Métamodélisation et sémantique statique

Exercice 1 : Comprendre SimplePDL

EMOF (OMG) ou Ecore (Eclipse) sont des méta-métamodèles. Un extrait d'Ecore est donné à la figure 1. Leur objectif est de permettre la définition de métamodèles. La figure 2 donne le métamodèle du langage SimplePDL, un langage très simplifié de description des procédés de développement. Ce métamodèle est conforme à EMOF/Ecore. Il a été dessiné en utilisant les conventions traditionnellement utilisées qui sont empruntées au diagramme de classe UML.

1.1 *Concepts Ecore.* Le métamodèle de SimplePDL est conforme à Ecore. Indiquer, pour chaque élément du métamodèle de SimplePDL, l'élément d'Ecore auquel il « correspond ».

1.2 *Signification de SimplePDL.* Expliquer ce que décrit le métamodèle SimplePDL.

1.3 *Description d'un procédé particulier.* On s'intéresse à un procédé simple composé de quatre activités : concevoir, programmer, tester et documenter. Programmer ne peut commencer que quand la conception est terminée. Le test peut démarrer dès que la conception est commencée. Documenter ne peut commencer que quand la programmation est commencée et ne peut peut s'achever que si la programmation est terminée. Le test ne peut être terminé que si la conception et la programmation sont terminée.

1.3.1 Dessiner le modèle de ce procédé. On utilisera une boîte pour représenter une activité et une flèche pour les relations de précédence.

1.3.2 Montrer que le modèle de procédé ainsi construit est bien conforme à SimplePDL.

1.4 Expliquer les contraintes OCL portant sur SimplePDL données ci-dessous.

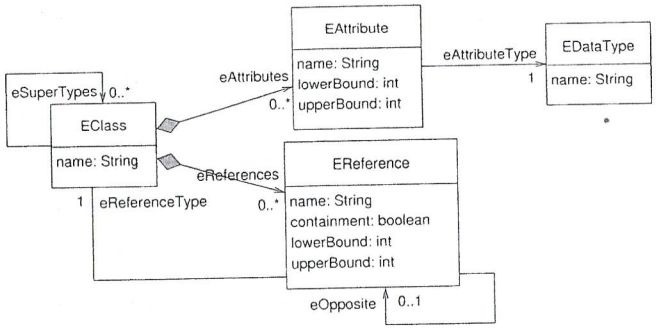
```
context ProcessElement
def: process: Process =
      Process.allInstances()->select(p | p.processElements->includes(self))
            ->asSequence()->first()

context WorkSequence
inv previousWDinSameProcess: self.predecessor.process = self.process
inv nextWDinSameProcess: self.successor.process = self.process
```

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

1. une dépendance ne peut pas être réflexive.
2. deux sous-activités différentes d'un même processus ne peuvent pas avoir le même nom.
3. le nom d'une activité doit être composé d'au moins un caractère.
4. les dépendances du modèle de processus ne provoquent pas de blocage.

FIGURE 1 – Version très simplifiée du méta-métamodèle Ecore



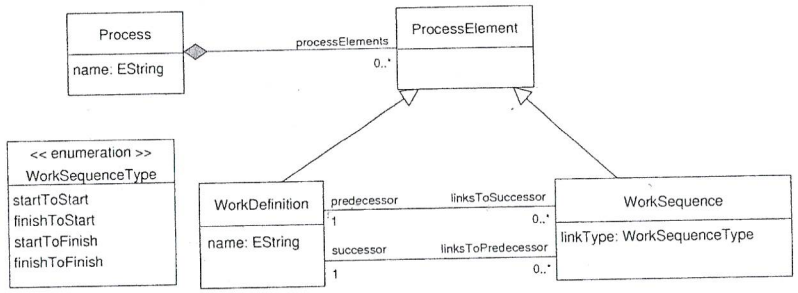


FIGURE 2 – Méta-modèle de SimplePDL conforme à EMOF/Ecore

Exercice 2 : Mise en œuvre avec OCLinEcore

OCLinEcore propose un éditeur qui offre une syntaxe concrète textuelle pour un métamodèle Ecore. Il permet d'ajouter des éléments OCL directement sur ce métamodèle. Ils sont ensuite sauvegardés dans des éléments `EAnnotation` dans le `.ecore`. Le listing 1 présente un exemple avec SimplePDL.

2.1 Expliquer les différents éléments présents sur le listing 1.

2.2 Comparer les approches OCL et OCLinEcore.

Exercice 3 : Méta-modèle des réseaux de Petri

L'objectif de cet exercice est de construire un métamodèle des réseaux de Petri.

3.1 Proposer un métamodèle des réseaux de Petri. On utilisera Ecore.

3.2 Dessiner quelques modèles de réseau de Petri qui sont conformes au métamodèle défini mais non valides.

3.3 Définir des contraintes OCL pour exprimer les propriétés qui n'ont pas été capturées par le métamodèle ECore.

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;
  }
}
```

TD4
GLS

Métamodélisation et sémantique statique.

Exercice 1 : Comprendre SimplePDL.

1.1

PDL
Process element
Process
Work Definition
Work Sequence.

EClass

ID	name	eReferences	eAttributes	eSuperTypes
C1	Process	processElements R1	name A1	
C2	ProcessElement			
C3	WorkDefinition	LinksToSuccessor R2 LinksToPredecessor R3	name A2	ProcessElement
C4	WorkSequence	predecessor R4 successor R5	LinkType : WorkSeqType A3.	ProcessElement

EAttribute

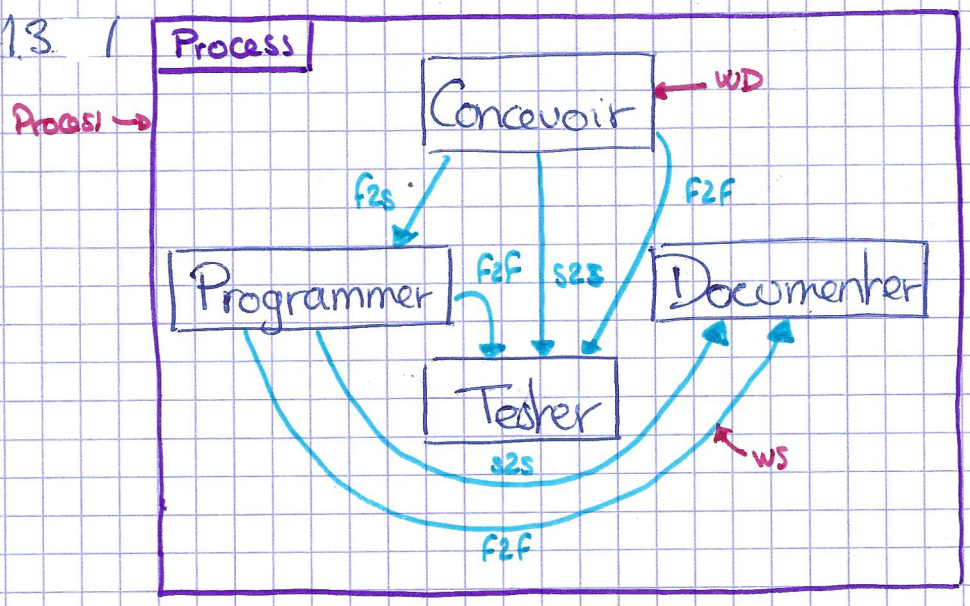
ID	name	lowerBound	upperBound	eAttributeType
A1	name	1	1	String
A2	name	1	1	String
A3	LinkType	1	1	WorkSeqType

EReference

ID	name	containment	lowerBound	upperBound	eOpposite	eRefType
R1	ProcessElt	T	0	*		Process
R2	LinksToSuc	F	0	*	Pred	WkDef
R3	LinksToPred	F	0	*	Succ	WkDef
R4	Pred	F	1	1	LinkSucc	WkSeq
R5	Successor	F	1	1	LinkPred	WkSeq

1.2 / Modèle de processus

1.3 /



F2S : Finish to start
S2S : Start to start
F2F : Finish to finish

1.4/

1.5/ ① context Work Sequence

inv notReflexive : self.predecessor <> self.successor

② context Process

$\forall w_1, w_2$ Process.processElements , w_1, w_2 : WkDefinition
$w_1.name = w_2.name \Rightarrow w_1 = w_2$

inv uniqueNames
self.processElements
→ select (p : ProcessElt | p.oclIsKindOf (WD))
→ collect (p : ProcessElt | p.oclAsType (WD))
→ forAll (w_1, w_2 : WorkDef | $w_1.name = w_2.name$ implies $w_1 = w_2$)

③ context WorkDefinition

inv validName: not self.name.oclIsUndefined()
and self.name <> ''.