



Formations OpenEmbeDD

Kermeta 1.2

Premier niveau

Plan

- **Installer *Kermeta***
- **Environnement *Kermeta* dans Eclipse**
- ***Kermeta* : le langage**
- **Ingénierie Dirigée par les Modèles**
- **Modelisation orientée Aspects**
- **Autres fonctionnalités**

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

Kermeta

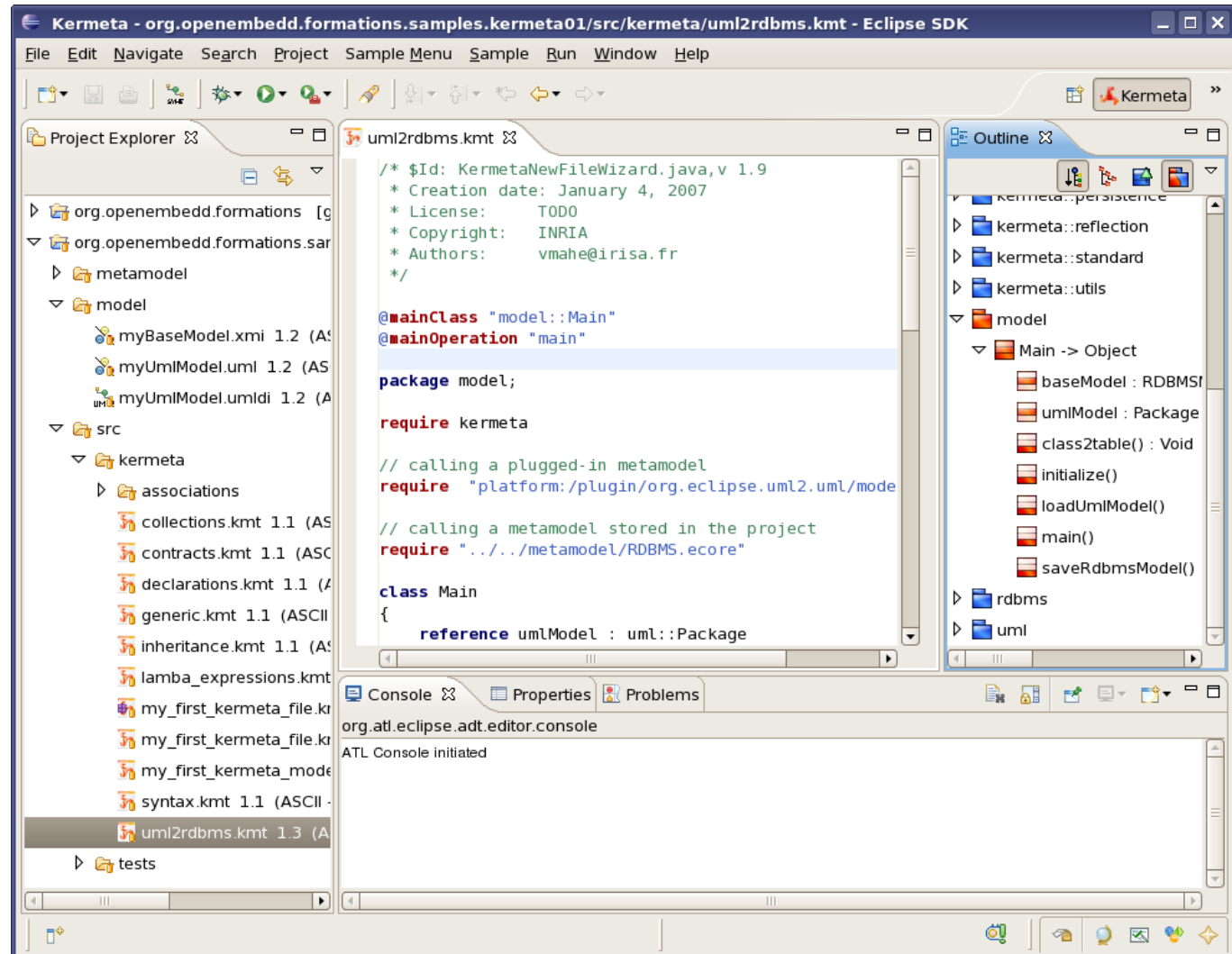
Un environnement

I - Installer *Kermeta*

- Télécharger un Eclipse SDK 3.4.1
 - ... avec l'environnement Java
 - Le déballer
 - Lancer Eclipse(un Eclipse 3.4.1 déjà fonctionnel peut faire l'affaire)
- Paramétrer le site update d'OpenEmbeDD
 - *Help -> Software Updates -> Available Software*
 - Add a New Remote Site
 - « OpenEmbeDD experimental »
 - <http://openembedd.org/experimental/update>
(pour avoir accès à la plus récente version)
- Installer ce dont vous avez besoin
 - Sélectionner « *Generic Modelling Tools* » + « *Samples* »
 - Cliquer sur le bouton « Install »
 - Finir l'installation puis redémarrer Eclipse

II - Environnement : Eclipse et Kermeta

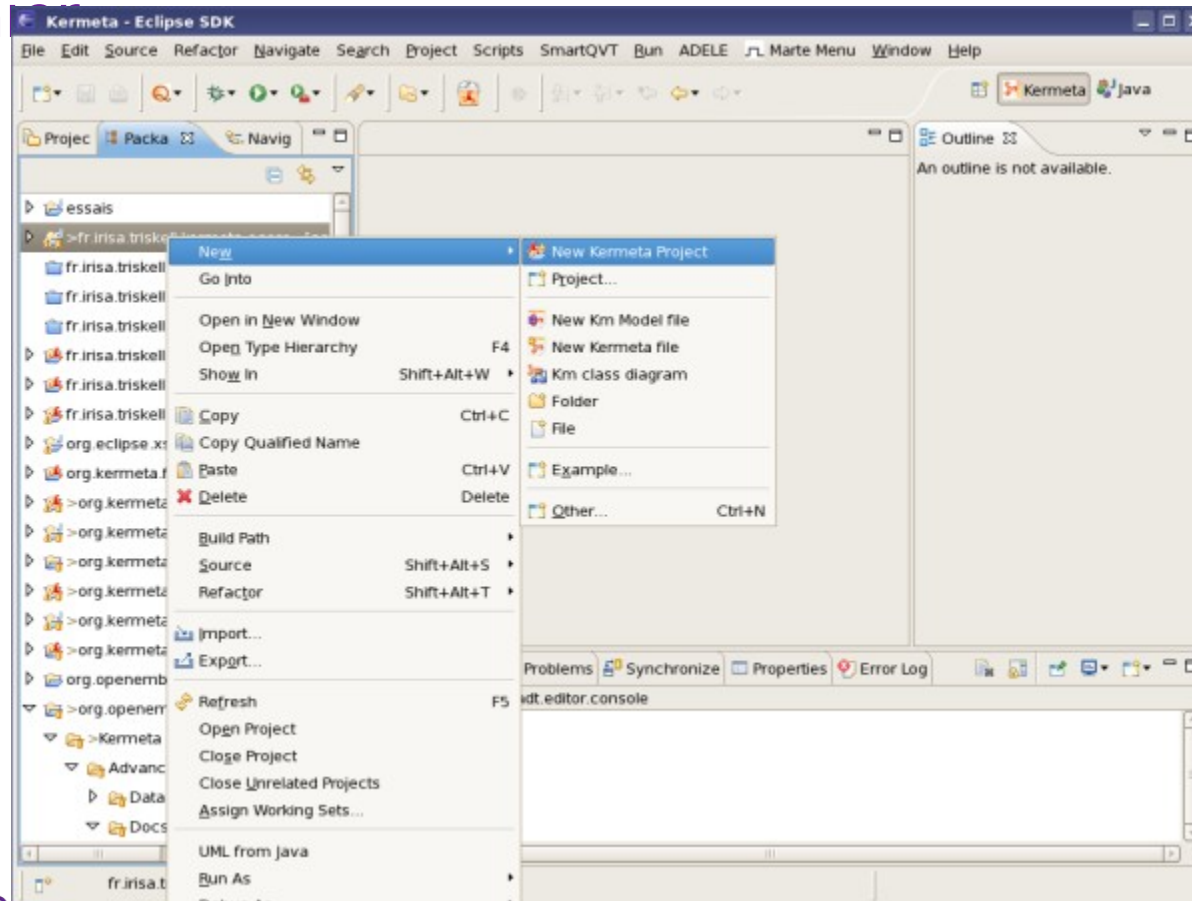
- Kermeta est totalement intégré à Eclipse :**



II – Environnement : la perspective

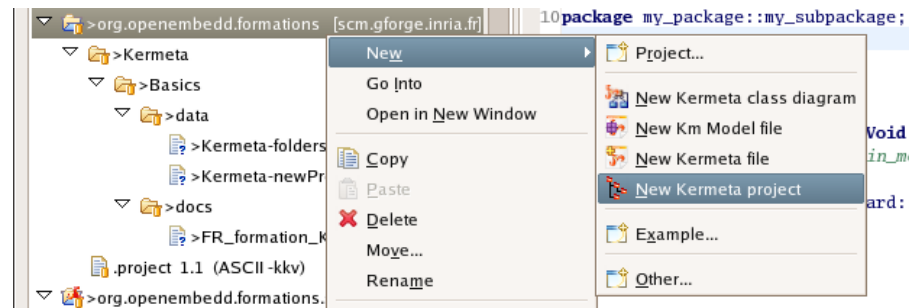
Kermeta dispose de sa propre perspective :

- Raccourcis contextuels adhoc
- Assistants « Nouveau projet » & « Nouveau fichier »

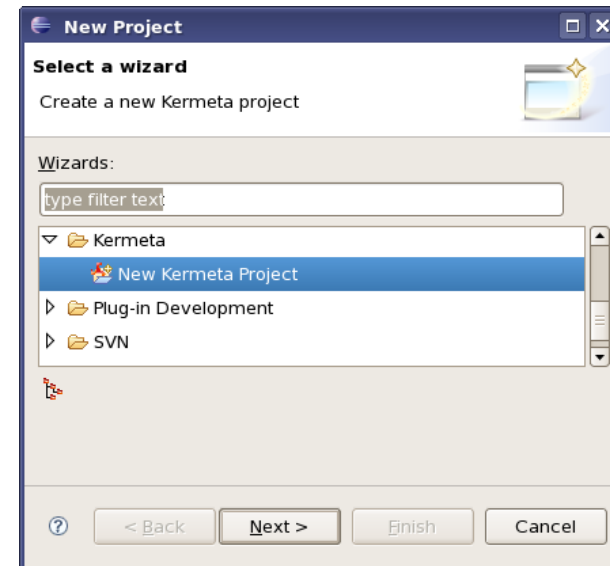


II – Environnement : nouveau projet

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

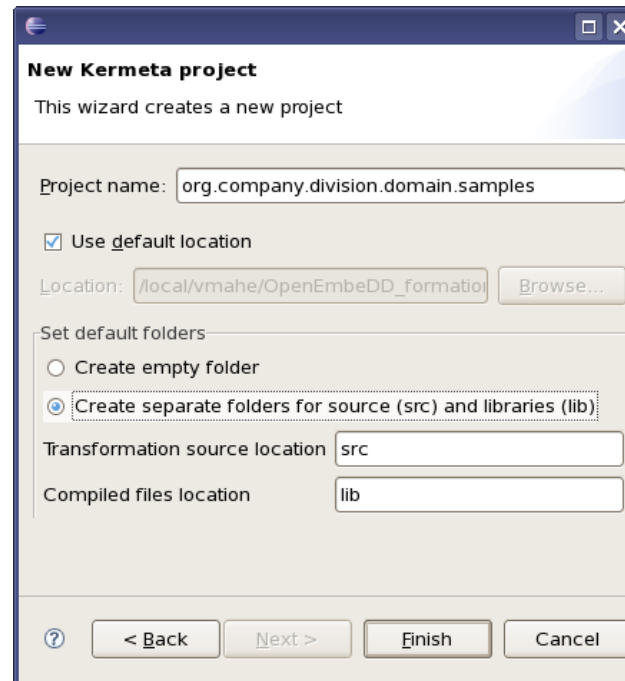


- ou bien par le menu général -> projet :

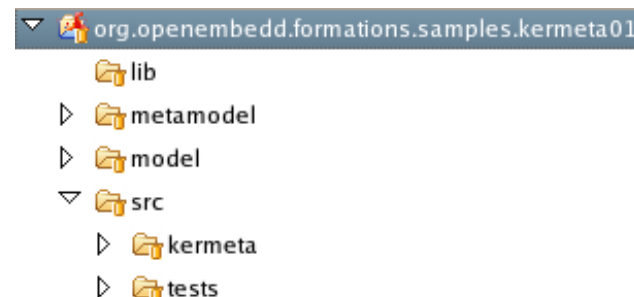


II – Environnement : projet Kermeta

- L'assistant réalise une structuration des projets Kermeta telle que préconisée :



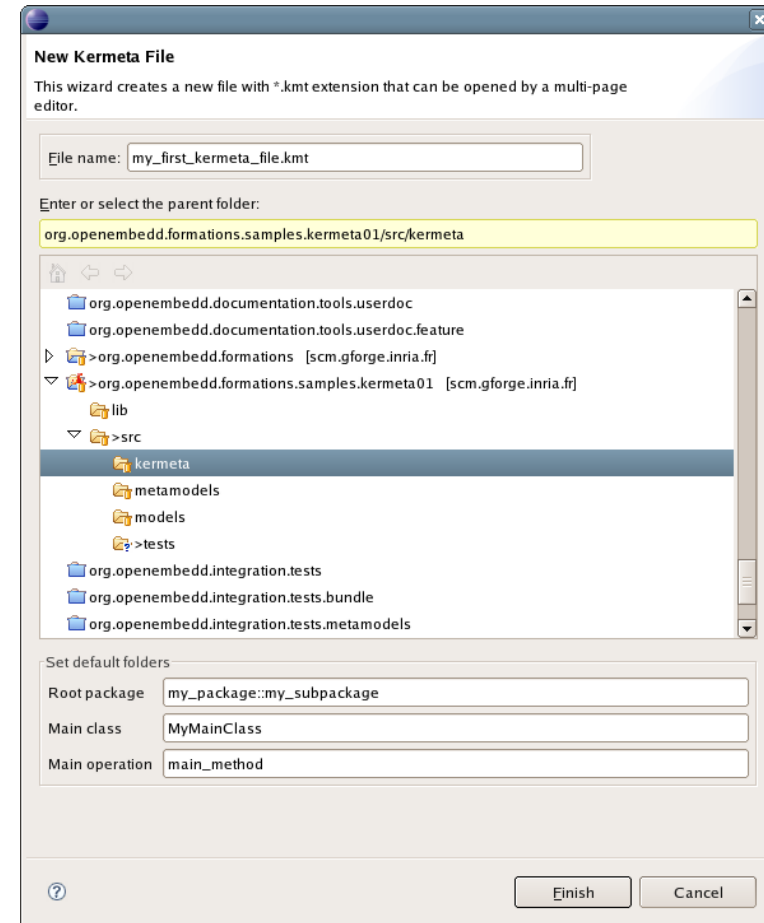
Ce qui donne :



II – Environnement : nouveau fichier

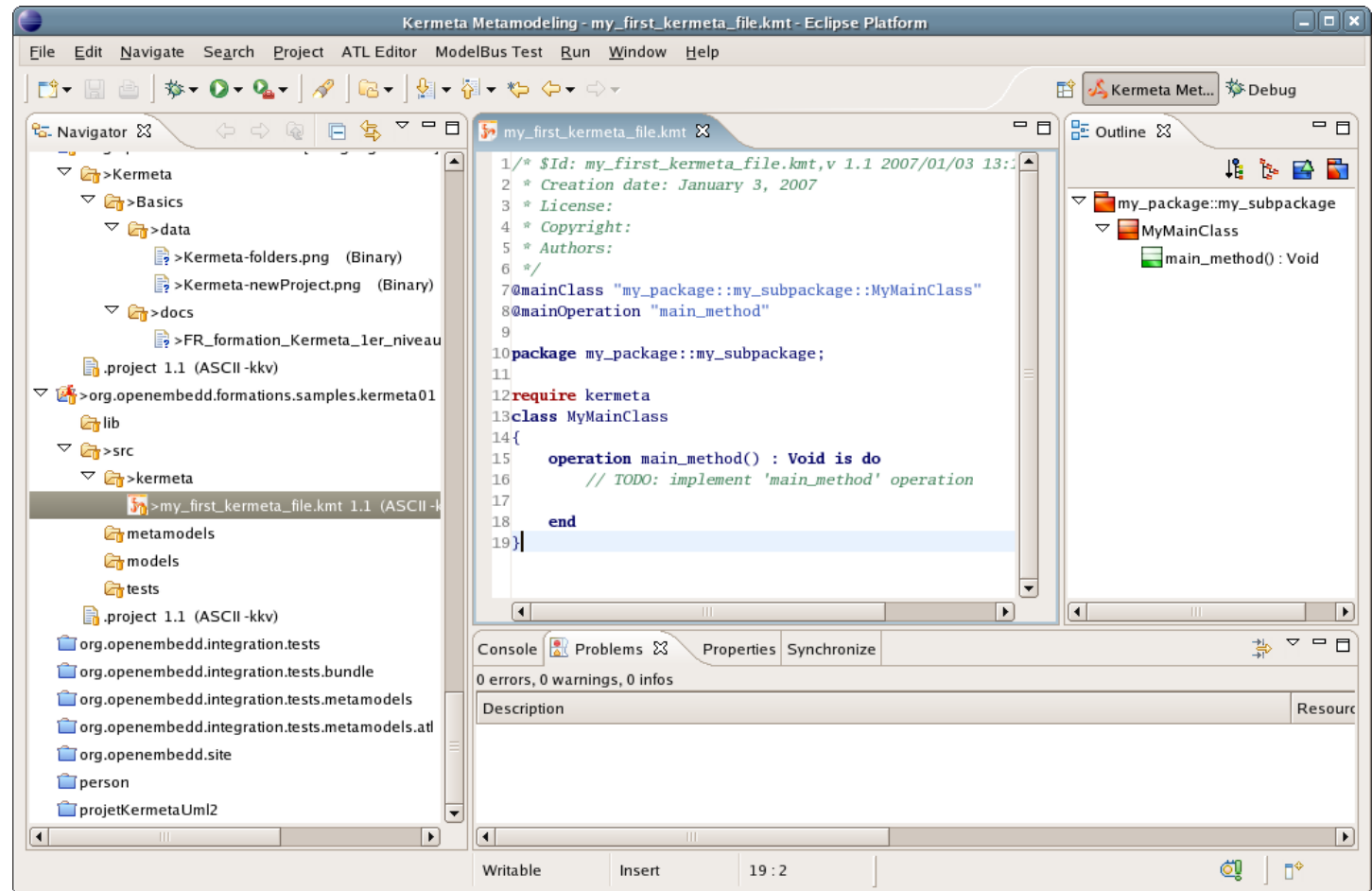
Un assistant dédié :

- Nom
- Emplacement dans le projet
- Paquetage de référence
- Classe principale
- Méthode de lancement



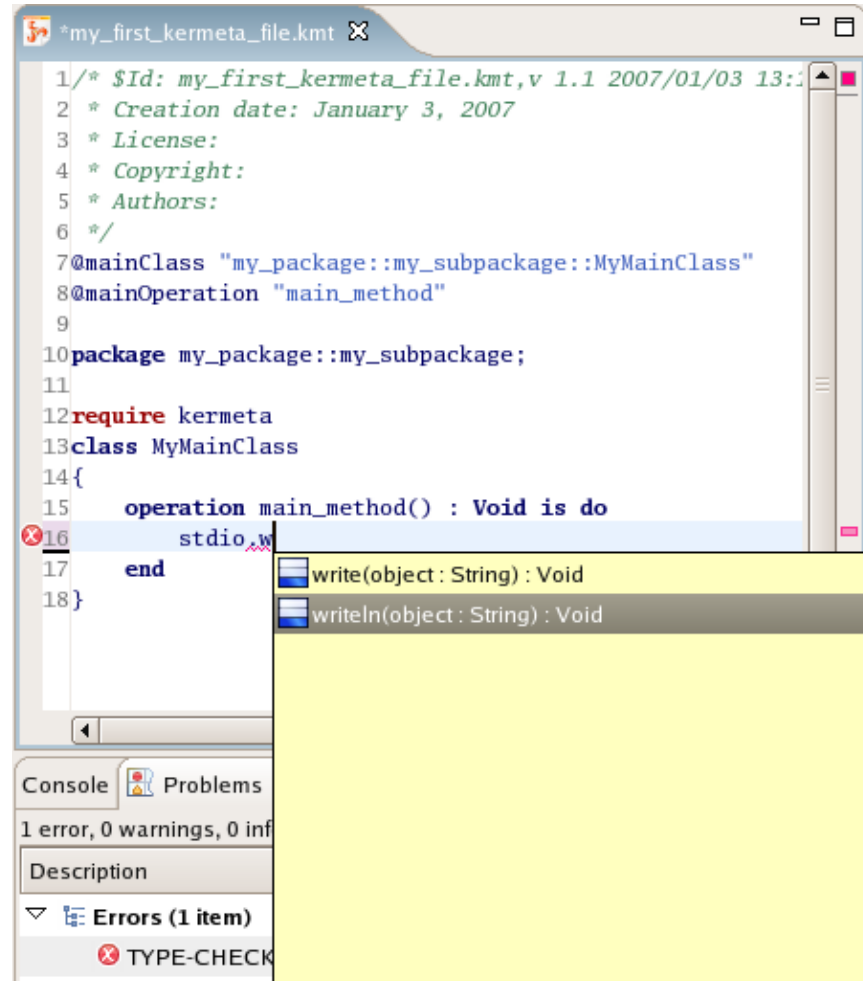
II – Environnement : édition d'un fichier

- L'assistant génère le nouveau fichier et l'ouvre dans l'éditeur de texte :



II – Environnement : édition d'un fichier

- Ajoutons un peu de code au fichier :



```

1 /* $Id: my_first_kermeta_file.kmt,v 1.1 2007/01/03 13:11:11 $ */
2 * Creation date: January 3, 2007
3 * License:
4 * Copyright:
5 * Authors:
6 */
7 @mainClass "my_package::my_subpackage::MyMainClass"
8 @mainOperation "main_method"
9
10 package my_package::my_subpackage;
11
12 require kermeta
13 class MyMainClass
14 {
15     operation main_method() : Void is do
16         studio.w
17     end
18 }
  
```

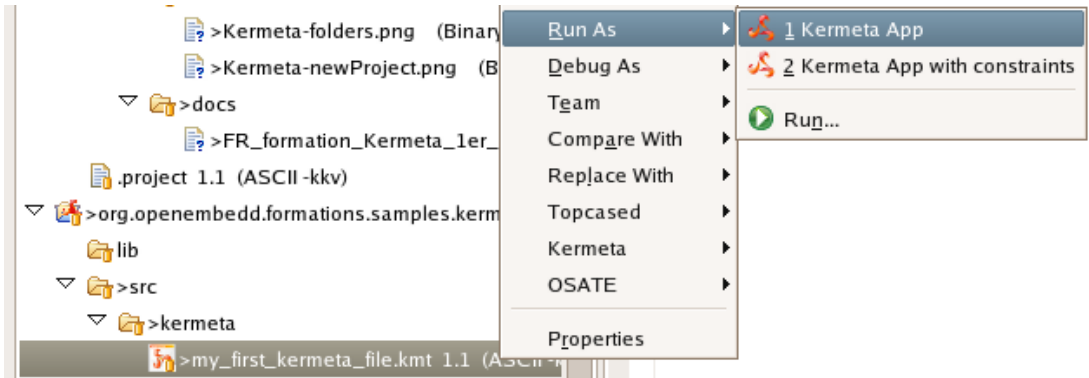
The screenshot shows an IDE window titled "*my_first_kermeta_file.kmt". The code is in Kermeta, a modeling language. A code completion menu is open at line 16, showing suggestions: "write(object : String) : Void" and "writeln(object : String) : Void". The bottom panel shows a "Problems" view with one error: "TYPE-CHECK".

II – Environnement : exécuter Kermeta

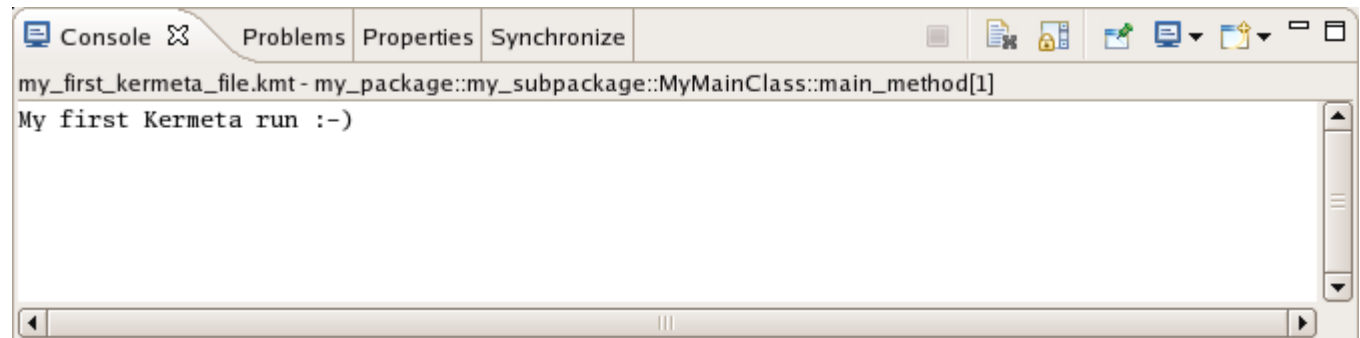
- Complétons :

```
13 class 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 :



II – Environnement : *KUnit*

Tests unitaires avec KUnit

```
tests_suite01.kmt
/* $Id: tests_suite01.kmt,v 1.1 2007/01/09 08:17:59 vmahe Exp $
 * Creation date: January 3, 2007
 * License:
 * Copyright:
 * Authors:
 */
@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

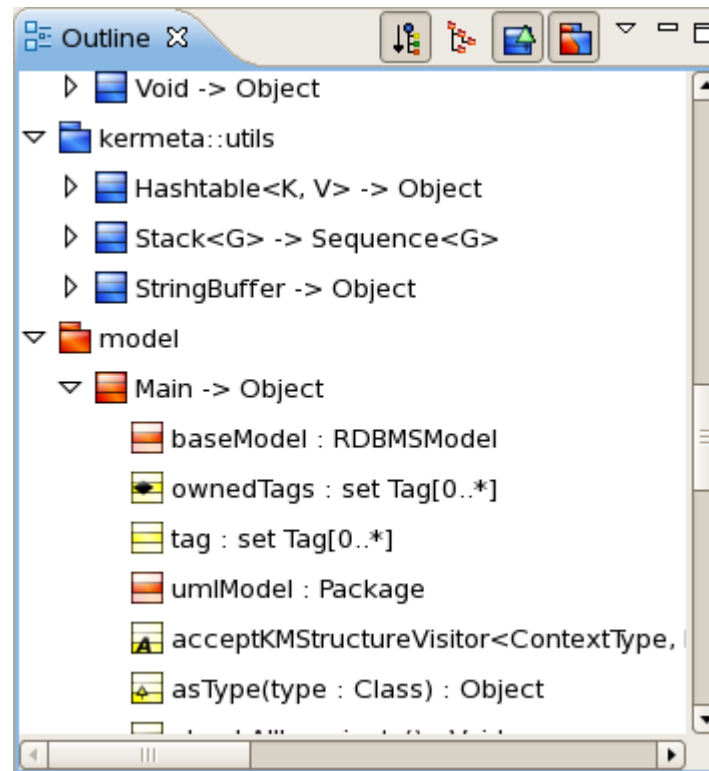
    method setUp() is do
        a := 0
        b := 1
    end

    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 – Environnement : divers

- « Outline » : affichage des librairies



Kermeta

Le langage

III – 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
}
```


III – 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
}
    
```

III – 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
    // v1 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
}
```

III – 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 to 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 block
end
```

III – 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  
{  
  /* avoid writing kermeta::language::structure::Object */  
  reference anAnnotatedObject : Object  
}
```

III – Langage Kermeta : généralités

• Variables

- Syntaxe : a..z, A..Z, 0..9, « ~ », « _ »
- Mots réservés : utilisables précédés de « ~ »

• Énumérations

- Déclaration **enumeration** Seasons { spring; summer; autumn; winter; }
- Usage **operation** x (val : Seasons) **is do**
if val == Seasons.spring **then** stdio.writeln("It's Spring") **end**
end

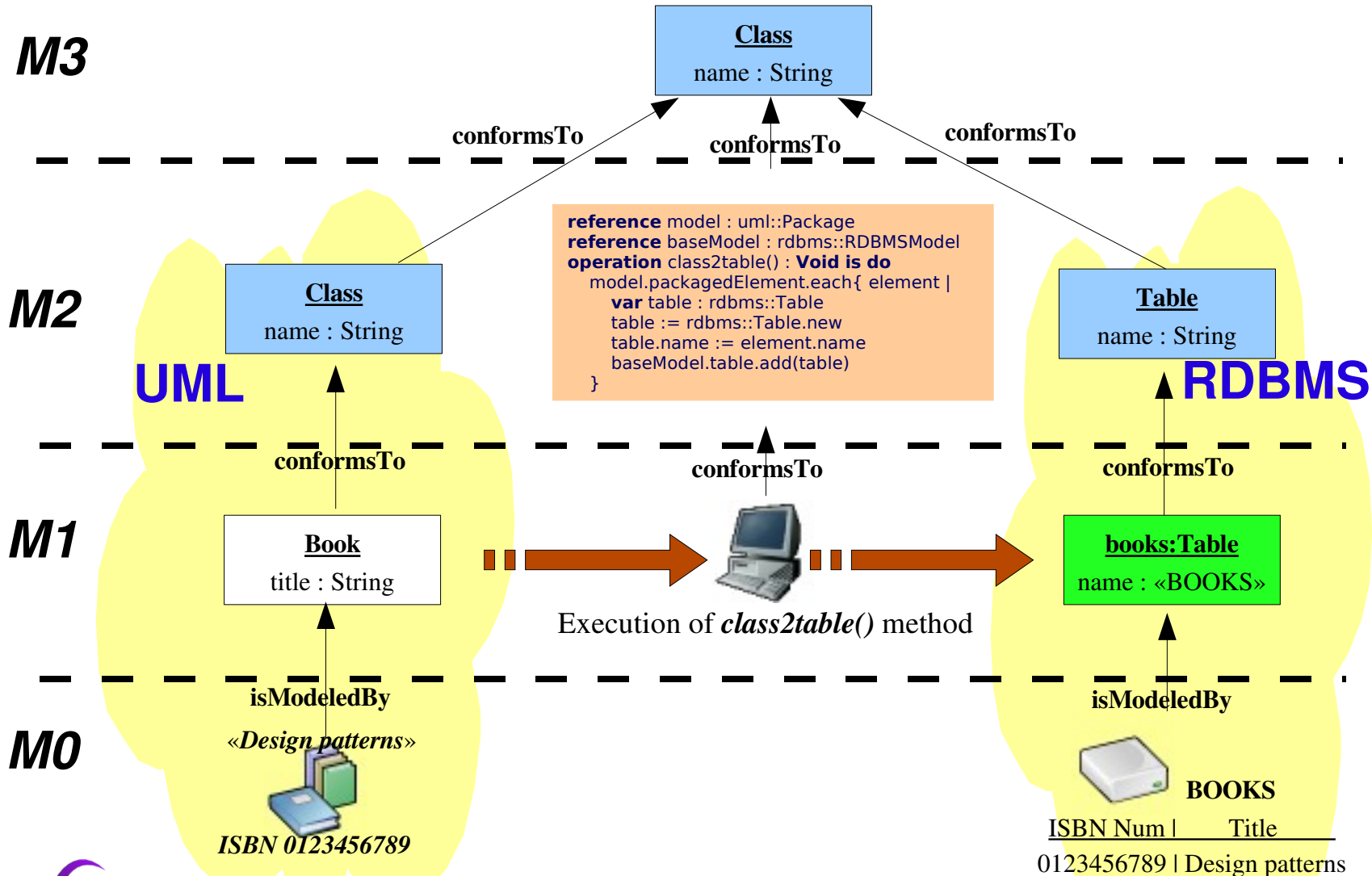
• Types primitifs

- Integer \Leftrightarrow Java Integer
- String \Leftrightarrow Java String mais peu de méthodes (append, ...)
- Boolean \Leftrightarrow Java Boolean
- Character [incomplet]
- Real [incomplet]

IDM

**modèles,
méta-modèles,
méta-méta-modèles**

IV – Modèles : méta et méta-méta



VI – Modèles : chargement

• Déclaration du méta-modèle

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

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

// calling a plugged-in metamodel (the best)
require "http://www.eclipse.org/uml2/2.1.0/UML"
```

• Chargement d'un modèle

```
operation loadUmlModel() is do
    var inputRep : kermeta::persistence::EMFRepository init kermeta::persistence::EMFRepository.new
    var inputRes : kermeta::persistence::EMFResource
    inputRes := inputRep.getResource("../models/myUmlModel.uml",
                                    "platform:plugin/org.eclipse.uml2.uml/model/UML.ecore")

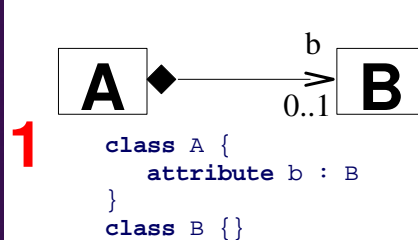
    inputRes.load()
    var pack : uml::Package
    pack := inputResource.instances.one
    umlModel := pack.packagedElement.one
end
```

• 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")

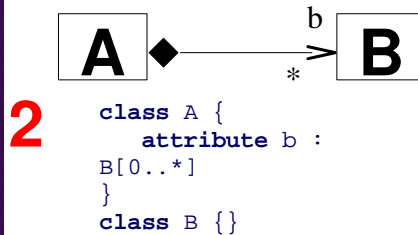
    outputResource.instances.add(baseModel)
    outputResource.save()
end
```


IV – Modèles : associations

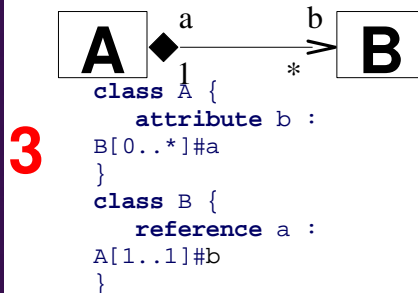


usage

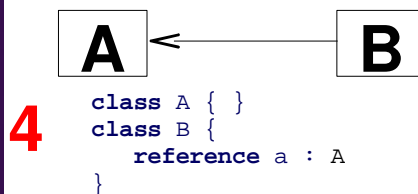
```
var a1 : A init A.new
var b1 : B init B.new
a1.b := b1
var b2 : B
b2 := a1.b
```



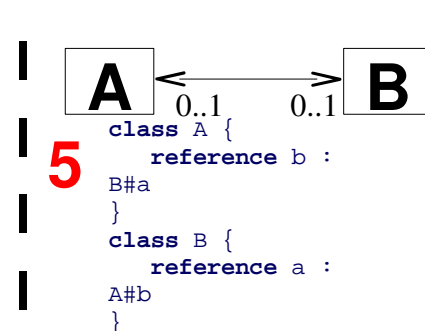
```
var a1 : A init A.new
var b1 : B init B.new
a1.b.add(b1)
var bees : OrderedSet<B>
bees := a1.b
```



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

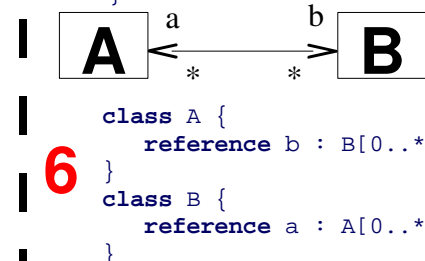


```
var a1 : A init A.new
var b1 : B init B.new
b1.a := a1
var a2 : A
a2 := b1.a
```

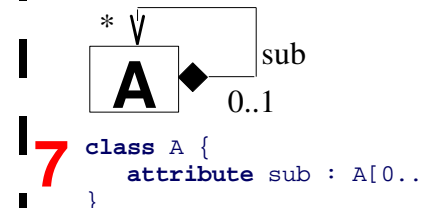


usage

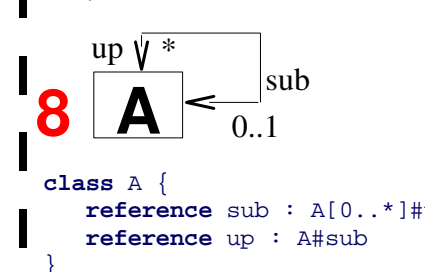
```
var a1 : A init A.new
var b1 : B init B.new
a1.b := b1
var b2 : B
b2 := b2.a.b
var a2 : A
a2 := b1.a
```



```
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
```



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



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

IV – 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 }`

IV – 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..*]

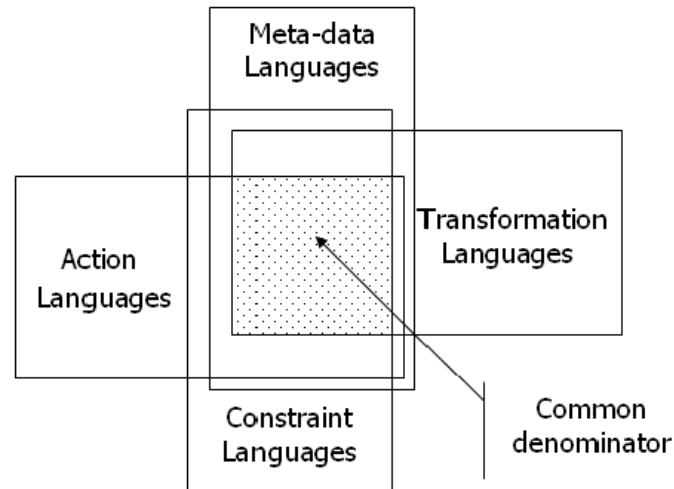
// Fill in myCol1
myCol1.add(10)
myCol1.add(50)
    
```

```

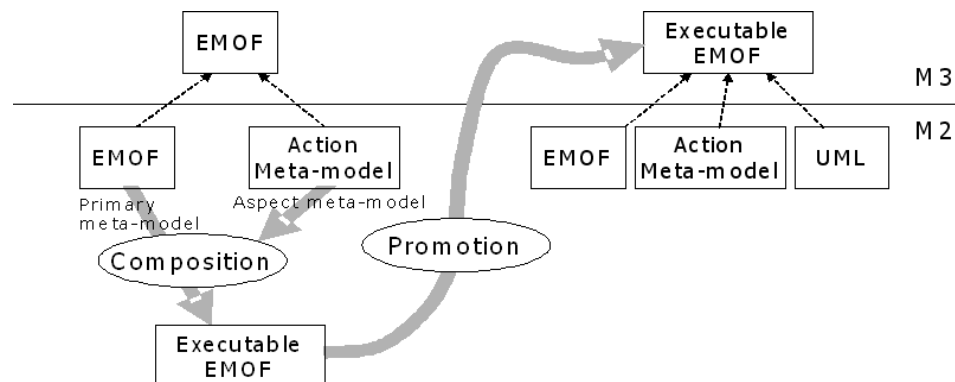
init kermeta::standard::Set<Integer>.new
init kermeta::standard::OrderedSet<String>.new
init kermeta::standard::Bag<Boolean>.new
init kermeta::standard::Sequence<Package>.new
    
```

IV – Modèles : un langage d'action

- À la croisée des chemins, Kermeta :



- Kermeta ajoute une sémantique à EMOF :



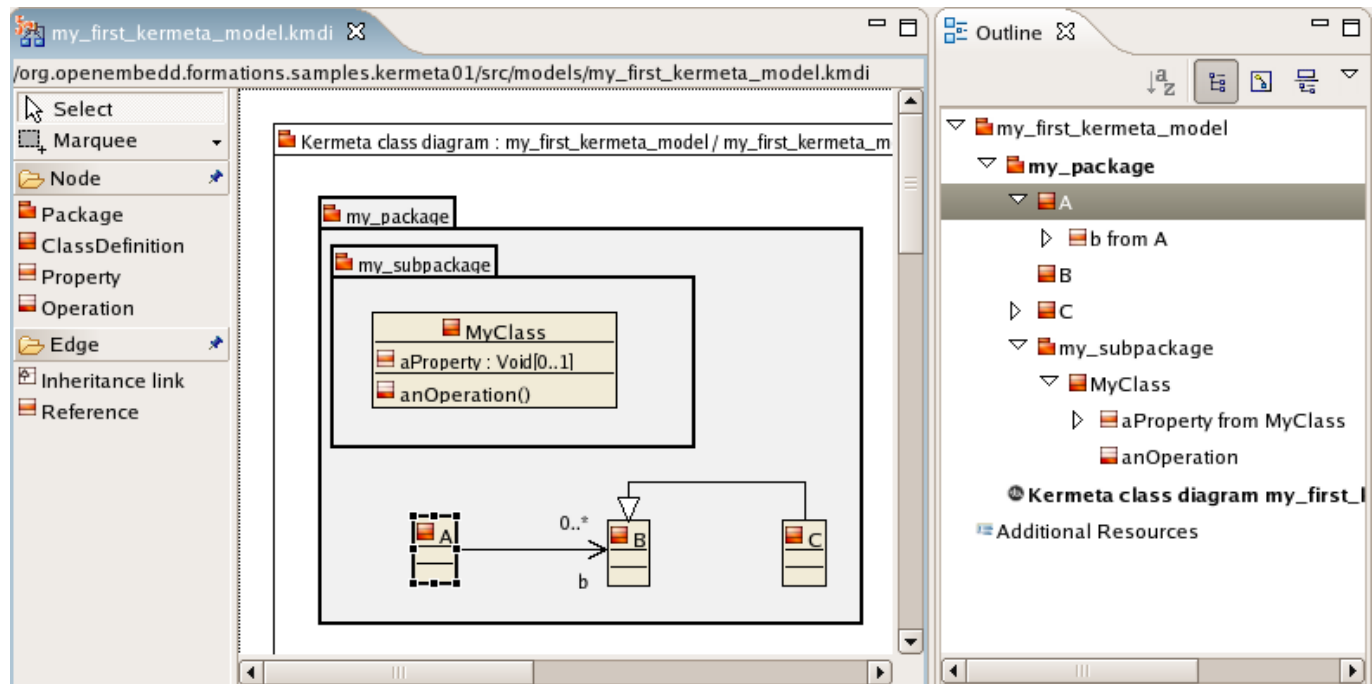
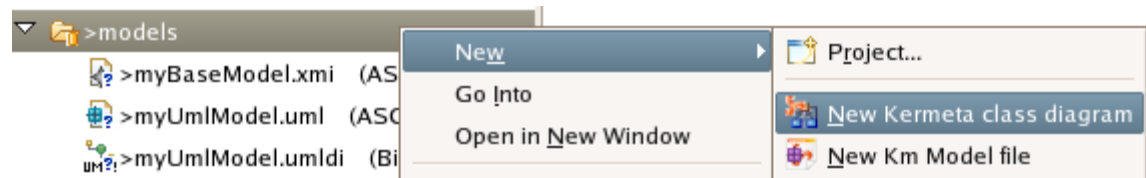
IV – Modèles : un langage d'action

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

The screenshot displays the Kermeta IDE interface. At the top, a list of files is shown, including >declarations.kmt, >generic.kmt, >inheritance.kmt, and >my_first_kermeta_file.kmt. A context menu is open over the selected file, showing options: Kermeta, OSATE, Properties, and a submenu with 'Generate Ecore' and 'Compile to Kermeta model (=>km)'. Below this, the IDE's Navigator and Editor views are visible. The Navigator shows a tree structure under 'kermeta' with various association and collection files. The Editor shows the 'my_first_kermeta_file.km' file, which is a Resource Set containing a package 'my_package' with a subpackage 'my_subpackage' and a class 'MyMainClass'. The class has a 'main_method' of type 'Void Type' containing a 'Block' with a 'Call Feature writeln' (with a string literal 'My first Kermeta run :-') and a 'Call Variable stdio'. The IDE also shows tags for 'mainClass' and 'mainOperation'. A context menu is open over the 'my_first_kermeta_file.km' file in the Navigator, showing options: New, Open, Open With (with a submenu for 'Km Model Editor' and 'Sample Reflective Ecore Model Editor'), and Copy.

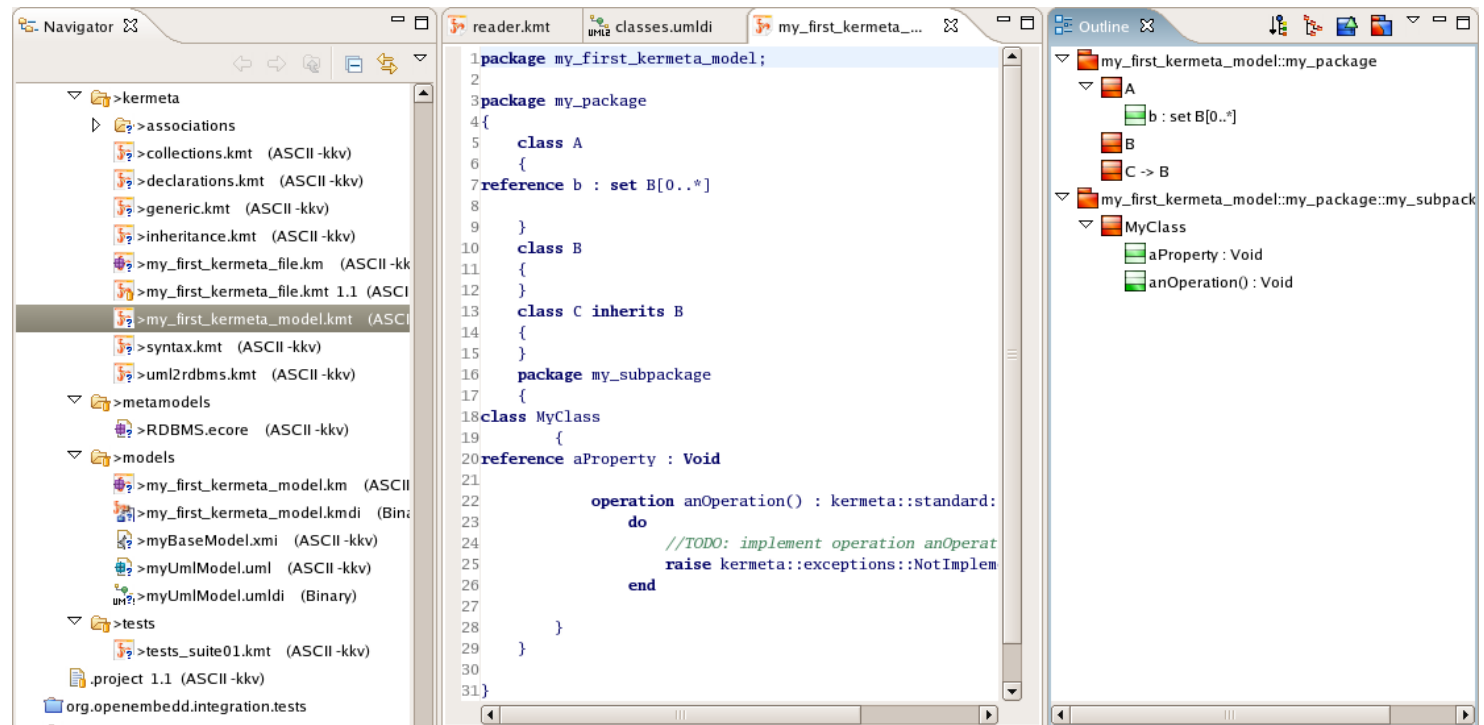
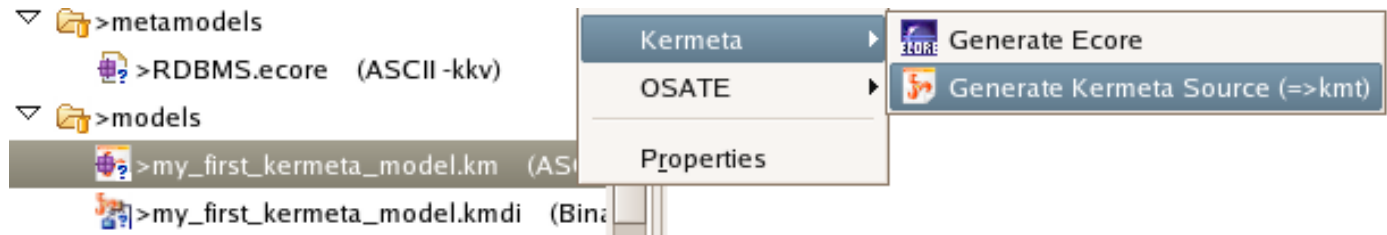
IV – Modèles : éditeur graphique

- Kermeta possède son propre éditeur graphique :



IV – Modèles : éditeur graphique

• Passer du graphique au code :

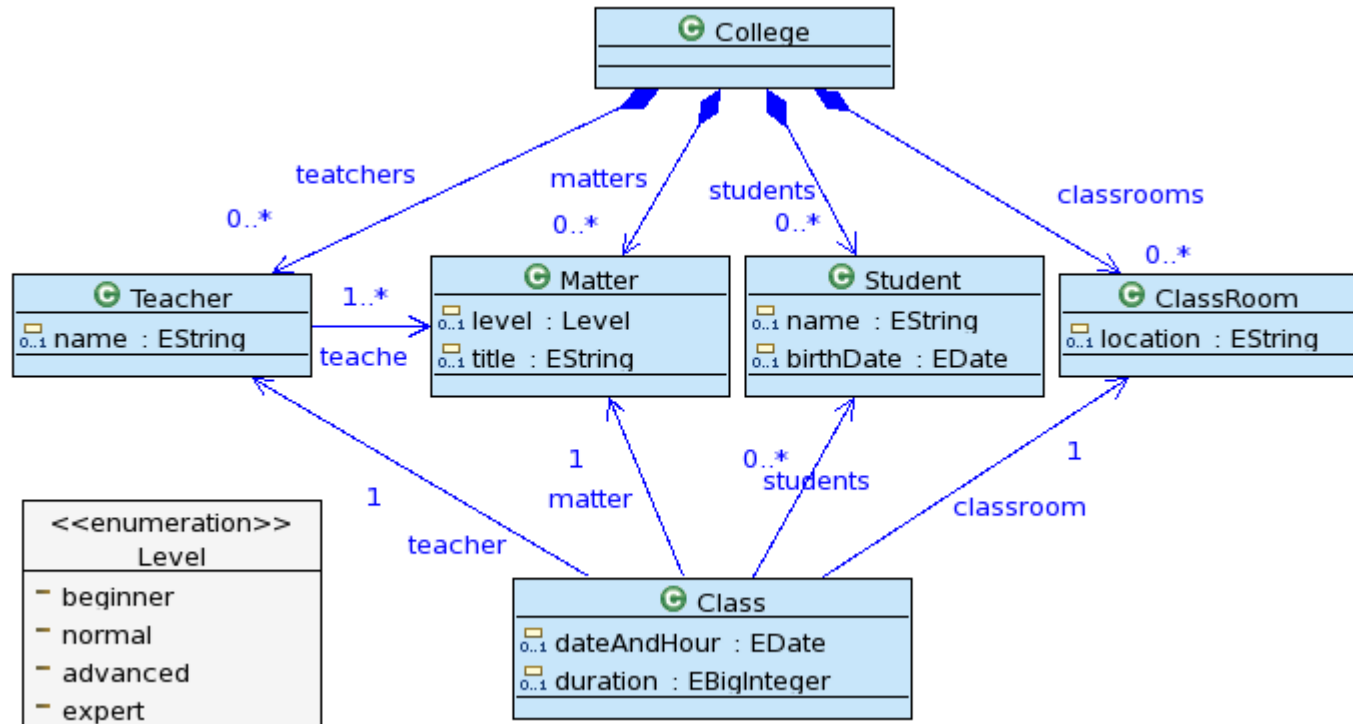


Aspects

Enrichissez vos méta-modèles

V – Aspects : enrichissez vos méta-modèles

- Imaginez un méta-modèle d'école(s)



V – Aspects : enrichissez vos méta-modèles

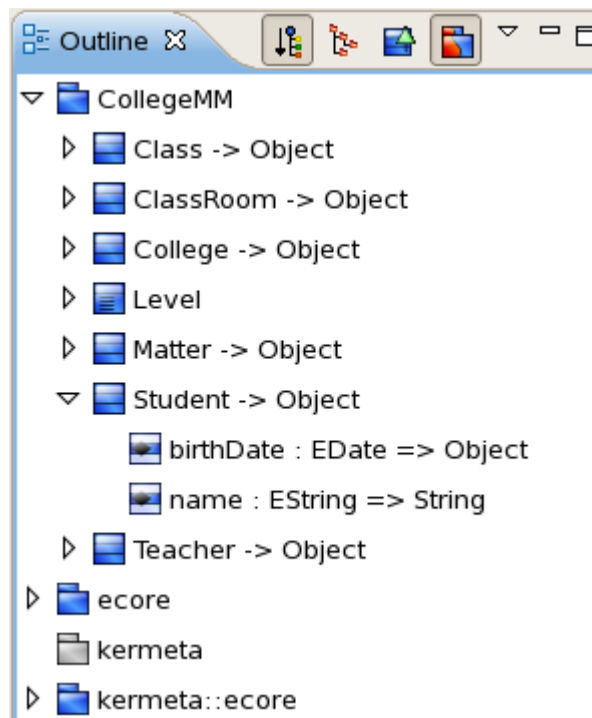
• Créez un fichier kermeta qui le référence

```
package CollegeMM;
```

```
require kermeta
```

```
require "platform:/resource/org.openembedd. formations.samples.kermeta01/metamodel/CollegeMM.ecore"
```

• Vous obtenez l'outline correspondant



V – Aspects : enrichissez vos méta-modèles

- Un aspect permet d'ajouter une classe

```
package CollegeMM;
```

```
require kermeta
```

```
require "platform:/resource/org.openembedd. formations. samples. kermeta01/metamodel/CollegeMM.ecore"
```

```
aspect class Note {
```

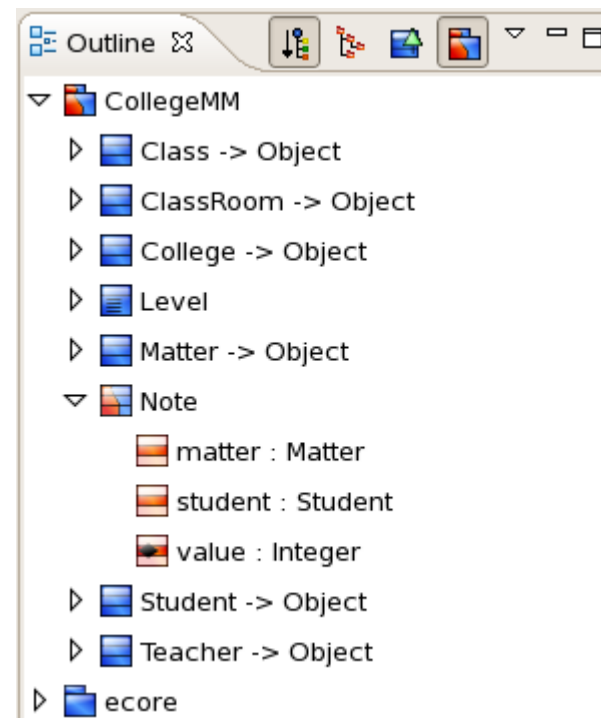
```
    attribute ~value : kermeta::standard::Integer
```

```
    reference student : Student
```

```
    reference matter : Matter
```

```
}
```

- L'outline affiche l'aspectisation



V – Aspects : enrichissez vos méta-modèles

• Ajout des opposites + nouvelle opération

```

package CollegeMM;

require kermeta
require "platform:/resource/org.openembedd. formations.samples.kermeta01/metamodel/CollegeMM.ecore"

aspect class Note {
    attribute ~value : kermeta::standard::Real
    // add the opposite for managing notes from students/matters
    reference student : Student#notes
    reference matter : Matter#notes
}

aspect class Student {
    reference notes : Note[0..*]#student
    property average : kermeta::standard::Real
    getter is do
        var total : kermeta::standard::Real
        notes.each{ n | total := total + n.~value }
        result := total / notes.size.toReal
    end
}

aspect class Matter {
    reference notes : Note[0..*]#matter
    property average : kermeta::standard::Real
    getter is do
        var total : kermeta::standard::Real
        notes.each{ n | total := total + n.~value }
        result := total / notes.size.toReal
    end
}
    
```

V – Aspects : enrichissez vos méta-modèles

• Factoriser le traitement via l'héritage

```

package CollegeMM;

require kermeta
require "platform:/resource/org.openembedd.formations.samples.kermeta01/metamodel/CollegeMM.ecore"

aspect class Note {
    attribute ~value : kermeta::standard::Real
    // add the opposite for managing notes from students/matters
    reference student : Student#notes
    reference matter : Matter#notes
}

aspect class Notable {
    operation average(notes : Note[0..*]) : kermeta::standard::Real is do
        var total : kermeta::standard::Real
        notes.each{ n | total := total + n.~value }
        result := total / notes.size.toReal
    end
}

aspect class Student inherits Notable {
    reference notes : Note[0..*]#student
    property averageNote : kermeta::standard::Real
    getter is do
        result := average(notes)
    end
}

aspect class Matter inherits Notable {
    reference notes : Note[0..*]#matter
    property average : kermeta::standard::Real
    getter is do
        result := average(notes)
    end
}
    
```

Kermeta

Autres fonctionnalités

VI – Autres : 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
    pre noVoidInput is
      do first != void and second != void end

    post noVoidOutput is
      do result != void end

  // operation body
  is do
    result := first
    result.append(second)
  end
}
```

VI – Autres : programmation par contrats

• Programme de test :

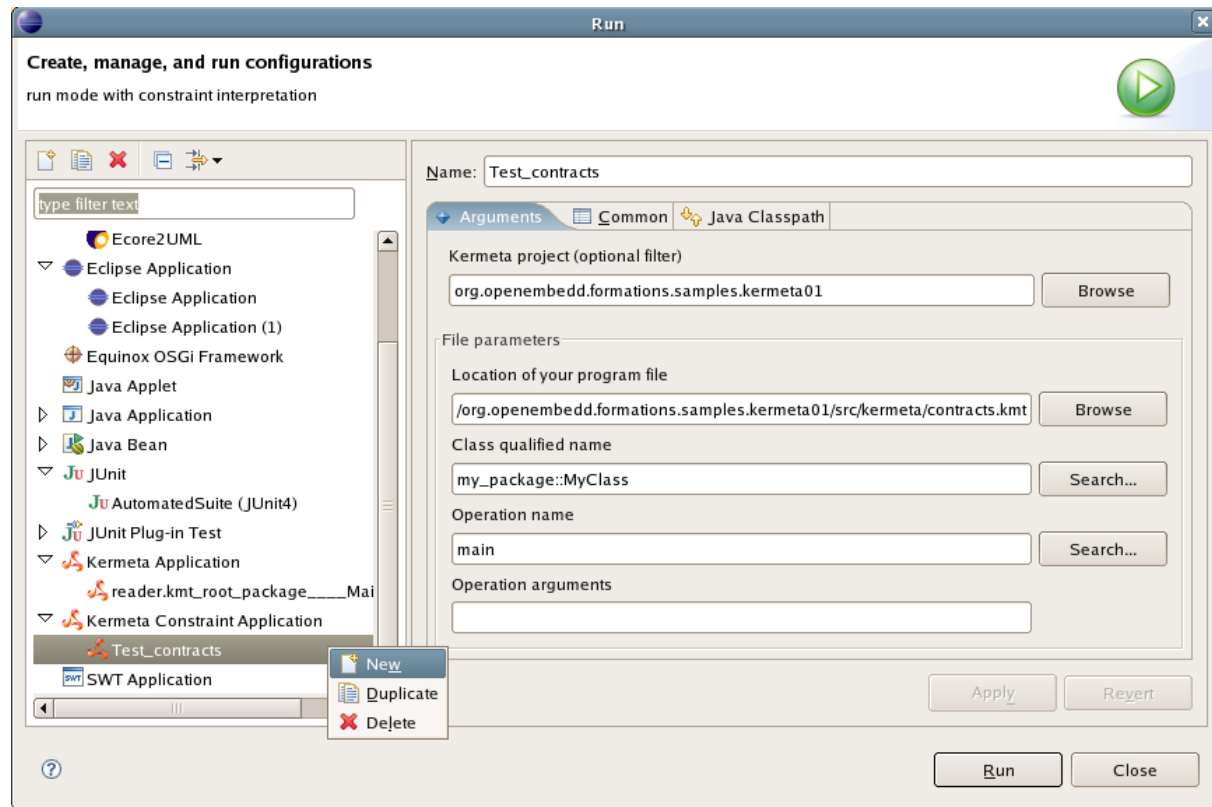
```
class MyClass
{
  operation main() : Void is do
    // new tool : its stringTable must be initialized
    var st1 : StringTool init StringTool.new
    st1.stringTable := Set<String>.new
    var s1 : String
    var s2 : String

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

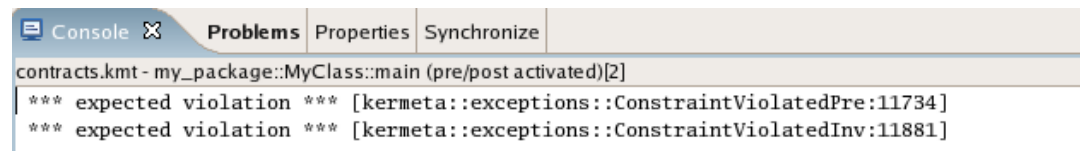
    do
      // new tool without table
      var st2 : StringTool init StringTool.new
      st2.checkInvariants
    rescue (err : ConstraintViolatedInv)
      stdio.writeln("expected err " + err.toString)
    end
  end
}
```


VI – Autres : programmation par contrats

Lancement en mode « contrats » :



Résultat :



VI - Autres : passerelle Java / Kermeta

- **Passerelle Java :**
possibilité d'appeler des types et fonctions Java

```

/** An implementation of a StdIO class in Kermeta using existing Java standard input/output */
class StdIO {
  /** write the object to standard output */
  operation write(object : Object) : Void is do
    result ?= extern fr::irisa::triskell::kermeta::runtime::basetypes::StdIO.write(object)
  end
  /** read an object from standard input */
  operation read(prompt : String) : String is do
    result ?= extern fr::irisa::triskell::kermeta::runtime::basetypes::StdIO.read(prompt)
  end
}

```

```

/** Java Implementation of wrapper called from kermeta */
public class StdIO{
  // ..... //
  // Implementation of method read(prompt : String)
  public static RuntimeObject read(RuntimeObject prompt) {
    java.lang.String input = null;
  }
}

```

VI – Autres fonctionnalités

- **Expressions dynamiques**
interprétation à la volée de code passé en variable
- **λ expressions**
créer ses propres fonctions
- **ModelType**
rapprocher des méta-modèles
- **Fonctionnalités en cours de développement :**
 - Clone élaboré
 - ...



Formations OpenEmbeDD

Kermeta 1.2

Premier niveau



Février 2009

Formation Kermeta : premier niveau

1

Plan

- **Installer *Kermeta***
- **Environnement *Kermeta* dans Eclipse**
- ***Kermeta* : le langage**
- **Ingénierie Dirigée par les Modèles**
- **Modelisation orientée Aspects**
- **Autres fonctionnalités**

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



Février 2009

Formation Kermeta : premier niveau

2

Cette formation s'adresse d'abord à un public familier des langages de programmation modernes (Java, C++,...).

En conséquence, nous présentons d'abord les éléments familiers dans Kermeta (IDE, instructions) pour amener ensuite progressivement les auditeurs vers les concepts spécifiques à la modélisation.

De plus, la bonne intégration de Kermeta dans Eclipse est un de ses atouts, qu'il convient d'offrir à la connaissance de l'utilisateur avant que celui-ci plonge dans le monde des modèles.

Cliquez pour ajouter un titre

Kermeta

Un environnement











I - Installer Kermeta

- Télécharger un Eclipse SDK 3.4.1
 - ... avec l'environnement Java
 - Le déballer
 - Lancer Eclipse
 (un Eclipse 3.4.1 déjà fonctionnel peut faire l'affaire)
- Paramétrer le site update d'OpenEmbeDD
 - *Help -> Software Updates -> Available Software*
 - Add a New Remote Site
 - « OpenEmbeDD experimental »
 - <http://openembedd.org/experimental/update>
(pour avoir accès à la plus récente version)
- Installer ce dont vous avez besoin
 - Sélectionner « *Generic Modelling Tools* » + « *Samples* »
 - Cliquer sur le bouton « Install »
 - Finir l'installation puis redémarrer Eclipse



Février 2009

Formation Kermeta : premier niveau

4

La plate-forme OpenEmbeDD offre plusieurs outils de MDE :

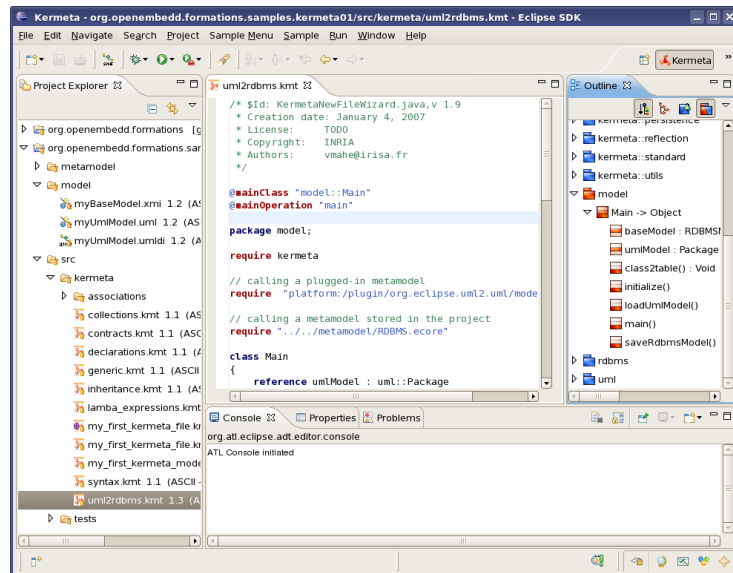
- Les modeleurs Topcased : UML2, AADL, SysML, SDL
- Le langage ATL
- Des outils de MDE orientés Temps Réel et Embarqué
- Des transformations (dont vous pouvez vous inspirer)
- ...

Des outils pour les développeurs sont disponibles en modules additionnels :

- Intégration d'OpenOffice.org dans Eclipse
- Plug-in Subclipse (SVN Subversion)
- ...

II - Environnement : Eclipse et Kermeta

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



L'utilisateur Kermeta dispose sous Eclipse de plusieurs outils :

- Une perspective Kermeta
- Une vue « Package » avec des icônes dédiées
- Un éditeur textuel avec coloration syntaxique et complétion
- Des menus contextuels dédiés
- Une vue « Outline » de l'élément édité
- Le signalement des problèmes et warnings
- Une console textuelle
- Des assistants à la création d'éléments Kermeta
- Des configurations de lancement dédiées

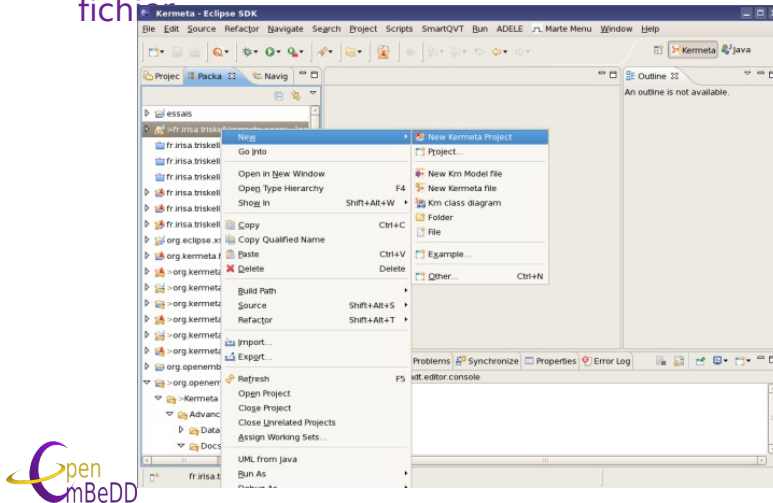
Des outils plus évolués sont disponibles en modules additionnels :

- Un éditeur graphique des objets Kermeta
- Un navigateur tactile (TouchGraph)

II – Environnement : la perspective

Kermeta dispose de sa propre perspective :

- Raccourcis contextuels adhoc
- Assistants « Nouveau projet » & « Nouveau fichier »

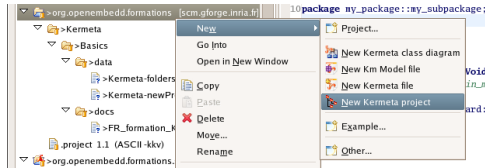


6

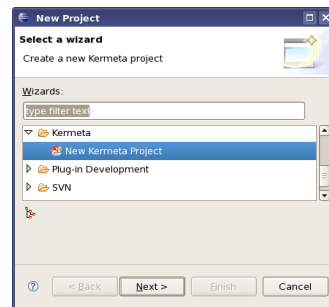
- Placer le nouveau fichier dans « src/kermeta »
- Utiliser une hiérarchie de paquetages cohérente avec les tâches à mener et les modèles à traiter par le projet.
- Classe « Main » et méthode « main() » pourront être supprimées après création si le nouveau code n'est pas un point d'entrée des traitements (pas de lancement).

II – Environnement : nouveau projet

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



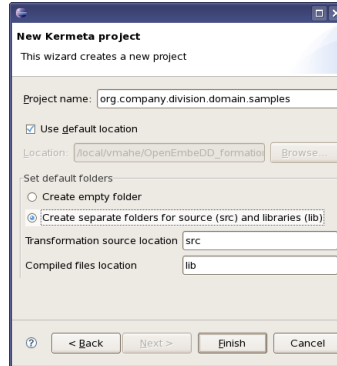
- ou bien par le menu général -> projet :



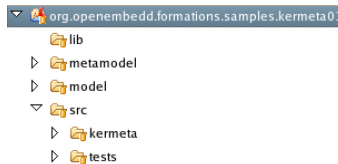
Le menu contextuel ici n'apparaît que sous la perspective Kermeta, ce qui n'empêche pas de créer des projets et fichiers Kermeta sous n'importe quelle perspective.

II – Environnement : projet Kermeta

- L'assistant réalise une structuration des projets Kermeta telle que préconisée :



Ce qui donne :



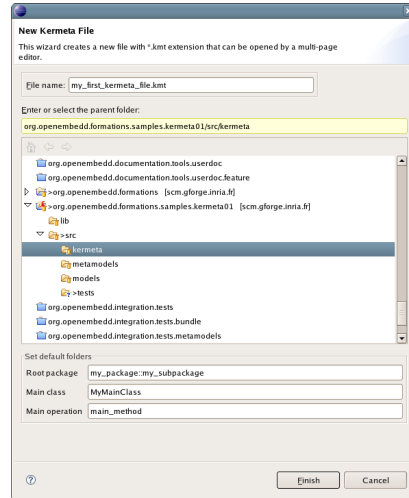
La séparation entre le « code » Kermeta, les modèles et les méta-modèles permet de faciliter la maintenance et le suivi des applications de traitement de modèles.

Des tests unitaires peuvent (et donc doivent) être faits dans l'environnement Kermeta, comme nous le verrons par la suite.


II – Environnement : nouveau fichier







Un assistant dédié :

- Nom
- Emplacement dans le projet
- Paquetage de référence
- Classe principale
- Méthode de lancement



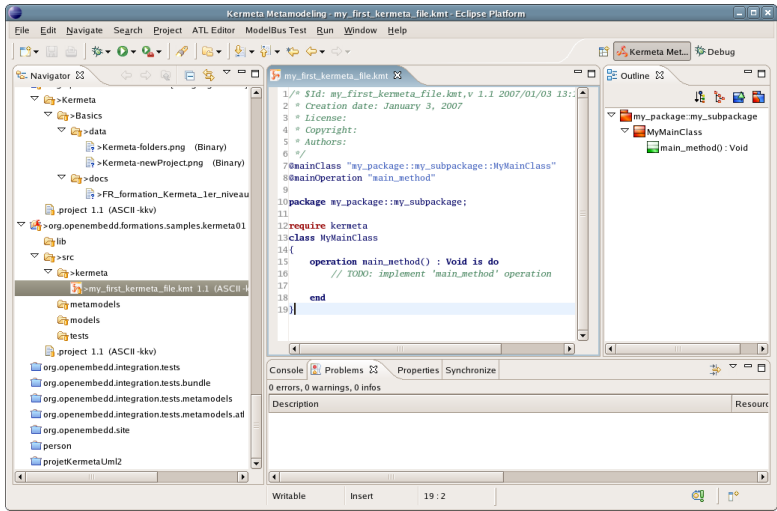
- Placer le nouveau fichier dans « src/kermeta »
- Utiliser une hiérarchie de paquetages cohérente avec les tâches à mener et les modèles à traiter par le projet.
- Classe « Main » et méthode « main() » pourront être supprimées après création si le nouveau code n'est pas un point d'entrée des traitements (pas de lancement).




II – Environnement : édition d'un fichier

- L'assistant génère le nouveau fichier et l'ouvre dans l'éditeur de texte :





Février 2009

Formation Kermeta : premier niveau

10

Les déclarations :

```
@mainClass "my_package::my_subpackage::MyMainClass"
@mainOperation "main_method"
```

sont là pour la configuration d'un point de lancement éventuel du code Kermeta.

Il est conseillé de les supprimer si le fichier n'est pas un point de lancement des traitements.

La vue « Outline » est mise à jour automatiquement et présente les différentes entités kermeta du fichier courant. Nous verrons la grande utilité de cette vue un peu plus loin.

II – Environnement : édition d'un fichier

- Ajoutons un peu de code au fichier :


```







1/* $Id: my_first_kermeta_file.kmt,v 1.1 2007/01/03 13:
2 * Creation date: January 3, 2007
3 * License:
4 * Copyright:
5 * Authors:
6 */
7@mainClass "my_package::my_subpackage::MyMainClass"
8@mainOperation "main_method"
9
10package my_package::my_subpackage;
11
12require kermeta
13class MyMainClass
14{
15    operation main_method() : Void is do
16        stdout,
17    end
18}
  
```

The screenshot shows an IDE window titled 'my_first_kermeta_file.kmt'. The code is a Kermeta script. At line 16, there is a red squiggly line under 'stdout,'. A dropdown menu is open, showing two options: 'write(object : String) : Void' and 'writeln(object : String) : Void'. Below the code editor, there is a 'Console' and 'Problems' tab. The 'Problems' tab shows '1 error, 0 warnings, 0 info'. The error is described as 'TYPE-CHECK'.

Notez la vérification syntaxique en cours de frappe ...

... et la complétion (appelée par **Ctrl+Space**)



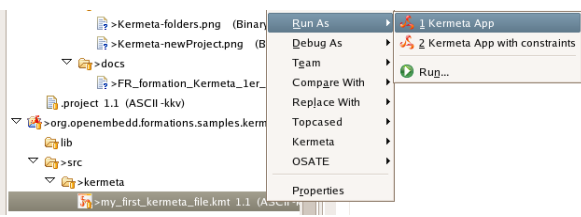







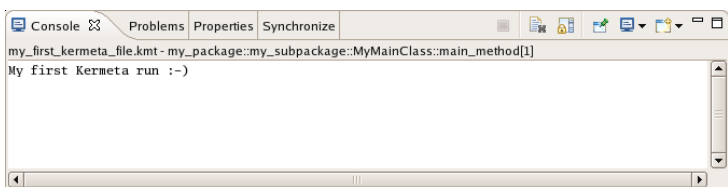
II – Environnement : exécuter Kermeta


- Complétons :


```

13 class 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 :










12

Une configuration plus poussée peut être faite par le menu « Run... », comme pour les programmes écrits sous Eclipse dans d'autres langages.

Il est possible de passer des paramètres à la méthode lancée, mais celle-ci ne peut avoir que des paramètres String déclarés (et donc fixés une fois pour toutes).



II – Environnement : KUnit

Tests unitaires avec KUnit

```

tests_suite01.kmt
/* $Id: tests_suite01.kmt,v 1.1 2007/01/09 08:17:59 vmahe Exp $
 * Creation date: January 3, 2007
 * License:
 * Copyright:
 * Authors:
 */

@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
}

```

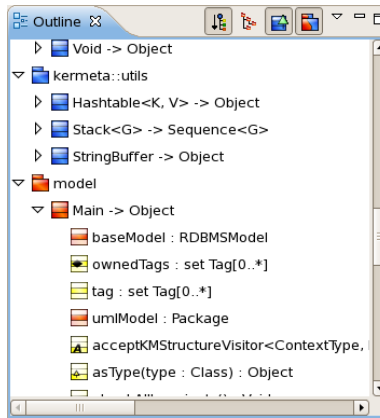

Févr

À la différence de Java, vous pouvez mettre plusieurs classes dans un même fichier source Kermeta.

Il importe cependant de structurer les différents tests unitaires compte-tenu de leur grand nombre dans un projet d'application non triviale.

II – Environnement : divers







• « Outline » : affichage des librairies



Penser à trier la vue par ordre alphabétique, pour faciliter les recherches d'information.

Très utile quand on cherche une structure inhabituelle dans une bibliothèque insérée mal connue. C'est LA VUE pour les utilisateurs Kermeta débutants.

Signification des icônes :

-  • Classe (avec « A » => abstraite)
-  • Propriété (triangle => héritée, crayon => dérivée, **diamant => composition**)
-  • Opération (triangle => méthode redéfinie)
-  • Datatype
-  • Enum
-  • Package

Signification des couleurs :

- Rouge : éléments créés / modifiés dans le KMT courant
- Bleu : éléments définis dans un méta-modèle importé (require)
- Mixte Rouge/Bleu : élément importé étendu par aspects dans le document courant
- Jaune : élément hérité d'une classe supérieure

Cliquez pour ajouter un titre

Kermeta

Le langage

III – 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
}
```

Exemple d'opération générique avec un type moins général :

```
class Utils {
    operation max<T : Comparable>(a : T, b : T) : T is do
        result := if a > b then a else b end
    end
}
```

III – 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
}
    
```

Éléments de syntaxe introduits :

- Mot-clé « inherits »
- « operation » redéfinie => « method »
- Mot-clé « super » pour appel de la méthode héritée

À noter :

- Nom unique pour les méthodes et les attributs => pas de surcharge (impossible d'avoir deux méthodes avec le même nom mais des paramètres différents, ni une méthode et un attribut de même nom).

III – 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
    // v1 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
}
end
```



Février 2009

Formation Kermeta : premier niveau

18

IMPORTANT :


Le choix entre « attribut » et « reference » se fait sur la base de la composition. Un attribut => un objet ne peut être pointé que par un seul attribut (une affectation => le précédent objet l'ayant eu comme attribut est désormais « void »).









Éléments de syntaxe introduits :

- « attribute », « reference », « var », « property »
- Déclaration
- Initialisation
- Affectation
- Méthode « toString » : nécessaire pour les objets autres que de type String
- Affectation du retour d'une méthode : « result »
- « package » : racine valable pour toutes les classes du fichier. NB : il est suivi d'un point-virgule.
- « require kermeta » : lien avec le paquetage Kermeta(contient toutes les bibliothèques de base du langage)

Remarques :

- Un choix technique dans EMOF empêche actuellement l'écriture de propriétés « write-only », même si elles sont prévues dans le langage. Malheureusement, un bug empêche en plus l'affectation aux « setter » (ce qui est plus gênant).



III – Langage Kermeta : généralités

Bloc de code :

 Conditions :

 Boucle :

 Exceptions :

```


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

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

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

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 block
end
      
```



Février 2009

Formation Kermeta : premier niveau

19

Éléments de syntaxe introduits :

- « do ... end »
- « if ... then ... else »
- « from ... until ... loop ... end »
- « raise »
- « do ... rescue() ... end »

Remarques :

- Différentes écritures et indentations possibles
- Les méthodes sans paramètres d'entrée peuvent être écrites sans parenthèses, comme des accesseurs ou variables dérivées

III – Langage Kermeta : généralités

- Commentaires**
 - Fin de ligne

// a "line" comment
 - Plusieurs lignes

/* 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
anAnr
end
class ForComments(
...
operation myAnnotatedMethod() is abstract
...
)
@descr "a named annotation"

```

- Sucre syntaxique**

```

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

class X
{
/* avoid writing kermeta::language::structure::Object */
reference anAnnotatedObject : Object
}

```

Février 2009


Formation Kermeta : premier niveau









20

Les annotations nommées peuvent être assimilées aux stéréotypes d'UML et permettent d'étendre le méta-modèle Kermeta. Nous les retrouverons dans la partie MODELES du cours. Deux d'entre elles sont utilisées pour repérer le point d'entrée par défaut d'un programme : « mainClass » et « mainOperation ».

Les annotations anonymes correspondent à l'annotation nommé « @kdoc » utilisée pour la documentation des programmes Kermeta.


Toutes les annotations sont visibles dans les bulles d'aide de l'éditeur (laisser traîner la souris sur une référence à un élément annoté pour voir l'information).



III – Langage Kermeta : généralités

- **Variables**
 - Syntaxe : a..z, A..Z, 0..9, « ~ », « _ »
 - Mots réservés : utilisables précédés de « ~ »
- **Énumérations**
 - Déclaration `enumeration Seasons { spring; summer; autumn; winter; }`
 - Usage `operation x (val : Seasons) is do
if val == Seasons.spring then stdio.writeln("It's Spring") end
end`
- **Types primitifs**
 - Integer <=> Java Integer
 - String <=> Java String mais peu de méthodes (append, ...)
 - Boolean <=> Java Boolean
 - Character [incomplet]
 - Real [incomplet]



Février 2009

Formation Kermeta : premier niveau

21

Il est fréquent de rencontrer des variables au nom correspondant à un mot réservé Kermeta dans les manipulations de modèles et méta-modèles.
Voir la vue Outline pour plus d'informations sur les méthodes disponibles sur les types primitifs.

Syntaxe limitée pour les primitifs Real et Character :

- 'a' impossible
- 3.14 et '3.14' impossibles

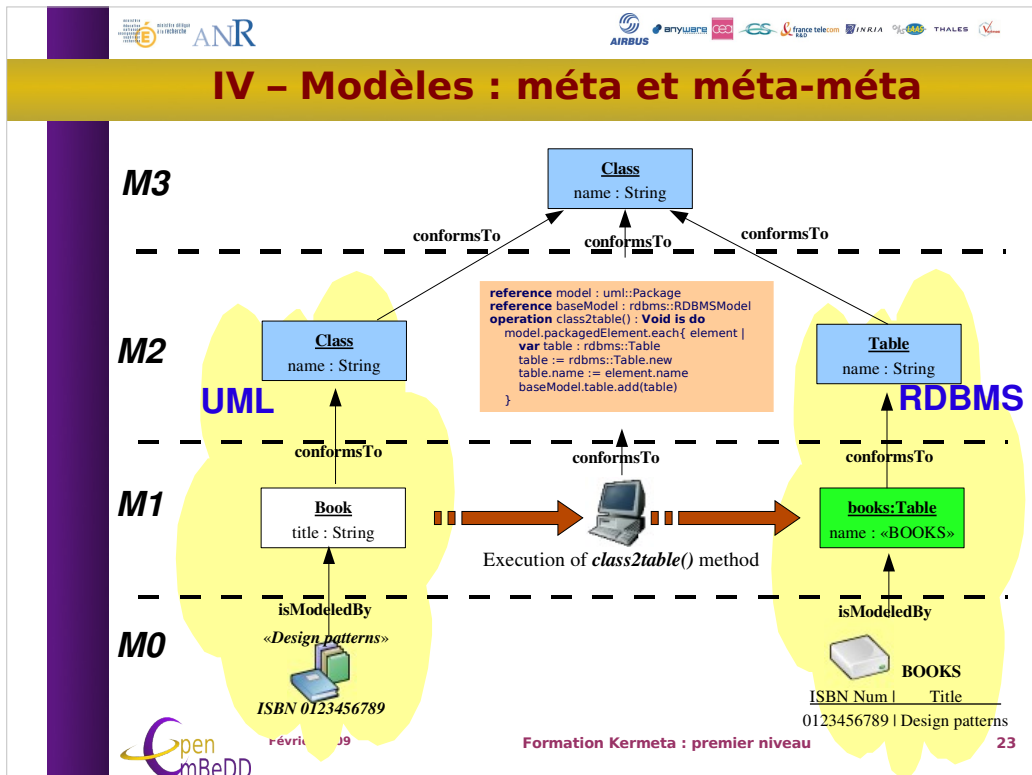
Utiliser :

- monChar := "toto".elementAt(0)
- monReal := "3.14".toReal

Cliquez pour ajouter un titre

IDM

**modèles,
méta-modèles,
méta-méta-modèles**



Les niveaux M0 à M3 peuvent être repérés par le nombre de « M » du niveau (instances, modèles, méta-modèles, et méta-méta-modèles).

En blanc : UML (diagramme d' instances et diagramme de classes).

En bleu ciel : EMOF (en fait, Kermeta remplace EMOF en l'étendant : voir plus loin).

En orange : Kermeta (méta-modèle avec comportement)

En vert : l'approche « base de données » => RDBMS*

Chaque classe du modèle UML donne une table dans la base de données.

On peut imaginer facilement la transformation inverse.

Pour éviter trop de niveaux méta, les méta-modèles sont décrits dans MOF, qui peut se décrire lui même. Ecore est une norme d'enregistrement de MOF dans un fichier (sérialisation).

* RDBMS = Relational DataBase Management System

VI – Modèles : chargement

- **Déclaration du méta-modèle**

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

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

// calling a plugged-in metamodel (the best)
require "http://www.eclipse.org/uml2/2.1.0/UML"
```

- **Chargement d'un modèle**

```
operation loadUmlModel() is do
    var inputRep : kermeta::persistence::EMFRepository init kermeta::persistence::EMFRepository.new
    var inputRes : kermeta::persistence::EMFResource
    inputRes := inputRep.getResource("../models/myUmlModel.uml",
                                     "platform:/plugin/org.eclipse.uml2.uml/model/UML.ecore")

    inputRes.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.xml",
                                                       "../metamodels/RDBMS.ecore")

    outputResource.instances.add(baseModel)
    outputResource.save()
end
```



Février 2009

Formation Kermeta : premier niveau

24

Rem: comme un modèle conforme EMF connaît son méta-modèle (stocké dans le XMI), il est possible de simplifier le chargement (création de la ressource + load) :

```
inputRes : EMFResource init
    inputRep.getResource(nomDuFichierModele)
```

Soit deux commandes (avec l'instance d'EMFRepository) au lieu des quatre ci-dessus. Cela réduit aussi les erreurs (plus besoin de connaître l'emplacement du méta-modèle).

Pour sauvegarder un nouveau modèle, il faut par contre préciser son méta-modèle (afin qu'il soit enregistré dans le XMI).

IV – 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 {}
```

usage

```
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
}
```

usage

```
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
  }
}
```

usage

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

usage

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

6

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

usage

```
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..*]
}
```

usage

```
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
}
```

usage

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

Février 2009

Formation Kermeta : premier niveau

25

Remarques :

- Une propriété à multiplicité > 1 devient par défaut un OrderedSet (2, 3, 6). Ces collections sont automatiquement initialisées lors du « new » de l'objet possédant la propriété.
- Les multiplicités ne sont pas contrôlées pour l'instant : [4..4] => possibilité de 0, 1, ..10 éléments présents dans cette propriété (impossible en l'absence de constructeur dans le langage)*.
- **Le diamant => un « attribute » et donc la perte de référence pour l'ancien propriétaire de l'objet attribué lors de sa ré-affectation à un nouvel attribut**

* voir si le contrôle des invariants vérifie ces multiplicités.

IV – 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 }`

each : effectuer un traitement sur chacun des éléments de la collection

forAll : vrai si tous les éléments de la collection vérifient la condition

select : retourne une collection contenant tous les éléments de la collection vérifiant la condition

reject : retourne une collection contenant tous les éléments de la collection NE vérifiant PAS la condition

collect : retourne une collection contenant une valeur (objet ou type primitif) pour tous les éléments de la collection

detect : retourne le premier objet de la collection vérifiant la condition

exists : retourne vrai si au moins un élément de la collection vérifie la condition

times : sur un entier n, exécute n fois le bloc de code

NB : la façon de créer ses propres lambda expressions est présentée dans le chapitre « Fonctionnalités avancées ».

IV – Modèles : collections

- 4 types de collections

	Not Ordered	Ordered
Unique	Set	OrderedSet
Not Unique	Bag	Sequence

- Usage :

```
var myColl1 : set Integer[0..*]
var myColl2 : oset String[0..*]
var myColl3 : bag Boolean[0..*]
var myColl4 : seq Package[0..*]

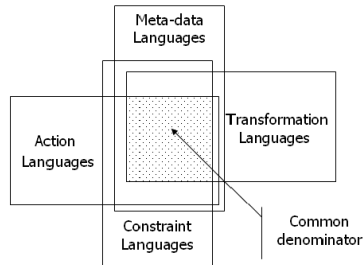
init kermeta::standard::Set<Integer>.new
init kermeta::standard::OrderedSet<String>.new
init kermeta::standard::Bag<Boolean>.new
init kermeta::standard::Sequence<Package>.new

// Fill in myColl1
myColl1.add(10)
myColl1.add(50)
```

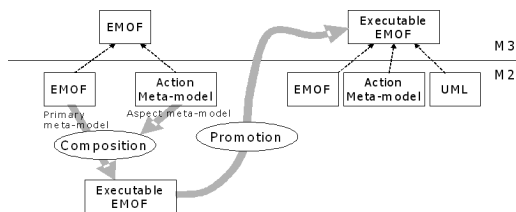
Actuellement, un bug sur la déclaration « bag » simple => utiliser la déclaration complète (sans multiplicité).

IV – Modèles : un langage d'action

- À la croisée des chemins, Kermeta :



- Kermeta ajoute une sémantique à EMOF :



On retrouve cela dans les bibliothèques Kermeta :

kermeta::language::structure : EMOF

kermeta::language::behavior : comportement ajouté à EMOF

IV – Modèles : un langage d'action

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

Février 2009

Formation Kermeta : premier niveau

29

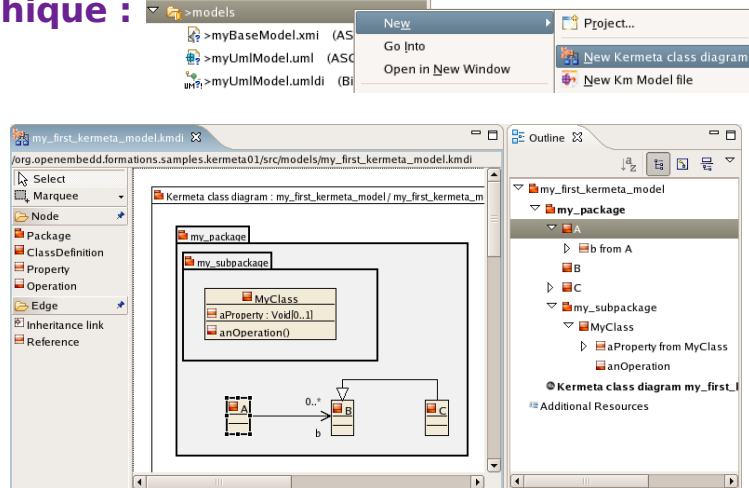
Voir les tags correspondant à « mainClass » et « mainOperation »

Il est possible de commencer l'écriture d'un traitement Kermeta dans l'éditeur graphique.

29

IV – Modèles : éditeur graphique

- Kermeta possède son propre éditeur graphique :



À noter qu'il est bien plus délicat d'écrire le code des procédures (même si cela peut être envisagé dans l'Outline).

Les principaux intérêts :

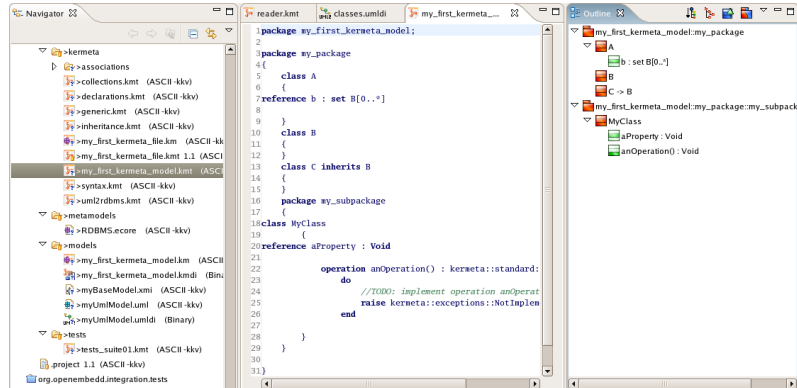
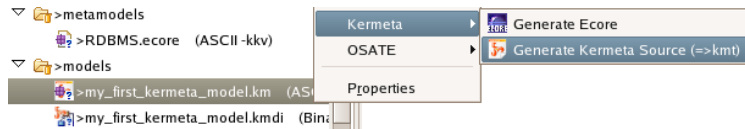
- Création d'un nouveau modèle Kermeta (en vue ou non d'une exécution)
- Insertion de classes provenant d'autres « modèles » Kermeta

Limites :

- Pour inclure un package dans un autre, double-cliquer sur le premier => crée un sous-diagramme

IV – Modèles : éditeur graphique

• Passer du graphique au code :



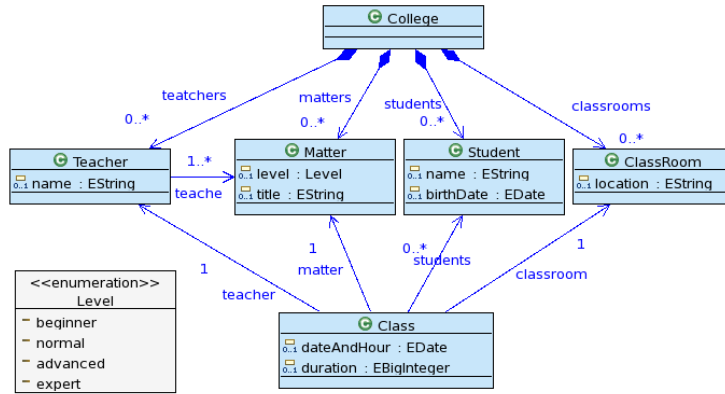
Cliquez pour ajouter un titre

Aspects

Enrichissez vos méta-modèles

V – Aspects : enrichissez vos méta-modèles

- Imaginez un méta-modèle d'école(s)



V – Aspects : enrichissez vos méta-modèles

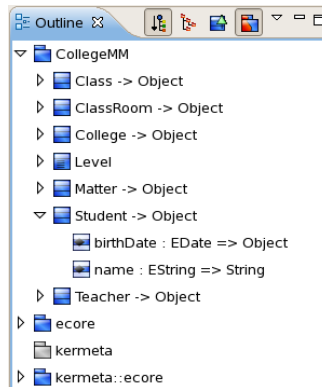
- **Créez un fichier kermeta qui le référence**

```
package CollegeMM;
```

```
require kermeta
```

```
require "platform:/resource/org.openembedd.formations.samples.kermeta01/metamodel/CollegeMM.ecore"
```

- **Vous obtenez l'outline correspondant**



Les éléments (re)définis dans le fichier courant sont en rouge.

Les éléments définis extérieurement (*Show imported types*) au fichier courant sont en bleu.

Les éléments hérités indirectement (*Flatten inheritance*) sont en jaune.

V – Aspects : enrichissez vos méta-modèles

- Un aspect permet d'ajouter une classe

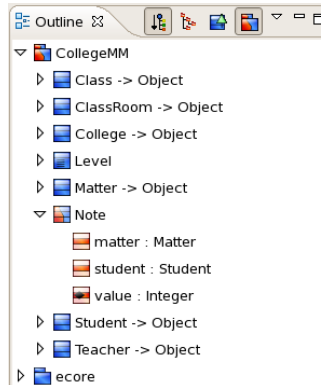
```
package CollegeMM;

require kermeta
require "platform:/resource/org.openembedd.formations.samples.kermeta01/metamodel/CollegeMM.ecore"

aspect class Note {
    attribute ~value : kermeta::standard::Integer

    reference student : Student
    reference matter : Matter
}
```

- L'outline affiche l'aspectisation



Le package CollegeMM est maintenant rouge/bleu, indiquant qu'il fait l'objet d'un aspect (au moins) dans le fichier courant.

De même pour la classe Note du package.

Il est maintenant possible de saisir des notes dans un « modèle de collège », bien que le méta-modèle originel ne prévoyait pas ce type.

REM : vous n'avez pas besoin de toucher au méta-modèle d'origine. Ça fonctionne même si celui-ci est propriétaire et packagé dans un plug-in.

V – Aspects : enrichissez vos méta-modèles

• Ajout des opposites + nouvelle opération

```
package CollegeMM;

require kermeta
require "platform:/resource/org.openembedd.formations.samples.kermeta01/metamodel/CollegeMM.ecore"

aspect class Note {
  attribute ~value : kermeta::standard::Real
  // add the opposite for managing notes from students/matters
  reference student : Student#notes
  reference matter : Matter#notes
}

aspect class Student {
  reference notes : Note[0..*]#student
  property average : kermeta::standard::Real
  getter is do
    var total : kermeta::standard::Real
    notes.each{ n | total := total + n.-value }
    result := total / notes.size.toReal
  end
}

aspect class Matter {
  reference notes : Note[0..*]#matter
  property average : kermeta::standard::Real
  getter is do
    var total : kermeta::standard::Real
    notes.each{ n | total := total + n.-value }
    result := total / notes.size.toReal
  end
}
```

V – Aspects : enrichissez vos méta-modèles

• Factoriser le traitement via l'héritage

```
package CollegeMM;

require kermeta
require "platform:/resource/org.openembedd. formations.samples.kermeta01/metamodel/CollegeMM.ecore"

aspect class Note {
  attribute ~value : kermeta::standard::Real
  // add the opposite for managing notes from students/matters
  reference student : Student#notes
  reference matter : Matter#notes
}

aspect class Notable {
  operation average(notes : Note[0..*]) : kermeta::standard::Real is do
    var total : kermeta::standard::Real
    notes.each{ n | total := total + n.-value }
    result := total / notes.size.toReal
  end
}

aspect class Student inherits Notable {
  reference notes : Note[0..*]#student
  property averageNote : kermeta::standard::Real
  getter is do
    result := average(notes)
  end
}

aspect class Matter inherits Notable {
  reference notes : Note[0..*]#matter
  property average : kermeta::standard::Real
  getter is do
    result := average(notes)
  end
}
```

La nécessité des opposite interdit la totale factorisation de la méthode « average() », par impossibilité de mettre la référence « notes » dans la classe « Notable ».

Il est possible aussi de surcharger une opération, pour une classe spécialisée, comme dans les langages Objet sérieux.

Cliquez pour ajouter un titre

Kermeta

Autres fonctionnalités

VI – Autres : 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
    pre noVoidInput is
      do first != void and second != void end

    post noVoidOutput is
      do result != void end

  // operation body
  is do
    result := first
    result.append(second)
  end
end
}
```

Attention :

pre et post conditions doivent être déclarées avant le corps de la méthode (avant son « is »).

VI – Autres : programmation par contrats

- Programme de test :

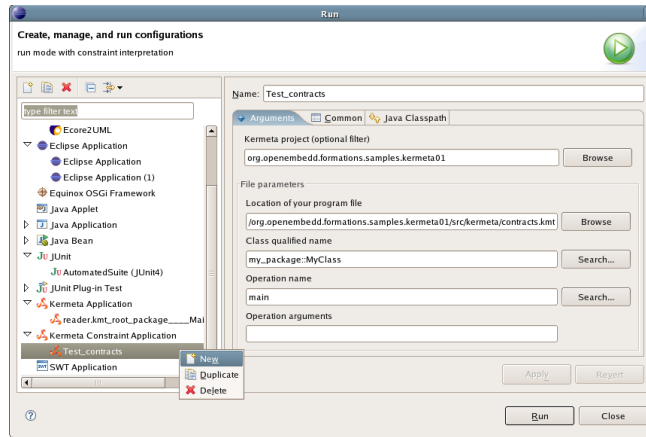
```
class MyClass
{
  operation main() : Void is do
    // new tool : its stringTable must be initialized
    var st1 : StringTool init StringTool.new
    st1.stringTable := Set<String>.new
    var s1 : String
    var s2 : String

    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
}
```

VI – Autres : programmation par contrats

Lancement en mode « contrats » :



Résultat :

```
contracts.kmt - my_package::MyClass:main (pre/post activated)[2]
*** expected violation *** [kermeta::exceptions::ConstraintViolatedPre:11734]
*** expected violation *** [kermeta::exceptions::ConstraintViolatedInv:11881]
```

VI - Autres : passerelle Java / Kermeta

■ Passerelle Java :

possibilité d'appeler des types et fonctions Java

```
/** An implementation of a StdIO class in Kermeta using existing Java standard input/output */
class StdIO {
  /** write the object to standard output */
  operation write(object : Object) : Void is do
    result ?= extern fr::irisa::triskell::kermeta::runtime::basetypes::StdIO.write(object)
  end
  /** read an object from standard input */
  operation read(prompt : String) : String is do
    result ?= extern fr::irisa::triskell::kermeta::runtime::basetypes::StdIO.read(prompt)
  end
}

/** Java Implementation of wrapper called from kermeta */
public class StdIO{
  // ..... //
  // Implementation of method read(prompt : String)
  public static RuntimeObject read(RuntimeObject prompt) {
    java.lang.String input = null;
  }
}
```

VI – Autres fonctionnalités

- **Expressions dynamiques**
interprétation à la volée de code passé en variable
- **λ expressions**
créer ses propres fonctions
- **ModelType**
rapprocher des méta-modèles
- **Fonctionnalités en cours de développement :**
 - Clone élaboré
 - ...