Formations OpenEmbeDD

Kermeta

Premier niveau

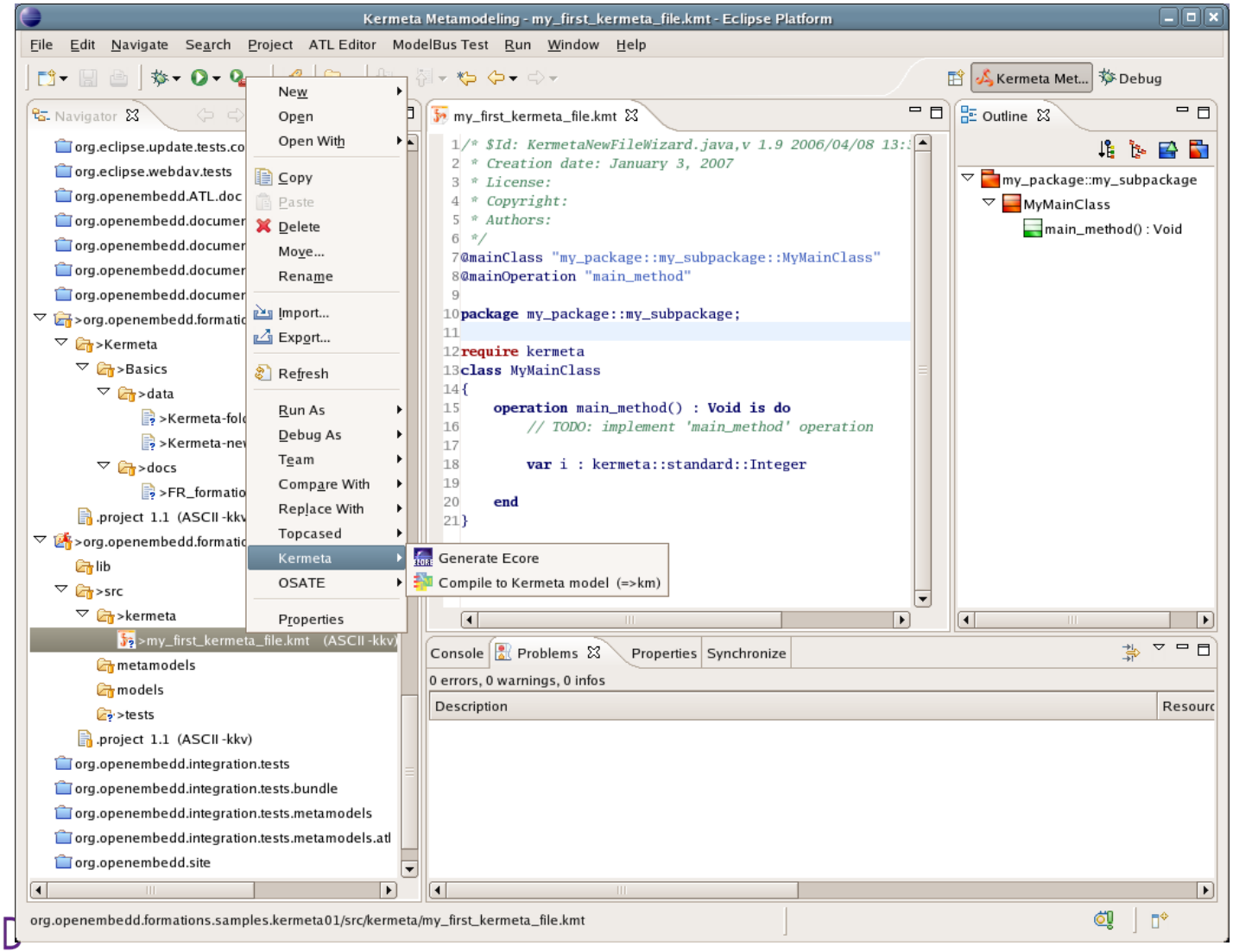
Plan

- **Environnement *Kermeta* dans Eclipse**
- **Le langage Kermeta**
  - Langage objet, syntaxe impérative, typage fort
  - Instructions de base, déclarations, généricité
- **Modèles**
  - Modèle, méta, et méta-méta
  - Associations, collections,...
  - Langage d'action pour les modèles (MOF)
  - Kermeta est aussi un méta-modèle
- **Fonctionnalités avancées / expérimentales**
  - Programmation par contrats

  Plus d'information : http://kermeta.org/documents/

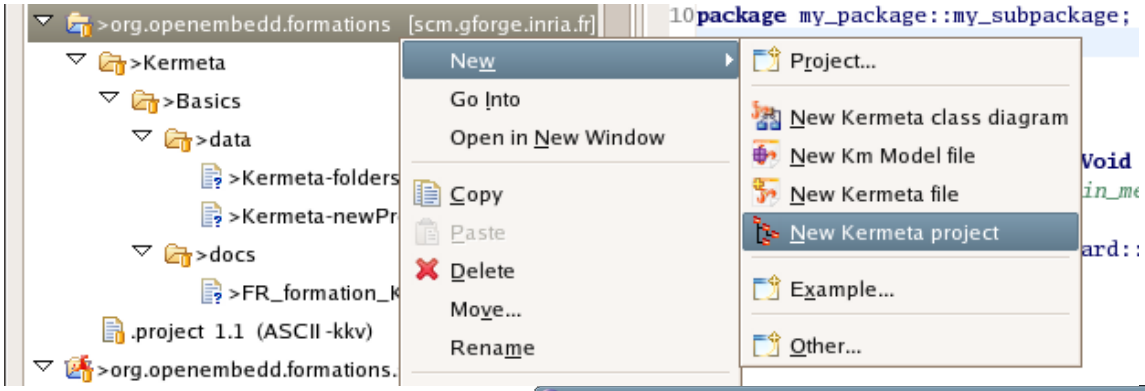
I – Environnement : Eclipse et *Kermeta*

- ***Kermeta*** **est totalement intégré à Eclipse :**

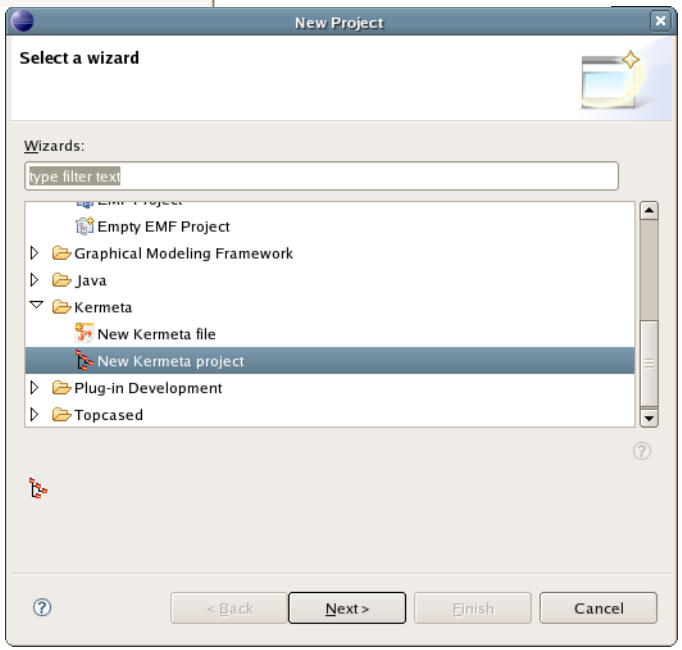


I – Environnement : nouveau projet

- Un projet kermeta se crée par appel à l'assistant via le menu contextuel :

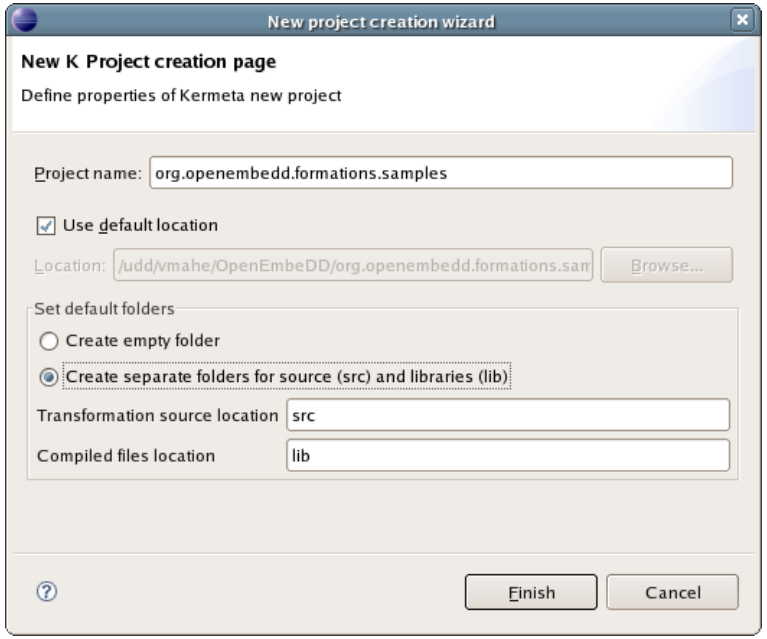


- ou bien par le menu général -> projet :

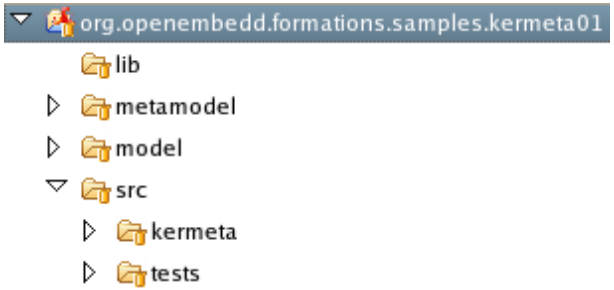


I – Environnement : projet Kermeta

- Nous préconisons un nommage et une structuration des projets Kermeta tels que ci-dessous :

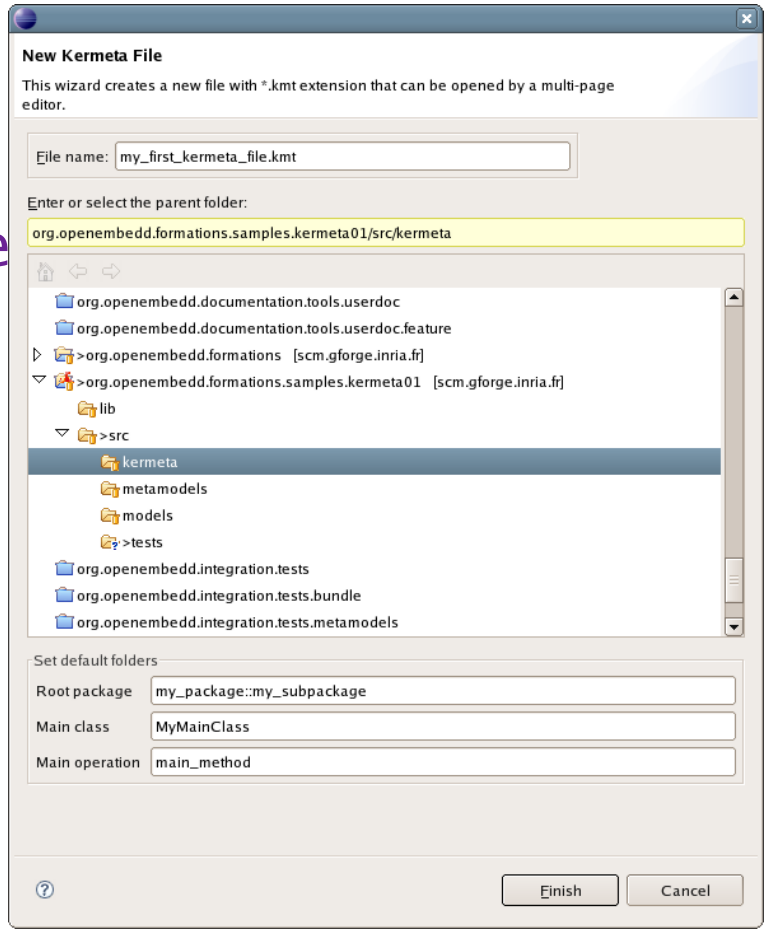


- Ainsi que :



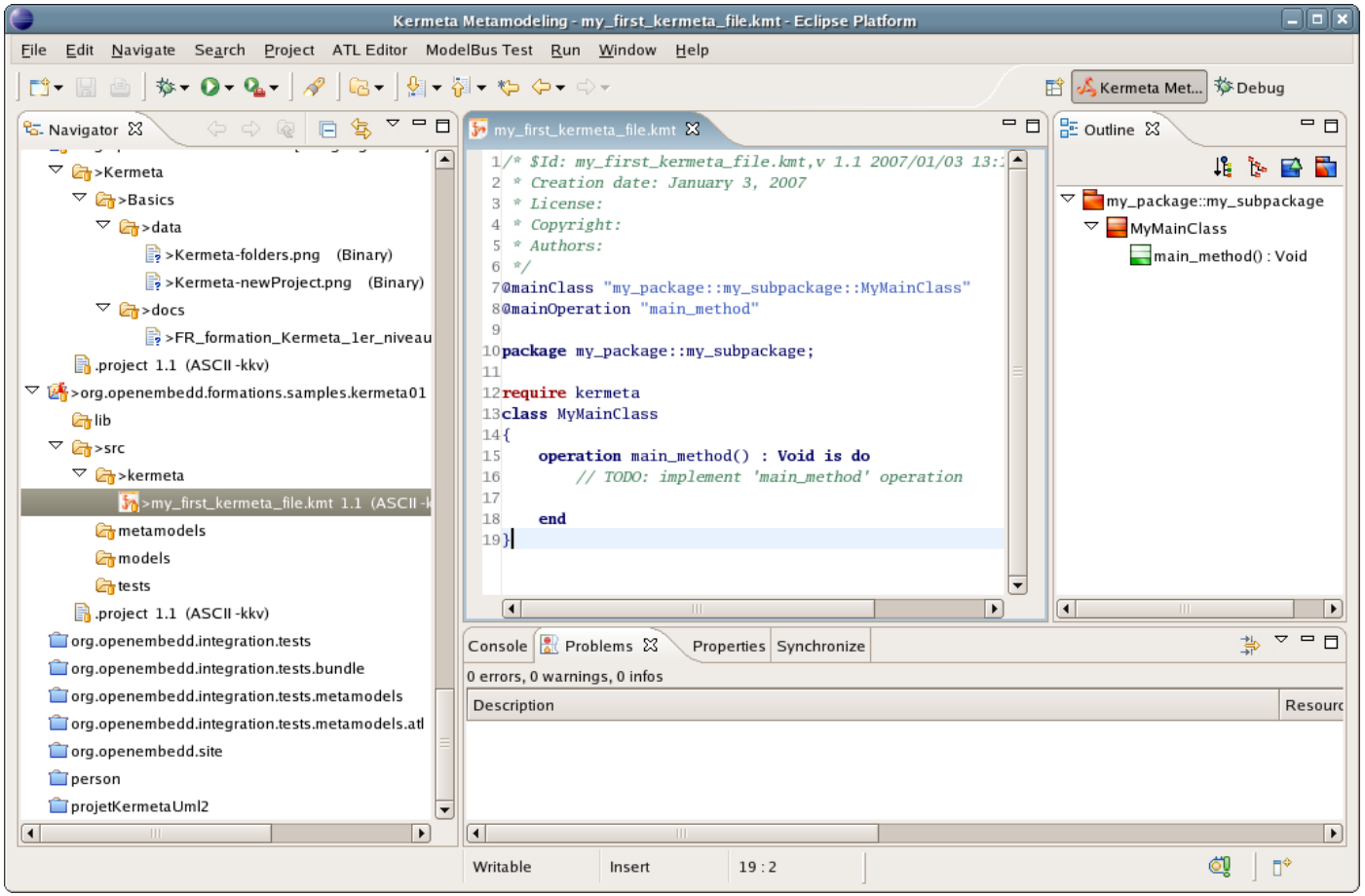
I – Environnement : nouveau fichier

- **Un assistant dédié :**
  - Nom
  - Emplacement dans le proje
  - Paquetage de référence
  - Classe principale
  - Méthode de lancement



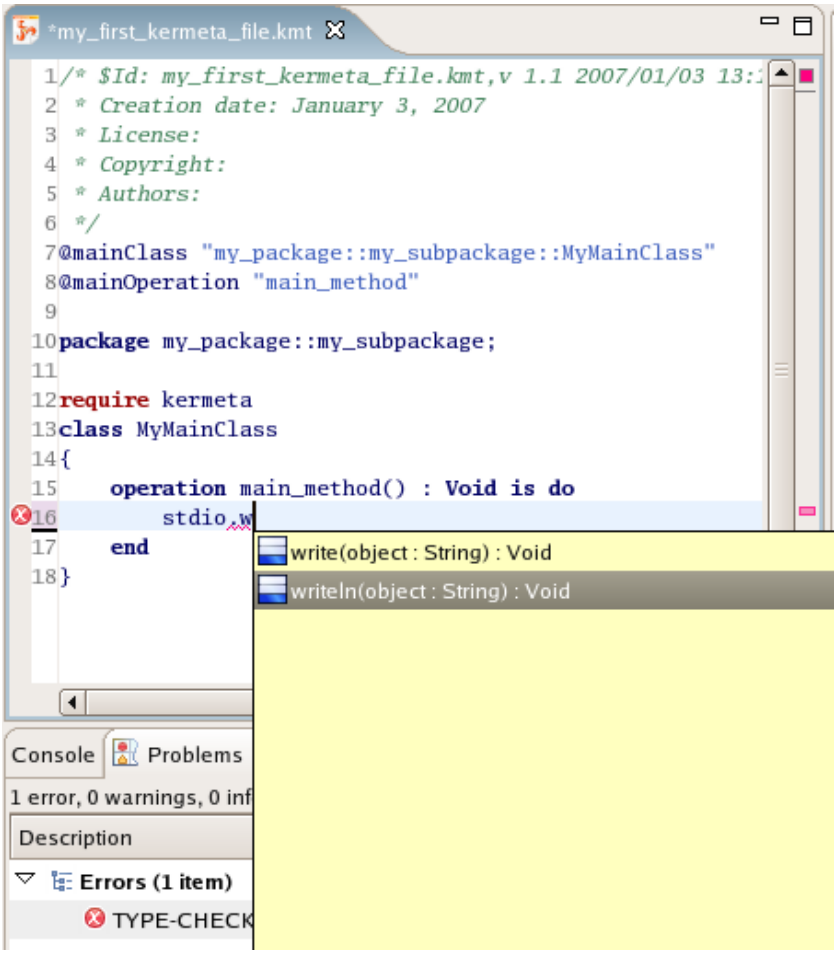
I – Environnement : édition d'un fichier

- L'assistant renseigne les principales informations et ouvre le nouveau fichier dans l'éditeur de texte :



I – Environnement : édition d'un fichier

- Ajoutons un peu de code au fichier :

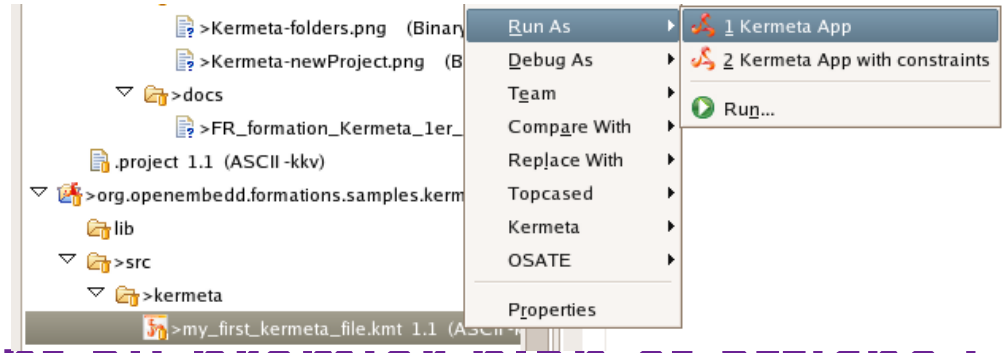


I – Environnement : exécuter Kermeta

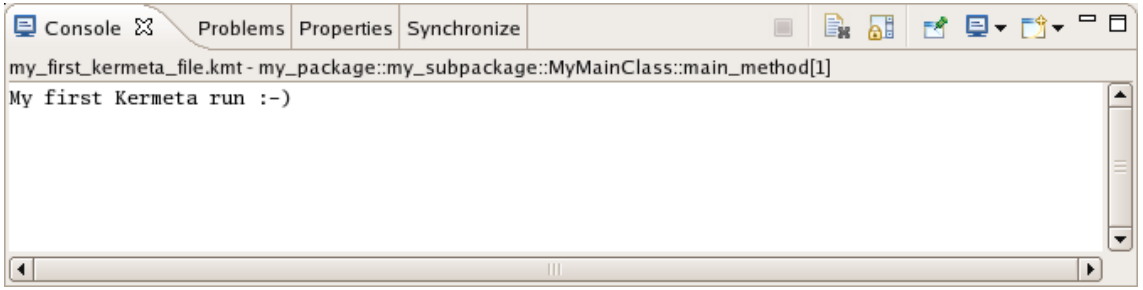
- Complétons :

```
13class MyMainClass
14{
15    operation main_method() : Void is do
16        stdio.writeln("My first Kermeta run :-)")
17    end
18}
```

- Puis lançons :

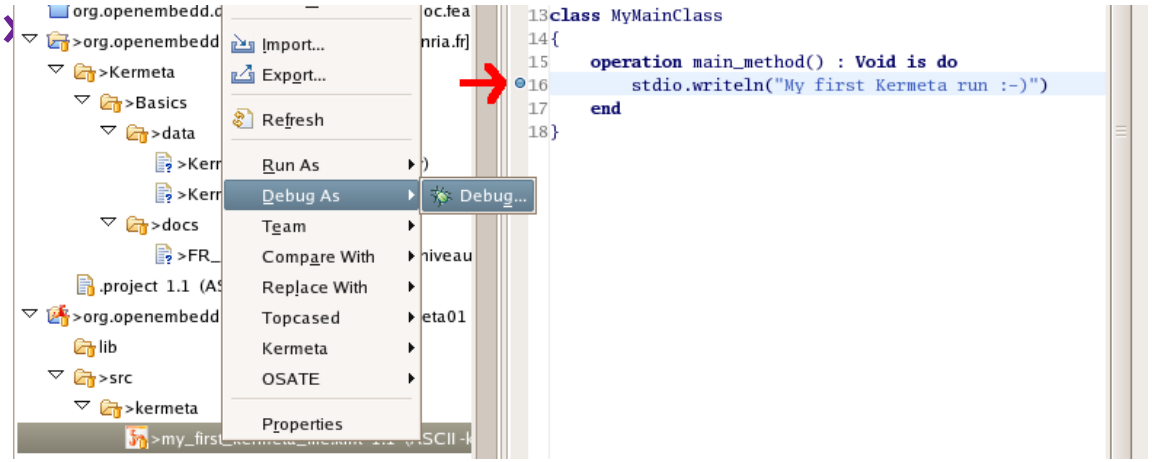


- La console vient au premier plan et affiche :

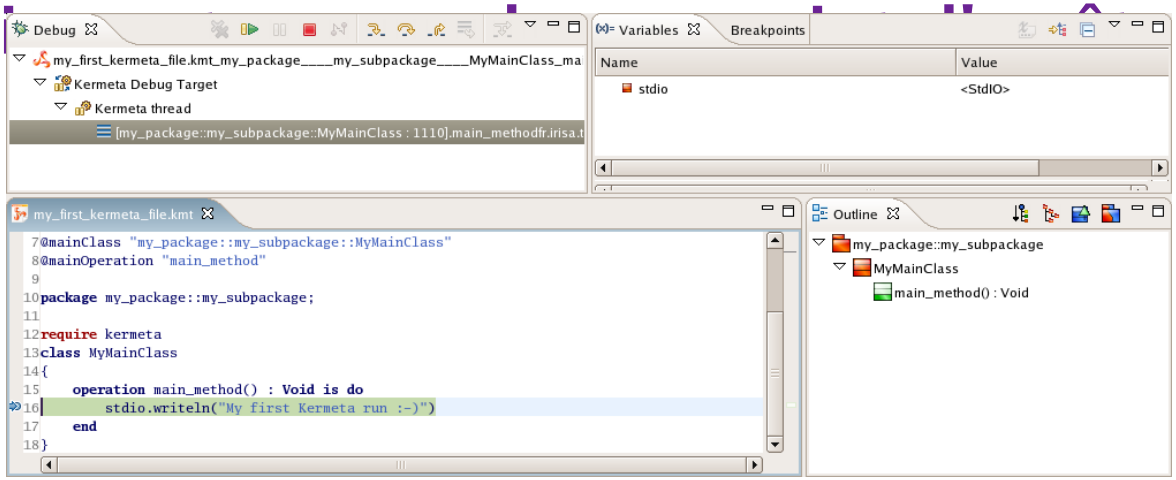


I – Environnement : déboguer Kermeta

- Poser un point d'arrêt puis lancer en mode « Debug »

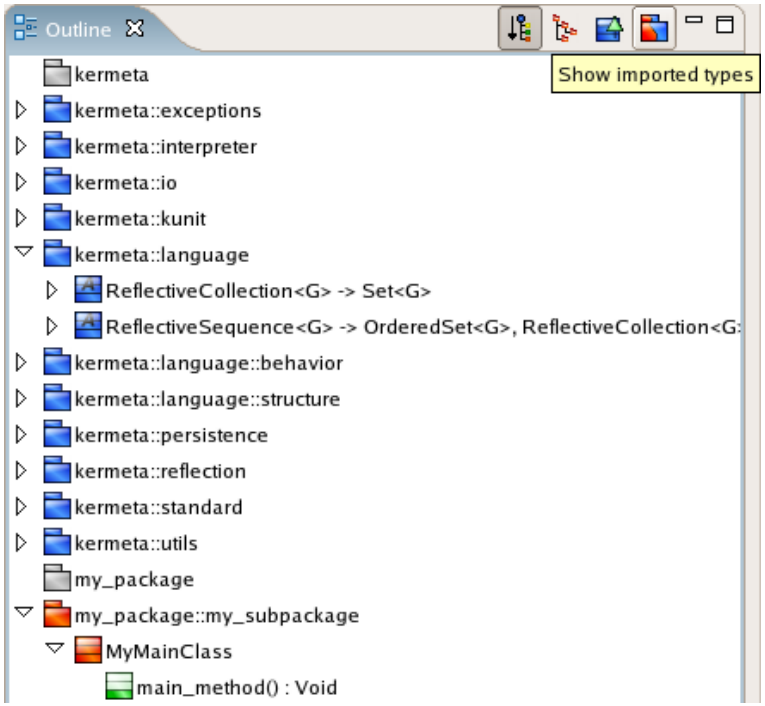


- L'exécut



I – Environnement : divers (a)

- « Outline » : affichage des librairies



I – Environnement : divers (b)

Tests unitaires avec Kunit :

tests_suite01.kmt

```
@mainClass "my_package::subpackage::MyTestSuite"
@mainOperation "runTests"

package my_package::subpackage;

require kermeta
require "platform:/resource/org.openembedd.formations.samples.kermeta01/src/kermeta/my_first_kermeta_file.kmt"

class MyTestSuite inherits kermeta::kunit::TestRunner
{
    operation runTests() : Void is do
        // Here, we run our first test case
        run(FirstTestCase)
        printTestResult
    end
}
class FirstTestCase inherits kermeta::kunit::TestCase
{
    reference a : kermeta::standard::Integer
    reference b : kermeta::standard::Integer

    method setUp() is do
        a := 0
        b := 1
    end

    method tearDown() is do     // TODO
    end

    // test methods name must begin with "test" to be processed
    operation testSuccessDemo() is do
        assertTrueWithMsg(b > a, "The testSuccessDemo should demonstrate a test success")
    end
    operation testFailureDemo() is do
        assertTrueWithMsg(a > b, "The testFailureDemo should demonstrate a test failure")
    end
    operation testErrorDemo() is do
        var i : kermeta::standard::Integer init 1/0
        assertTrueWithMsg(a > (b/a), "The testErrorDemo should demonstrate a error interception")
    end
}
```

II – Langage Kermeta : généralités

- C'est un langage Objet

```
class MyMainClass
{
    operation main_method() : Void is do
        stdio.writeln("My first Kermeta run :-)")
    end
}
```

- Sa syntaxe est impérative
- Le code bénéficie d'un typage fort, vérifié à la volée
- Kermeta offre la généricité :

```
class Queue<G>
{
     reference elements : oset G[*]
     operation enqueue(e : G) : Void is do
         elements.add(e)
     end
     operation dequeue() : G is do
         result := elements.first
         elements.removeAt(0)
     end
}
```

II – Langage Kermeta : généralités

- **Héritage multiple :**

```
abstract class AText
{
    operation addOp(textToAdd : kermeta::standard::String) is abstract
}
class LeftHand inherits AText
{
        reference text  : kermeta::standard::String
        method addOp(textToAdd : kermeta::standard::String) is
                do
                        if text != void then
                                text.append(textToAdd)
                        else
                                text := textToAdd
                        end
                end
}
class RightHand inherits AText
{
        reference text  : kermeta::standard::String
        method addOp(textToAdd : kermeta::standard::String) is
                do
                        if text != void then
                                textToAdd.append(text)
                                text := textToAdd
                        else
                                text := textToAdd
                        end
                end
}
class CapitalText inherits LeftHand, RightHand
{
        method addOp(textToAdd : kermeta::standard::String) from LeftHand is
                do
                        super(textToAdd)
                end
}
```

II – Langage Kermeta : généralités

- **Éléments de syntaxe :**

```
package my_package::subpackage;

require kermeta

class SyntaxClass
{
    // composition attributes
    attribute myAtt : X
    // pointer-like attributes
    reference myObj : X
    // affectation to an "attribute" deletes former
container attribute
    operation main() : Void is do
        // temporary variable declaration +
initialization
        var v1 : SyntaxClass init SyntaxClass.new
        var v2 : SyntaxClass init SyntaxClass.new
        var anObj : X       // declaration without
initialization
        anObj := X.new      // affectation with a new X
object

        v1.myAtt := anObj
        // v& has an attribute
        stdio.writeln(v1.myAtt.toString)

        v2.myAtt := v1.myAtt            // transfert of
 "anObj" from v1 to v2
        // v1 has loose its attribute (print <void>)
        stdio.writeln(v1.myAtt.toString)
    end
}
class X
{
    method toString() : kermeta::standard::String is do
        result := "I'm an X object"
    end
}
```

```
class Rectangle
{
    attribute length :
kermeta::standard::Integer
    attribute width :
kermeta::standard::Integer


        // read-only property derived from
length & width
    property surface :
kermeta::standard::Integer
        getter is do
            result := length * width
        end
}
class Cube
{
    attribute width :
kermeta::standard::Integer
    attribute surface :
kermeta::standard::Integer
    attribute volume :
kermeta::standard::Integer


        // read-write property
    property edge : kermeta::standard::Integer
        getter is do
            result := width
        end
        setter is do
           width := edge
           surface := edge * edge * 6
           volume := edge * edge * edge
        end
}
```

II – Langage Kermeta : généralités

Bloc de code :

```
do
    // my code : locally declared variables are not visibles outside the block
end
```

Conditions :

```
var boolCond : kermeta::language::structure::Boolean init true
// conditional block
if boolCond then
    // block for true value of the condition
else
    // block for false value of the condition
end
// conditional expression => affectation
var s : kermeta::standard::String
s := if boolCond then "its true !" else "its a joke ;-)" end
```

Boucle :

```
from
    var i : kermeta::standard::Integer init 0
until
    i == 10
loop
    /* code to be done 10 times
     .... */
    i := i + 1              // don't forget increment the counter :-)
end
```

Exceptions :

```
operation raiseException() is do
    raise kermeta::exceptions::Exception.new
end

operation handleException() is
    do    // some code which raise an exception
         self.raiseException
         rescue (e : kermeta::exceptions::Exception)
         // do something  if exception of Exception type has been raised in the "do"
block
    end
```

II – Langage Kermeta : généralités

- **Commentaires**
  - Fin de ligne
  - Plusieurs lignes

```
// a "line" comment

/* a multi line
     comment */
```

  - Annotation nommée

```
@descr    "a named annotation"
operation myAnnotatedMethod() is abstract
```

  - Annotation anonyme

```
/** anonymous multi line annotation */
reference anAnnotatedObject :
kermeta::language::structure::Object
```

  - affichage en bulle d'aide

```
operation main() is do
  myAnnotatedMethod
  anAnn
end
```

```
class ForComments{
  ...
  operation myAnnotatedMethod() is abstract
  ...
}
@descr "a named annotation"
```

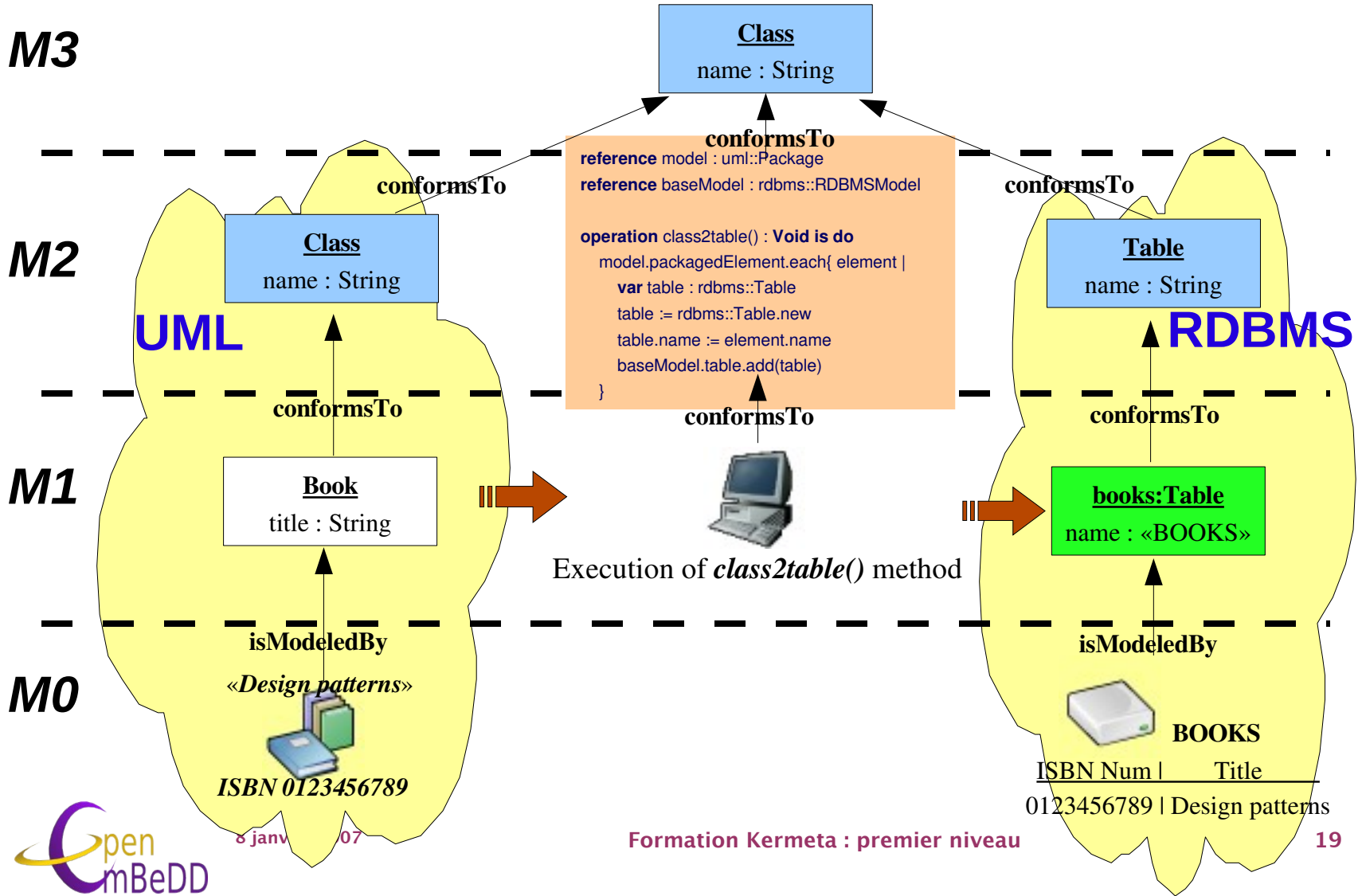
- **Sucre syntaxique**

```
package root_package;
require kermeta
using kermeta::language::structure

class X
{
        /** anonymous multi line annotation */
        reference anAnnotatedObject : Object
}
```

II – Langage Kermeta : généralités

- **Variables**
  - Syntaxe : a..z, A..Z, 0..9, « ~ », « _ »
  - Mots réservés : utilisables précédés de « ~ »
- **Énumérations**
  - Déclaration
  - Usage
- **Types primitifs**
  - Integer <=> Java Integer
  - String <=> Java String mais peu de méthodes (append, ...)
  - Boolean <=> Java Boolean
  - Character [incomplet]
  - Real [incomplet]

III – Modèles : méta et méta-méta



III – Modèles : chargement

- **Déclaration du méta-modèle**

```
// calling a plugged-in metamodel
require  "/plugin/org.eclipse.uml2.uml/model/UML.ecore"

// calling a metamodel stored in the project
require "../metamodels/RDBMS.ecore"
```

- **Chargement d'un modèle**

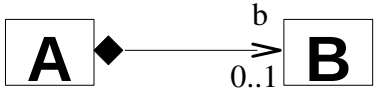
```
operation loadUmlModel() is do
    var inputRepository : kermeta::persistence::EMFRepository init
kermeta::persistence::EMFRepository.new
    var inputResource : kermeta::persistence::EMFResource
    inputResource ?= inputRepository.createResource("../models/myUmlModel.uml",

"platform:/plugin/org.eclipse.uml2.uml/model/UML.ecore")
    inputResource.load()
    var pack : uml::Package
    pack ?= inputResource.instances.one
    umlModel ?= pack.packagedElement.one
end
```

- **Sérialisation d'un modèle**

```
operation saveRdbmsModel() is do
    var outputRepository : kermeta::persistence::EMFRepository init
kermeta::persistence::EMFRepository.new
    var outputResource : kermeta::persistence::EMFResource
    outputResource ?= outputRepository.createResource("../models/myBaseModel.xmi",

"../metamodels/RDBMS.ecore")
    outputResource.instances.add(baseModel)
    outputResource.save()
end
```

III – Modèles : associations

1




```
class A {
   attribute b : B
}
class B {}
```

usage

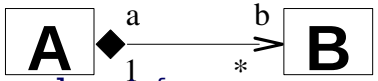
```
var a1 : A init A.new
var b1 : B init B.new
a1.b := b1
var b2 : B
b2 := a1.b
```

2


```
class A {
   attribute b :
B[0..*]
}
class B {}
```

```
var a1 : A init A.new
var b1 : B init B.new
a1.b.add(b1)
var bees : OrderedSet<B>
bees := a1.b
```

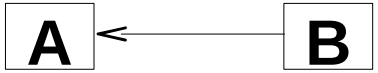
3




```
class A {
   attribute b :
B[0..*]#a
}
class B {
   reference a :
A[1..1]#b
}
```

```
var a1 : A init A.new
var b1 : B init B.new
a1.b.add(b1)
var a2 : A
a2 := b1.a
```

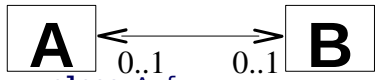
4




```
class A { }
class B {
   reference a : A
}
```

```
var a1 : A init A.new
var b1 : B init B.new
b1.a := a1
var a2 : A
a2 := b1.a
```

5




```
class A {
   reference b :
B#a
}
class B {
   reference a :
A#b
}
```

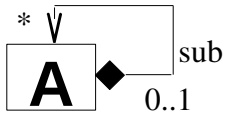
```
var a1 : A init A.new
var b1 : B init B.new
a1.b := b1
var b2 : B
b2 := b2.a.b
var a2 : A
a2 := b1.a
```

6


```
class A {
   reference b : B[0..*]
}
class B {
   reference a : A[0..*]
}
```

```
var a1 : A init A.new
var b1 : B init B.new
a1.b.add(b1)
var bees : OrderedSet<B>
bees := a1.b
var aees : OrderedSet<A>
aees := b1.a
```

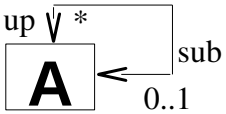
7




```
class A {
   attribute sub : A[0..*]
}
```

```
var a1 : A init A.new
var a2 : A init A.new
a1.sub.add(a2)
var a3 : A
a3 := a1.sub.first
```

8




```
class A {
   reference sub : A[0..*]#up
   reference up : A#sub
}
```

```
var a1 : A init A.new
var a2 : A init A.new
a1.sub.add(a2)
var a3 : A
a3 := a2.up
```

III – Modèles : collections

- **Fonctions à la OCL déjà implémentées sur les collections :**
  - aCollection.**each** { e | do /* traiter "e" */ end }
  - aBoolean := aCollection.**forAll** { e | /* condition */ }
  - aCollection2 := aCollection.**select** { e | /* condition */ }
  - aCollection2 := aCollection.**reject** { e | /* condition */ }
  - aCollection2 := aCollection.**collect** { e | /* valeur */ }
  - anObject := aCollection.**detect** { e | /* condition */ }
  - aBoolean := aCollection.**exists** { e | /* condition */ }
- **Autres**
  - 10.**times** { i | do /* code à exécuter 10 fois */ end }

III – Modèles : collections

- 4 types de collections

	Not Ordered	Ordered
Unique	**Set**	**OrderedSet**
Not Unique	**Bag**	**Sequence**

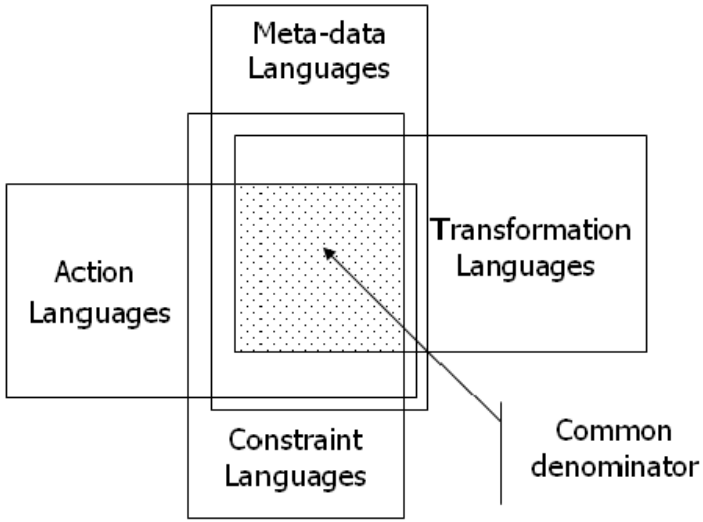
- Usage :

```
var myCol1 : set Integer[0..*]
var myCol2 : oset String[0..*]
var myCol3 : bag Boolean[0..*]
var myCol4 : seq Package[0..*]

// Fill in myCol1
myCol1.add(10)
myCol1.add(50)
```

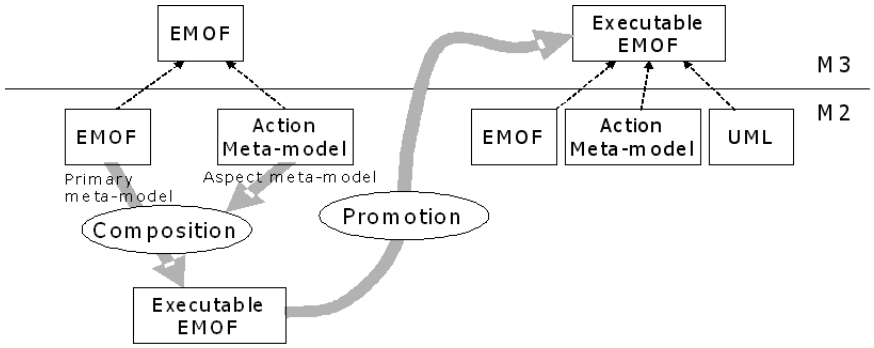
```
init kermeta::standard::Set<Integer>.new
init kermeta::standard::OrderedSet<String>.new
init Bag<Boolean>.new
init kermeta::standard::Sequence<Package>.new
```

III – Modèles : un langage d'action

- À la croisée des chemins, Kermeta :

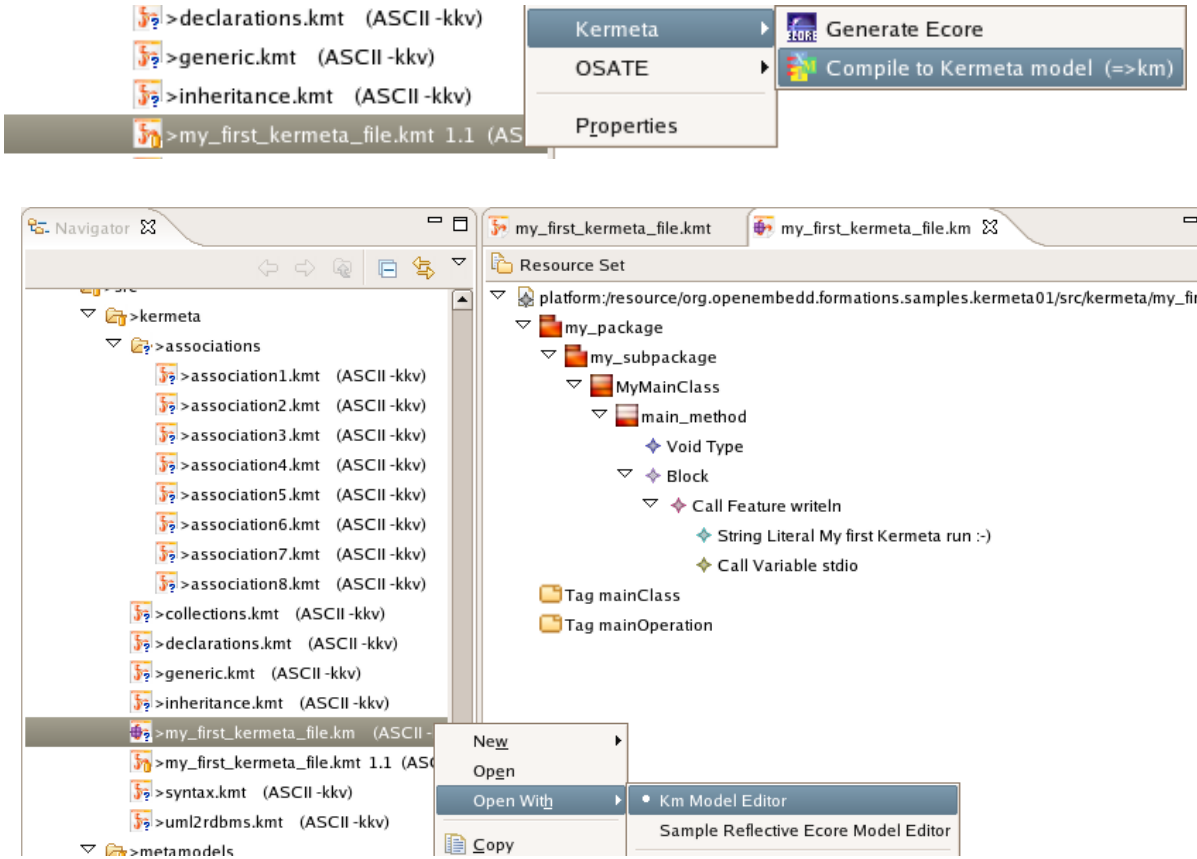


- Kermeta ajoute une sémantique à EMOF :



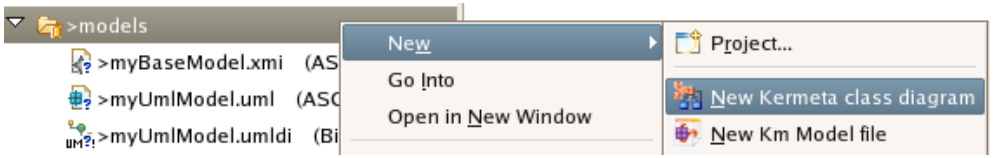
III – Modèles : un langage d'action

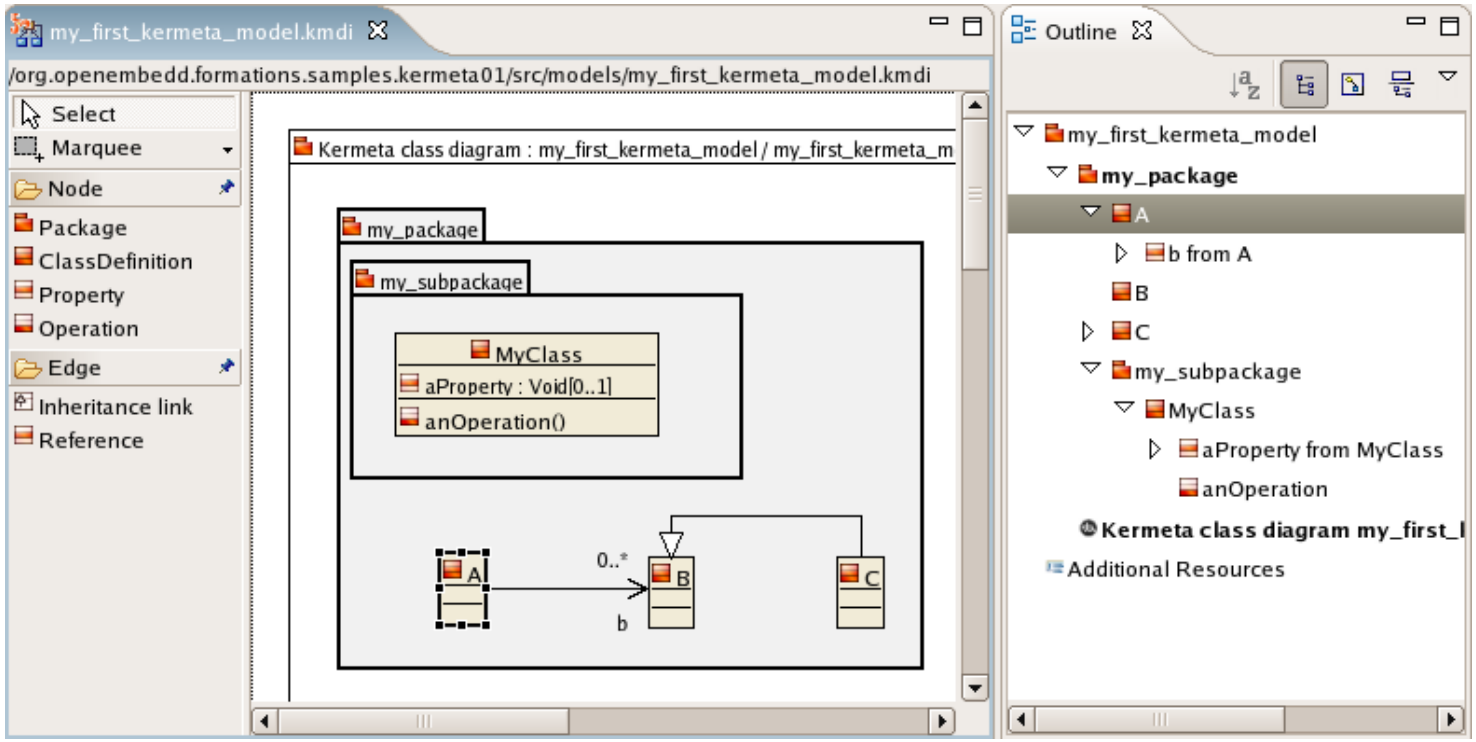
- **Kermeta est aussi un méta-modèle**
  - Transformation en KM (ou Ecore) :



III – Modèles : éditeur graphique

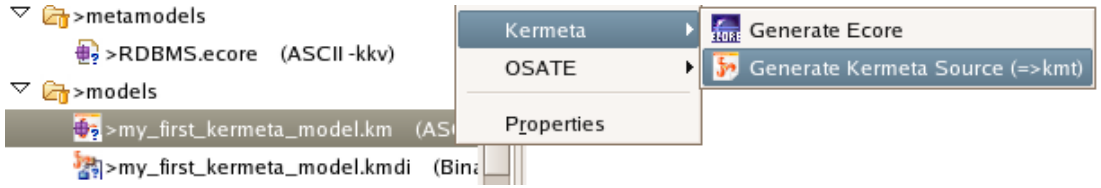
- Kermeta possède son propre éditeur graphique :

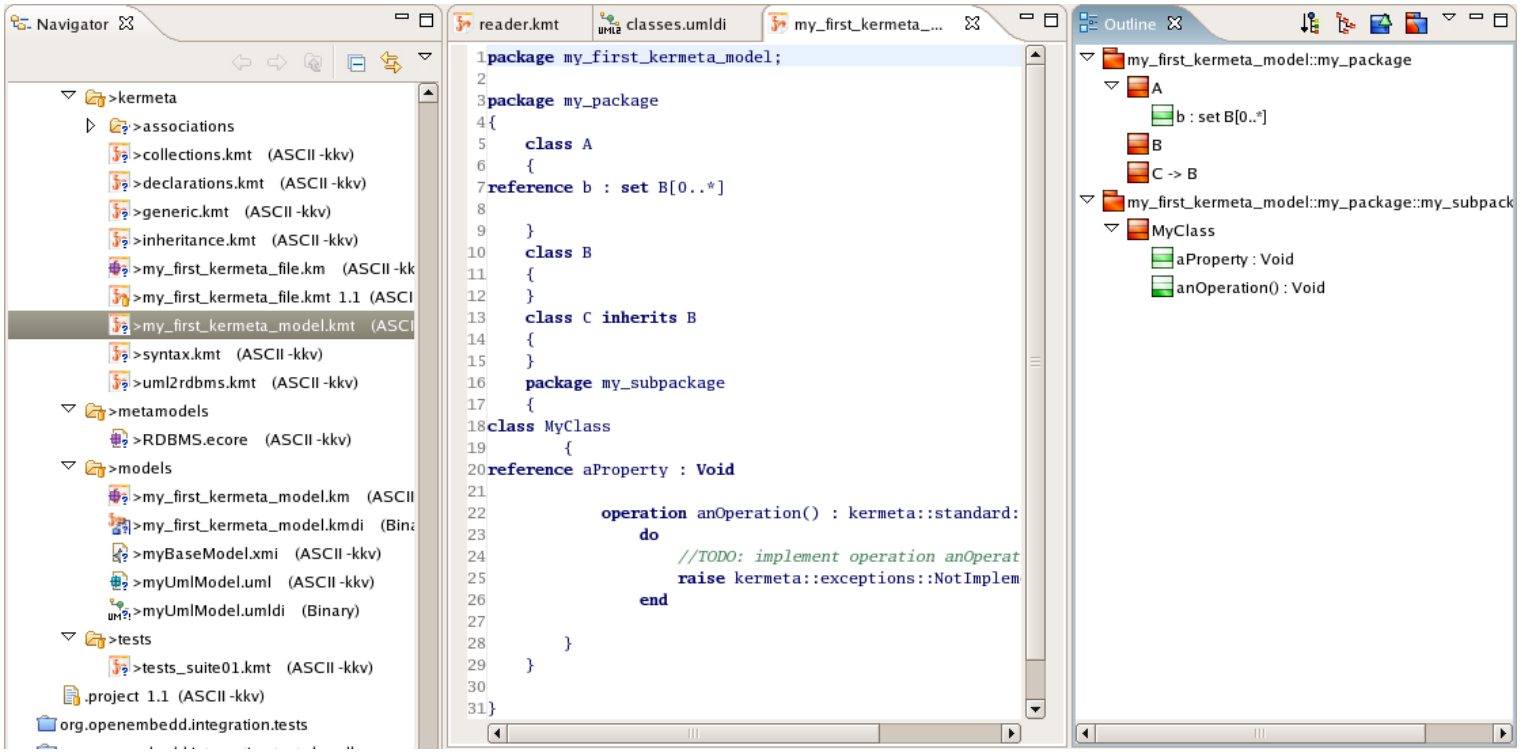




III – Modèles : éditeur graphique

- **Passer du graphique au code :**





IV – Avancé : λ expressions

Il est possible de définir ses propres λ expressions :

- /// TODO ///

IV – Avancé : programmation par contrats

- **Syntaxe :**

```
class StringTool
{
     reference stringTable : Collection<String>

    // an invariant constraint
     inv noVoidTable is
          do stringTable != void end

    // an operation with contracts
    operation concatenate(first : String,
                          second : String) : String is
              pre noVoidInput is
                  do first != void and second != void end
              do          // operation body
                  result := first
                  result.append(second)
              end

              post noVoidOutput is
                  do result != void end
}
```

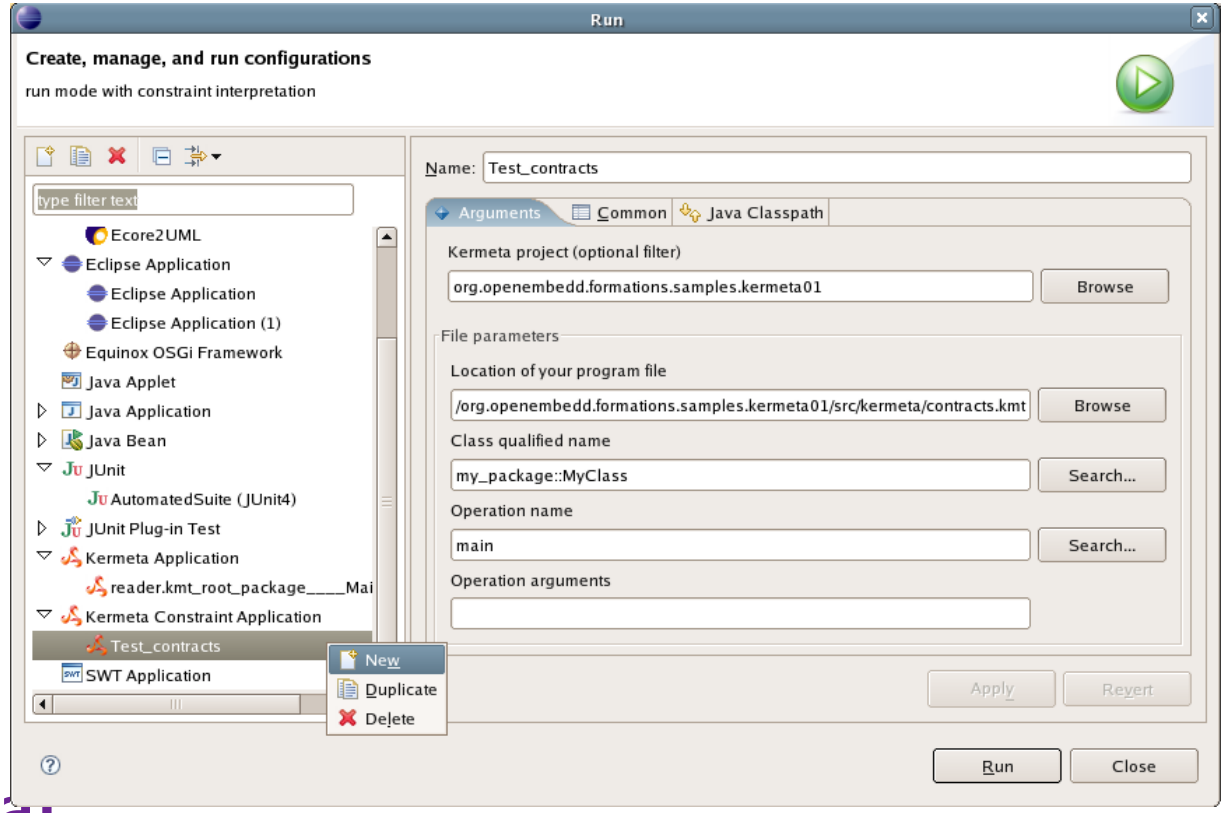
IV – Avancé : programmation par contrats

- **Programme de test :**

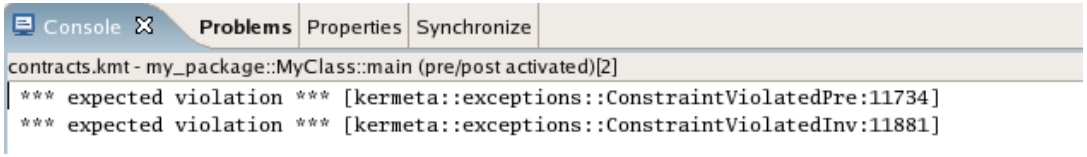
```
class MyClass
{
  operation main() : Void is do
    // new tool : its stringTable must be initialized
    var st1 : StringTool init StringTool.new
    st1.stringTable := Set<String>.new
    var s1 : String
    var s2 : String

    do
      // void strings should raise exception
      st1.concatenate(s1, s2)
      rescue (err : ConstraintViolatedPre)
        stdio.writeln("expected err " + err.toString)
        end

    do
      // new tool without table
      var st2 : StringTool init StringTool.new
      st2.checkInvariants
      rescue (err : ConstraintViolatedInv)
        stdio.writeln("expected err " + err.toString)
    end
  end
}
```

IV – Avancé : programmation par contrats

- **Lancement en mode « contrats » :**



- **Résultat :**



IV – Autres fonctionnalités avancées

- **Expressions dynamiques**

  interprétation à la volée de code passé en variable

- **Passerelle Java :**

  possibilité d'appeler des types et fonctions Java

- **Fonctionnalités en cours de développement :**
  - Type « Model »
  - Clone élaboré
  - ...