Transformation de modèle à modèle

Version d'Eclipse à utiliser : `/mnt/n7fs/ens/tp_dupont/modelling-2023-09/eclipse/eclipse`

ATL [1] est un outil de transformation de modèle à modèle développé par le groupe ATLAS. Cette technologie permet d'exprimer des transformations sous la forme de *règles* : plutôt que de donner les étapes précises de la transformation, on se contente de dire en quoi se transforme tel ou tel objet, et c'est ATL qui se charge de gérer les ressources, les modèles, et d'appeler les bonnes règles au bon moment et dans le bon ordre.

ATL est une technologie extrêmement riche [2] dont nous n'exploiterons qu'une petite partie.

1 Le métamodèle TaskMaster

Exercice 1 : TaskMaster

Dans cette partie, nous instrumenterons, en plus de SimplePDL, le métamodèle TaskMaster, donné figure 1. Il s'agit d'une façon de présenter des scénarios *orienté événement*, plutôt que orienté « activités » comme SimplePDL.

Un modèle TaskMaster a pour racine un groupe d'événements (`EventBundle`), qui se compose d'événements (`Event`). Un événement peut définir plusieurs dépendances (`Requirement`). Une dépendance peut être *causale* sur un autre événement (`Causal`), autrement dit le fait qu'un événement ne peut arriver qu'après ceux sur lequel il a une dépendance se soient passés, ou une dépendance « physique » (`Physical`), c'est-à-dire un besoin particulier en stock (`Supply`), consommable (papier, café, etc.) ou non (ordinateurs, bureau, etc.).

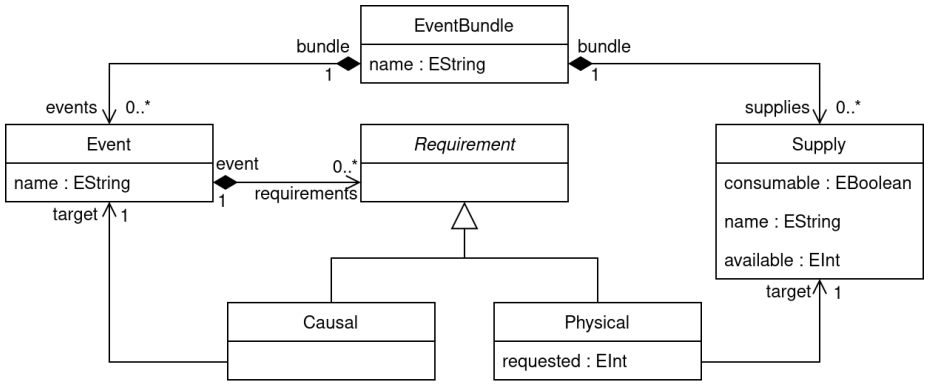

FIGURE 1 – Métamodèle TaskMaster

1.1. Importer le fichier Ecore du métamodèle TaskMasterdans un nouveau projet Eclipse.
1.2. Créer un `genmodel` pour TaskMaster et générer les sources.

1. `https://www.eclipse.org/atl/`
2. Documentation : `https://www.eclipse.org/atl/documentation/`

1.3. (*Optionnel*) On peut récupérer une transformation M2T Acceleo qui permet de transformer un modèle TaskMaster en DOT, ce qui permet de le visualiser. Pour cela, créer un nouveau projet Acceleo et remplacer le code du fichier `TaskMaster2dot.mtl` par le code Acceleo fourni.

Le cas échéant, modifier `Generate.java` pour charger correctement le métamodèle.

2 Projet ATL

Exercice 2 : Créer une transformation ATL
Le but est de transformer un modèle de procédé en modèle TaskMaster, à l'aide d'ATL.
2.1. *Projet ATL.* Créer un nouveau projet ATL (*File* > *New* > *Project...* puis *ATL* > *ATL Project*).
Remarque : on peut aussi créer un projet autre et lui rajouter la "nature ATL" avec clic-droit puis Configure > Convert to ATL Project).
2.2. *Fichier ATL.* Pour créer le fichier ATL, suivre les étapes suivantes.
2.2.1. Créer un fichier ATL (*File* > *New* > *Other...* puis *ATL* > *ATL File*). Sélectionner le projet ATL précédemment créé et renseigner le nom de fichier (il doit avec l'extension `.atl`). Faire *Next*. L'écran suivant permet de configurer les entrées/sorties de la transformation.
2.2.2. Ajouter une entrée (avec *Add*). Prendre soin de renommer le métamodèle (`MM` par défaut) en `SimplePDL`. Expliciter le métamodèle en entrée, `SimplePDL.ecore`, en allant chercher la ressource (avec *Browse Workspace*), puis faire *OK*.
2.2.3. De même, ajouter une sortie. Renommer le métamodèle en `TM` et donner le chemin vers le métamodèle en sortie, `TaskMaster.ecore`. Faire *OK*, puis faire *Finish*. Le fichier ATL créé s'ouvre avec un morceau de code minimal.

Remarque : on peut configurer et lancer ATL via le menu des *Run Configurations* (*Run* > *Run Configurations* puis double-clic sur *ATL Transformation*).

3 Transformation de SimplePDL à TaskMaster

L'objectif de cette partie est de concevoir une transformation de SimplePDL vers TaskMaster. La figure 2 donne un aperçu du résultat de la transformation sur un exemple.

Une transformation ATL se base sur un ensemble de *règles*, dont l'entrée (`from`) est un objet d'un métamodèle d'entrée, et les sorties (`to`) sont des objets des métamodèles de sortie.

Exercice 3 : Processus
Pour chaque élément *Process*, on veut créer un élément *EventBundle* de même nom. On ne se soucie pas pour le moment des enfants de ces éléments :

```
rule Process2EventBundle {
   from process : SimplePDL!Process
   to  bundle : TM!EventBundle(name <- process.name)
}
```

3.1. Ajouter ce morceau de code et exécuter la transformation sur un modèle SimplePDL valide.
3.2. Constater que ATL crée bien un modèle conforme à TaskMaster, qui contient un unique *EventBundle* du même nom que le procédé présent dans le modèle en entrée.

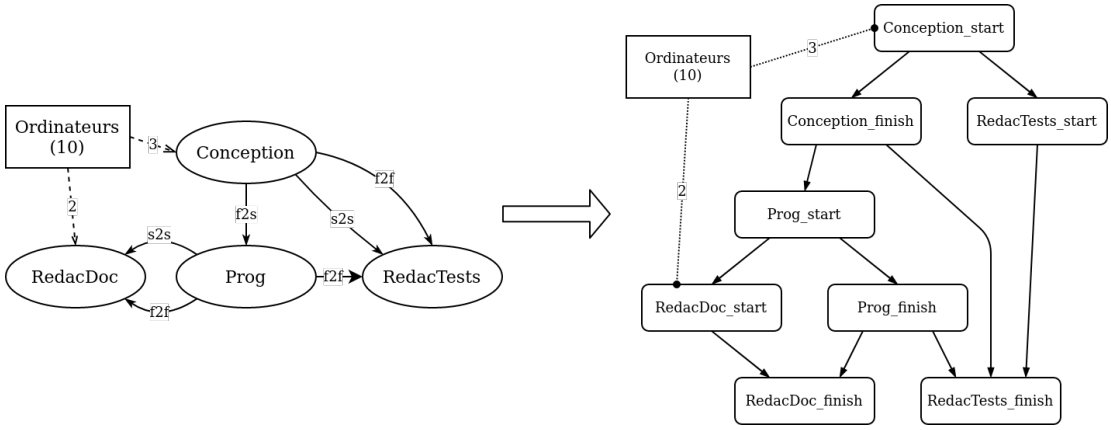

FIGURE 2 – Exemple de transformation de SimplePDL à TaskMaster

Exercice 4 : Activités

Une activité (*WorkDefinition*) de SimplePDL correspond à deux événements (*Event*) dans TaskMaster : un événement de début et un événement de fin. On notera que la fin arrive forcément après le début ; il faut donc aussi créer une dépendance causale entre ces deux événements.

4.1. Ajouter la règle suivante qui crée deux événements et une dépendance causale par tâche :

```
rule WD2Task {
  from wd : SimplePDL!WorkDefinition
  to  event_start : TM!Event(name <- wd.name + '_start'),
      event_finish: TM!Event(name <- wd.name + '_finish'),
      finish_after_start: TM!Causal(event <- event_finish, target <- event_start)
}
```

4.2. Exécuter ce code. On constate que les événements n'apparaissent pas sous l'élément racine `EventBundle` mais à côté (il ne sera donc pas valide). Pourquoi ?

4.3. Il nous faut rattacher les événements créés au "bundle" créé par la première règle. Pour cela, une première étape consiste à récupérer le *process* dans lequel se trouve la tâche en entrée. Comme OCL, ATL permet de définir des *helper*. On peut par exemple reprendre *getProcess* :

```
helper context SimplePDL!ProcessElement
def: getProcess() : SimplePDL!Process =
   SimplePDL!Process.allInstances()->select(e | e.processElements->includes(self));
```

À noter que, comme pour OCL, il est nécessaire de donner un contexte pour un helper. On note aussi que l'on peut faire référence à des éléments d'un métamodèle dans le typage en le préfixant avec le nom du métamodèle, suivi du point d'exclamation.

On peut maintenant indiquer à ATL que les événements créés doivent être rajoutés dans le "bundle" qui correspond au process auquel appartient l'entrée :

```
rule WD2Task {
   from wd : SimplePDL!WorkDefinition
   to
      event_start : TM!Event(bundle <- wd.getProcess(), name <- wd.name + '_start'),
      event_finish: TM!Event(bundle <- wd.getProcess(), name <- wd.name + '_finish'),
      finish_after_start: TM!Causal(event <- event_finish, target <- event_start)
}
```

Remarque : Ici, ATL devine tout seul que l'on veut affecter à l'attribut `bundle` le résultat de la transformation de l'élément `wd.getProcess()` (le process contenant la *WorkDefinition*) car il n'y a pas d'ambiguïté. Il se charge tout seul de récupérer le résultat de cette transformation, et s'assure par ailleurs que la règle associée a bien été exécutée avant (pour éviter d'avoir `null`).
4.4. Exécuter la transformation, les éléments devraient apparaitre dans le modèle en sortie.

Exercice 5 : Dépendances
Une dépendance entre tâches se transforme assez naturellement en dépendance *causale*. Le type de la *WorkSequence* détermine quels événements (`xxx_start` ou `xxx_finish`) sont en relation.

De façon similaire à l'exercice précédent, il faudrait que l'on arrive à récupérer les événements résultants de la transformation du prédécesseur et du successeur de la *WorkSequence* ; mais cette fois-ci, la règle qui transforme les activités crée plusieurs éléments, et on n'en veut qu'un seul (`event_start` ou `event_finish`). ATL ne peut pas choisir tout seul. Il faut l'aider grâce à la méthode :

```
thisModule.resolveTemp(<entrée>, '<variable>')
```

Elle permet de récupérer une variable créée par la règle qui a été appliquée à l'entrée donnée. Par exemple, le code ci-dessous :

```
rule WS2Requirement {
   from ws : SimplePDL!WorkSequence
   to  req : TM!Causal(
         event <- thisModule.resolveTemp(ws.successor, 'event_finish'),
...
```

affecte l'attribut `event` (= événement dépendant) de la dépendance causale avec l'objet associé à la variable `event_finish` créée lors de la transformation de `ws.successor` (par la règle `WD2Task`).

Bien sûr, comme le deuxième argument de `resolveTemp` est une chaîne de caractères, on peut se servir du + pour le faire varier :

```
event <- thisModule.resolveTemp(ws.successor, 'event_' +
      (if ws.linkType = #s2s or ws.linkType = #f2s then
         'start'
      else
         'finish'
      endif)),
```

Remarque : Faire un *helper* pour encapsuler la conditionnelle permettrait de gagner en lisibilité.
5.1. Compléter la transformation afin que les *WorkSequence* soient bien transformées.
5.2. Compléter la transformation pour prendre en compte les ressources. On exploitera pour cela les objets *Physical* et *Supply* de TaskMaster.

4 Mini-projet

Exercice 6 Écrire la transformation de SimplePDL en réseau de Petri en utilisant ATL.