École de technologie supérieure
Département de Génie logiciel et des TI

Document thématique

OCL en 7 exemples – Partie IV: les pré et post conditions

Auteur(s) : Sylvie Ratté
Date de création : 9 février, 2005
Réviseurs : Sylvie Ratté
Date de révision : 15 mars, 2005

Références

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. The examples presented 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

L'utilisation des pre et des posts conditions en OCL est présentée.

- Pour la classe « Cours », une opération « addPreRequis » permet d'ajouter un prérequis au cours visé.
- Pour la classe « Etudiant », deux opérations sont prévues. La première, « abandonCours » permet à l'étudiant d'abandonner un cours, la seconde, « nouveauCode », permet d'attribuer un nouveau code permanent à un étudiant.

2 Pre et post conditions en OCL

2.1 Syntaxe

```
context Typename::operationName(param1 : Type1, …): ReturnType
pre precondname1 : …param1… …self…
pre precondname2 : …param1… …self…
…
post postcondname1 : …result… …param1… …self…
post postcondname2 : …result… …param1… …self…
```

Chaque pré condition peut utiliser les paramètres et « self » et il en est de même pour les post conditions. Dans ce dernier cas, la variable « result », réservée par OCL, permet d'indiquer le résultat attendu s'il y a lieu.

2.2 Exemples

academique-7.use

```
class Cours
attributes
  titre: String
  sigle: String
operations
  addPreRequis(c : Cours)
end

class Etudiant < Personne
operations
   abandonCours(c : Cours)
   nouveauCode(n : String)
end
```

Opération « nouveauCode » de la classe Etudiant

```
context Etudiant::nouveauCode(n : String)
  pre  CodeLegal:    n.size() = 12
  post EstNouveau:   codePermanent.oclIsNew
  post EstNumero:    codePermanent.code = n
```

« oclIsNew » est vrai si « codePermanent n'existait pas dans l'état précédent.

Opération « abandonCours » de la classe Etudiant

Tentative No 1 :

```
context Etudiant::abandonCours(c: Cours)
pre Estinscrit: self.inscritsDans->includes(c)
post Plusinscrit: self.inscritsDans->excludes(c)
```

Examinons deux cas (avec c = C3):

```
Pre-état : self.inscritsDans = {C1,C2,C3}
Post-état : self.inscritsDans = {C1,C2}
```

Tout est OK

```
Pré-état : self.inscritsDans = {C1,C2,C3}
Post-état : self.inscritsDans = { }
```

Tout est OK! Mais ce n'est pas ce que l'on veut.

Tentative No 2

```
context Etudiant::abandonCours(c: Cours)
  pre  Estinscrit:   self.inscritsDans->includes(c)
  post Plusinscrit:  self.inscritsDans = inscritsDans@pre->excluding(c)
```

Examinons deux cas (avec c = C3):

```
Pre-état : self.inscritsDans = {C1,C2,C3}
Post-état : self.inscritsDans = {C1,C2}
```

Vrai

```
Pré-état : self.inscritsDans = {C1,C2,C3}
Post-état : self.inscritsDans = { }
```

Faux. Tout est donc parfait.

Mais, qu'arrive-t-il aux autres éléments de Etudiant? OCL adopte la philosophie « rien d'autres ne change que ce qui est spécifié ».

Cette particularité n'est pas implémenté en USE. En d'autres termes, USE ne vérifiera pas si les autres éléments sont effectivement non modifiés.

Post conditions : spécification du « quoi » et non du « comment »

```
context A::sort(s : Sequence(Integer)) : Sequence(Integer)
post MemeTaille:result->size = s->size
post MemesElements: result->forAll(i | result->count(i) = s->count(i))
post EstTriee: Sequence{1..( result->size-1)}->
                    forAll(i | result.at(i) <= result.at(i+1))
-- Notez l'utilisation d'une sequence ici pour imiter une boucle de 1 à
result->size-1
```

L'algorithme n'est pas précisé. Seule la condition qui exprime le fait qu'une séquence soit triée est spécifiée.

Exercice : ajouterCours

Etudiant::ajouterCours(c: Cours)

Objectif : ajouter c à l'ensemble des cours suivis par "self"

Préconditions :

- self n'est pas déjà inscrit à c
- self a déjà suivi les pré requis de c

Postconditions:

- C est ajouté à l'ensemble des cours suivis par self.

Ajouter un pré requis à un cours

Cours::addPreRequis(c: Cours)

Objectif : faire de c un nouveau pré requis de self

Préconditions :

- c n'est pas déjà un pré requis de self
- self n'est pas (transitivement) un pré requis de c (ce qui nous permettra de nous assurer qu'il n'y a pas de cycles)

Postconditions:

- c est ajouté au pré requis

Exercice : completerCours

Etudiant::completerCours(c: Cours, g: Note)

self complète le cours c et obtient la nite « g ».

c est enlevé des cours suivis par self

c est ajouté dans le relevé de notes de self avec la note g.

Préconditions :

…exercice…

Postconditions:

…exercice…

4 Vérification des pre et des post conditions

4.1 Structure d'un scénario USE

…

…commandes pour générer quelques objets

…

!openter <expr-source> <nom-operation> ([<liste-expr-arguments>])

…

…commandes USE qui réalisent les effets (les changements d'états) pour l'opération.

…

!opexit [<val-retour>]

…

…commandes USE illustrant l'effet/les valeurs produites par l'opération

…

Exemple avec "abandonCours":

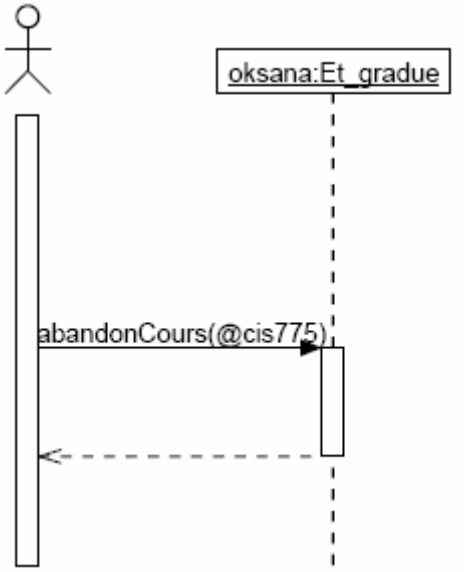
```
info vars
```

```
…
cis775 : Cours = @cis775
oksana: Et_gradue = @oksana
…
```

```
!openter oksana abandonCours(cis775)
```

```
precondition 'estInscrit' is true
```

```
info vars
      …
      c: Cours = @cis775
      self: Et_gradue = @oksana
-- imiter l'effet de l'abandon
!delete (oksana,cis775) from Inscrits
-- Lors de la sortie, les post conditions sont vérifiées.
!opexit
      postcondition 'PlusInscrit' is true
```



4.2 Détails des commandes « openter » et « opexit »

!openter oksana dropCourse(cis775)

1. **L'expression "source"** est évaluée pour obtenir l'objet "récepteur"
2. Les **arguments** sont évalués
3. self est lié à l'objet obtenu suite à l'évaluation de oksana et le paramètre c est lié à l'objet obtenu suite à l'évaluation de cis775.
4. Toutes les pré conditions de l'opération sont évaluées.
5. Si les pré conditions sont satisfaites, l'appel et ses liens sont empilés sur la pile. Et l'état pré est sauvegardée afin de pouvoir être utilisé par les post conditions.

info opstack

```
1. Etudiant::abandonCours(c : Cours) | oksana.abandonCours(@cis775)
```

!opexit

1. L'opération active est dépilée.
2. Si un résultat optionnel est fourni, ce dernier est lié à la variable spéciale OCL « result ».
3. Toutes les post conditions sont évaluées dans le nouveau contexte.
4. Tous les liens locaux sont éliminés.

4.3 Conseils sur l'utilisation de USE

Remarques

- Cet exemple permet de tester la spécification de l'opération « abandonCours » dans un contexte précis.
- L'utilisateur est responsable de créer les tests suffisants pour révéler les défauts de la spécification.
- Les actions que devraient réaliser l'opération sont simulées de manière abstraite en utilisant des commandes USE (par exemple ici l'effacement d'une paire dans une association).

Méthodologie

- Écrivez la spécification de votre opération.
- Déterminez quelles sont les commandes USE qui vous permettront de capturer le corps de l'opération (son effet).
- Créez un ensemble de contexte et d'appels à l'opération afin de tester ses fonctionnalités (son effet) contre ses pré/post conditions.
- Lorsque vous êtes satisfait de la spécification et de son implémentation abstraite, vous pouvez codez l'opération en fournissant une implémentation concrète de vos lignes de commandes USE.

À faire

Ouvrir la documentation USE concernant les pré/post conditions et tester le modèle « employee » venant avec la distribution. Vous y trouverez des exemples d'opérations retournant des résultats.

Examiner également l'exemple academique-8 et son script associé. J'y ai incorporé le test des cycles dans les pré requis de cours.
