MDE
Exercices

Nous allons voir comment combiner DresdenOCL et AspectJ pour générer des tests équivalents aux contraintes OCL. Mais tout d'abord, nous allons voir comment utiliser l'outil Acceleo pour générer un squelette de classes Java, ce qui nous permettra d'éviter d'écrire ce code à la main !

Au delà des étapes techniques, il est important d'observer le fonctionnement des outils et de comprendre comment se réalisent les transformations.

1. Il faut d'abord s'assurer que les outils sont installés.
   (a) Pour AspectJ faire `Help -> Install new software` et puis choisir *AspectJ Development Tools*.
   (b) Pour Acceleo faire `Help -> Marketplace`, chercher `Acceleo` et choisir *UML to Java generator*.
2. Comme Acceleo présuppose que le modèle UML est de structure Model/Package, nous allons partir d'un modèle de la bonne forme !
   (a) Sauvegardez, si nécessaire, vos projets actuels dans un archive zip,
   (b) vider le Workspace
   (c) et finalement chargez le ficher `mde.zip`.
3. L'outil AspectJ permet d'instrumenter du code Java pour, par exemple, faire exécuter des tests à des moments bien précis, comme après l'appel à un constructeur ou encore avant/après l'appel à une méthode.
   (a) Vous devez donc créer un nouveau projet AspectJ qui recevra le code produit par DresdenOCL.
4. Nous sommes maintenant prêt à générer un squelette de classes Java à l'aide de l'outil Acceleo. Il s'agit donc de produire, à partir du modèle UML, les classes Java correspondantes.
   (a) Sélectionnez le modèle UML et faites `run as -> configuration`.
   (b) Dupliquez la configuration (si vous voulez conserver l'originale). Assurez vous maintenant que la nouvelle configuration indique les choses suivantes :
      i. le projet AspectJ est le projet destination,
      ii. le répertoire pour le code source est bien le répertoire src du projet AspectJ,
      iii. le fichier de sortie est le répertoire classes.
   (c) Faites finalement `Apply` et `Run`.
   (d) Le code source AspectJ/Java généré devrait apparaître dans le répertoire src. Faites `refresh`, pour faire relire le disque, si nécessaire.

5. L'étape précédent produira du code Java correspondant aux classes et méthodes du modèle UML. Observez bien les classes et méthodes Java produites pour vous en convaincre.

   La génération de code source est toujours une approximation, qu'il faut corriger. Dans notre cas, vous pouvez vérifier qu'il manque des références à des classes comme HashSet. Ceci peut être corrigé en ajoutant `import java.util.*;` au début des fichiers.

   Un autre problème est que le code généré utilise une méthode `set` pour faire l'affectation. On peut la remplacer par l'affectation `=`, tout en corrigeant le code pour que la bonne valeur soit affectée.

6. Pour faire exécuter les tests, il nous faudra un programme Java minimal. Nous allons donc :
   (a) ajouter une classe Java `Main` au projet AspectJ au même endroit que les autres classes (utilisez le bouton de droite et faites ajouter une fonction `main`).
   (b) Dans la fonction `main` ajoutez `Employe E = new Employe();` Pour créer un Employé de nom vide.

7. Pour faire générer des tests correspondant aux contraintes OCL, on utilisera la démarche suivante.
   (a) Pour bien comprendre le processus, nous allons nous concentrer sur la toute première contrainte du fichier `.ocl`, celle qui vérifie que le nom d'un Employé est non-vide.
   (b) Il est possible, mais non obligatoire, de donner un nom à une contrainte en le plaçant entre le `inv` et le `:`. Ceci peut néanmoins permettre de la repérer plus facilement.
   (c) Comme d'habitude, avant de pouvoir utiliser DresdenOCL, il faut charger le modèle UML et ouvrir le fichier .ocl.
   (d) Pour faire générer le code, il suffit de choisir dans le menu du haut `DresdenOCL -> Generate AspectJ Contraint Code` et ne sélectionner que la toute première contrainte. Il faut s'assurer que le répertoire source est bien le src du projet AspectJ pour que les contraintes y soient ajoutées.
   (e) Assurez-vous que le code AspectJ pour les contraintes est bien présent dans le répertoire `src/paq/paq.constraints`. Il peut être nécessaire de faire `refresh`.

8. Nous avons maintenant un programme (`main`) qui crée un Employé au nom vide, ce qui viole donc la contrainte OCL pour laquelle nous avons généré un test AspectJ/Java.
   (a) Pour s'en assurer faites `run (as AspectJ/Java application)`. Un message d'erreur devrait s'afficher.
   (b) Si vous modifiez le code, soit en enlevant la création de cette instance de la classe Employé, soit en ajoutant un nouveau constructeur qui affectera un nom non-vide (et en enlevant l'affectation par défaut du nom à une chaîne vide), vous pourrez vérifier que la propriété n'est plus violée.
   (c) D'une façon similaire on peut vérifier d'autres invariants ainsi que les pré et post-conditions des opérations.