Manipulation du métamodèle UML avec Kermeta

1 Installation de Kermeta

Nous utilisons Kermeta 1. La version d'Eclipse Juno Modeling Tools étant difficilement compatible avec Kermeta 1, nous allons utiliser une autre version d'Eclipse, plus ancienne. Il faut voir les choses du bon côté : on va utiliser une version déjà toute prête. Cette version est disponible à partir de l'adresse suivante pour linux 32 bits
http ://www.kermeta.org/releases/1.4.1/eclipse_package/kermeta-1.4.1-galileo-linux_x86.zip
et pour d'autres versions à l'adresse http ://gforge.inria.fr/frs/ ?group_id=32. Une copie est disponible localement sous /auto_home/cnebut/IDM/.

2 Le métamodèle UML

2.1 Récupérer le modèle ecore de UML

Tout d'abord vous devez créer un nouveau projet Kermeta dans votre Workspace. Dans le répertoire *metamodels*, copiez le fichier *UML.ecore*. Le fichier UML.ecore est disponible grâce aux divers plugs-ins installés dans Eclipse : vous pouvez le récupérer :

- en allant explorer le jar de plugin uml2 (de nom de type *org.eclipse.uml2.uml_2.2.2.v200811051031.jar*) dans le répertoire d'installation d'Eclipse
- ou en le récupérant ici sur la page des TPs IDM via http ://www.lirmm.fr/~nebut

2.2 Reconnaitre les parties du métamodèle UML vues en CM/TD

Ouvrez le métamodèle `uml.ecore`. Vous pouvez le naviguer facilement. Reconnaissez-y les parties que vous avez vues en CM/TD, notamment la modélisation des packages, des classes, des attributs et des associations, et de l'héritage.

2.3 Créer un modèle UML avec TopCaseD

Pour définir un modèle conforme au métamodèle UML, on peut soit passer par l'éditeur réflexif d'Eclipse (création d'instance dynamique), soit utiliser un éditeur graphique adéquate comme TopCased ou Papyrus. Nous vous proposons d'éditer avec Topcased le modèle UML donné Figure 1, il nous servira dans la suite. Pour créer un modèle UML avec Topcased, depuis le répertoire model : clic droit→new UML model with Topcased. Notez qu'après édition, vous obtenez deux fichiers : l'un pour la représentation graphique, et l'autre, suffixé uml, qui est en fait le modèle sérialisé en xmi. C'est celui-là que nous pourrons manipuler avec Kermeta. Vérifiez que le modèle que vous avez dessiné se représente sous forme d'instance du métamodèle UML de la façon dont vous l'auriez pensé.

3 Premières manipulations avec Kermeta

Créez un fichier Kermeta (new → Other → Kermeta File).

Pour manipuler un modèle UML avec Kermeta, il faut ajouter la ligne suivante dans le fichier Kermeta :

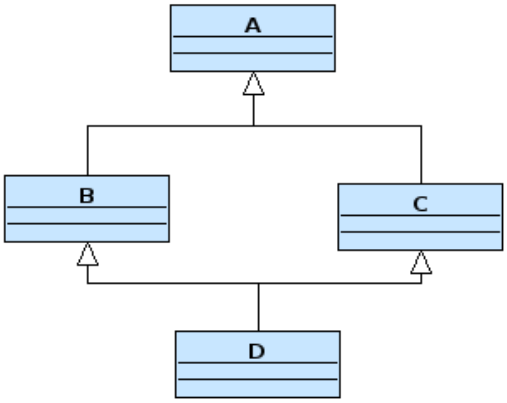

FIGURE 1 – Modèle UML de test

```
require "platform:/plugin/org.kermeta.uml2/src/kermeta/transformations/
    UmlTool.kmt"
using uml
```

On ne met par contre pas de ligne du genre :

```
require "platform:/resource/tp2/metamodele/UML.ecore"
```

qui vous indiquerait qu'il y a des erreurs dans le ECore, et ceci pour de sombres raisons partiellement expliquées ici :
http ://www.kermeta.org/documents/faq/ecore2kmt_errors

3.1 Mise en route

Ecrivez en kermeta une méthode qui affiche tous les noms des packages d'un modèle donné en paramètre.

Pour tester cette méthode, vous devrez charger le modèle de test précédemment construit, récupérer son élément racine de type Model, et le passer en paramètre de votre méthode.

3.2 Manipulations des hiérarchies

Créez les méthodes suivantes :

- `parents(c:Classifier): set Classifier[0..*]` qui retourne les parents directs du Classifier c
- `allParents(c:Classifier): set Classifier[0..*]` qui retourne tous les parents du Classifier c

Testez ces deux méthodes avec votre modèle de test. Pour cela, vous récupèrerez tous les classifiers d'un modèle, et pour chacun vous appelerez ces deux méthodes, puis vous ferez les affichages vous permettant de vérifier un fonctionnement correct.

On notera que ces deux méthodes auraient été mieux situées directement dans la classe `Classifier`, où elles sont prévues par UML (sans le paramètre `Classifier` bien sûr).

4 Codage de quelques méthodes utilitaires pour UML

Développez puis testez les méthodes suivantes dans une classe utilitaire :

- getAllClasses(model : uml : :model) qui retourne l'ensemble des classes du modèle
- getAllPackages(model : uml : :model) qui retourne l'ensemble des packages du modèle