

Présentation d'OCL 2

Un langage de requête pour exprimer la sémantique statique des modèles

Benoît Combemale[†], Xavier Crégut[‡], Marc Pantel[‡]

[†] IRISA CNRS Laboratory, *University of Rennes 1*
benoit.combemale@irisa.fr

[‡] IRIT CNRS Laboratory, *University of Toulouse*
{xavier.cregut, marc.pantel}@enseeiht.fr

dernière mise à jour le 24 octobre 2010

Support disponible à l'adresse (*teaching part*) :
<http://perso.univ-rennes1.fr/benoit.combemale/>

1 Contexte et Motivation

- Intérêt des modèles
- Modèles et méta-modèles
- OCL, un langage de requête pour exprimer des contraintes
- Exemples d'utilisation d'UML

2 Présentation générale d'OCL

- Objectifs initiaux
- Historique
- Propriétés du langage
- Les différentes utilisations d'OCL

3 Syntaxe du langage

- Les types OCL
- Contexte d'une expression OCL
- Syntaxe des différentes utilisations
- Navigation dans le modèle
- Les opérations de la bibliothèque standard
- Les opérateurs sur les collections

4 Outils : éditeurs et évaluateurs OCL

5 Conclusion

1 Contexte et Motivation

- Intérêt des modèles
- Modèles et méta-modèles
- OCL, un langage de requête pour exprimer des contraintes
- Exemples d'utilisation d'UML

2 Présentation générale d'OCL

3 Syntaxe du langage

4 Outils : éditeurs et évaluateurs OCL

5 Conclusion

Rôle d'un modèle

- On utilise des modèles pour mieux comprendre un système.

Pour un observateur A , M est un modèle de l'objet O , si M aide A à répondre aux questions qu'il se pose sur O . (Minsky)

- Un modèle est une simplification, une abstraction du système.

- Exemples :

- une carte routière
- une partition de musique
- un diagramme UML

- Un modèle peut permettre de :

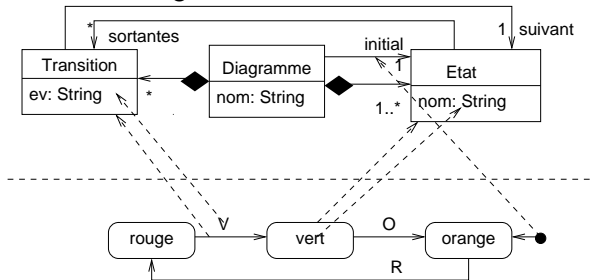
- de comprendre,
- de communiquer,
- de construire

Modèles et méta-modèles

Définition : Méta-modèle = modèle du modèle.

⇒ Il s'agit de décrire la structure du modèle.

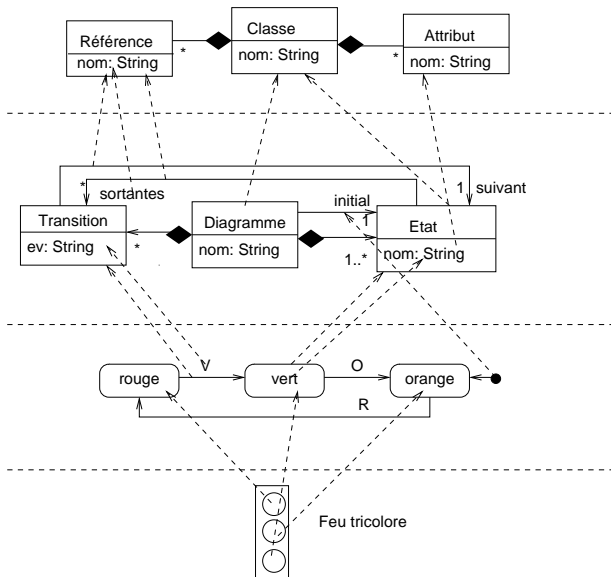
Exemple : Structure d'un diagramme à état



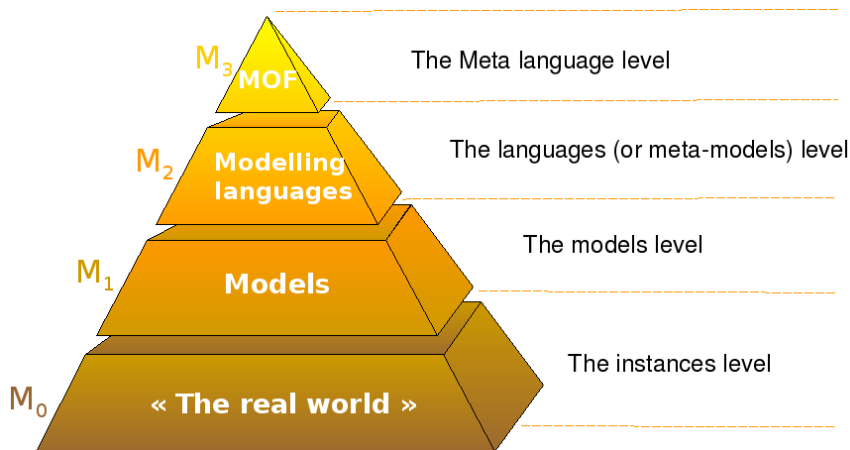
Conformité : On dit qu'un modèle est conforme à un méta-modèle si :

- chaque élément du modèle est instance d'un élément du méta-modèle et ;
- chaque contrainte exprimée sur le méta-modèle est respectée sur le modèle.

Exemple : le monde réel est un feu tricolore



Pyramide de l'OMG



Pyramide de l'OMG

Explications

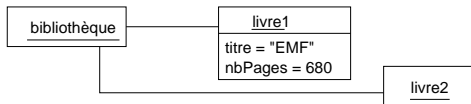
- M3 : méta-méta-modèle :
 - réflexif : se décrit en lui-même
 - pour définir des méta-modèles, des langages (comme UML)
 - Exemple : MOF de l'OMG, Ecore de Eclipse/EMF, KM3 de AMMA...
- M2 : méta-modèle : langage de modélisation pour un domaine métier
 - Exemples : UML2, SPEM...
- M1 : modèle : un modèle du monde réel
 - Exemples : un modèle de feu tricolore, un modèle de bibliothèque...
- M0 : le monde réel
 - Exemples : un feu tricolore, une bibliothèque...

Remarque : Le numéro permet de préciser l'objectif du « modèle ». Dans la suite, les notions de modèle et méta-modèle sont suffisantes.

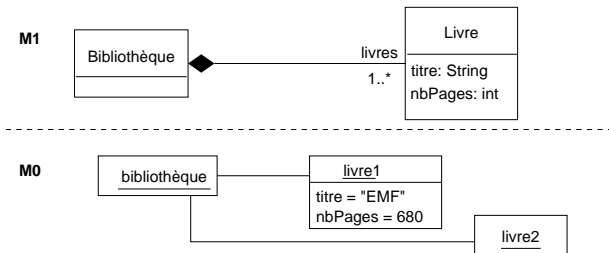
Pas nouveau : Par exemple Grammarware (EBNF, syntaxe de Java, Programme Java, exécution du programme)

Exemple : le monde réel est une bibliothèque

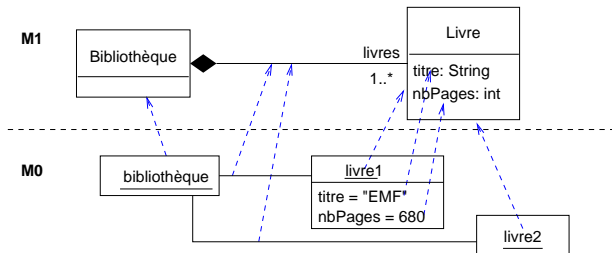
M0



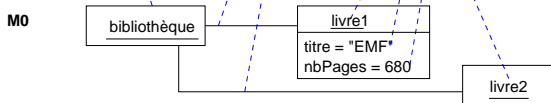
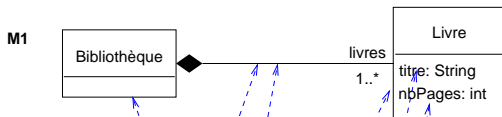
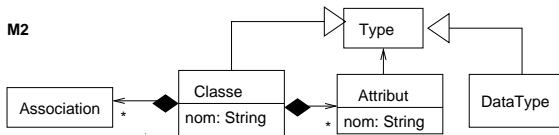
Exemple : le monde réel est une bibliothèque



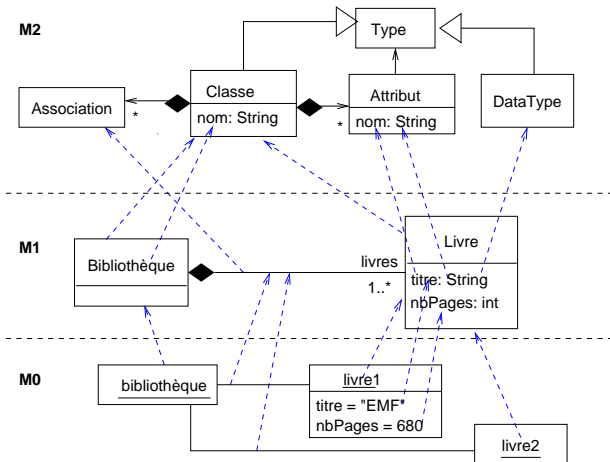
Exemple : le monde réel est une bibliothèque



Exemple : le monde réel est une bibliothèque

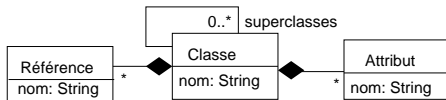


Exemple : le monde réel est une bibliothèque

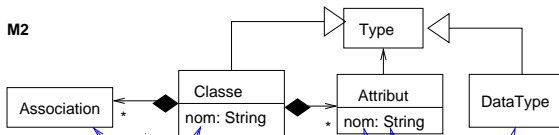


Exemple : le monde réel est une bibliothèque

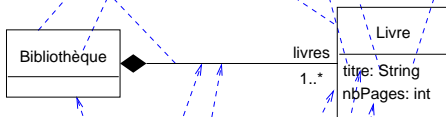
M3



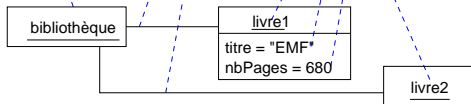
M2



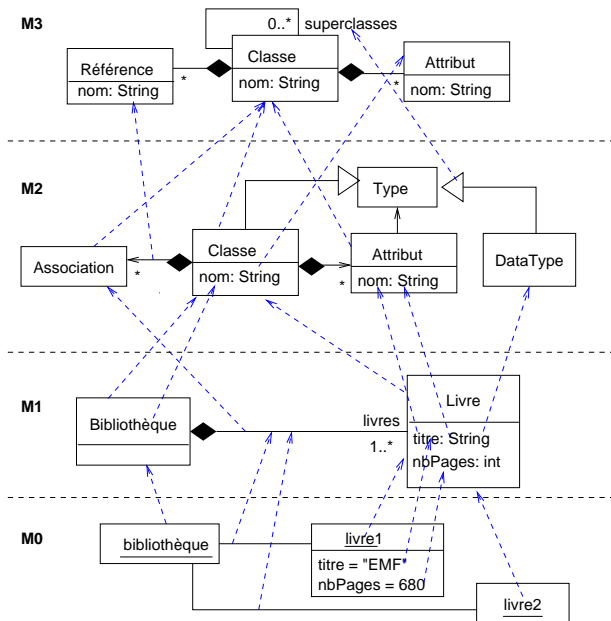
M1



M0



Exemple : le monde réel est une bibliothèque



Intérêt des méta-modèles

- La sémantique d'un langage de programmation est définie sur le langage (M2), pas sur le programme (M1).
- L'objectif est de faire pareil avec les modèles.
- Pouvoir raisonner sur l'ensemble des modèles :
 - limiter les instances possibles d'un méta-modèle en ajoutant des contraintes non capturées par le méta-modèle.

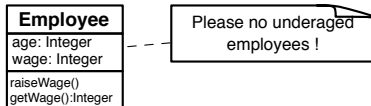
Exemple : Ajouter une contrainte OCL pour exprimer que le nombre de pages d'un livre est positif.

context Livre **inv**: nbPages > 0

- construire une syntaxe concrète (textuelle ou graphique)
- transformer le modèle (restructuration, raffinement...)
- ...

Préciser la sémantique statique d'un modèle

- Les langages graphiques (p.ex. UML) ne décrivent très souvent qu'un aspect partiel du système.
- Les contraintes sont souvent décrites (si elles existent) sous la forme de notes marginales en langage naturel



- ⇒ presque toujours ambigu,
 - ⇒ imprécis,
 - ⇒ vérification automatique impossible.
- ⇒ Les langages formels sont un bon complément au langage naturel

Utilisation d'OCL

Objectif général

Objectif : OCL est avant tout un **langage de requête** qui permet de calculer une *expression sur un modèle en s'appuyant sur sa syntaxe* (son méta-modèle).

⇒ Une expression exprimée une fois, pourra être évaluée sur tout modèle conforme au méta-modèle correspondant.

Exemple : pour une bibliothèque particulière on peut vouloir demander :

- Quels sont les livres possédées par la bibliothèque ? Combien y en a-t-il ?
- Quels sont les auteurs dont au moins un titre est possédé par la bibliothèque ?
- Quels sont tous les titres dans la bibliothèque écrits par Martin Fowler ?
- Quel est le nombre de pages du plus petit ouvrage ?
- Quel est le nombre moyen de pages des ouvrages ?
- Quels sont les ouvrages de plus 100 pages écrits par au moins trois auteurs ?
- ...

Utilisations : Tout ce qui s'appuie sur de telles expressions !

Utilisation d'OCL

Programmation par contrat

Principe : Établir formellement les responsabilités d'une classe et de ses méthodes.

Moyen : définition de propriétés (expressions booléennes) appelées :

- **invariant** : propriété définie sur une **classe** qui doit toujours être vraie, de la création à la disparition d'un objet.

Un invariant lie les requêtes d'une classe (état externe).

- **précondition** : propriété sur une **méthode** qui :
 - doit être vérifiée par l'appelant pour que l'appel à cette méthode soit possible ;
 - peut donc être supposée vraie dans le code de la méthode.

postconditions : propriété sur une **méthode** qui définit l'effet de la méthode, c'est-à-dire :

- spécification de ce que doit écrire le programmeur de la méthode ;
- caractérisation du résultat que l'appelant obtiendra.

Exercice : Invariant pour une Fraction (état = numérateur et dénominateur) ?

Exercice : Pré- et postconditions de racine carrée et de pgcd ?

Utilisation d'OCL

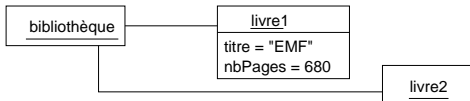
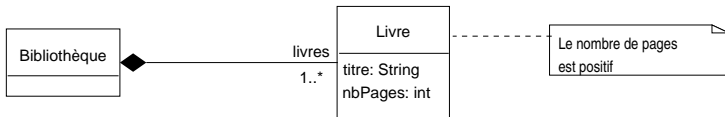
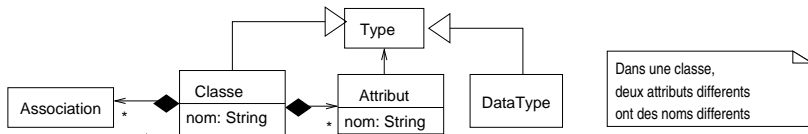
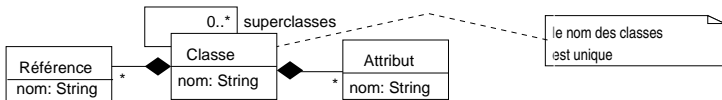
Diagrammes d'UML

OCL peut être utilisé sur différents diagrammes d'UML :

- diagramme de classe :
 - définir des préconditions, postconditions et invariants :
Stéréotypes prédéfinis : «precondition», «postcondition» et «invariant»
 - caractérisation d'un **attribut dérivé** (p.ex. le salaire est fonction de l'âge)
 - spécifier la **valeur initiale** d'un attribut (p.ex. l'attribut *salaire* d'un employé)
 - spécifier le **code d'une opération** (p.ex. le salaire annuel est 12 fois le salaire mensuel)
- diagramme d'état :
 - spécifier une garde sur une transition
 - exprimer une expression dans une activité (affectation, etc.)
 - ...
- diagramme de séquence :
 - spécifier une garde sur un envoi de message
- ...

Utilisation d'OCL

Méta-modélisation : préciser la sémantique statique d'un modèle

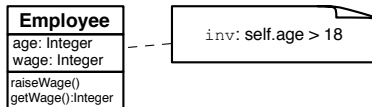


- 1 Contexte et Motivation
- 2 **Présentation générale d'OCL**
 - Objectifs initiaux
 - Historique
 - Propriétés du langage
 - Les différentes utilisations d'OCL
- 3 Syntaxe du langage
- 4 Outils : éditeurs et évaluateurs OCL
- 5 Conclusion

The *Object Constraint Language*

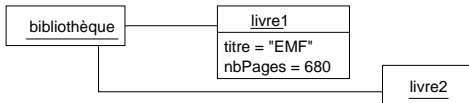
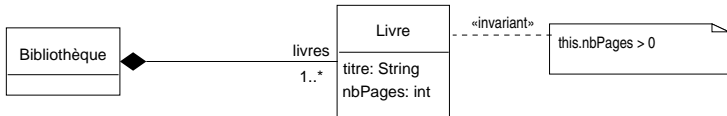
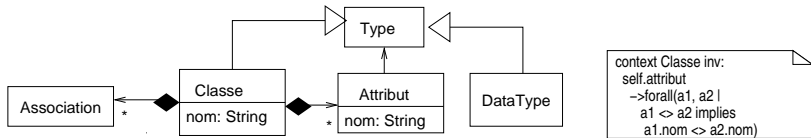
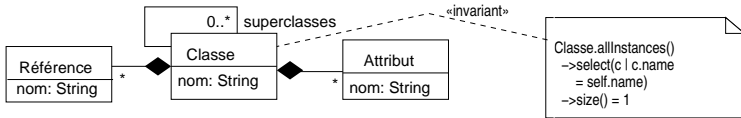
Objectifs initiaux

- Les langages formels traditionnels (e.g. Z) requièrent de la part des utilisateurs une bonne compréhension des fondements mathématiques.
- *Object Constraint Language (OCL)* a été développé dans le but d'être :
 - formel, précis et non ambigu,



- utilisable par un large nombre d'utilisateurs,
- un langage de spécification (et non de programmation !),
- supporté par des outils.

Préciser la sémantique statique d'un modèle



The *Object Constraint Language*

Historique

- Développé en 1995 par IBM,
- Inclu dans le standard UML jusqu'à la version 1.1 (1997),
- OCL 2.0 *Final Adopted Specification* (ptc/06-05-01), May 2006.

The *Object Constraint Language*

Propriétés du langage

- **Langage de spécification sans effet de bord**

- une expression OCL calcule une valeur... **et** laisse le modèle inchangé !
 - ⇒ l'état d'un objet ne peut pas être modifié **par** l'évaluation d'une expression OCL
- l'évaluation d'une expression OCL est instantanée
 - ⇒ l'état des objets ne peut donc pas être modifié **pendant** l'évaluation d'une expression OCL
- OCL n'est pas un langage de programmation !

- OCL est un **langage typé** :

- Chaque expression OCL a un type
- OCL définit des types primitifs : **Boolean**, **Integer**, **Real** et **String**
- Chaque *Classifier* du modèle est un nouveau type OCL
- *Intérêt* : vérifier la cohérence des expressions
exemple : il est interdit de comparer un String et un Integer

- 1 Contexte et Motivation
- 2 Présentation générale d'OCL
- 3 Syntaxe du langage
 - Les types OCL
 - Contexte d'une expression OCL
 - Syntaxe des différentes utilisations
 - Navigation dans le modèle
 - Les opérations de la bibliothèque standard
 - Les opérateurs sur les collections
- 4 Outils : éditeurs et évaluateurs OCL
- 5 Conclusion

Les types OCL de base

Les types de base (*Primitive*) sont **Integer**, **Real**, **Boolean** et **String**. Les opérateurs suivants s'appliquent sur ces types :

Opérateurs relationnels	<code>=, <>, >, <, >=, <=</code>
Opérateurs logiques	<code>and, or, xor, not, if ... then ... else ... endif</code>
Opérateurs mathématiques	<code>+, -, /, *, min(), max()...</code>
Opérateurs pour les chaînes de caractères	<code>concat, toUpper, substring...</code>

Attention : Concernant l'opérateur `if ... then ... else ... endif` :

- la clause **else** est nécessaire et,
- les expressions du **then** et du **else** doivent être de même type.

Attention : `and`, `or`, `xor` ne sont pas évalués en court-circuit !

Priorité des opérateurs

Liste des opérateurs dans l'ordre de priorité décroissante :

- | | | |
|----|---|--------------------------------------|
| 1 | @pre | |
| 2 | . | <i>— notation pointée et fléchée</i> |
| 3 | not | <i>— opérateurs unaires</i> |
| 4 | * / | |
| 5 | + - | <i>— opérateurs binaires</i> |
| 6 | if-then-else-endif | |
| 7 | < > <= >= | |
| 8 | = <> | |
| 9 | and or xor | |
| 10 | implies | <i>— implication</i> |

Remarque : Les parenthèses peuvent être utilisées pour changer la priorité.

Les autres types OCL

- Tous les éléments du modèle sont des types (*OclModelElementType*),
 - y compris les énumérations : *Gender :: male*,
- Type *Tuple* : enregistrement (produit cartésien de plusieurs types)
$$Tuple \{a : Collection(Integer) = Set\{1, 3, 4\}, b : String = 'foo'\}$$
- *OclMessageType* :
 - utilisé pour accéder aux messages d'une opération ou d'un signal,
 - offre un rapport sur la possibilité d'envoyer/recevoir une opération/un signal.
- *VoidType* :
 - a seulement une instance *oclUndefined*,
 - est conforme à tous les types.

Contexte d'une expression OCL

Une expression est définie sur un **contexte** qui identifie :

- une **cible** : l'élément du modèle sur lequel porte l'expression OCL

T	Type (Classifier : Interface, Classe...)	context Employee
M	Opération/Méthode	context Employee::raiseWage(inc: Int)
A	Attribut ou extrémité d'association	context Employee::job : Job

- le **rôle** : indique la **signification** de cette expression (pré, post, invariant...) et donc contraint sa **cible** et son **évaluation**.

rôle	cible	signification	évaluation
inv	T	invariant	toujours vraie
pre	M	précondition	avant tout appel de M
post	M	postcondition	après tout appel de M
body	M	résultat d'une requête	appel de M
init	A	valeur initiale de A	création
derive	A	valeur de A	utilisation de A
def	T	définir une méthode ou un attribut	

Syntaxe d'OCL

inv (invariant) doit toujours être vrai (avant et après chaque appel de méthode)

context Employee

inv: self.age > 18

context e : Employee

inv age_18: e.age > 18

pre (precondition) doit être vraie avant l'exécution d'une opération

post (postcondition) doit être vraie après l'exécution d'une opération

context Employee::raiseWage(increment : **Integer**)

pre: increment > 0

post my_post: self.wage = self.wage@pre + increment

context Employee::getWage() : **Integer**

post: result = self.wage

Remarques : result et @pre : utilisables seulement dans une postcondition

- exp@pre correspond à la valeur de expr avant l'appel de l'opération.
- result est une variable prédéfinie qui désigne le résultat de l'opération.

Syntaxe d'OCL

- **body** spécifie le résultat d'une opération

```
context Employee::getWage() : Integer
  body: self .wage
```

- **init** spécifie la valeur initiale d'un attribut ou d'une association

```
context Employee::wage : Integer
  init : 900
```

- **derive** spécifie la règle de dérivation d'un attribut ou d'une association

```
context Employee::wage : Integer
  derive : self .age * 50
```

- **def** définition d'opérations (ou variables) qui pourront être (ré)utilisées dans des expressions OCL.

```
context Employee
  def: annualIncome : Integer = 12 * wage
```

La navigation dans le modèle

Accès aux informations de la classe

- Une expression OCL est définie dans le contexte d'une classe
 - en fait : un type, une interface, une classe, etc.
- Elle s'applique sur un objet, instance de cette classe :
⇒ cet objet est désigné par le mot-clé **self**.
- Étant donné un accès à un objet (p.ex. `self`), une expression OCL peut :
 - accéder à la valeur des attributs :
 - `self.nbPages`
 - `unLivre.nbPages`
 - appeler toute requête définie sur l'objet :
 - `self.getNbPages()`
 - `unLivre.getNbPages()`

Rappel : Une requête (notée {isQuery} en UML) est une opération :

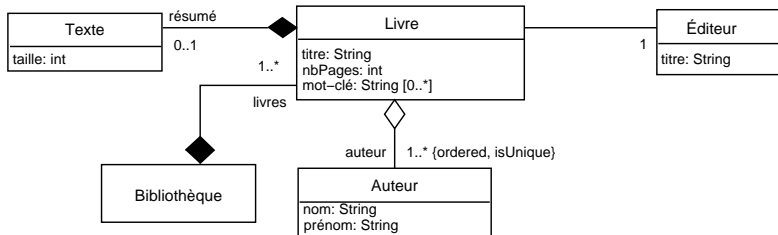
- qui a un type de retour (calcule une expression);
- et n'a pas d'effet de bord (ne modifie pas l'état du système).

Attention : une opération avec effet de bord est proscrite en OCL !

- parcourir les associations...

Correspondance entre association et OCL

- Exemple de diagramme de classe



- pour atteindre l'autre extrémité d'une association, on utilise :
 - le rôle, p.ex. : `unLivre.résumé`
 - à défaut le nom de la classe en minuscule : `unLivre.éditeur`
- La manière dont une association est vue en OCL dépend :
 - de sa multiplicité : un exactement (`1`), optionnel (`0..1`), ≥ 2
 - de ses qualificatifs : `{isUnique}`, `{isOrdered}`

Correspondance entre association et OCL

association avec multiplicité ≤ 1

- multiplicité 1 : nécessairement un objet à l'extrémité (invariant implicite)
 - `unLivre. éditeur`
- multiplicité 0..1 (optionnel) :
 - utiliser l'opération `oclIsUndefined()`
 - `unLivre.résumé.oclIsUndefined()` est :
 - vraie si pas de résumé,
 - faux sinon
 - Exemple d'utilisation :

```
if unLivre.résumé.oclIsUndefined() then
    true
else
    uneLivre.résumé. taille  >= 60
endif
```

Correspondance entre association et OCL

association avec multiplicité ≥ 2

- les éléments à l'extrémité d'une association sont accessibles par une collection
- OCL définit quatre types de collection :
 - **Set** : pas de double, pas d'ordre
 - **Bag** : doubles possibles, pas d'ordre
 - **OrderedSet** : pas de double, ordre
 - **Sequence** : doubles possibles, ordre
- Lien entre associations UML et collections OCL

UML	Ecore	OCL
		Bag
isUnique	Unique	Set
isOrdered	Ordered	Sequence
isUnique, isOrdered	Unique, Ordered	OrderedSet

- Exemple : `unLivre.auteur` : la collection des auteurs de unLivre

Les collections OCL

- *Set* : ensemble d'éléments *sans* doublon et *sans* ordre

Set {7, 54, 22, 98, 9, 54, 20..25}

— *Set*{7,54,22,98,9,20,21,23,24,25} : *Set(Integer)*

— *ou Set*{7,9,20,21,22,23,24,25,54,98} : *Set(Integer)*, *ou ...*

- *OrderedSet* : ensemble d'éléments *sans* doublon et *avec* ordre

OrderedSet {7, 54, 22, 98, 9, 54, 20..25}

— *OrderedSet*{7,9,20,21,22,23,24,25,54,98} : *OrderedSet(Integer)*

- *Bag* : ensemble d'éléments *avec* possibilité de doublon et *sans* ordre

Bag {7, 54, 22, 98, 9, 54, 20..25}

— *p.ex.* : *Bag*{7,9,20,21,22,22,23,24,25,54,54,98} : *Bag(Integer)*

- *Sequence* : ensemble d'éléments *avec* possibilité de doublon et *avec* ordre

Sequence{7, 54, 22, 98, 9, 54, 20..25}

— *Sequence*{7,54,22,98,9,54,20,21,22,23,24,25} : *Sequence(Integer)*

N.B. : Les collections sont génériques : *Bag(Integer)*, *Set(String)*, *Bag(Set(Livre))*

Opérations de la bibliothèque standard pour les collections

Pour tous les types de Collection

- size ()**: **Integer** — *nombre d'éléments dans la collection self*
- includes**(object: T): **Boolean** — *est-ce que object est dans self ?*
- excludes**(object: T): **Boolean** — *est-ce que object n'est pas dans self ?*
- count**(object: T): **Integer** — *nombre d'occurrences de object dans self*
- includesAll**(c2: **Collection**(T)): **Boolean**
 — *est-ce que self contient tous les éléments de c2 ?*
- excludesAll** (c2: **Collection**(T)): **Boolean**
 — *est-ce que self ne contient aucun des éléments de c2 ?*
- isEmpty**(): **Boolean** — *est-ce que self est vide ?*
- notEmpty**(): **Boolean** — *est-ce que self est non vide ?*
- sum**(): T — *la somme (+) des éléments de self*
 — *l'opérateur + doit être défini sur le type des éléments de self*
- product**(c2: **Collection**(T2)): **Set**(**Tuple**(premier: T, second: T2))
 — *le produit (*) des éléments de self*

Opérations de la bibliothèque standard pour les collections

En fonction du sous-type de *Collection*, d'autres opérations sont disponibles :

- union
- intersection
- append
- flatten
- =
- ...

Une liste exhaustive des opérations de la bibliothèque standard pour les collections est disponible dans [OMG OCL 2.0, §11.7].

Opérations de la bibliothèque standard pour tous les objets

OCL définit des opérations qui peuvent être appliquées à tous les objets

- *oclIsTypeOf*(*t* : *OclType*) : *Boolean*

Le résultat est vrai si le type de *self* et *t* sont identiques.

context Employee

inv: self .**oclIsTypeOf**(Employee) — *is true*

inv: self .**oclIsTypeOf**(Company) — *is false*

- *oclIsKindOf*(*t* : *OclType*) : *Boolean*

vrai si *t* est le type de *self* ou un super-type de *self*.

- *oclIsNew*() : *Boolean*

Uniquement dans les post-conditions

vrai si le récepteur a été créé au cours de l'exécution de l'opération.

- *oclIsInState*(*t* : *OclState*) : *Boolean*

Le résultat est vrai si l'objet est dans l'état *t*.

Opérations de la bibliothèque standard pour tous les objets

OCL définit des opérations qui peuvent être appliquées à tous les objets

- $oclAsType(t : OclType) : T$
Retourne le même objet mais du type t
Nécessite que $oclIsKindOf(t) = true$
- $allInstances()$
 - prédéfinie pour les classes, les interfaces et les énumérations,
 - le résultat est la collection de toutes les instances du type au moment de l'évaluation.

context Employee

inv: Employee.allInstances() \rightarrow **forAll**(p1, p2
| p1 <> p2 **implies** p1.name <> p2.name)

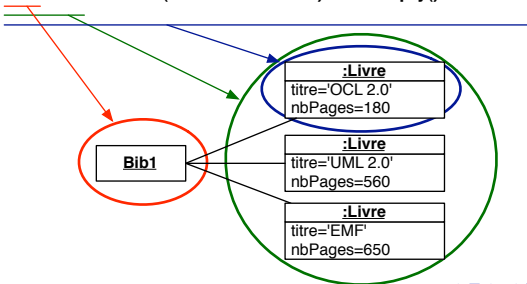
Opérateur *select* (resp. *reject*)

Permet de spécifier le sous-ensemble de tous les éléments de *collection* pour lesquels l'expression est vraie (resp. fausse pour *reject*).

- *collection* → *select(elem : T|expr)*
- *collection* → *select(elem|expr)*
- *collection* → *select(expr)*

context Bibliothèque inv:

self.livres->select(name = 'OCL 2.0')->notEmpty()



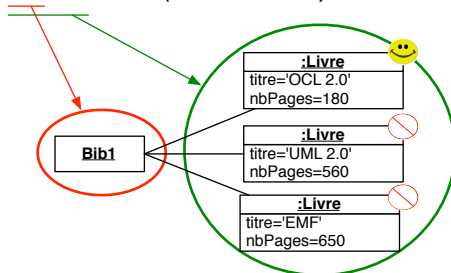
Opérateur *exists*

Retourne vrai si l'expression est vraie pour au moins un élément de la collection.

- $collection \rightarrow exists(elem : T | expr)$
- $collection \rightarrow exists(elem | expr)$
- $collection \rightarrow exists(expr)$

context Bibliothèque inv:

self.livres->exists(name = 'OCL 2.0')

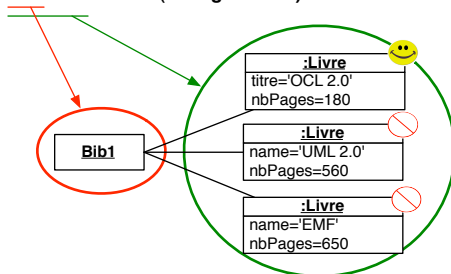


Opérateur *forAll*

Retourne vrai si l'expression est vraie pour tous les éléments de la collection.

- $collection \rightarrow forAll(elem : T | expr)$
- $collection \rightarrow forAll(elem | expr)$
- $collection \rightarrow forAll(expr)$

context Bibliothèque inv:
self.livres->forAll(nbPages < 200)

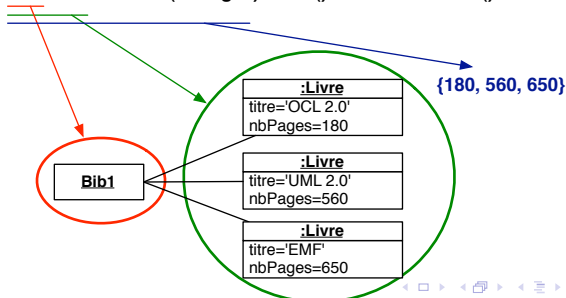


Opérateur *collect*

Retourne la collection des valeurs (*Bag*) résultant de l'évaluation de l'expression appliquée à tous les éléments de collection.

- $collection \rightarrow collect(elem : T | expr)$
- $collection \rightarrow collect(elem | expr)$
- $collection \rightarrow collect(expr)$

```
context Bibliothèque def moyenneDesPages : Real =  
  self.livres->collect(nbPages)->sum() / self.livres->size()
```



Opérateur *iterate*

Forme générale d'une itération sur une collection et permet de redéfinir les précédents opérateurs.

```
collection ->iterate(elem : Type;  
    answer : Type = <value>  
    | <expression_with_elem_and_answer>)
```

```
context Bibliothèque def moyenneDesPages : Real =  
    self . livres ->collect(nbPages)->sum() / self.livres->size()
```

-- *est identique à* :

```
context Bibliothèque def moyenneDesPages : Real =  
    self . livres ->iterate(l : Livre ;  
        lesPages : Bag{Integer} = Bag{  
            | lesPages->including(l.nbPages))  
        ->sum() / self.livres ->size()
```

Plusieurs itérateurs pour un même opérateur

Remarque : les opérateurs *forAll*, *exist* et *iterate* acceptent plusieurs itérateurs :

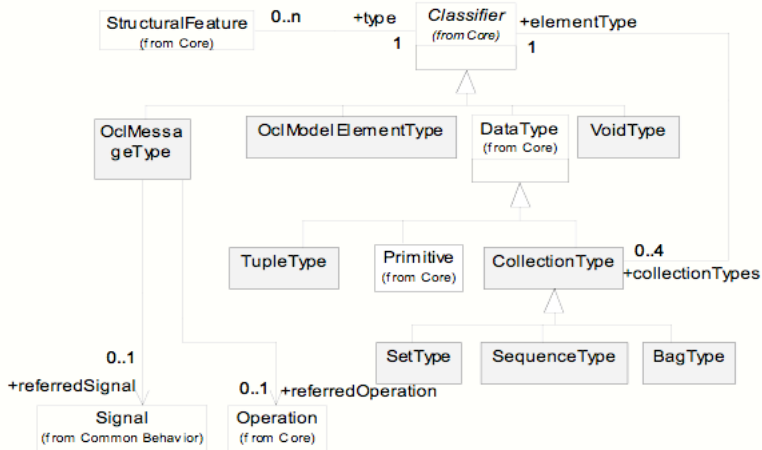
```
Auteur.allInstances() -> forAll(a1, a2 |  
    a1 <> a2 implies  
    a1.nom <> a2.nom or a1.prénom <> a2.prénom)
```

Bien sûr, dans ce cas il faut nommer tous les itérateurs !

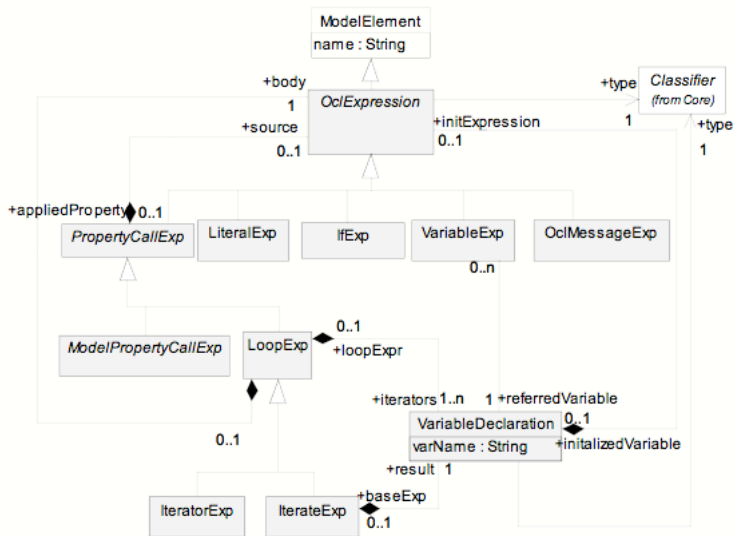
Le méta-modèle d'OCL

- OCL 2.0 à (bien sûr...) un méta-modèle qui définit sa syntaxe abstraite
 - ⇒ une expression OCL est un modèle !
- Méta-modèle des types OCL :
 - ⇒ OCL est un langage typé,
 - ⇒ définit des types supplémentaires à ceux d'UML (collection, tuple, ...)
- Méta-modèle des expressions OCL
 - ⇒ définit les expressions OCL possibles.

OCL Types metamodel



OCL Expression metamodel



- 1 Contexte et Motivation
- 2 Présentation générale d'OCL
- 3 Syntaxe du langage
- 4 Outils : éditeurs et évaluateurs OCL**
- 5 Conclusion

Outils supportant le standard OCL

Il existe de nombreux outils qui permettent de vérifier la syntaxe, la sémantique et d'évaluer une expression OCL :

- Use 2.3 (cf. <http://www.db.informatik.uni-bremen.de/projects/USE/>)
- Dresden OCL Toolkit 2.0 (cf. <http://dresden-ocl.sourceforge.net/>)
- LCI OCL Evaluator OCLE 2.0.4 (cf. <http://lci.cs.ubbcluj.ro/ocle/>)
- The Kent OCL library v1 (cf. <http://www.cs.kent.ac.uk/projects/ocl/>)
- Octopus OCL (cf. <http://octopus.sourceforge.net/>)
- RoclET (<http://www.roclet.org>)
- Topcased (<http://www.topcased.org>)
- ...

Plus d'outils : <http://www.klasse.nl/ocl/ocl-tools.html> ...

Testez vos requêtes OCL...

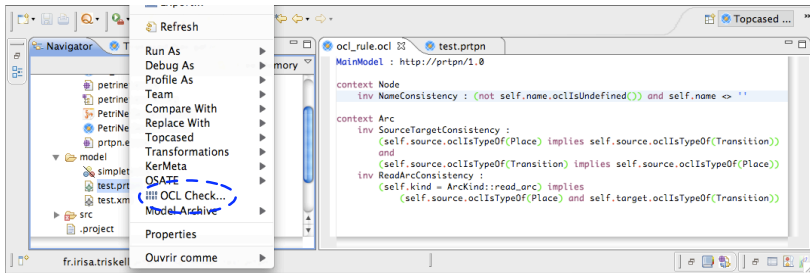
Utilisation de l'atelier Topcased

- Atelier *open source* couvrant les différentes phases de développement et intégrant les contraintes de certification
- Différents langages de modélisation (UML, AADL, SysML, SAM,...)
- Éditeurs graphiques
- Outils de traçabilité, de génération, de transformation et de V & V

Testez vos requêtes OCL...

Utilisation de l'atelier Topcased

- Atelier *open source* couvrant les différentes phases de développement et intégrant les contraintes de certification
- Différents langages de modélisation (UML, AADL, SysML, SAM et OCL,...)
- Éditeurs graphiques
- Outils de traçabilité, de génération, de transformation et de V & V



Site Internet : <http://www.topcased.org> & Serveur de développement : <http://gforge.enseeiht.fr/>
Documentation de l'outil OCL : Help → Help contents → Topcased User Guide → OCL Tools

Testez vos requêtes OCL...

Utilisation de Kermeta

- Modèles, Métamodèles, métamétamodèles, DSLs...
 - "Méta-bla-bla" : trop complexe pour l'ingénieur λ
- Par contre, les concepts suivants lui sont familiers :
 - Programmation orientée objets (Java, C#, C++...)
 - UML (au moins les diagrammes de classes)
 - Notion de *design by contract* (pré,post, invariants)
- Kermeta utilise ces points pour supporter la métamodélisation

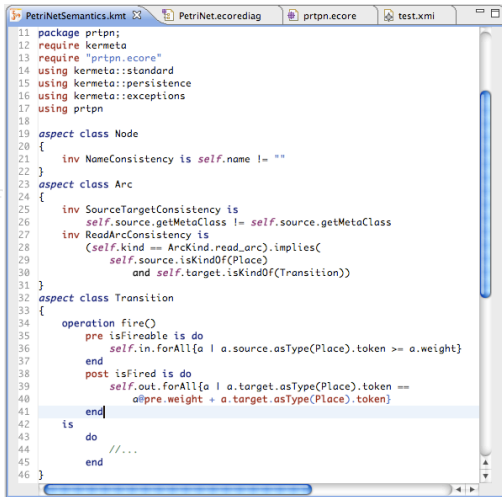


"Kermeta - Breathe life into your metamodels"

Testez vos requêtes OCL...

Utilisation de Kermeta : expression des invariants et des pre-, post- conditions

- Use AOM to weave your static semantics into your metamodel !
- The activation of the checking of the pre - post conditions depends of the run configuration
- If the boolean statement is evaluated to FALSE then the pre or post condition is violated and an exception `CONSTRAINTVIOLATEDPRE` or `CONSTRAINTVIOLATEDPOST` is raised.

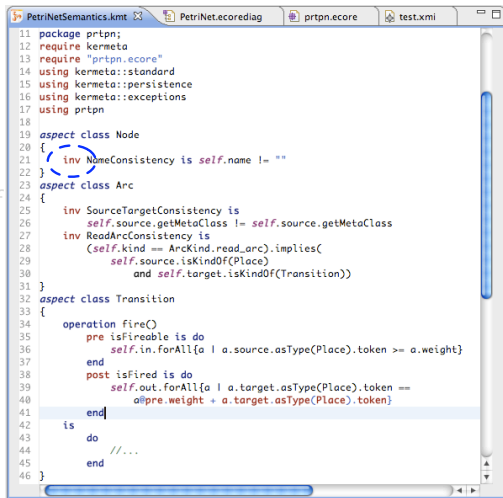


```
11 package prtpn;  
12 require kermeta  
13 require "prtpn.ecore"  
14 using kermeta::standard  
15 using kermeta::persistence  
16 using kermeta::exceptions  
17 using prtpn  
18  
19 aspect class Node  
20 {  
21     inv NameConsistency is self.name != ""  
22 }  
23 aspect class Arc  
24 {  
25     inv SourceTargetConsistency is  
26         self.source.getMetaClass != self.source.getMetaClass  
27     inv ReadArcConsistency is  
28         (self.kind == ArcKind.read_arc).implies(  
29             self.source.isKindOf(Place)  
30             and self.target.isKindOf(Transition))  
31 }  
32 aspect class Transition  
33 {  
34     operation fire()  
35     pre isFireable is do  
36         self.in.forAll{a | a.source.asType(Place).token >= a.weight}  
37     end  
38     post isFired is do  
39         self.out.forAll{a | a.target.asType(Place).token ==  
40             @pre.weight + a.target.asType(Place).token}  
41     end  
42 is  
43 do  
44     //...  
45 end  
46 }
```

Testez vos requêtes OCL...

Utilisation de Kermeta : expression des invariants et des pre-, post- conditions

- Use AOM to weave your static semantics into your metamodel !
- The activation of the checking of the pre - post conditions depends of the run configuration
- If the boolean statement is evaluated to FALSE then the pre or post condition is violated and an exception
CONSTRAINTVIOLATEDPRE or
CONSTRAINTVIOLATEDPOST is raised.



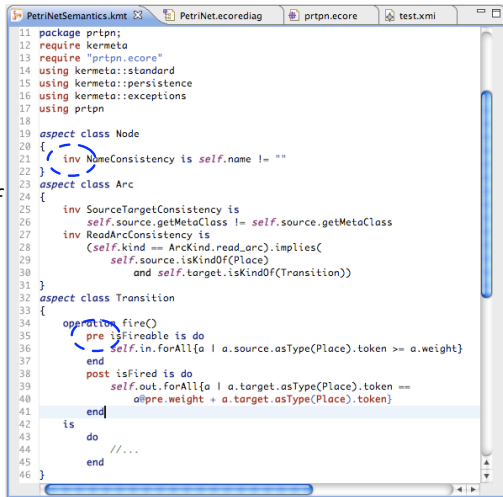
```
11 package prtpn;  
12 require kermeta  
13 require "prtpn.ecore"  
14 using kermeta::standard  
15 using kermeta::persistence  
16 using kermeta::exceptions  
17 using prtpn  
18  
19 aspect class Node  
20 {  
21   inv NameConsistency is self.name != ""  
22 }  
23 aspect class Arc  
24 {  
25   inv SourceTargetConsistency is  
26     self.source.getMetaClass != self.source.getMetaClass  
27   inv ReadArcConsistency is  
28     (self.kind == ArcKind.read_arc).implies(  
29       self.source.isKindOf(Place)  
30       and self.target.isKindOf(Transition))  
31 }  
32 aspect class Transition  
33 {  
34   operation fire()  
35     pre isFireable is do  
36       self.in.forAll{a | a.source.asType(Place).token >= a.weight}  
37     end  
38     post isFired is do  
39       self.out.forAll{a | a.target.asType(Place).token ==  
40         a@pre.weight + a.target.asType(Place).token}  
41     end  
42   is  
43     do  
44       //...  
45     end  
46 }
```

Testez vos requêtes OCL...

Utilisation de Kermeta : expression des invariants et des pre-, post- conditions

- Use AOM to weave your static semantics into your metamodel !
- The activation of the checking of the pre - post conditions depends of the run configuration
- If the boolean statement is evaluated to FALSE then the pre or post condition is violated and an exception

CONSTRAINTVIOLATEDPRE or
CONSTRAINTVIOLATEDPOST is
raised.



```
11 package prtpn;
12 require kermeta
13 require "prtpn.ecore"
14 using kermeta::standard
15 using kermeta::persistence
16 using kermeta::exceptions
17 using prtpn
18
19 aspect class Node
20 {
21     inv NameConsistency is self.name != ""
22 }
23 aspect class Arc
24 {
25     inv SourceTargetConsistency is
26         self.source.getMetaClass != self.source.getMetaClass
27     inv ReadArcConsistency is
28         (self.kind == ArcKind.read_arc).implies(
29             self.source.isKindOf(Place)
30             and self.target.isKindOf(Transition))
31 }
32 aspect class Transition
33 {
34     operation fire()
35     pre isFireable is do
36         self.in.forAll{a | a.source.asType(Place).token >= a.weight}
37     end
38     post isFired is do
39         self.out.forAll{a | a.target.asType(Place).token ==
40             a@pre.weight + a.target.asType(Place).token}
41     end
42     is
43     do
44         //...
45     end
46 }
```

Testez vos requêtes OCL...

Utilisation de Kermeta : expression des invariants et des pre-, post- conditions

- Use AOM to weave your static semantics into your metamodel !
- The activation of the checking of the pre - post conditions depends of the run configuration
- If the boolean statement is evaluated to FALSE then the pre or post condition is violated and an exception

CONSTRAINTVIOLATEDPRE or
CONSTRAINTVIOLATEDPOST is
raised.

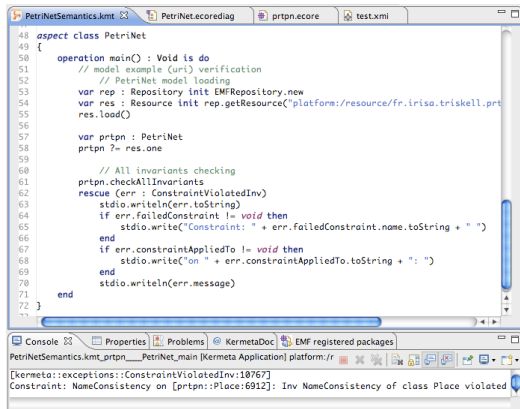
```
11 package prtpn;  
12 require kermeta  
13 require "prtpn.ecore"  
14 using kermeta::standard  
15 using kermeta::persistence  
16 using kermeta::exceptions  
17 using prtpn  
18  
19 aspect class Node  
20 {  
21   inv NameConsistency is self.name != ""  
22 }  
23 aspect class Arc  
24 {  
25   inv SourceTargetConsistency is  
26     self.source.getMetaClass != self.source.getMetaClass  
27   inv ReadArcConsistency is  
28     (self.kind == ArcKind.read_arc).implies(  
29       self.source.isKindOf(Place)  
30       and self.target.isKindOf(Transition))  
31 }  
32 aspect class Transition  
33 {  
34   operation fire()  
35     pre isFireable is do  
36       self.in.forAll{a | a.source.asType(Place).token >= a.weight}  
37     end  
38     post isFired is do  
39       self.out.forAll{a | a.target.asType(Place).token ==  
40         a@pre.weight + a.target.asType(Place).token}  
41     end  
42   is  
43     do  
44       //...  
45     end  
46 }
```

Testez vos requêtes OCL...

Utilisation de Kermeta : vérification des invariants

Two methods in Kermeta to check the well-formedness rules of a model element :

- CHECKINVARIANTS, check only the current model element
- CHECKALLINVARIANTS, checks recursively the element being a containment link with the checked element



The screenshot shows the Kermeta IDE with several tabs: PetriNetSemantics.kmt, PetriNet.ecorediag, prtpn.ecore, and test.xml. The main editor displays the code for the PetriNetSemantics.kmt file, which includes a class PetriNet with a main() operation. The code performs model verification, loading a resource and checking invariants. The console at the bottom shows the execution results, indicating a constraint violation: "Constraint: NameConsistency on [prtpn:Place:6912]: Inv NameConsistency of class Place violated".

```
48 aspect class PetriNet
49 {
50   operation main() : Void is do
51     // model example (uri) verification
52     // PetriNet model loading
53     var rep : Repository init EMFRepository.new
54     var res : Resource init rep.getResource("platform:/resource/fr.irisa.triskell.prt
55     res.load()
56
57     var prtpn : PetriNet
58     prtpn ?= res.one
59
60     // All invariants checking
61     prtpn.checkAllInvariants
62     rescue (err : ConstraintViolatedInv)
63       stdio.writeln(err.toString)
64       if err.failedConstraint != void then
65         stdio.write("Constraint: " + err.failedConstraint.name.toString + " ")
66       end
67       if err.constraintAppliedTo != void then
68         stdio.write("on " + err.constraintAppliedTo.toString + ": ")
69       end
70       stdio.writeln(err.message)
71     end
72   }
73 }
```

Console: [kermeta:exceptions::ConstraintViolatedInv:10767]
Constraint: NameConsistency on [prtpn:Place:6912]: Inv NameConsistency of class Place violated

Testez vos requêtes OCL...

Utilisation de Kermeta : vérification des invariants

Two methods in Kermeta to check the well-formedness rules of a model element :

- CHECKINVARIANTS, check only the current model element
- CHECKALLINVARIANTS, checks recursively the element being a containment link with the checked element

The screenshot shows the Kermeta IDE with a PetriNet model loaded. The main editor displays the Kermeta script for the PetriNet model, which includes a `main()` operation that initializes the repository, loads the model, and checks all invariants. The console window at the bottom shows the execution results, indicating a constraint violation: `[kermeta::exceptions::ConstraintViolatedInv:10767] Constraint: NameConsistency on [prtpn::Place:6912]: Inv NameConsistency of class Place violated`.

```
48 aspect class PetriNet
49 {
50   operation main() : Void is do
51     // model example (uri) verification
52     // PetriNet model loading
53     var rep : Repository init EMFRepository.new
54     var res : Resource init rep.getResource("platform:/resource/fr.irisa.triskell.prt
55     res.load()
56
57     var prtpn : PetriNet
58     prtpn ?= res.one
59
60     // All invariants checking
61     prtpn.checkAllInvariants
62     rescue (err : ConstraintViolatedInv)
63       stdio.writeln(err.toString)
64       if err.failedConstraint != void then
65         stdio.write("Constraint: " + err.failedConstraint.name.toString + " ")
66       end
67       if err.constraintAppliedTo != void then
68         stdio.write("on " + err.constraintAppliedTo.toString + ": ")
69       end
70       stdio.writeln(err.message)
71     end
72   end
73 }
```

Console
Properties
Problems
KermetaDoc
EMF registered packages

PetriNetSemantics.kmt_prtpn__PetriNet_main [Kermeta Application] platform:/r
[kermeta::exceptions::ConstraintViolatedInv:10767]
Constraint: NameConsistency on [prtpn::Place:6912]: Inv NameConsistency of class Place violated

Testez vos requêtes OCL...

Utilisation de Kermeta : vérification des invariants

Two methods in Kermeta to check the well-formedness rules of a model element :

- CHECKINVARIANTS, check only the current model element
- CHECKALLINVARIANTS, checks recursively the element being a containment link with the checked element

```
48 aspect class PetriNet
49 {
50   operation main() : Void is do
51     // model example (uri) verification
52     // PetriNet model loading
53     var rep : Repository init EMFRepository.new
54     var res : Resource init rep.getResource("platform:/resource/fr.irisa.triskell.prt
55     res.load()
56
57     var prtpn : PetriNet
58     prtpn ?= res.one
59
60     // All invariants checking
61     prtpn.checkAllInvariants
62     rescue (err : ConstraintViolatedInv)
63       stdio.writeln(err.toString)
64       if err.failedConstraint != void then
65         stdio.write("Constraint: " + err.failedConstraint.name.toString + " ")
66       end
67       if err.constraintAppliedTo != void then
68         stdio.write("on " + err.constraintAppliedTo.toString + ": ")
69       end
70       stdio.writeln(err.message)
71     end
72   }
73 }
```

Console
PetriNetSemantics.kmt_prtprn__PetriNet_main [Kermeta Application] platform:/r
[kermeta::exceptions::ConstraintViolatedInv:10767]
Constraint: NameConsistency on [prtpn::Place:6912]: Inv NameConsistency of class Place violated

Testez vos requêtes OCL...

Utilisation de Kermeta : vérification des invariants

Two methods in Kermeta to check the well-formedness rules of a model element :

- CHECKINVARIANTS, check only the current model element
- CHECKALLINVARIANTS, checks recursively the element being a containment link with the checked element

The screenshot shows the Kermeta IDE with a PetriNet model loaded. The main editor displays the PetriNet model's code, which includes a `main()` method that checks invariants. Two sections of the code are circled with dashed blue lines: the `main()` method signature and the `checkAllInvariants` method call. The bottom console window shows the execution results, indicating a constraint violation: `[kermeta::exceptions::ConstraintViolatedInv:10767] Constraint: NameConsistency on [prtpn::Place:6912]: Inv NameConsistency of class Place violated`.

```
48 aspect class PetriNet
49 {
50   operation main() : Void is do
51     // model example (uri) verification
52     // PetriNet model loading
53     var rep : Repository init EMFRepository.new
54     var res : Resource init rep.getResource("platform:/resource/fr.irisa.triskell.prt
55     res.load()
56
57     var prtpn : PetriNet
58     prtpn ?= res.one
59
60     // All invariants checking
61     prtpn.checkAllInvariants
62     rescue (err : ConstraintViolatedInv)
63       stdio.writeln(err.toString)
64       if err.failedConstraint != void then
65         stdio.write("Constraint: " + err.failedConstraint.name.toString + " ")
66       end
67       if err.constraintAppliedTo != void then
68         stdio.write("on " + err.constraintAppliedTo.toString + ": ")
69       end
70       stdio.writeln(err.message)
71     end
72   end
73 }
```

Console: [kermeta::exceptions::ConstraintViolatedInv:10767] Constraint: NameConsistency on [prtpn::Place:6912]: Inv NameConsistency of class Place violated

- 1 Contexte et Motivation
- 2 Présentation générale d'OCL
- 3 Syntaxe du langage
- 4 Outils : éditeurs et évaluateurs OCL
- 5 Conclusion**

Conclusion

- OCL est un **langage formel de requête**
 - Une requête est exprimée en terme des éléments de la syntaxe du modèle (diagramme de classe ou méta-modèle)
 - OCL est utilisé partout une expression est utile :
 - programmation par contrat
 - attribut dérivés, corps d'une méthode, garde sur une transition, ...
- **OCL ne remplace pas les explications en langage naturel.**
 - Les deux sont *complémentaires*!
 - *Comprendre* (informel)
 - *Lever les ambiguïtés* (OCL)
- **Éviter les expressions OCL trop compliquées**
 - éviter les navigations complexes
 - bien choisir le contexte (associer l'invariant au bon type!)
 - éviter d'utiliser **allInstances** () :
 - rend souvent les invariants plus complexes
 - souvent difficile d'obtenir toutes les instances dans un système (sauf BD!)
 - décomposer une conjonction de contraintes en plusieurs (inv, post, pre)
 - Toujours nommer les extrémités des associations (indiquer le rôle des objets)