École de technologie supérieure
Département de Génie logiciel et des TI

Document thématique

OCL en 7 exemples – Partie I: les notions de base

Auteur(s) : Sylvie Ratté
Date de création : 9 février, 2005
Réviseurs : Sylvie Ratté
Date de révision : 10 janvier, 2011

Remerciements

A special thanks to Math Dwyer from University of Nebraska, John Hatcliff and Rod Howell from Kansas State University (KSU) to have given me the authorization to use and adapt their class material. All the examples in USE are originally theirs. The chronological order in which the subjects are presented is inspired by their slides for the course CIS771: Software Specification at KSU.

1 Introduction

Afin de pouvoir réaliser MDA, nos modèles doivent être de plus en plus précis. OCL est un langage qui permet d'apporter cette précision en offrant une manière d'exprimer des contraintes.

Ce document, et les suivants, introduit les notions d'OCL en utilisant un exemple qui est raffiné peu à peu. Il s'agit d'un modèle d'une structure « académique ».

Les spécifications USE ainsi que les contraintes OCL sont augmentées au fur et à mesure des besoins.

Les exemples étudiés sont disponibles dans les fichiers correspondants.

Toute spécification USE débute par le mot « model » suivi du nom de ce modèle.

La stratégie de vérification que nous adoptons avec cet outil consiste à créer différents instantanés du modèle : certains respectant toutes les contraintes, d'autres permettant de violer ces contraintes.

2 Le monde académique

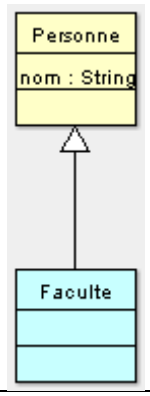
2.1 Trois exemples de base

academique-1.use

Illustration de la syntaxe de base dans une spécification USE (à gauche) et sa représentation graphique en ArgoUML (à droite).

```
class Personne
attributes
  nom : String
end

class Faculte < Personne
end
```



academique-2.use

Les enseignants peuvent être habituellement des membres de la faculté ou des étudiants gradués. En UML, il sera possible de représenter cet héritage multiple.

Premier essai en USE :

```
class Enseignant < Faculte, Et_gradue
end
```

Malheureusement, cette manière d'écrire signifie plutôt que l'Enseignant implémente à la fois le comportement de la classe Faculte ET de la classe Et_gradue.

Deuxième essai :

```
class Enseignant < Personne
end

class Etudiant < Personne
end

class Faculte < Enseignant
end

class Et_gradue < Etudiant, Enseignant
end
```

Mais cela signifierait que tout étudiant gradué est aussi un enseignant.

Inspiration Java :

En Java, nous pourrions faire en sorte que Faculte et Et_gradue implémentent tous les deux l'interface Enseignant mais USE ne propose pas cette notion.

Nous aurions pu également implémenter Enseignant comme une classe « adapter ». C'est la solution qui a été adoptée ici. Des contraintes OCL viendront plus tard limiter cette personne à appartenir à Faculte ou Et_gradue.

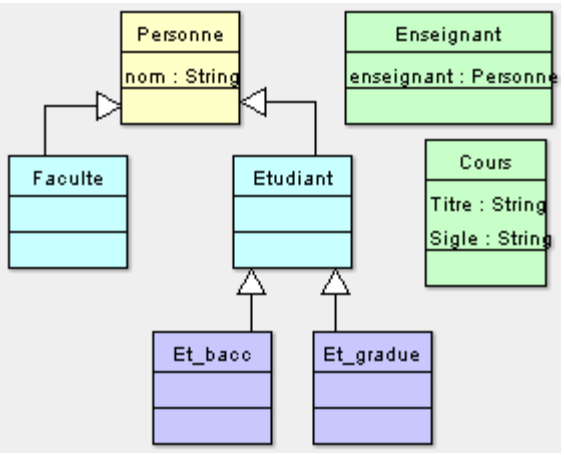
```
class Cours
attributes
  titre: String
  sigle: String
end

class Etudiant < Personne
end

class Et_bacc < Etudiant
end

class Et_gradue < Etudiant
end

class Enseignant
attributes
  enseignant:Personne
end
```



academique-3.use

Ajout ici de quelques relations :

- Un enseignant enseigne un cours.
- Un étudiant est inscrit à un cours.
- Un étudiant est sur la liste d'attente pour s'inscrire à un cours.

La syntaxe utilisée par USE est simple :

```
association <nom-de-l-association> between
  <classe-partiticipante-1>[<multiplicité>] role <role-1>
  <classe-partiticipante-2>[<multiplicité>] role <role-2>
end
```

Remarques :

- Les « *aggregation* » et les « *composition* » sont également disponibles (remplacez simplement le mot « association » par l'un des deux précédents).
- Les associations ternaires et quaternaires sont également possibles. Il suffit de poursuivre l'énumération des classes participantes, de leur multiplicité et de leur rôle.
- Si le rôle est absent, le nom de ce rôle sera celui de la classe participante dont le premier caractère sera placé en minuscule.

```
association Enseignement between
  Enseignant[1] role enseignant
  Cours[*] role cours
end

association InscritsDans between
  Etudiant[1..*] role etudiants
  Cours[*] role cours
end

association EnAttentePour between
  Etudiant[*] role listeAttente
  Cours[*] role listeAttente
end
```

2.2 Vérification par instantanés

Un exemple de script pour tester le modèle.

```
Fichier : Academique-script-3.cmd
      !create sylvie:Faculte
      !set sylvie.nom := 'Sylvie Ratté'
      !create pierre:Faculte
      !set pierre.nom := 'Pierre Dumouchel'
      !create michel:Et_gradue
      !set michel.nom := 'Michel Halde'
      !create fouad:Et_gradue
      !set fouad.nom := 'Fouad Khelouiati'
      !create denis: Et_bacc
      !set denis.nom := 'Denis Tremblay'
      !create marc: Et_bacc
      !set marc.nom := 'Marc Labrèche'
      !create log670:Cours
      !set log670.sigle := 'LOG670'
      !set log670.titre := 'Lg + ou - formels'
      !create mgl806:Cours
      !set mgl806.sigle := 'MGL806'
      !set mgl806.titre := 'Métho + ou - formels'
      !create instrSylvie:Enseignant
      !set instrSylvie.enseignant := sylvie
      !create instrMichel: Enseignant
      !set instrMichel.enseignant := michel
      !insert (instrMichel, log310) into Enseignement
      !insert (instrSylvie, mgl806) into Enseignement
      !insert (marc, log310) into InscritsDans
      !insert (denis, log310) into EnAttentePour
      !insert (fouad, mgl806) into InscritsDans
```

3 Le monde académique contraint

Quelques invariants de base :

- Tous les enseignants sont membre d'une faculté ou sont des étudiants gradués.
- Un étudiant ne peut être sur la liste d'attente d'un cours à moins d'y être inscrit.
- Un étudiant gradué ne peut enseigner un cours auquel il est inscrit.

3.1 Un exemple

Forme de base des invariants en OCL :

```
context <id> :<classe>
inv <nom> : <expr-bool> -- premier invariant de la <classe>
…
inv <nom> : <expr-bool> -- nième invariant de la <classe>
```

ou encore

```
context <classe>
inv <nom> : <expr-bool> -- premier invariant de la <classe>
…
inv <nom> : <expr-bool> -- nième invariant de la <classe>
```

Exemple:

```
context c :Cours
  inv SigleDe6caracteres:
         c.sigle.size() = 6

context Cours
  inv SigleDe6caracteres:
         self.sigle.size() = 6
```

En USE : Les contraintes peuvent être incorporées à la définition des classes.

```
class Cours
attributes
  titre: String
  sigle: String
constraints
  inv SigleDe6caracteres:
         self.sigle.size() = 6
end
```

3.2 Quelques notions en OCL

Les types disponibles

- Les noms de classe peuvent être utilisés comme type.
- Les types de base : Integer, Real, Boolean, String.
- Les types Collection : Set, Bag, Sequence.
- Les types énumérés.
- Les types définis dans le modèle.
- Les types spéciaux : OclAny, OclType

Les valeurs

- Boolean : true, false
- Integer : 1, -5, 34, 26456
- Real : 1.5, 3.14
- String: 'ceci est un exemple'

Les operations de base

- Boolean: and, or, xor, not, implies, if-then-else
- Integer: +, -, *, /, abs
- Real, +, -, *, /, floor
- String: toUpper, concat, size, toLower

Les operations pré définies

- `o.oclIsKindOf(t:OclType):Boolean`
  retourne vrai si le type de « o » est « t » ou s'avère être un sous-type de « t ».
- `o.oclIsTypeOf(t :OclType) :Boolean`
  retourne vrai si le type de « o » est « t ».
- `o.oclAsType(t :OclType) : oclType`
  similaire à la conversion de type en Java
  « o » devient du type « t ».

```
context e : Etudiant
inv: e.IsKindOf(Personne) -- vrai
inv: e.IsTypeOf(Personne) -- faux
inv: e.IsKindOf(Cours)    -- faux
inv: e.IsKindOf(Et_grad)  -- faux
```

Les types Collection

- Collection : un type abstrait possédant les types concrets Set, Bag, Sequence et OrderedSet comme sous-types.
- Set : ensemble sans duplicata.
- Bag : ensemble admettant des duplicata.
- OrderedSet : Set ordonné.
- Sequence : Bag ordonné.

Navigation simple

`Objet.rôle`

- Multiplicité >= 1 : le résultat est un ensemble (Set) des objets référés par le rôle.
  ex. `instrSylvie.coursEnseignes` retourne un ensemble de Cours `{mgl606}`
- Multiplicité = 1 : le résultat peut être considéré comme étant un ensemble singleton ou comme étant du même type que l'objet référé par le rôle.
  ex. `mgl806.enseignant` retourne un objet de type Enseignant.
  ex. `mgl806.enseignant ->size()` retournerait la cardinalité 1 puisque l'ensemble résultant est un singleton.

ex. `mgl806.enseignant.enseignant.nom` retournerait le nom de la personne qui enseigne le cours.

Quelques opérateurs concernant les collections

- `Collection->notEmpty()`
- `Collection->isEmpty()`
- `Collection->size()`

Quelques opérateurs concernant les ensembles

- ```
  Set->union(s2 : Set(T)) : Set(T)
  ex. Set{1,2,4}->union(Set{3,4,5,6,2}) = {1,2,4,3,5,6}
  ex. instrSylvie.cours->union(instrMichel.cours) = {mgl806,log310}
  ```
- ```
  Set->intersection(s2: Set(T)) : Set(T)
  ex. Set{1,2,4}->union(Set{3,4,5,6,2}) = {4,2}
  ex. instrSylvie.cours-> intersection(instrMichel.cours)-
  >isEmpty() = true
  ```
- ```
  Set - (s2: Set(T) : Set(T)
  ex. Set{1,2,4} -  Set{3,4,5,6,2} = {1}
  ```

3.3 Quelques invariants

academique-4.use

Nous exprimerons ici trois invariants du système :

Un enseignant doit être un étudiant gradué ou un membre de la faculté

- Acteur principal : un Enseignant
- Objet nécessaire : la personne qui enseigne
- Navigation : l'attribut *personne* nous donne accès à la personne qui enseigne.
- Voir l'invariant à la page suivante.

Aucun étudiant gradué ne peut enseigner un cours dans lequel il est inscrit

- Acteur principal : un étudiant gradué considéré comme un enseignant.
- Ensembles nécessaires :
  - les cours enseignés -- seront accessibles à partir de la classe Enseignant
  - les cours inscrits sont accessibles par la classe Etudiant.
- Navigation :
  - En partant de la classe Enseignant on peut accéder aux étudiants gradués via l'attribut « enseignant ».
  - En partant de la classe Et_gradue, la navigation est plus difficile (il faudrait passer par plusieurs séquences de navigation).
  ```

EXERCICE :

Un étudiant ne peut être sur la liste d'attente d'un cours auquel personne n'est inscrit

-

Un enseignant doit être un étudiant gradué ou un membre de la faculté

```
context i:Enseignant

   inv EnseignantEstFaculteOuGradue:

         i.enseignant.oclIsKindOf(Faculte)
      or i.enseignant.oclIsKindOf(Et_gradue)
```

l'enseignant (Personne) est un membre de la faculté ou est un étudiant gradué.

Aucun étudiant gradué ne peut enseigner un cours dans lequel il est inscrit

```
context i:Enseignant
  inv PasDEnseignementPendantInscription:
  i.enseignant.oclIsKindOf(Et_gradue) implies
  i.cours -> intersection(i.enseignant.oclAsType(Et_gradue).cours) ->isEmpty
```

On considère uniquement les enseignants qui son des étudiants gradués.

Naviguer pour obtenir l'ensemble des cours enseignés.

À partir du type Personne (via attribut "enseignant"), convertir le résultat en Et_gradue.

Obtenir ensuite la liste des cours auxquels est inscrit cet étudiant gradué (qui est aussi un enseignant).

L'intersection entre les cours enseignés et les cours auxquels il est inscrit doit donc être vide.

Quelques exercices suggérés

Augmentez le modèle afin d'incorporer les notions pertinentes et insérez les contraintes suivantes :

- Chaque enseignant appartient à un seul département. Chaque département possède au moins un enseignant.
- Chaque département possède une banque de cours. Les cours appartiennent à un seul département.
- Chaque département possède un ensemble de cours obligatoires.
- Chaque étudiant est inscrit dans un seul département.
- Un étudiant gradué est encadré par un directeur qui est membre du corps professoral.
- Un membre du corps professoral peut être associé à au plus cinq jury (maîtrise ou doctorat) d'étudiants gradués.
- Un membre du corps professoral est toujours membre du jury des étudiants qu'il encadre.
- Seuls les membres du corps professoral peuvent donner les cours obligatoires.