Transformation de modèles avec Kermeta

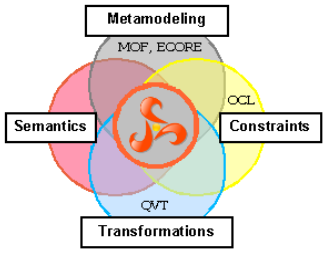
Ingénierie Dirigée par les modèles

Faculté des sciences / Université de Montpellier 2

Octobre 2013

Sommaire

Kermeta



Les figures et la plupart des exemples sont issus de la documentation de Kermeta.

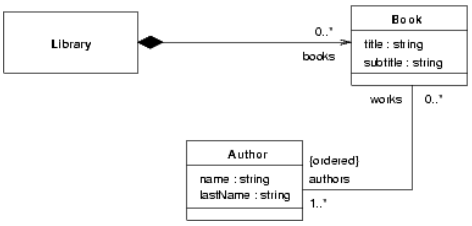
Kermeta

- MOF compliant (EMOF compliant to be precise)
- Model oriented
- Imperative
- Object-Oriented
- Aspect-Oriented
- Statically Typed

Kermeta 1, Kermeta 2

- Kermeta 1 : interprété
- Kermeta 2 : compilé (vers Scala)
- En TP : utilisation de la V1

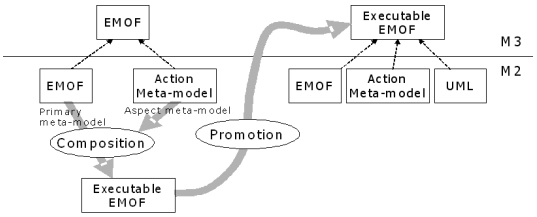
Model-Oriented : exemple



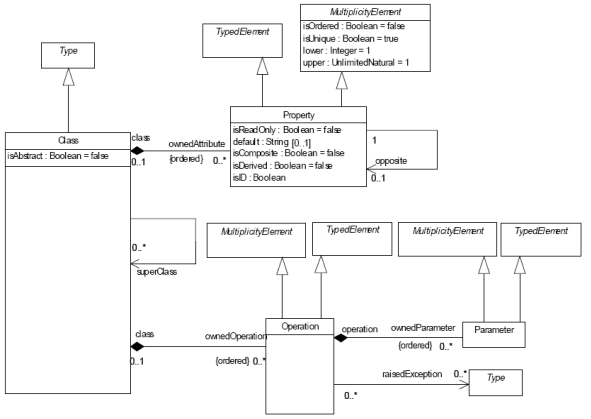
Model-oriented : exemple

```
class Library
{
    attribute books : set Book[0..*]
}

class Book
{
    attribute title : String
    attribute subtitle : String
    reference authors : oset Author[1..*]#works
}


class Author
{
    attribute name : String
    attribute lastName : String
    reference works : set Book[0..*]#authors
}
```

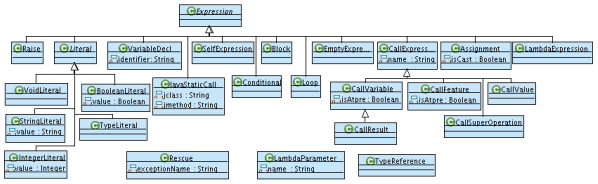

Construction de Kermeta



Construction de Kermeta, EMOF



Construction de Kermeta, Expression



Intégration de Kermeta à Eclipse

- Repose sur EMF
- Interpréteur
- Débugueur

Sommaire

Créer un métamodèle ECore

- Avec un éditeur graphique ou sous forme arborescente (éditeur réflexif)
- On obtient une structure statique :
  - des classes, des packages, des propriétés
  - des méthodes mais sans comportement opérationnel
  - des propriétés dérivées mais sans règle de dérivation opérationnelle

Créer un métamodèle Kermeta

- Structure : avec une syntaxe textuelle ou via ecore2kmt
- Dynamique : un langage OO, avec facilités de parcours de modèle
- Des méta-modèles avec une sémantique opérationnelle : méthodes avec comportement opérationnel, propriétés dérivées avec règle de dérivation opérationnelle, etc.

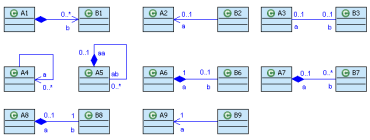
Créer un métamodèle avec Kermeta

Quelques éléments du langage

- attribute vs reference (attention : un objet n'est attribut que d'un composite)
- gestion des associations

```
class Library
{
    attribute books : set Book[0..*]
}

class Book
{
    attribute title : String
    attribute subtitle : String
    reference authors : oset Author[1..*]#works
}


class Author
{
    attribute name : String
    attribute lastName : String
    reference works : set Book[0..*]#authors
}
```


Attribute, reference, associations



```
class A1 {
    attribute b : B1[0..*] }
class B1 {}
class A2 {}
class B2 {
    reference a : A2 }
class A3 {
    reference b : B3#a}
class B3 {
    reference a : A3#b}
class A4 {
    reference a : A4[0..*]}
class A5 {
    attribute ab : A5[0..*]#aa
    reference aa : A5#ab}
```

```
class A6 {
    attribute b : B6#a}
class B6 {
    reference a : A6[1..1]#b}
class A7 {
    attribute b : B7[0..*]#a}
class B7 {
    reference a : A7#b }
class A8 {
    attribute b : B8[1..1]#a}
class B8 {
    reference a : A8#b}
class A9 {}
class B9 {
    reference a : A9[1..1]}
```


Propriétés dérivées

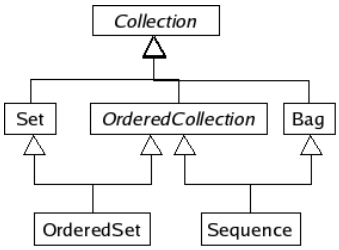
```
class B
{
    attribute period : Real
    property frequency : Real
        getter is do
            result result := 1/period
        end
        setter is do
            period := 1/value
        end
}
```


Types de base

- Integer
- String
- boolean

```
// Simple datatypes examples
var myVar1 : Integer init 10
var myVar2 : Integer
var myVar4 : String init "a new string value"
var myVar5 : boolean
```

Collections



Collections

Name	Description	Unique ?	Ordered ?
set	Represents an unordered collection of objects with no doublet (Set)	True	False
oset	Represents an ordered collection of objects with no doublet (OrderedSet)	True	True
seq	Represents an ordered collection of objects (Sequence)	False	True
bag	Represents a basic collection of objects (Bag)	False	False

Collections

```
class A {
attribute x1 : seq String[0..*]  // allows duplicates, ordered
attribute x2 : set String[0..*]  // doesn't allow duplicates, not
    ordered
attribute x3 : bag String[0..*]  // allows duplicates, not ordered
attribute x4 : oset String[0..*] // (default if no modifier) doesn'
    t allow duplicates, ordered
}
```


Héritage

- héritage simple et multiple. Mot-clef : `inherits`
- affectation conditionnée par le type : `?=`
- transtypage .asType(atype)
- isKindOf vs isInstanceOf
- `abstract`
- `operation` (définition) vs `method` (surcharge)
- redéfinition invariante sur le type de retour et les paramètres
- pas de surcharge

Divers

- déclarer une variable : `var foo : String // This is a variable declaration`
- les énumérations : `enumeration Size { small; normal; big; huge; }`
- block : `do ... end`
- `if ... then ... else ... end`
- boucle : `from ...until ... loop ... end`
- `result`
- `;`

Sommaire

Accès aux propriétés et références

```
class A6 {
    attribute b : B6[0..*]#a
}

class B6 {
    reference a : A6#b
}
```

```
var a6 : A6 init A6.new
var b6 : OrderedSet<B6>
b6 := a6.b // get the b attribute

var a6 : A6 init A6.new
var b6 : B6 init B6.new
a6.b.add(b6)// add b6 to the
    attribute b of A.

a6.b.removeAt(0) // Also valid : a6.
    b.remove(b6)

var a6 : A6 init A6.new
var b6 : B6 init B6.new
a6.b.add(b6)
assert(b6.a == a6) // this assertion
    is true.

var a6 : A6 init A6.new
var b6 : B6 init B6.new
a6.b.add(b6) // add ab6 to the
    attribute "b"
var a6c : A6 init b6.container()
assert(a6c.equals(a6)) // this
    assertion is true
```


Parcourir des collections : opérateurs ocl-like

```
aCollection.each { e |
    /* do something with each element e of this collection */
}
```

```
aBoolean := aCollection.forAll { e | /* put here a condition */
} // return true if the condition is true for all elements in the
    collection.
```

```
aCollection2 := aCollection.select{ e |
    /* put here a condition that returns true for elements that
        must be included in the resulting Collection */
}
```

```
aCollection2 := aCollection.reject{ e |
    /* put here a condition that returns true for elements that
        must be exclude in the resulting Collection */
}
```


Parcourir des collections : opérateurs ocl-like

```
aCollection2 := aCollection.collect{ e |
    /* put here an expression, for example e.name */
}
```

```
anObject := aCollection.detect{ e | /* a condition */} // returns
    an element (usually the first) that fulfills the condition.
```

```
anObject := aCollection.detect{ e | /* a condition */}
// returns an element (usually the first) that fulfills the
    condition.
```

```
aBoolean := aCollection.exists{ e | /* a condition */}
// returns true if at least one element fulfills the condition.
```

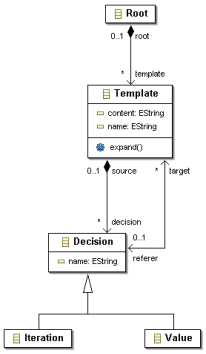

Sommaire

Créer une transformation de modèles

- revient à manipuler un ou des modèles sources, un ou des modèles cibles
- modéliser la transformation
- y appliquer toutes les règles du GL OO

Sommaire

Métamodèle d'exemple



Charger un modèle

```
@mainClass "root::TestCSLoading"
@mainOperation "main"
package root;
require kermeta
require "cs.ecore"
using kermeta::standard
using kermeta::persistence

class TestCSLoading
{
    operation initialize(uri : String, mm_uri : String) : Set<Object> is do
        /* Initialize the EMF repository */
        var repository : EMFRepository init EMFRepository.new
        /* Create an EMF Resource */
        var resource : Resource init repository.createResource(uri, mm_uri)
        /* Load the resource */
        resource.load
        /* Get the loaded __root__ instances (a Set<Object>) */
        result := resource // a resource is a collection of objects contained
    end
```


Parcourir les instances

```
operation main() is do
  var instances : Set<Object> init self.initialize("./test.cs", "./cs.ecore")
  instances.each{ o |
    if (o == void) then stdio.writeln("Void object!")
    else
      stdio.writeln("-----------------------------")
      stdio.writeln("Objet : " + o.getMetaClass.typeDefinition.qualifiedName
        + " ( " + o.getMetaClass.typeDefinition.ownedAttribute.size.toString+
            " attr.)" )
    end
    var template : cs::Template // Print instances which type is cs::Template
    if (cs::Template.isInstance(o))
    then
      template ?= o
      stdio.writeln("  name : " + template.name)
      stdio.writeln("  decision : " + template.decision.toString)
      stdio.writeln("  content : " + template.content) stdio.writeln("  referer
          : " + template.referer.toString)
    end
    // Print instances which type is cs::Decision
    if (cs::Decision.isInstance(o))
    then
      decision ?= o
      stdio.writeln("  name : " + decision.name)
    end
  }
}
```


Enregistrer un modèle

```
operation save(uri : String, mm_uri : String) is do
        /* Initialize the EMF repository */
        var repository : EMFRepository init EMFRepository.new
        /* Create an EMF Resource */
        var resource : Resource init repository.createResource(uri, mm_uri)
        /* Adds the rootInstance to the resource)
 resource.instances.add(theRootInstance)
        /* save the resource */
        resource.save()
    end
```