Sémantique statique d'un DSML avec OCL

La version d'Eclipse à utiliser est la suivante :

```
/mnt/n7fs/ens/tp_babin/gls/eclipse-gls/bin/eclipse-gls
```

1 Compléter un méta-modèle par des contraintes statiques

Exercice 1 : Vérification statique avec OCL
Nous avons utilisé Ecore (ou EMOF) pour définir un méta-modèle pour les processus. Il est rappelé à la figure 1. Certaines contraintes sur les modèles de processus ont pu être exprimées. C'est par exemple le cas de celles qui concernent les multiplicités. La propriété `opposite` permet aussi d'exprimer une contrainte entre deux références pour retrouver une notion proche de la relation d'association bidirectionnelle d'UML.

Cependant, le langage de méta-modélisation (Ecore ou EMOF) ne permet pas d'exprimer toutes les contraintes que doivent respecter les modèles de processus. Aussi, on complète la description structurelle du méta-modèle réalisée en Ecore (ou EMOF) par des contraintes exprimées en OCL.

Le méta-modèle Ecore et les contraintes OCL définissent la syntaxe abstraite du langage de modélisation considéré.

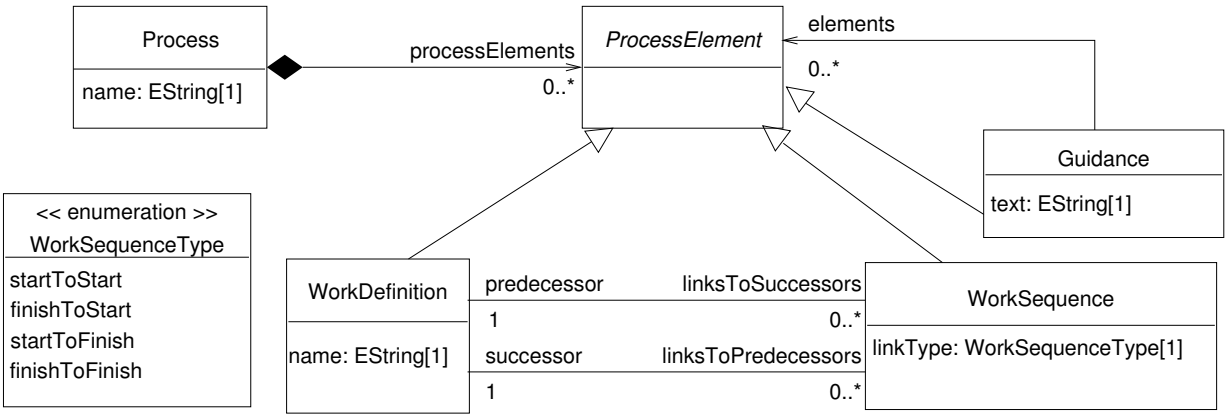

FIGURE 1 – Nouveau méta-modèle de SimplePDL

1.1 Expliquer les éléments apparaissant sur le fichier OCLinEcore du listing 1 correspondant au métamodèle de SimplePDL.

Listing 1 – Le métamodèle SimplePDL en OCLinEcore avec des éléments OCL

```
package simplepdl : simplepdl = 'http://simplepdl'
{
  enum WorkSequenceType { serializable }
  {
    literal startToStart;
    literal finishToStart = 1;
    literal startToFinish = 2;
    literal finishToFinish = 3;
  }
  class Process
  {
    attribute name : String;
    property processElements : ProcessElement[*] { ordered composes };
  }
  abstract class ProcessElement
  {
    property process : Process { derived readonly transient volatile !resolve }
    {
      derivation: Process.allInstances()
          ->select(p | p.processElements->includes(self))
          ->asSequence()->first();
    }
  }
  class WorkDefinition extends ProcessElement
  {
    property linksToPredecessors#successor : WorkSequence[*] { ordered };
    property linksToSuccessors#predecessor : WorkSequence[*] { ordered };
    attribute name : String;
  }
  class WorkSequence extends ProcessElement
  {
    attribute linkType : WorkSequenceType;
    property predecessor#linksToSuccessors : WorkDefinition;
    property successor#linksToPredecessors : WorkDefinition;
    invariant previousWDinSameProcess: self.process = self.predecessor.process;
    invariant nextWDinSameProcess: self.process = self.successor.process;
  }
  class Guidance extends ProcessElement
  {
    property element : ProcessElement[*] { ordered };
    attribute text : String;
  }
}
```

1.2 *Utiliser l'éditeur OCLinEcore pour saisir les contraintes OCL.* Voyons comment il est possible d'attacher à des éléments d'un métamodèle des contraintes OCL.

1.2.1 *Enregistrement des métamodèles.* Il est nécessaire dans une première étape d'enregistrer les métamodèles en cliquant sur les fichiers `.ecore` avec le bouton droit et en sélectionnant *Register Epackages*. On peut vérifier la bonne inscription des métamodèles dans la vue « EPackage Registry » (menu *Window* > *Show view* > *Other*, puis *EPackages registration / EMF registered packages*).

1.2.2 *Ouverture du fichier.* Cliquer avec le bouton droit sur `SimplePDL.ecore` puis faire *Open With* > *OCLinEcore (Ecore) Editor*. Lire le contenu de ce fichier.

1.2.3 *Ajout de nouvelles règles OCL.* L'éditeur OCLinEcore est un éditeur textuel permettant d'éditer un métamodèle Ecore et de définir des contraintes OCL sur ce modèle. Dans chaque élément class de notre métamodèle nous pouvons définir des *invariants* exprimés en OCL. On pourrait par exemple définir un invariant specifiant que l'attribut *name* d'un élément *Process* ne doit pas être *"Process"* (cf listing 2). Le contexte de la contrainte est la classe la contenant (ici *Process*), il n'est donc pas nécessaire de le spécifier comme précedemment.
Remarque : La syntaxe OCL utilisée dans l'éditeur OCLinEcore n'est pas exactement la même que celle proposée par l'OMG. Par exemple, on écrira *invariant* et non *inv*.

```
class Process
{
      invariant nameForbidden : name <> 'Process';
      attribute name : String[1] { ordered };
      property processElements : ProcessElement[*] { ordered composes };
}
```

Listing 2 – Une contrainte OCL interdisant un certain nom pour un élément *Process*

Une fois le fichier Ecore modifié et sauvegardé, on peut valider une instance (le fichier *.xmi*). Cliquer droit dans l'éditeur reflexif sur la racine du modèle et faire *Validate*. Si des contraintes ne sont pas vérifiées un message d'erreur apparaitra et une erreur est affichée dans la vue *Error Log*. Cliquer sur l'erreur pour obtenir plus de détails.

1.3 *Compléter les contraintes de SimplePDL.* Exprimer les contraintes suivantes sur SimplePDL et les évaluer sur des exemples de modèles de processus :

1. deux activités différentes d'un même processus ne peuvent pas avoir le même nom.
2. une dépendance ne peut pas être réflexive.
3. le nom d'une activité doit être composé d'au moins un caractère.

2 Application aux réseaux de Petri

Exercice 2 : Sémantique statique des réseaux de Petri
Définir les contraintes OCL définissant la sémantique statique du métamodèle des réseaux de Petri. Les valider sur différents modèles de réseaux de Petri.