Définition de syntaxes concrètes textuelles

Version d'Eclipse à utiliser : `/mnt/n7fs/ens/tp_dupont/modelling-2024-09/eclipse/eclipse`

La syntaxe abstraite d'un DSML (exprimée en Ecore ou un autre langage de métamodélisation) ne peut pas être manipulée directement. Le projet EMF d'Eclipse permet d'engendrer un éditeur arborescent et les classes Java pour 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 (TP 6).

Pour la définition des syntaxes concrètes textuelles, nous utiliserons l'outil *Xtext* [1] de openArchitectureWare (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 [2] ! 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 [3]) puis d'écrire une transformation de modèle à modèle pour obtenir un modèle conforme au métamodèle existant du DSML.

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.n7.pdl1` par exemple [4]), le nom du langage (`fr.n7.PDL1`), et l'extension pour les fichiers contenant les modèles correspondants (`pdl1`).

1. `https://www.eclipse.org/Xtext/`
2. Voir la documentation de *Xtext* : `http://www.eclipse.org/Xtext/documentation`
3. Attention, la syntaxe engendrée automatiquement est très verbeuse.
4. Le nom du projet doit être en minuscules et contenir au moins un point « . ».

Plusieurs projets sont alors créés, en particulier `fr.n7.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. Le fichier *Xtext* correspondant est donné au listing 1.

```
process ex1 {
      wd a
      wd b
      wd c
      ws s2s from a to b
      ws f2f from b to c
}
```

Listing 1 – Description *Xtext* pour la syntaxe PDL1

```
1  grammar fr.n7.PDL1 with org.eclipse.xtext.common.Terminals
2  generate pDL1 "http://www.n7.fr/PDL1"
3
4  Process : 'process' name=ID '{'
5            processElements+=ProcessElement*
6        '}' ;
7
8  ProcessElement : WorkDefinition | WorkSequence | Guidance ;
9
10 WorkDefinition : 'wd' name=ID ;
11
12 WorkSequence : 'ws' linkType=WorkSequenceType
13       'from' predecessor=[WorkDefinition]
14       'to' successor=[WorkDefinition] ;
15
16 enum WorkSequenceType : start2start='s2s'
17       | finish2start='f2s'
18       | start2finish='s2f'
19       | finish2finish='f2f' ;
20
21 Guidance : 'note' texte=STRING ;
```

1.2.1. Expliquer les différents éléments du fichier Xtext du listing 1.

1.2.2. Remplacer le texte de PDL1.xtext par celui du listing 1.

1.3. *Engendrer l'outillage Xtext pour PDL1.* Dans le répertoire `src` du projet principal, il faut sélectionner le fichier `GeneratePDL1.mwe2`, 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 se placer dans l'Eclipse de déploiement. Ensuite, on peut créer dans le projet `fr.n7.simplepdl.exemples` un fichier avec l'extension `.pdl1`, par exemple `ex1.pdl1`. Saisir un exemple de processus et regarder les possibilités de l'éditeur (colorisation, complétion syntaxique...).
Attention : Le projet doit avoir la nature Xtext. On l'ajoute soit avec clic droit puis *Configure / Convert to Xtext Project*, soit avec un clic droit sur le fichier ex1.pdl1, *Open with ...* puis *PDL1 Editor* (répondre oui quand Eclipse demande de convertir le projet en un projet Xtext).
1.5. *Visualiser avec l'éditeur Ecore.* Un fichier écrit avec Xtext est en fait un modèle conforme au métamodèle que Xtext génère. Pour s'en convaincre, dans l'Eclipse de déploiement, faire un clic droit sur le fichier `.pdl1` puis *Open With > Other... > Sample Ecore Model Editor*.
1.6. *Consulter le métamodèle PDL1 engendré.* Le métamodèle PDL1 a été engendré. Le fichier `PDL1.ecore` est présent dans le répertoire `model/generated` du projet principal. Pour l'ouvrir avec l'éditeur graphique, faire un clic droit et sélectionner *Initialize Ecore Diagram...*.

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, sur papier ou dans un fichier avec l'extension `.pdfl2`, la syntaxe PDL2 du processus donné à la question 1.2. Il s'agit d'écrire dans la syntaxe PDL2 le processus présenté dans le sujet dans la syntaxe PDL1.
2.2. Engendrer l'éditeur pour PDL2, le déployer et l'utiliser.
2.3. Vérifier la syntaxe PDL2 proposée pour l'exemple de processus en l'ouvrant avec l'éditeur syntaxique pour PDL2.
2.4. Visualiser le métamodèles `PDL2` et constater qu'il est très différent de `PDL1.ecore`.

Exercice 3 : Nouvelle syntaxe textuelle pour SimplePDL : PDL3
Définir une nouvelle syntaxe concrète textuelle, PDL3, pour SimplePDL dont voici un exemple :

```
process : ex1
workdefinitions : a; b; c;
worksequences : a s2s b; b f2f c;
```

2 Transformations vers SimplePDL

Exercice 4 : De PDL1 à SimplePDL
Les fichiers écrits avec l'éditeur engendré par Xtext ne sont pas compatibles avec le métamodèle SimplePDL. C'est évident pour des métamodèles éloignés comme PDL2 mais aussi pour ceux qui sont structurellement proches comme PDL1. Il nous faut donc les transformer en modèles conformes à SimplePDL si l'on veut réutiliser l'outillage défini pour ce métamodèle. On peut naturellement le faire avec une transformation ATL.

Écrire une transformation ATL d'un modèle conforme au méta-modèle PDL1 engendré par Xtext à un modèle conforme à SimplePDL. On pourra reprendre/s'inspirer du début de code (incomplet) du listing 3. *Il est conseillé pour cela de se placer dans l'Eclipse de déploiement, et de chercher les méta-modèles parmi les* Registered Packages.

Listing 2 – Description *Xtext* pour la syntaxe PDL2

```
1  grammar fr.n7.PDL2 with org.eclipse.xtext.common.Terminals
2
3  generate pDL2 "http://www.n7.fr/PDL2"
4
5  Process :
6        'process' name=ID '{'
7           processElements+=ProcessElement*
8        '}' ;
9
10 ProcessElement :
11       WorkDefinition | Guidance ;
12
13 WorkDefinition :
14       'wd' name=ID '{'
15             ( 'starts' 'if' (linksToPredecessors+=DependanceStart)+ )?
16             ( 'finishes' 'if' (linksToPredecessors+=DependanceFinish)+ )?
17       '}' ;
18
19 DependanceStart :
20       predecessor=[WorkDefinition] link=WorkSequenceKindStart ;
21
22 WorkSequenceKindStart:
23       Started2Start='started' | Started2Finish='finished' ;
24
25 DependanceFinish :
26       predecessor=[WorkDefinition] link=WorkSequenceKindFinish ;
27
28 WorkSequenceKindFinish:
29       Finished2Start='started' | Finished2Finish='finished' ;
30
31 Guidance :
32       'note' texte=STRING ;
```

Listing 3 – Idée de la transformation ATL de PDL1 vers SimplePDL

```
-- ... l'en-tête est volontairement omis
-- Note : XPDL = méta-modèle issu de Xtext
--      SPDL = méta-modèle SimplePDL
create OUT : SPDL from IN : XPDL;

-- Ce helper n'a pas de contexte. Il est défini au niveau du module.
helper def: convertLinkType(x : XPDL!WorkSequenceType) : SPDL!WorkSequenceType =
   if x = #start2start then #s2s
   else if x = #start2finish then #s2f
   else if x = #finish2start then #f2s
   else #f2f
   endif endif endif; -- pas de SinonSi en ATL

rule XProc2SProc {
   from xp : XPDL!Process
   to  sp : SPDL!Process(
      name <- xp.name,
      -- Ajouter les enfants, sinon les autres éléments créés seront orphelins !
      processElements <- xp.processElements)
}

rule XWD2SWD {
   from xwd : XPDL!WorkDefinition
   to  swd : SPDL!WorkDefinition(name <- xwd.name)
}

rule XWS2SWS {
   from xws : XPDL!WorkSequence
   to  sws : SPDL!WorkSequence(
         linkType <- thisModule.convertLinkType(xws.linkType), -- Appel à notre helper
         predecessor <- xws.predecessor, -- Conversion/appel de règle implicite
         successor <- xws.successor
      )
}
```

3 Mini-projet

Exercice 5 : Prendre en compte les ressources
Intégrer les ressources dans l'une des trois syntaxes concrètes proposées pour SimplePDL.