École de technologie supérieure
Département de Génie logiciel et des TI

Document thématique
UML et OCL

Auteur(s) : Sylvie ratté
Date de création : 6 février, 2005
Réviseurs : Sylvie Ratté
Date de révision : 10 janvier, 2011

Références

La section 1 est une traduction adaptée au vécu des étudiants de l'ÉTS de la dernière version de la norme de l'OMG concernant OCL (UML 2.0 OCL Specification, 2003).

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 of section 2 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.

L'exemple de l'avion, section 3, est une adaptation de celui de Warmer et Kleppe (2003).

1 Introduction

Tout le monde s'entend pour dire qu'un diagramme UML, tel un diagramme de classes, n'est jamais assez raffiné pour décrire tous les aspects d'une spécification. Il ne faut pas être un devin pour réaliser cela. Le besoin de décrire des contraintes additionnelles au sujet des objets du modèle s'est fait sentir très tôt. L'objectif poursuivi est d'éviter de laisser dans le flou toute contrainte sur le diagramme. Dans la vie de tous les jours, de telles contraintes sont souvent rédigées en langage naturel (au moyen de notes ou de zones de texte). La pratique nous a enseignée que cette manière de procéder ne fait que propager les ambiguïtés jusqu'à l'étape ultime : la programmation proprement dite.

Mais la science de l'informatique a évoluée, parfois en symbiose, parfois orthogonalement aux besoins de l'industrie. C'est ainsi qu'on a vu émerger de nombreux langages formels permettant justement de décrire ces contraintes essentielles de manières non ambiguës. Ces langages nécessitent non pas une **formation** mathématique de haut niveau mais plutôt une **habitude** mathématique de base.

Le langage OCL (Object Contraint Language) a été développé en gardant cette réserve en tête. OCL est un langage formel facile à lire et avec lequel il est relativement facile d'exprimer

les contraintes désirées. Le langage, développé à l'origine comme un langage de modèle d'affaire par la division d'assurance de la compagnie IBM, trouve ses racines dans la méthode Syntropy.

Le langage OCL est une langue pure de spécifications. Conséquemment, une expression OCL ne peut provoquer d'effets de bord (i.e. modifier l'état du système). Lorsqu'une expression OCL est évaluée, une valeur est simplement retournée. Cette évaluation ne peut, en aucun cas, changer quoi que ce soit dans le modèle. Ce qui signifie que l'état du système ne changera jamais en raison de l'évaluation d'une expression OCL, quoiqu'une expression OCL puisse être employée pour décrire (entre « spécifier ») un changement d'état (par exemple, une post condition).

Le langage OCL n'est pas un langage de programmation. En conséquence, il n'est pas possible de décrire en OCL la logique d'un algorithme ou le contrôle du flot d'exécution d'un éventuel programme. Il n'est d'ailleurs pas possible d'utiliser OCL pour démarrer des processus ou activer des opérations modifiant l'état du système. Puisque OCL est, d'abord et avant tout, un langage de modélisation, ses expressions sont, par définition, non exécutable.

Le langage OCL est une langue typée; chaque expression OCL possède un type bien déterminé. Toute expression OCL bien formée répond aux règles de conformité des types du langage. Ainsi, il n'est pas possible de comparer des entiers (Integer) avec des chaînes de caractères (String). Chaque classifieur (Classifier) défini dans un modèle UML représente un type distinct en OCL. De plus, OCL comprend un ensemble de types prédéfinis (voir le chapitre 11 intitulé « The OCL Standard Library », de la norme OMG concernant OCL).

Puisque OCL est un langage de spécification, toutes les questions d'implémentation sont évacuées de sa portée et ne peuvent être exprimées en OCL.

Finalement, l'évaluation des expressions OCL est instantanée. Ce qui veut dire que l'état des objets appartenant au modèle ne peut être modifié durant cette évaluation.

2 UML et OCL

2.1 UML, OCL et les outils

L'avantage évident de UML: UML est visuel mais…

- ☑ « Une image vaut mille mots ».
- ☒ Mais une image peut porter à confusion.
- ☑ La plupart des personnes peuvent examiner une description UML et se faire « une assez bonne idée » du système décrit.
- ☒ Est-ce que « assez bonne idée » s'avère suffisant?

Les défauts de UML

- ☒ Il y a plusieurs types de diagrammes et parfois, ce qui est couvert par un diagramme n'est pas très différent de ce qui est couvert par un autre.
- ☑ Grady Booch mentionne à ce propos que l'on peut modéliser 80% des problèmes avec 20% de UML.

☒ Plusieurs aspects de UML sont imprécis ce qui rend difficile le développement d'outils de vérification et de raisonnement puisque de tels outils doivent s'appuyer sur une sémantique solide.

☑ Des efforts sont réalisés par plusieurs groupes travaillant au sein de l'OMG et ailleurs (www.cs.york.ac.uk/puml).

UML/OCL et ses outils

- Vous trouverez sur OCL Portal de Dresden la liste des outils de modélisations incorporant OCL de même qu'un lien vers une liste très utile pour comparer les différents outils de modélisation.

- USE (Uml-based Specification Environment) supporte la vérification des contraintes OCL à travers la création d'instantanés du système construits grâce à des diagrammes de collaboration. Les diagrammes de séquence permettent de rendre compte de l'aspect dynamique et conséquemment, du respect des pré et des post conditions.
- Mathieu Martin, maintenant diplômé, a réalisé un outil permettant de traduire des modèles provenant de divers outils vers le format USE. Vous trouverez cet outil sur le site Web du cours.

2.2 OCL et USE

Le langage OCL permet, entre autre, la spécification des invariants (dans le contexte d'une classe et dans le contexte plus général du système), celle des pré/post conditions des méthodes, et celle des gardes dans les diagrammes d'états/transitions. Les contraintes OCL peuvent apparaître dans tous les diagrammes. Pour le moment, nous allons nous concentrer sur l'annotation des diagrammes de classes.

Tous les diagrammes présentées en figure ont d'abord été créés avec ArgoUML (afin d'obtenir des fichiers contenant les images). Ils ont ensuite été défini dans l'outil USE avec les commandes appropriées.

`[GUI]` précède une commande disponible dans l'environnement graphique.

`[TTY]` précède une commande disponible dans l'environnement terminal.

A Un modèle objet dans USE

- un ensemble de noms de classe;
- un ensemble d'attributs typés associés à chaque classe;
- un ensemble d'opérations – méthodes d'une classe qui n'ont pas d'effets de bord;
- un ensemble d'associations dont le nom et les rôles sont identifiés;
- ces éléments peuvent être utilisés dans les expressions OCL.

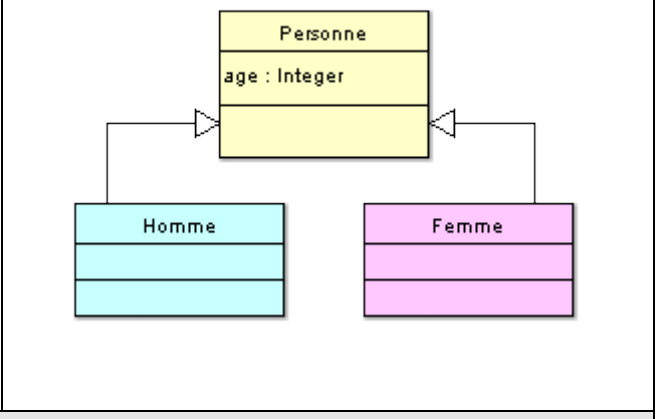
Créons d'abord un modèle simple grâce à un éditeur de texte standard. Nommez ce fichier Famille-1.use. Le contenu du fichier est présenté dans le tableau de la page suivante.

Famille-1.use

```
[GUI]  File|Open Famille-1.use
[TTY]  open Famille-1.use
```

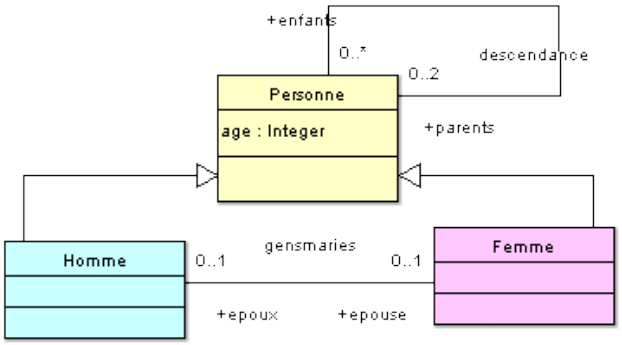
La figure 1 présente l'interface graphique USE résultant de l'exécution de la commande.

```
model Famille

-- classes

class Personne
attributes
  age : Integer
end

class Homme < Personne
end

class Femme < Personne
end
```



Famille-2.use

```
association gensmaries between
  Homme[0..1] role epoux;
  Femme[0..1] role epouse;
end

association descendance between
  Personne[0..2] role parents;
  Personne[*] role enfants;
end
```



Famille-3.use

```
constraints

context Personne
 inv PlusJeuneQueSesParents:
    parents->forAll(p | p.age > self.age)
```

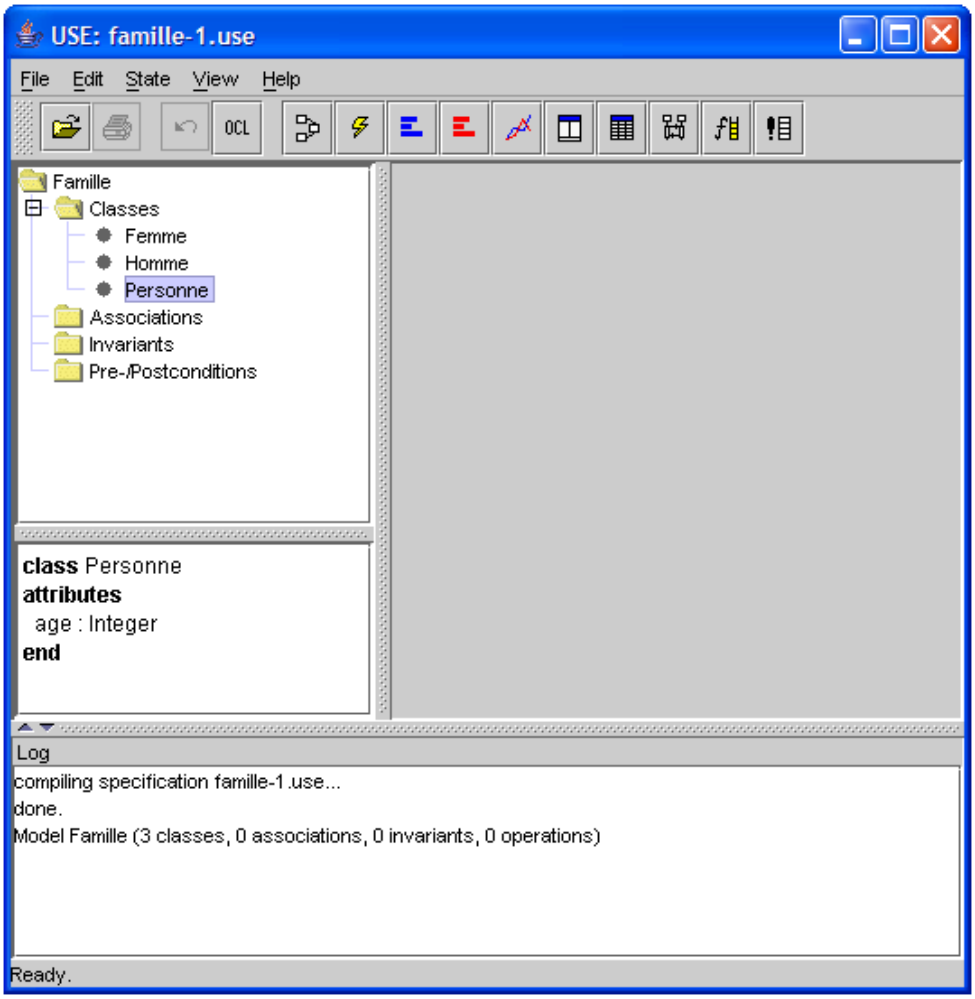


Figure 1. Ouverture d'une spécification en USE

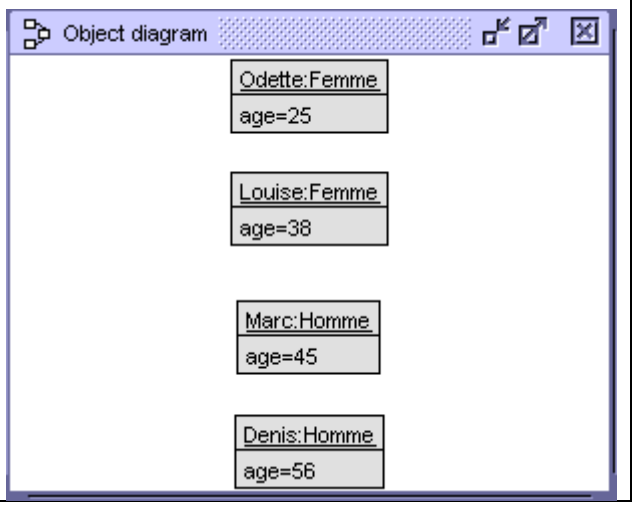
B Création et vérification d'un instantané

USE permet de créer des instantanés du système qui servent à vérifier les contraintes OCL. Chaque instantané correspond à une réalisation du système en illustrant les valeurs possibles des objets et leurs relations. Il est important de comprendre que ce procédé ne permet pas de capturer automatiquement l'ensemble de toutes les réalisations possibles. Cette couverture est de votre responsabilité.

Création d'un objet	
[GUI]	Dans l'environnement GUI, créer un nouveau diagramme de collaboration (bouton ) et glisser les classes dans le diagramme. Cette méthode n'est pas très indiquée à moins d'être pressés car elle ne permet pas de modifier beaucoup d'information.
[GUI]	State|Create object : permet au moins de donner un nom significatif à l'instance.
[TTY]	`!create nomDelInstance : Classe`
Pour modifier ou associer une valeur à un attribut :	
[TTY]	`!set nomInstance := valeur`

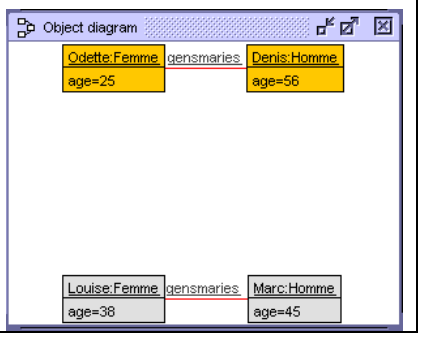
Ces commandes peuvent être enregistrées dans un fichier pour ensuite être rechargées rapidement dans l'environnement. Pour voir les commandes effectuées jusqu'à maintenant :	
[GUI]	View|Command list
Pour enregistrer ces commandes :	
[GUI]	File|Save script
[TTY]	write fichier

Exemple

Commandes	Résultat
!create Denis:Homme !create Marc:Homme !create Louise:Femme !create Odette:Femme !set Denis.age := 56 !set Marc.age := 45 !set Louise.age := 38 !set Odette.age := 25	

Créer une association entre deux instances	
[GUI]	Sélectionnez par click/shift click les deux objets. Avec le bouton droit faites apparaître le menu contextuel et choisissez « insert nomDeLaRelation link ».
[TTY]	!insert(objet1, objet2) into nomDeLaRelation

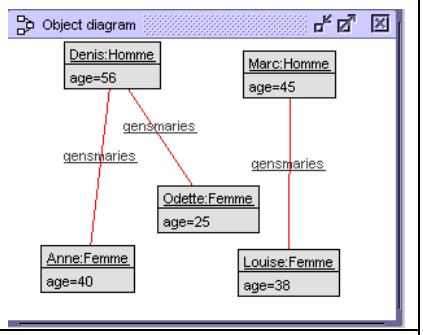
Exemple

Commandes	Résultat
!insert (Marc, Louise) into gensmaries !insert (Denis,Odette) into gensmaries	

Si j'ajoute:

```
!create Anne:Femme
!set Anne.age := 40
!insert (Denis, Anne) into gensmaries
```

On obtient évidemment une violation d'une contrainte de multiplicité. La violation de la contrainte est indiquée dans la fenêtre « log ».

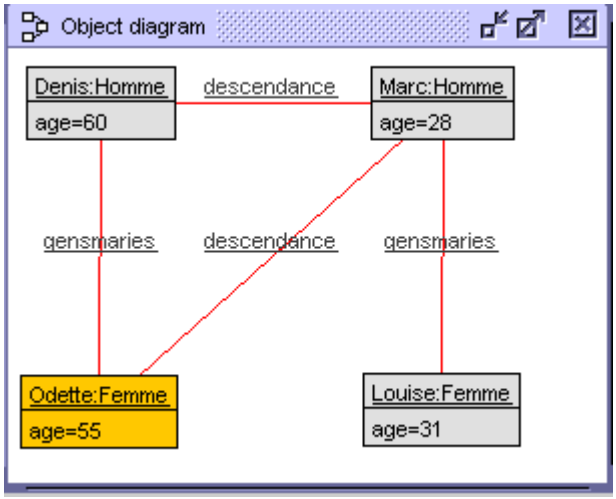


Log

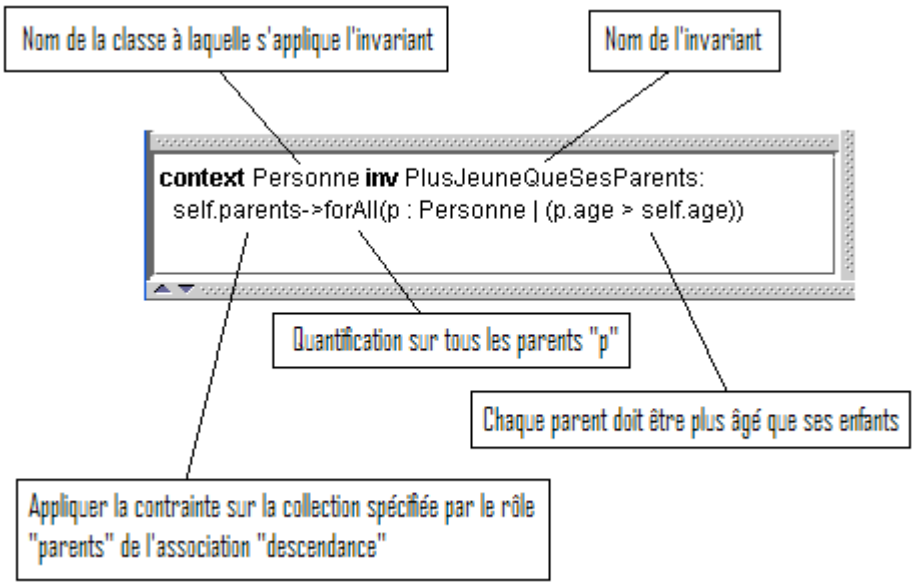
```
checking structure...
checking structure, ok.
checking structure...
checking structure, ok.
checking structure...
Multiplicity constraint violation in association `gensmaries':
  Object `Denis' of class `Homme' is connected to 2 objects of class `Femme'
  but the multiplicity is specified as `0..1'.
checking structure, found errors.
```

Exercice

Créez l'instantané suivant (l'association "descendance" n'est pas accessible dans l'interface graphique):

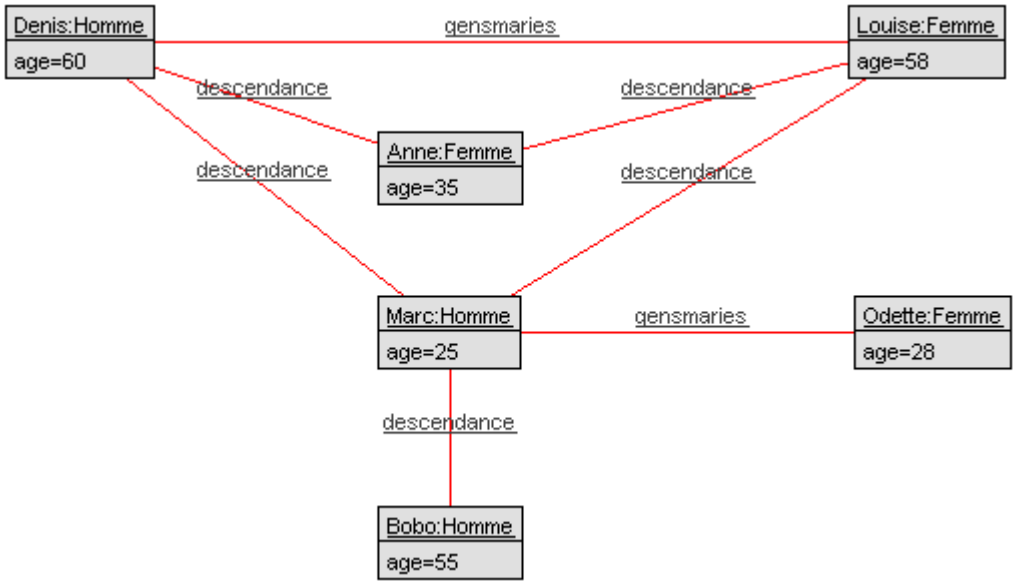


C Invariants OCL



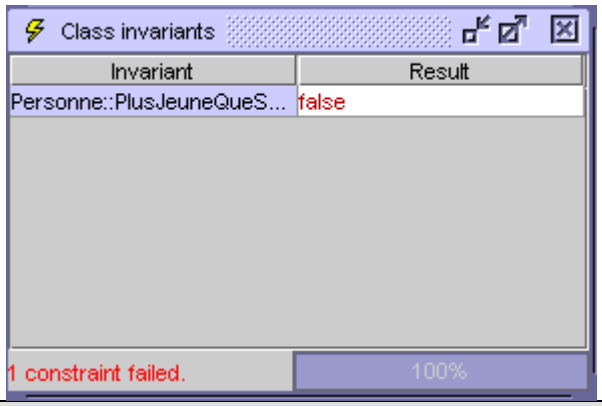
Exercice

Construisez le modèle nécessaire et l'instantané suivant afin de vérifier la contrainte "PlusJeuneQueSesParents"

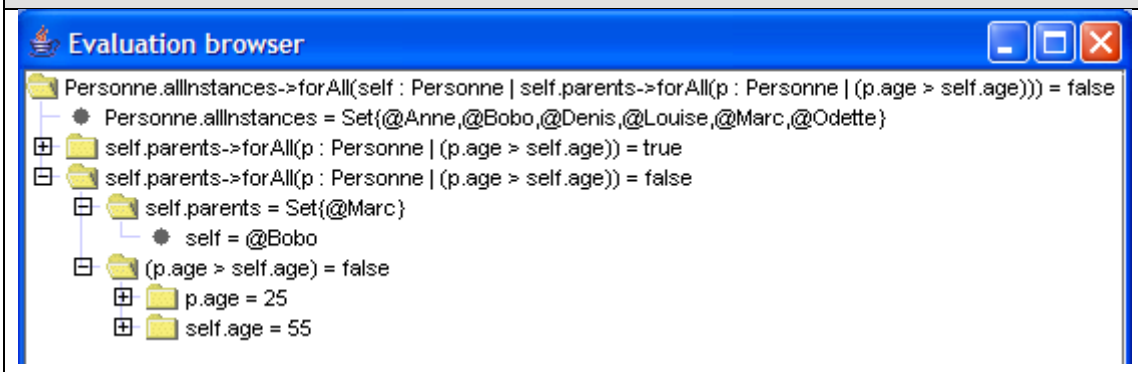


Vérifiez ensuite l'invariant :

[GUI] Sélectionnez le bouton  ou via le menu View | Create | Invariant pour faire afficher la fenêtre suivante :

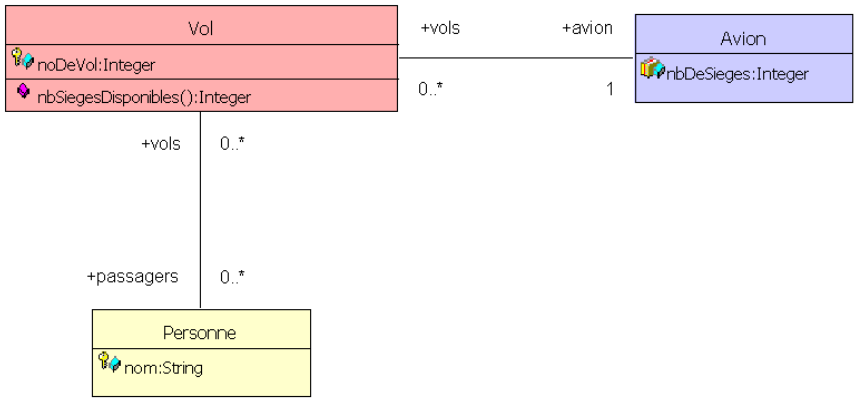


Ouvrez la fenêtre d'évaluation en double-cliquant sur l'invariant violé.



3 Exemple supplémentaire

L'association entre « Vol » et « Personne » tend à suggérer que le nombre de passagers est infini. Nous savons tous que ce nombre est directement lié aux nombres de sièges dans l'avion! L'idée consiste donc à offrir une manière d'exprimer cette contrainte.



Il va sans dire que l'expression de celle-ci directement sur le diagramme rendrait rapidement ce dernier illisible c'est pourquoi, les contraintes OCL doivent apparaître séparément. En OCL, la contrainte s'exprime comme suit :

```
context Vol
   inv : passagers->size() <= avion.nbDeSieges
```

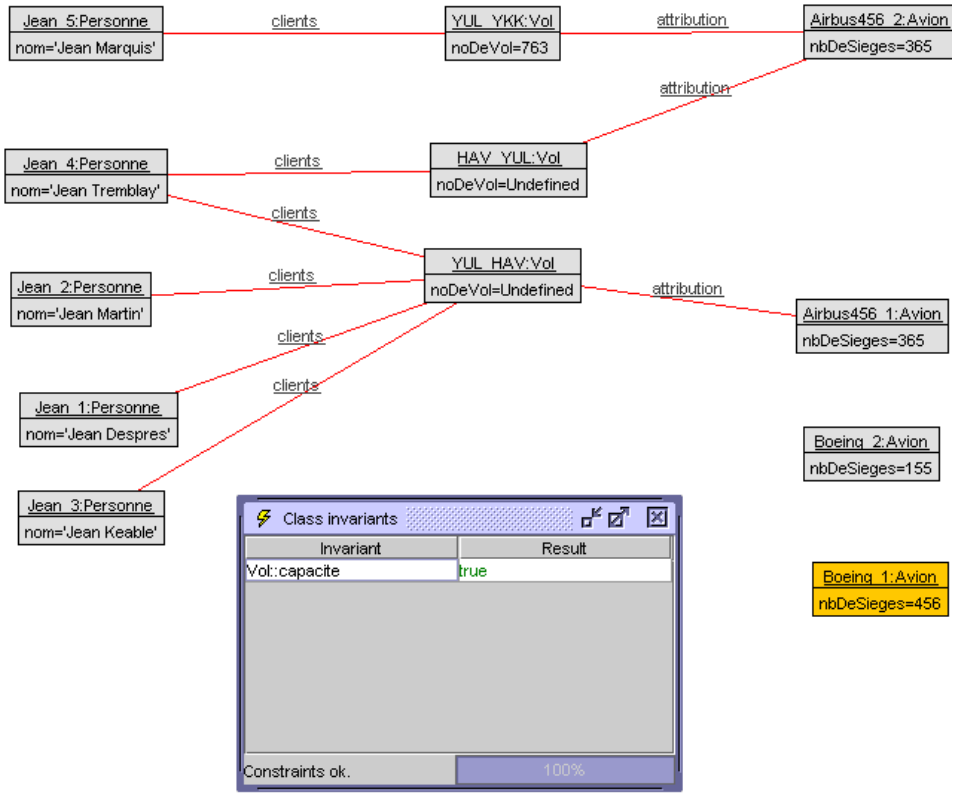
Le modèle USE est décrit dans le fichier : avion-1.use.

Et voici les commandes permettant de créer un instantané parmi bien d'autres avion1.cmd.

```
!create YUL_YKK:Vol
!create YUL_HAV:Vol
!create HAV_YUL:Vol
!create Airbus456_1:Avion
!create Airbus456_2:Avion
!create Boeing_1:Avion
!create Boeing_2:Avion
!create Jean_1:Personne
!create Jean_2:Personne
!create Jean_3:Personne
!create Jean_4:Personne
!create Jean_5:Personne
!create Jean-1:Personne
```

```
!set YUL_YKK.noDeVol := 444
!set YUL_YKK.noDeVol := 767
!set YUL_YKK.noDeVol := 763
!set Airbus456_1.nbDeSieges := 365
!set Airbus456_2.nbDeSieges := 365
!set Boeing_1.nbDeSieges := 456
!set Boeing_2.nbDeSieges := 155
!set Jean_1.nom := 'Jean Despres'
!set Jean_2.nom := 'Jean Martin'
!set Jean_3.nom := 'Jean Keable'
!set Jean_4.nom := 'Jean Tremblay'
!set Jean_5.nom := 'Jean Marquis'
```

```
!insert (HAV_YUL,Jean_4) into clients
!insert (YUL_HAV,Jean_4) into clients
!insert (YUL_HAV,Jean_2) into clients
!insert (YUL_HAV,Jean_1) into clients
!insert (Airbus456_2,HAV_YUL) into attribution
!insert (Airbus456_2,YUL_YKK) into attribution
!insert (YUL_YKK,Jean_5) into clients
!insert (Airbus456_1,YUL_HAV) into attribution
!insert (YUL_HAV,Jean_3) into clients
```



Caractéristiques de OCL

On se doit d'énoncer certains constats :

- les diagrammes UML sans OCL sont insuffisants;
- OCL seul devient trop cryptique;
- la solution se situe dans la complémentarité des spécifications.

OCL est un langage déclaratif fortement typé de la même puissance expressive que plusieurs langages de spécification dits « formels », mais sans la notation mathématique.

Qu'est-ce qu'une contrainte? C'est une restriction qui s'applique sur une ou plusieurs valeurs d'un modèle OO. Mais OCL est également utile pour exprimer plus que des contraintes. Par exemple, l'expression suivante définie le corps de l'opération « nbSiegesDisponibles » de la classe « Vol » (nous y reviendrons).

```
context Vol::nbSiegesDisponibles() : Integer
      body : avion.nbDeSieges – passagers->size()
```

Les expressions OCL peuvent être utilisées pour définir des attributs, des associations et des classes dérivés, des valeurs initiales, des pre et post conditions.