Définition de syntaxes concrètes textuelles

La version d'Eclipse à utiliser est la suivante :

```
/mnt/n7fs/ens/tp_dieumegard/eclipse_modeling_indigo/eclipse
```

Nous avons vu comment définir la syntaxe abstraite d'un DSML. Le principe est d'utiliser Ecore (ou un autre langage de métamodélisation) pour décrire le métamodèle et OCL (ou un autre langage de contrainte) pour exprimer les propriétés liées à la sémantique statique qui n'ont pas pu être capturées par le métamodèle.

La syntaxe abstraite ne peut pas être manipulée directement. Le projet EMF d'Eclipse permet d'engendrer un éditeur arborescent et les classes Java qui permettront de manipuler un modèle conforme à un métamodèle. Cependant, ce ne sont pas des outils très pratiques lorsque l'on veut saisir un modèle. Aussi, il est souhaitable d'associer à une syntaxe abstraite une ou plusieurs syntaxes concrètes pour faciliter la construction et la modification des modèles. Ces syntaxes concrètes peuvent être textuelles (l'objet de cette séance) ou graphiques (séance suivante).

Pour la définition des syntaxes concrètes textuelles, nous allons utiliser l'outil *xText* [1] de openArchitectureWare [2] (oAW). *xText* fait partie du projet TMF (Textual Modeling Framework). Il permet non seulement de définir une syntaxe textuelle pour un DSL mais aussi de disposer, au travers d'Eclipse, d'un environnement de développement pour ce langage, avec en particulier un éditeur syntaxique (coloration, complétion, outline, détection et visualisation des erreurs, etc).

xText s'appuie sur un générateur d'analyseurs descendants récursifs (LL(k)). L'idée est de décrire à la fois la syntaxe concrète du langage et comment le modèle EMF est construit en mémoire au fur et à mesure de l'analyse syntaxique.

1 Définir des syntaxes concrètes textuelles pour SimplePDL

Xtext consiste à définir une syntaxe concrète textuelle pour un DSML au travers d'une grammaire qui doit pouvoir être analysée en LL(k). En particulier, elle ne doit pas être récursive à gauche [3] ! Généralement, *Xtext* est utilisé pour engendrer le métamodèle correspondant. C'est ce que nous allons voir dans ce sujet. *Xtext* peut aussi travailler à partir d'un métamodèle existant. Cependant, il ne faut pas que la distance soit trop grande entre le métamodèle existant et la syntaxe textuelle souhaitée sinon il est conseillé de définir un métamodèle adapté à la syntaxe concrète (par exemple engendré automatiquement) puis d'écrire une transformation de modèle pour obtenir un modèle conforme au métamodèle existant du DSML.

1. http ://www.eclipse.org/Xtext/
2. http ://www.eclipse.org/workinggroups/oaw/
3. Voir la documentation de *Xtext* : `http://www.eclipse.org/Xtext/documentation`

Exercice 1 : Première syntaxe textuelle pour SimplePDL : PDL1
Comme toujours avec Eclipse, il faut commencer par créer un nouveau projet.

1.1 *Création du projet* Xtext. Sous Eclipse, on commence par définir un projet (New > Project, puis Xtext > Xtext project). Il faut indiquer d'abord le nom du projet principal (`fr.enseeiht.pdl1` par exemple [4]), le nom du langage (`PDL1`), et l'extension pour les fichiers contenant les modèles correspondants (`pdl1`).

Plusieurs projets sont alors créés, en particulier `fr.enseeiht.pdl1` qui contient dans `src` le fichier *Xtext* qui décrit la syntaxe de `PDL1`. Ce fichier nommé `pdl1.xtext` apparaît d'ailleurs dans l'éditeur. Il contient un squelette montrant la structure d'un fichier *Xtext*. La première ligne indique le langage qui est défini. La ligne suivante indique le métamodèle qui sera engendré. Le reste du fichier est composé des règles de production définissant la syntaxe concrète textuelle avec les actions qui permettent de construire le modèle EMF correspondant.

1.2 *Définition de la syntaxe PDL1.* Un exemple de la syntaxe textuelle souhaitée est donné dans l'exemple ci-dessous.

```
process dvp {
      wd a
      wd b
      wd c
      ws s2s from a to b
      ws f2f from b to c
}
```

Le fichier *Xtext* correspondant est donné au listing 1.

Remplacer le texte de PDL1.xtext par celui de la figure 1.

1.3 *Engendrer l'outillage Xtext pour PDL1.* Dans le répertoire `src` du projet principal, il faut sélectionner le fichier `GeneratePDL1.mwe`, cliquer à droite et sélectionner *Run As* > *MWE Workflow*. La génération prend un peu de temps...

1.4 *Utiliser l'éditeur PDL1 engendré.* Il faut lancer un second Eclipse depuis le premier pour avoir accès aux projets *Xtext* engendrés. Il suffit de faire un clique droit sur le nom du projet et de choisir « *Run As / Eclipse Application* ». Dans le deuxième Eclipse, on peut créer un projet et dans ce projet créer un fichier avec l'extension `.pdl1`, par exemple `process1.pdl1`. Saisir un exemple de processus et regarder les possibilités de l'éditeur (colorisation, complétion syntaxique...).

1.5 *Consulter le métamodèle PDL1 engendré.* Le métamodèle PDL1 a été engendré. Le fichier `PDL1.ecore` est présent dans le répertoire `src-gen` du projet principal. L'ouvrir avec l'éditeur graphique Ecore (cliquer à droite et sélectionner *initialize ecore_diagram diagram file*).

Exercice 2 : Autre syntaxe textuelle pour SimplePDL : PDL2
On considère une nouvelle syntaxe textuelle dont le fichier *Xtext* est donné au listing 2.

2.1 Donner la syntaxe PDL2 du processus donné à la question 1.2.

2.2 Engendrer l'éditeur pour PDL2 et l'utiliser (en relançant le second Eclipse).

2.3 Comparer les deux métamodèles, `PDL1`.ecore et `PDL2.ecore`.

4. Le nom du projet doit être en minuscules et contenir au moins un point « . ».

Listing 1 – Description *Xtext* pour la syntaxe PDL1

```
grammar fr.enseeiht.PDL1 with org.eclipse.xtext.common.Terminals

generate pDL1 "http://www.enseeiht.fr/PDL1"

Process :
      'process' name=ID '{'
         processElements+=ProcessElement*
      '}'
      ;

ProcessElement :
      WorkDefinition
      | WorkSequence
      | Guidance
      ;

WorkSequence :
      'ws' linkType=WorkSequenceType
            'from' predecessor=[WorkDefinition]
            'to' successor=[WorkDefinition]
      ;

Guidance:
      text=STRING
      ;

WorkDefinition :
      'wd' name=ID
      ;

enum WorkSequenceType :
      start2start = 's2s'
      | finish2start = 'f2s'
      | start2finish = 's2f'
      | finish2finish = 'f2f'
      ;
```

Listing 2 – Description *Xtext* pour la syntaxe PDL2

```
grammar fr.enseeiht.PDL2 with org.eclipse.xtext.common.Terminals

generate pDL2 "http://www.enseeiht.fr/PDL2"

Process :
      'process' name=ID '{'
         processElements+=ProcessElement*
      '}'
      ;

ProcessElement :
      WorkDefinition
      | Guidance
      ;

WorkDefinition :
      'wd' name=ID '{'
            ('starts' 'if' (linksToPredecessors+=DependanceStart)+)?
            ('finishes' 'if' (linksToSuccessors+=DependanceFinish)+)?
      '}'
      ;

DependanceStart :
      predecessor=[WorkDefinition] link=WorkSequenceKindStart
      ;

WorkSequenceKindStart:
      Started2Start = 'started'
      | Started2Finish = 'finished'
      ;

DependanceFinish :
      predecessor=[WorkDefinition] link=WorkSequenceKindFinish
      ;

WorkSequenceKindFinish:
      Finished2Start = 'started'
      | Finished2Finish = 'finished'
      ;

Guidance:
      text=STRING
      ;
```

2 Application aux réseaux de Petri

Exercice 3 : Définir des syntaxes textuelles pour les réseaux de Petri
Proposer une ou plusieurs syntaxes textuelles pour les réseaux de Petri et utiliser *xText* pour engendrer l'outillage Eclipse correspondant.