Transformation de modèle à texte

Attention : Version d'Eclipse à utiliser : `/mnt/n7fs/ens/tp_cregut/eclipse-gls/eclipse`

Intéressons nous aux transformations d'un modèle en un texte. On parle de transformation modèle vers texte (M2T). Nous allons utiliser l'outil Acceleo [1] de la société Obeo.

1 Transformation de modèle à texte avec Acceleo

Nous commençons par engendrer un fichier HTML à partir d'un modèle SimplePDL, puis nous engendrerons un fichier *dot* pour pouvoir visualiser graphiquement un modèle de procédé.

Exercice 1 : Comprendre la définition d'un syntaxe concrète textuelle avec Acceleo
Nous souhaitons engendrer un fichier HTML décrivant un modèle SimplePDL. Voici le fichier résultat pour le processus « developpement ».

```
<head><title>developpement</title></head>
<body>
<h1>Process "developpement"</h1>
<h2>Work definitions</h2>
 <ul>
  <li>Conception</li>
  <li>RedactionDoc requires Conception to be started, Conception to be finished.</li>
  <li>Programmation requires Conception to be finished.</li>
  <li>RedactionTests requires Conception to be started, Programmation to be finished.</li>
 </ul>
</body>
```

Le principe d'Acceleo est de s'appuyer sur des gabarits (*templates*) des fichiers à engendrer. Le template qui a produit le fichier HTML précédent est donné au listing 1.
1.1. Expliquer les différents éléments qui apparaissent sur le listing 1. On pourra s'appuyer sur la documentation fournie dans Eclipse (*Help* > *Help Contents* > *Acceleo*).

Exercice 2 : Créer et appliquer un template Acceleo
Pour créer et appliquer un template Acceleo, nous nous servons du métamodèle de SimplePDL.
Attention : On utilisera le métamodèle et les modèles SimplePDL des TP précédents.
2.1. *Créer un projet de génération Acceleo.* Pour créer un projet de génération Acceleo, faire *New* > *Other* > *Acceleo Model to Text* > *Acceleo Project*.
Donner un nom au projet (`fr.n7.simplepdl.toHTML`) puis faire *Next*. Dans le dialogue, définir les paramètres de génération Acceleo (fig. 1) :

1. le nom du module : `toHTML`
2. le métamodèle : `http://simplepdl` : cliquer sur +, cocher *Runtime Version* et utiliser le motif `*simp`
3. le type de l'élément sur lequel s'appliquera la transformation : `Process`

1. `www.acceleo.org`

Listing 1 – Template Acceleo pour engendrer un fichier HTML à partir d'un modèle SimplePDL

```
1  [comment encoding = UTF-8 /]
2  [module toHTML('http://simplepdl')]
3
4  [template public processToHTML(aProcess : Process)]
5  [comment @main/]
6  [file (aProcess.name + '.html', false, 'UTF-8')]
7  <head><title>[aProcess.name/]</title></head>
8  <body>
9  <h1>Process "[aProcess.name/]"</h1>
10 <h2>Work definitions</h2>
11 [let wds : OrderedSet(WorkDefinition) = aProcess.getWDs() ]
12    [if (wds->size() > 0)]
13   <ul>
14       [for (wd : WorkDefinition | wds)]
15    <li>[wd.toHTML()/]</li>
16       [/for]
17   </ul>
18    [else]
19   <b>None.</b>
20    [/if]
21 [/let]
22 </body>
23 [/file]
24 [/template]
25
26 [query public getWDs(p: Process) : OrderedSet(WorkDefinition) =
27    p.processElements->select( e | e.oclIsTypeOf(WorkDefinition) )
28       ->collect( e | e.oclAsType(WorkDefinition) )
29       ->asOrderedSet()
30 /]
31
32 [query public toState(link: WorkSequenceType) : String =
33    if link = WorkSequenceType::startToStart or link = WorkSequenceType::startToFinish then
34       'started'
35    else
36       'finished'
37    endif
38 /]
39
40 [template public toHTML(wd : WorkDefinition) post (trim()) ]
41 [wd.name /] [for (ws: WorkSequence | wd.linksToPredecessors)
42          before (' requires ') separator (', ') after('.')
43 ][ws.predecessor.name /] to be [ws.linkType.toState()
44 /][/for]
45 [/template]
```

4. le nom du template : `processToHTML`
5. cocher *Generate file* et *Main template*.

Faire *Finish* pour terminer la création du projet. Un nouveau projet est alors engendré.

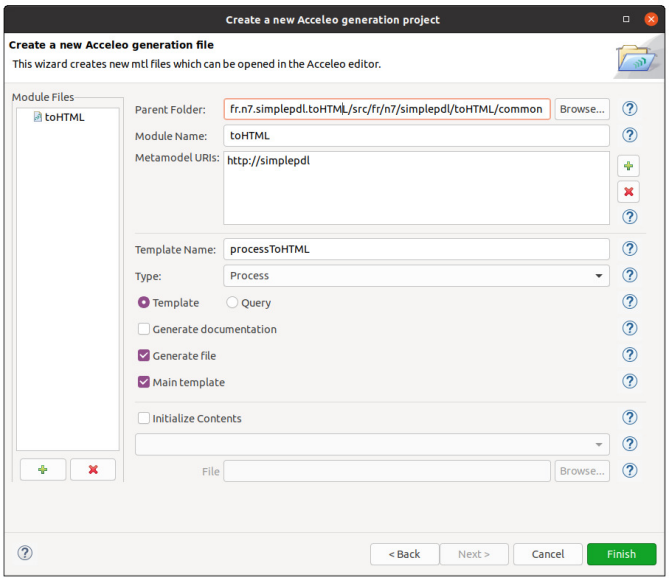


FIGURE 1 – Informations à fournir à l'assistant de création Acceleo

2.2. Le projet contient un dossier de sources (`src`). Dans le paquetage `fr.n7.simplepdl.toHTML.main`, un *template* de génération a été engendré (`toHTML.mtl`). Ouvrir ce fichier.
2.3. Remplacer le contenu du fichier `toHTML.mtl` par celui du listing 1 (disponible sur Moodle).
2.4. Exécuter la transformation M2T. Il suffit de cliquer droit sur le fichier `.mtl` puis faire *Run as > Launch Acceleo Application*. Dans la fenêtre de configuration de la transformation sélectionner le modèle d'entrée (dans le champ *Model*) ainsi qu'un dossier cible de la transformation (*Target*) où sera engendré le résultat de la transformation. Si Acceleo ne trouve pas le méta-modèle malgré le fait que tout est bien configuré, il faut modifier la méthode `registerPackages(ResourceSet)` dans le fichier *ToHTML.java* selon les commentaires présents dans le code de cette méthode.

Exercice 3 : Application à la génération d'un fichier .dot
Écrire une transformation modèle à texte qui permet de traduire un modèle de procédé en une syntaxe dot. Voici un exemple simple de fichier dot [2] pour le même modèle de processus.

```
digraph developpement {
        Conception -> RedactionDoc [arrowhead=vee label=f2f]
        Conception -> RedactionDoc [arrowhead=vee label=s2s]
        Conception -> Programmation [arrowhead=vee label=f2s]
        Conception -> RedactionTests [arrowhead=vee label=s2s]
        Programmation -> RedactionTests [arrowhead=vee label=f2f]
}
```

2. Voir `http://www.graphviz.org/Documentation.php` pour la documentation et des exemples du langage *dot*.

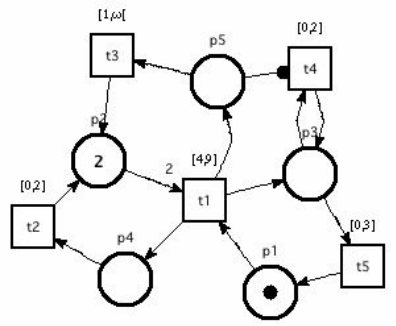
Une fois le fichier `.dot` obtenu, on obtient le graphe correspondant en PDF en faisant :

```
dot ficher.dot -Tpdf -o fichier.pdf
```

3.1. Créer un projet `fr.n7.simplepdl.todot`, écrire puis tester un template Acceleo `toDot.mtl` qui engendre un fichier `.dot` à partir d'un modèle de processus. On pourra ajouter un template principal à notre projet avec : *New* > *Other* > *Acceleo Model to Text* > *Acceleo Main Module File*.

2 Application aux réseaux de Petri

Exercice 4 : Transformations modèle à texte pour les réseaux de Petri
Nous allons maintenant définir une transformation modèle à texte pour les réseaux de Petri. L'objectif est d'engendrer la syntaxe textuelle utilisée par les outils Tina [3], en particulier `nd` (NetDraw). La figure 2(a) illustre les principaux concepts présents des réseaux de Petri temporels que nous considérons. Le fichier textuel de ce réseau est donné au listing 2(b) au format Tina.



(a) syntaxe graphique

```
1  net ifip
2  pl p1 (1)
3  pl p2 (2)
4  pl p3 (0)
5  pl p4 (0)
6  pl p5 (0)
7  tr t1 [4,9] p1 p2*2 -> p3 p4 p5
8  tr t2 [0,2] p4 -> p2
9  tr t3 [1,w[ p5 -> p2
10 tr t4 [0,2] p3 p5?1 -> p3
11 tr t5 [0,3] p3 -> p1
```

(b) syntaxe textuelle

FIGURE 2 – Exemple de réseau de Petri

4.1. Créer un projet `fr.n7.petrinet.totina` et écrire un template Acceleo `toTina.mtl` qui transforme un modèle de réseau de Petri en un fichier `.net`. Utiliser `nd` pour visualiser le fichier produit. Ne pas oublier de créer le lanceur `fr.n7.petrinet.totina.ui` et de déployer les greffons.
4.2. Créer un projet `fr.n7.petrinet.todot` et écrire un template Acceleo `toDot.mtl` pour transformer un modèle de réseau de Petri en un fichier `.dot` qui permettra de visualiser graphiquement le réseau. Ne pas oublier de créer le lanceur `fr.n7.petrinet.todot.ui` et de déployer les projets.

3. http ://www.laas.fr/tina/