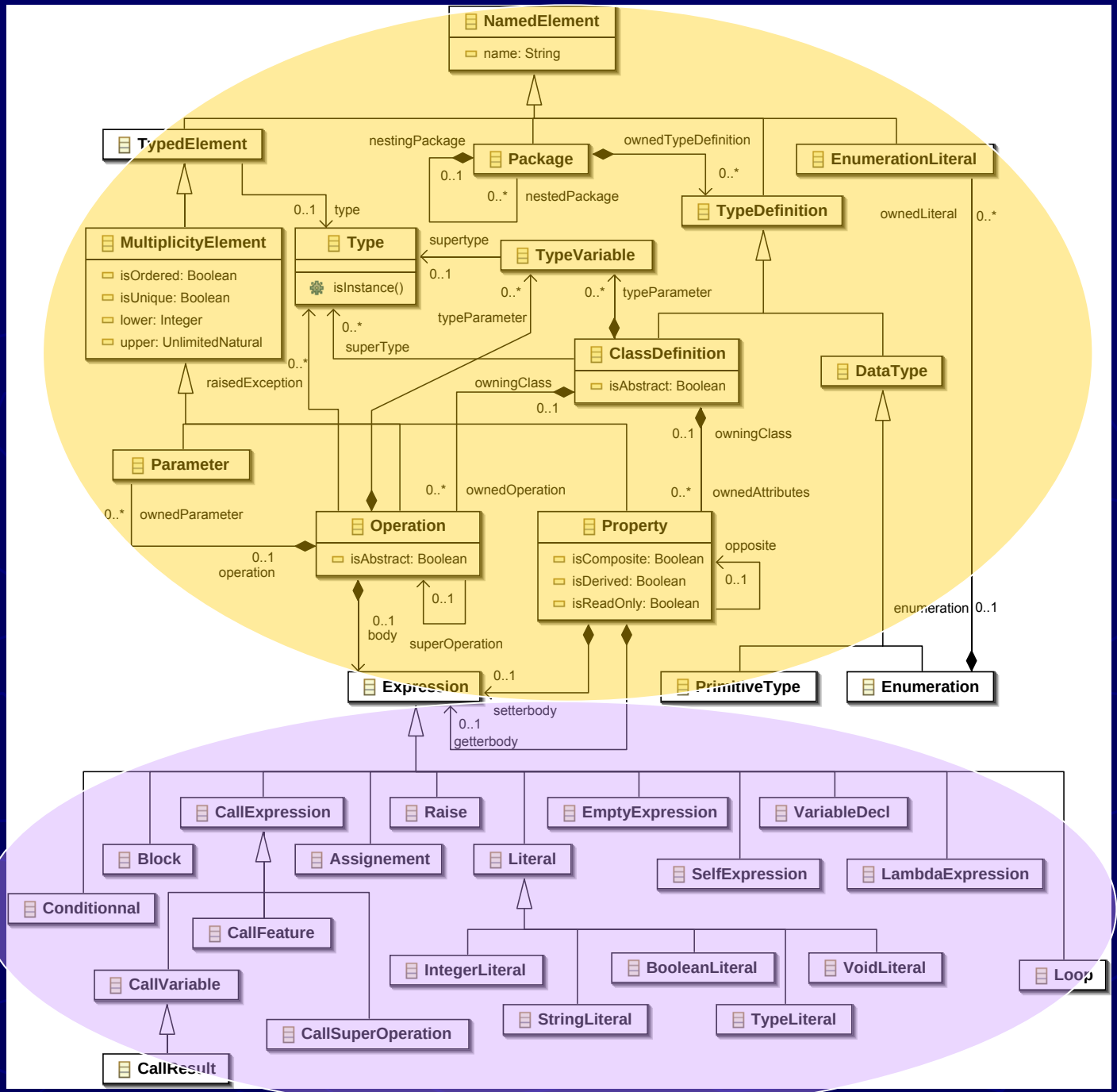
Choix pour Kermeta

- Compatible avec EMOF
- Orienté-objet
  - Héritage multiple
  - Rédefinition des méthodes et liaison dynamique
  - Réflexion (en lecture seule pour le moment)
- Typage statique
  - Généricité (classes et fonctions)
  - Type fonction

Construction par tissage d'aspect

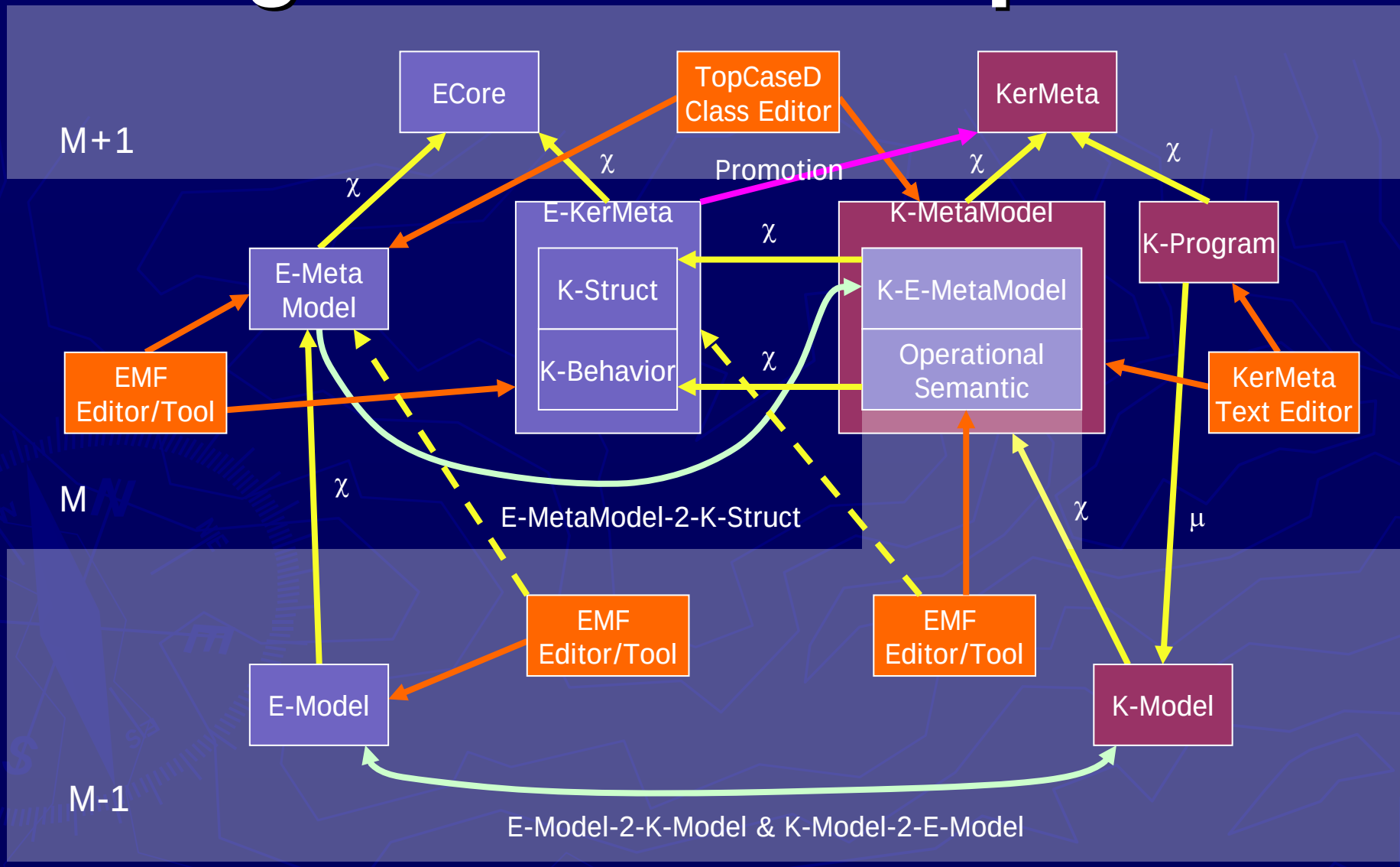


KerMeta Metamodel

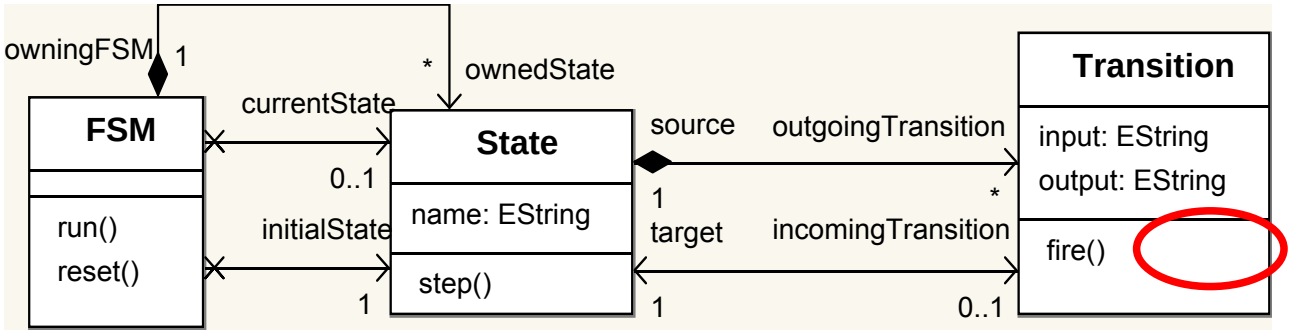


Intégration avec Eclipse / EMF

Integration avec Eclipse / EMF

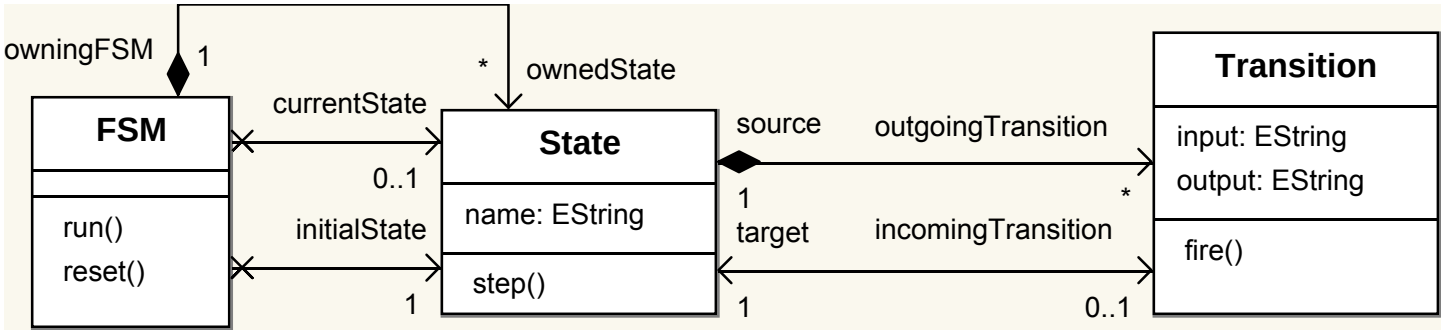


Example



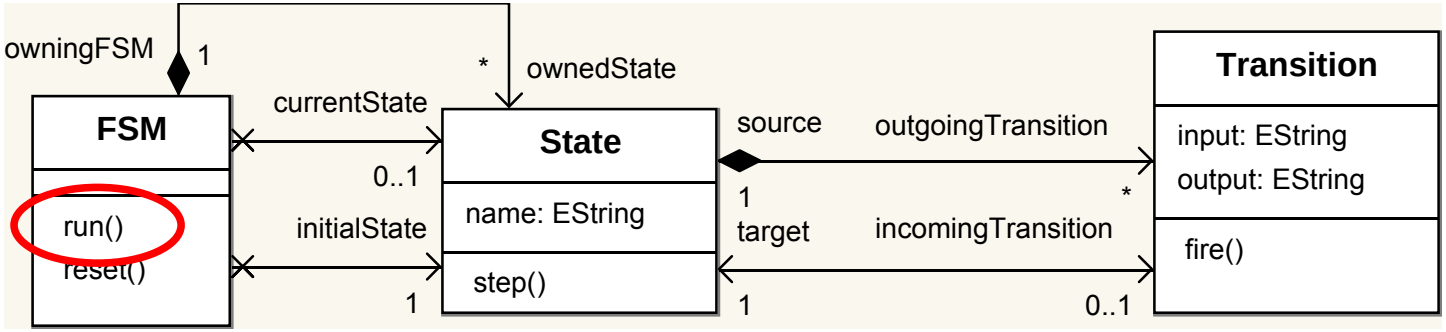
operation fire() : String

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



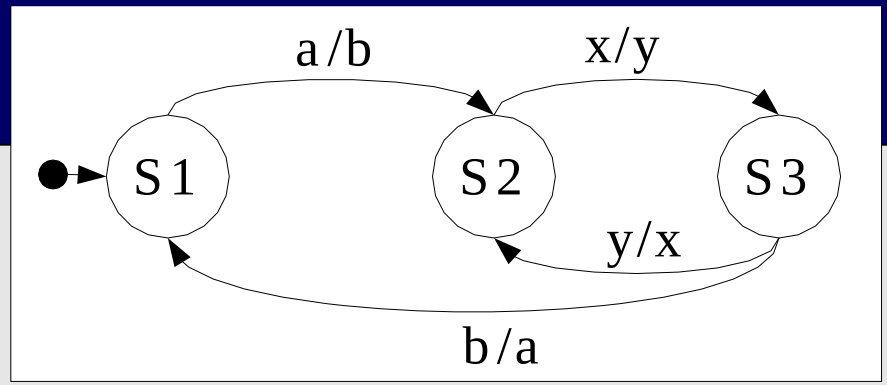
operation step(c : String) : String

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



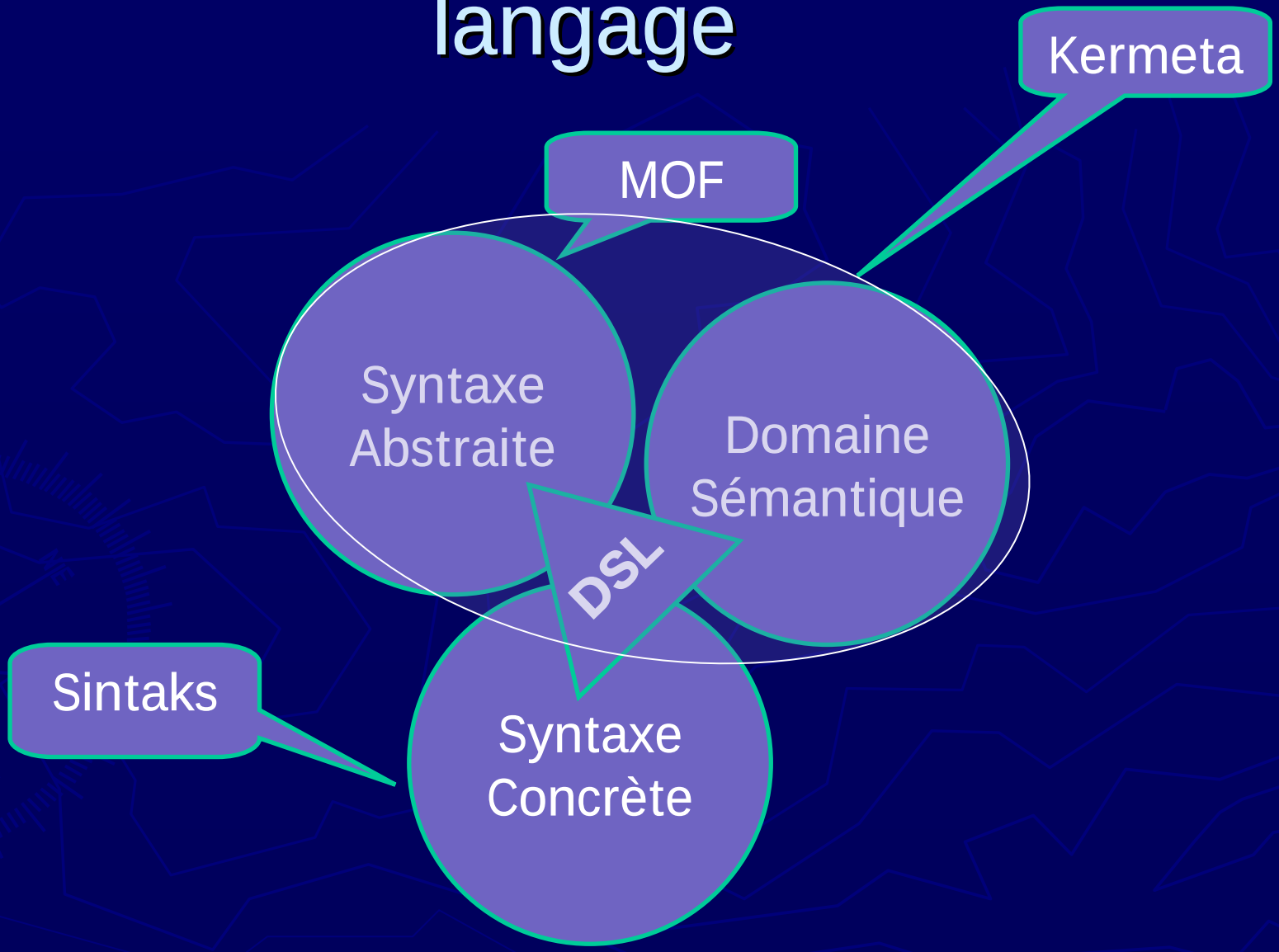

```
/**
 * Create a sample FSM
 */
operation createFSM() : FSM is do
 // The FSM
 result := FSM.new

 // Create the states of the FSM
 var s1 : State init State.new          s1.name := "s1"          result.ownedState.add(s1)
 var s2 : State init State.new          s2.name := "s2"          result.ownedState.add(s2)
 var s3 : State init State.new          s3.name := "s3"          result.ownedState.add(s3)

 // Create the transitions
 var t12 : Transition init Transition.new
 t12.input := "a"     t12.source := s1              t12.output := "b"    t12.target := s2
 var t23x : Transition init Transition.new
 t23x.input := "x"    t23x.source := s2  t23x.output := "y"  t23x.target := s3
 var t23y : Transition init Transition.new
 t23y.input := "y"    t23y.source := s2  t23y.output := "x"  t23y.target := s3
 var t31 : Transition init Transition.new
 t31.input := "b"     t31.source := s3              t31.output := "a"    t31.target := s1

 // Set the initial state
 result.initialState := s1
end
```

Retour sur la modélisation d'un langage



Motivation

- Comment créer des modèles conformes à des metamodèles ?
  - Par programme
  - Avec un éditeur réflexif, éditeur graphique
- Comment créer une représentation textuelle d'un modèle conforme à un metamodèle?
  - Syntaxe concrète générique
  - Syntaxe concrète spécifique
    - Grammarware : Parseur (text -> AST -> Model)
    - Modelware : text <-> Model

Kermeta

EMF

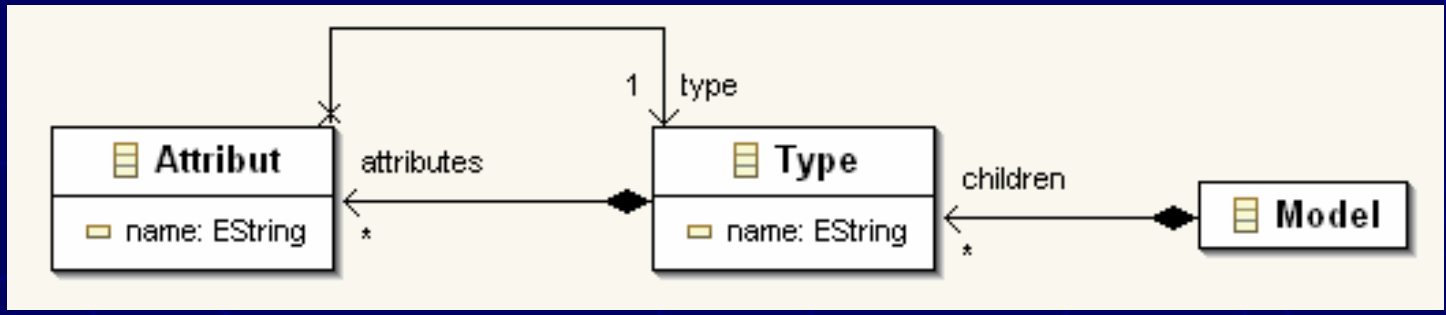
TopCaseD

HUTN

Sintaks

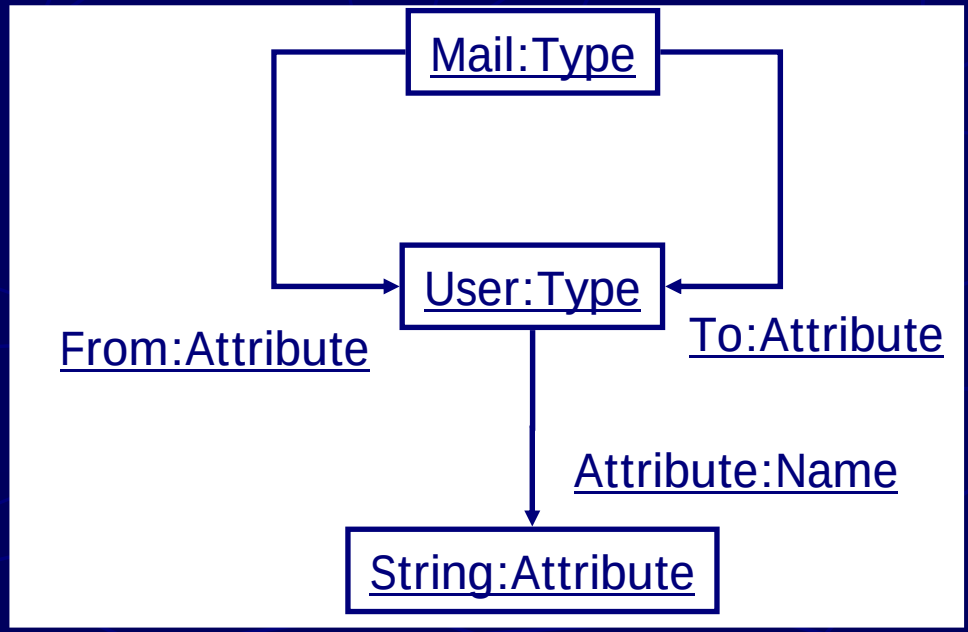
Exemple

- A very simple metamodel

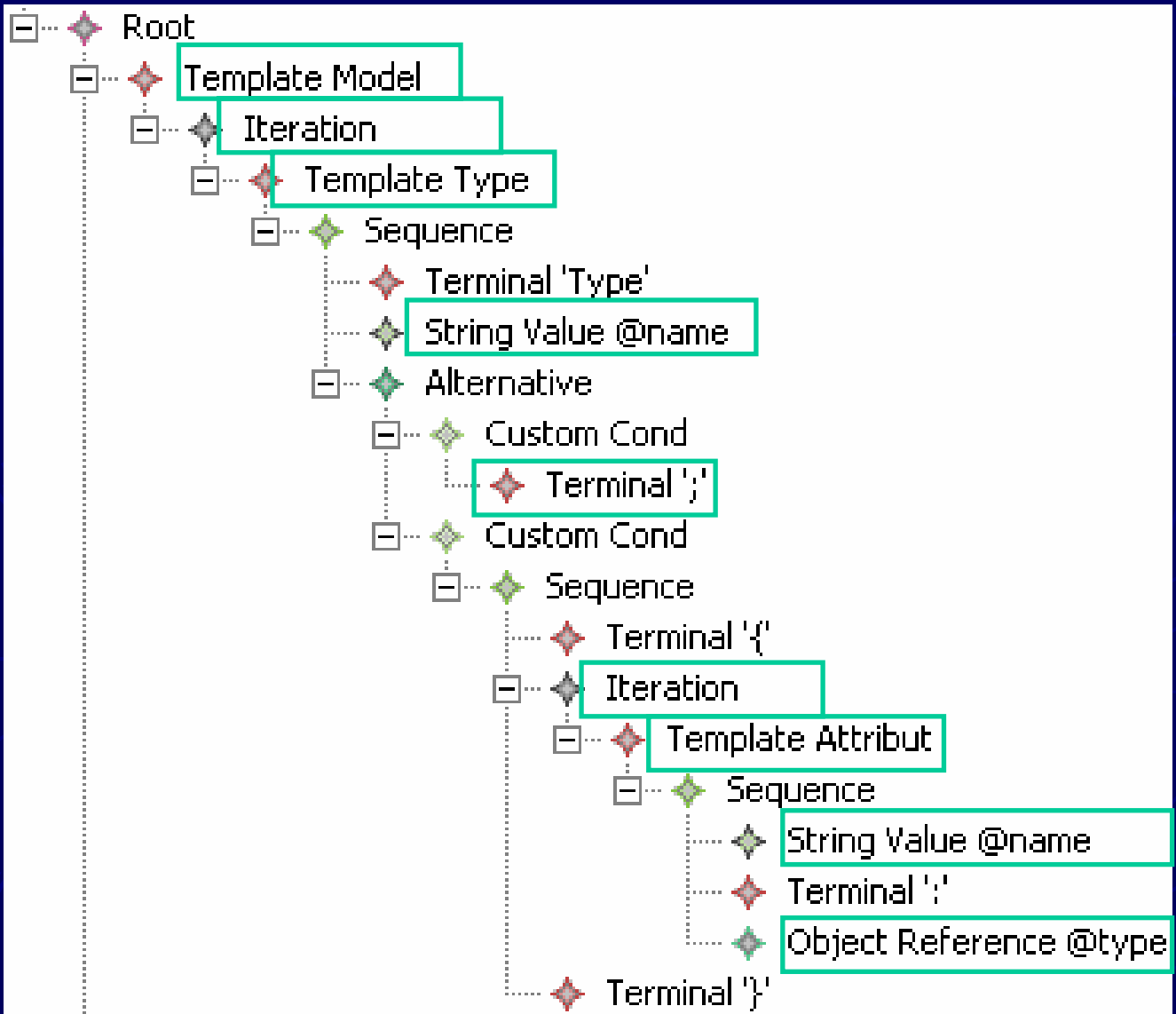


- Typical concrete syntax

```
Type Mail {
   From : User
   To : User
}
Type User {
   Name :
String
}
Type String;
```



Modèle de syntaxe concrète



Working on Model

Fill a collection

Working on Type

Extract the type name

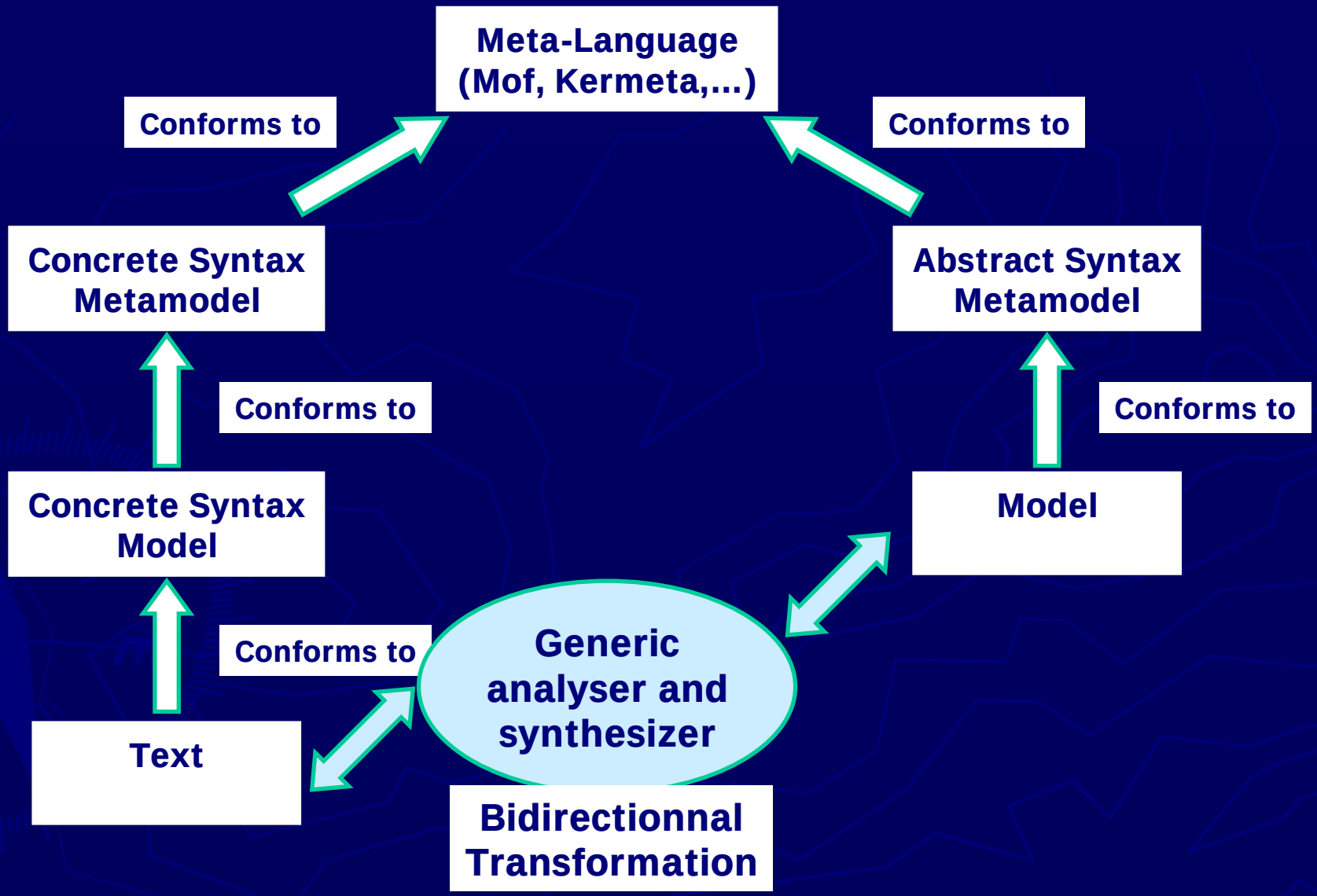
Just a semicolon

Fill a collection
Working on Attribute

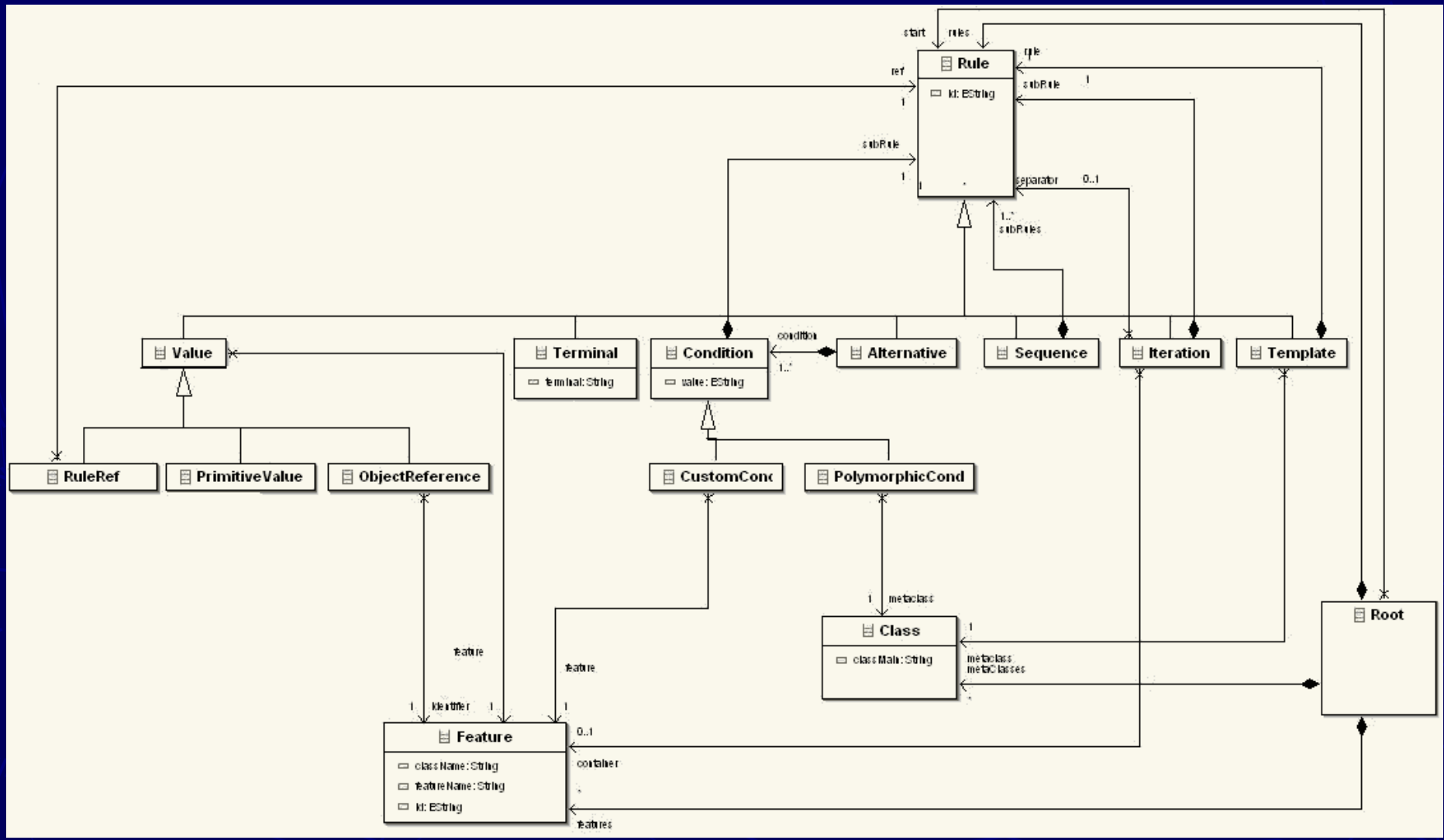
Extract the attribute name

Extract the ref to a type

Principe général



Metamodèle de syntaxe concrète



Découverte de Kermeta

- Google (« pierre alain muller home »)
- Rubrique « Teaching »
- Lien « TP Kermeta Polytech'Nice-Sophia »