Métamodèles, modèles et transformations de modèles : B.A BA Kermeta

1 Installation

La dernière version de Kermeta n'est pas installée en tant que plug-in Eclipse. La dernière version de Kermeta nécessite la version 3.5 d'Eclipse. Vous trouverez via cette adresse : http ://www.kermeta.org/download/ (puis section *Download and install*, première case *Beginner* du tableau pour accéder au *Kermeta+Eclipse bundle*) une archive contenant une version d'Eclipse avec les plugs-ins adéquates installés. Récupérez cette archive et décompressez-là dans votre répertoire personnel. Lancez l'Eclipse correspondant (modifiez le chemin vers le workspace !). Vous pourrez effacer ce répertoire d'installation d'Eclipse en fin de semestre.

2 Manipulations basiques sur des exemples de méta-modèles et de transformations

MinUML est un sous-ensemble minimaliste du méta-modèle UML, permettant juste de représenter une hiérarchie de classes.

Listing 1 – Métamodèle MinUML

```
package minuml;

require kermeta
using kermeta::standard

class ClassHierarchy
{
    attribute hierarchy: Class[0..*]
}

class Class
{
    attribute name: String[1..1]
    attribute ownedAttribute: Property[0..*]#~class
    reference superclass: Class[0..*]
}

class Property
{
    attribute name: String[1..1]
    reference redefinedProperty: Property[0..*]
    reference ~class: Class[1..1]#ownedAttribute
    reference type: Class[1..1]
}
```

On notera l'utilisation des mots-clefs : attribute, reference, et #.

attribute permet de déclarer une composition vers un autre élément.

reference permet de déclarer une référence vers un autre élément.

~ caractère de dé-spécialisation

permet de spécifier la propriété inverse

Voici le code source du méta-modèle MinDB, qui permet de définir des modèles de bases de données très simples :

Listing 2 – Métamodèle MinDB

```
package mindb;

require kermeta
using kermeta::standard

class Database
{
    attribute tables: Table[0..*]
}

class Table
{
    attribute name: String[1..1]
    attribute columns: Column[0..*]
}

class Column
{
    attribute name: String[1..1]
}
```

Vous trouverez ci-dessous une transformation simple de MinUML vers MinDB.

Listing 3 – Transformation MinUML vers MinDB

```
package minuml2mindb;

require kermeta
require "minuml.kmt"
require "mindb.kmt"
using kermeta::standard
using kermeta::persistence
using minuml
using mindb

class Minuml2Mindb
{
    operation loadClassHierarchy(filename: String): ClassHierarchy is do
        var repository: EMFRepository init EMFRepository.new
        var srcResource: EMFResource init EMFResource.new
        srcResource ?= repository.createResource(filename,"platform:/
            resource/IDM/minuml.ecore")
        srcResource.load()
        result ?= srcResource.instances.one
    end
```

```
    operation saveDatabase(db: Database, filename: String): Void is do
            var repository: EMFRepository init EMFRepository.new
        var tgtResource : EMFResource init EMFResource.new
        repository.registerEcoreFile("platform:/resource/IDM/mindb.ecore"
            )
        tgtResource ?= repository.createResource(filename,"platform:/
            resource/IDM/mindb.ecore")
        tgtResource.instances.add(db)
        tgtResource.save()
    end

    operation transform(source: ClassHierarchy): Database is do
        result := Database.new
        source.hierarchy.each{ cls |
            var table: Table init Table.new
            table.name := String.clone(cls.name)
            cls.ownedAttribute.each{ attr |
                var column: Column init Column.new
                column.name := String.clone(attr.name)
                table.columns.add(column)
            }
            result.tables.add(table)
        }
    end
}
```

Dans ce listing de transformation de MinUML vers MinDB, il y a trois opérations :

- *loadClassHierarchy* permet de charger un modèle stocké dans le fichier de nom *filename*, et conforme au méta-modèle MinUML, défini dans *minuml.ecore*. Dans la définition du chemin vers le méta-modèle, *platform :/resource* correspond au workspace courant.
- *saveDatabase* permet de sauvegarder le modèle de base de donnée obtenu
- *transform* réalise la transformation proprement dite.

Vous trouverez via http ://www.lirmm.fr/∼nebut/Enseignement/FMIN310 les sources des méta-modèles et de la transformation.

Question 1. Créez un nouveau projet Kermeta. Importez-y les deux métamodèles, et la transformation.

a- Les métamodèles sont donnés au format kermeta. Il existe une transformation pré-définie pour passer de kermeta à ecore, et réciproquement. Essayez cette transformation (clic-droit sur le fichier à transformer).

b- Modifiez le métamodèle minUML pour que chaque élément de modèle puisse porter un commentaire.

Question 2. Exécutez la transformation minUML vers minDB. Pour cela, il faut :

- Créer des modèles de test (ici : des modèles conformes à minUML), voir ci-dessous.
- Créer un programme de test. On fera un programme simple : chargement d'un modèle de test, application de la transformation, sauvegarde du modèle minDB résultat.

Pour créer un modèle de test, soit on dispose d'un éditeur graphique pour le métamodèle (ce n'est pas notre cas), soit d'une syntaxe textuelle et d'un parser vers un modèle (ce n'est pas non plus notre cas) soit on utilise la fonctionnalité d'EMF qui fournit un éditeur pour tout métamodèle ecore. Positionnez-vous sur l'élément racine du méta-modèle (celui qui contient tous les autres), et demandez la création d'une instance dynamique. Vous disposez alors d'un éditeur qui vous permet de définir des éléments de modèle conformes au méta-modèle.

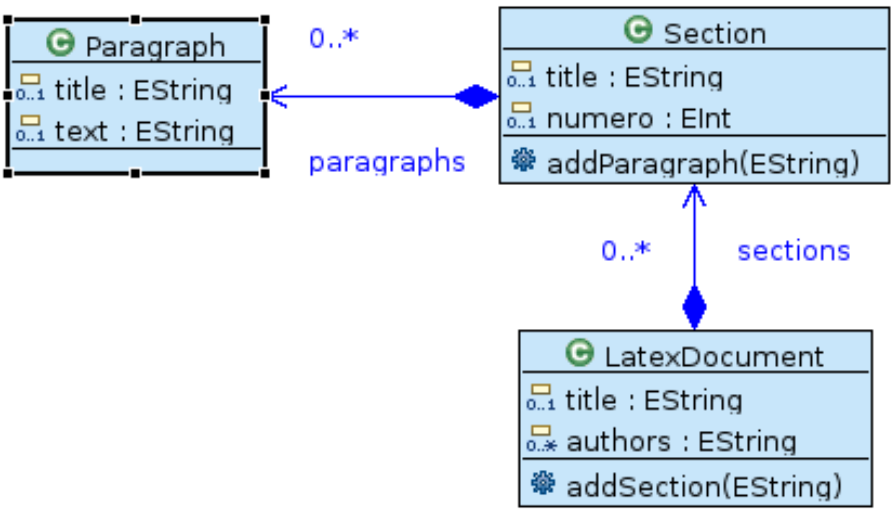

FIGURE 1 – Le méta-modèle LatexMM

Question 3. La transformation proposée est incomplète : elle ne prend pas en compte l'héritage de minUML. Complétez-là et testez votre nouvelle transformation.

3 Génération d'un dictionnaire très simple

On souhaite générer à partir d'un modèle minUML un dictionnaire des données très simple, qui recense les principaux termes utilisés dans le modèle, et les définit.

Question 4. Créez un métamodèle de dictionnaire très simple que l'on nommera dicodoc. Un dictionnaire contient le titre du modèle documenté ainsi que sa description, ainsi que des définitions de termes, chaque terme ayant un nom et un commentaire, et éventuellement un type. Prévoyez les méthodes qui vous seront utiles pour manipuler des modèles de documentation. Pour créer ce métamodèle, on peut soit les coder en kermeta, soit les créer avec l'éditeur ecore, on va utiliser cette deuxième solution. Créez un nouveau modèle Ecore. Demandez l'affichage des properties depuis le paquetage null, et définissez le nom de votre méta-modèle. Puis ajoutez des éléments de modèle conformes au méta-modèle Ecore, et définissez-en les propriétés.

Question 5. Créez une transformation de modèle qui génère un modèle dicodoc à partir d'un modèle minUML. Les termes mis dans le dictionnaire seront les noms de propriétés et de classe, s'il y a un commentaire on pourra l'utiliser. Testez votre transformation.

4 Formattage de la documentation en latex

On souhaite formatter le mini-dictionnaire grâce à latex. LateX est un langage permettant de créer des documents de différents formats (transparents, livres, articles). C'est un langage textuel à balises, qui est compilé vers du postcript, du pdf, ou du dvi.

La figure 1 donne un mini méta-modèle de document LateX, appelé LatexMM. Ce méta-modèle n'est absolument pas réaliste et capture peu des possibilités offertes par LateX, mais servira comme première base de travail.

Question 6. Ce méta-modèle vous est donné en utilisant comme syntaxe concrète un diagramme de classes UML. Pour pouvoir travailler sur ce méta-modèle, la première étape est de le rentrer sous forme de modèle ecore. Créez un nouveau projet Kermeta, puis un nouveau modèle Ecore. Demandez l'affichage des properties depuis le paquetage null, et définissez le nom

de votre méta-modèle. Puis ajoutez des éléments de modèle conformes au méta-modèle Ecore, et définissez-en les propriétés.

Question 7. Ecrivez une transformation de dicodoc vers latexMM. On fera un document avec deux sections : le première s'intitule "description du modèle" et contient la description du modèle. La seconde s'intitule "dictionnaire de données" et contient un paragraphe par terme, incluant le nom, la description et l'éventuel type. Testez votre transformation.

Question 8. Vous allez maintenant écrire une transformation qui permet de générer du code Latex à partir d'un modèle conforme à LatexMM. Cette transformation sera écrite en Kermeta. Vous devez donc transformer votre métamodèle LatexMM ecore en métamodèle LatexMM Kermeta, grâce à la transformation prédéfinie fourni par Kermeta. Regardez le code Kermeta obtenu. Pas de surprise, c'est une coquille vide : aucune méthode n'est implémentée. Implémentez en Kermeta les méthodes du méta-modèle.

Définissez ensuite une classe servant à la génération de LateX, et contenant une méthode GenLatex, que vous implémenterez également. Pour avoir une idée de la syntaxe LateX, un article LateX vous est fourni (exArticleMinimal.tex), ainsi que le pdf qui lui correspond (exArticleMinimal.pdf). Pour écrire dans un fichier, on utilisera la classe FileIO, qui possède entre autres les méthodes :

- readTextFile(filePath : String) : String qui retourne sous forme de chaîne le contenu du fichier dont le chemin est donnée en paramètre.
- writeTextFile(filePath : String, text : String) qui écrit le texte passé en paramètre dans le fichier dont le chemin est également passé en paramètre.

Rajoutez dans la classe LatexDocument du méta-modèle LatexMM (dans sa version Kermeta) la méthode *formatAuthors() :EString* qui retourne la chaîne résultant de la concaténation des noms des auteurs et implémentez-la en Kermeta. Utilisez à présent la transformation prédéfinie de Kermeta vers Ecore. La nouvelle méthode est bien présente dans le méta-modèle conforme à Ecore généré, mais le code Kermeta cette fois n'apparaît plus comme un élément du modèle mais comme une annotation de l'élément définissant la nouvelle opération.

Notez qu'il est assez courant de vouloir associer une syntaxe textuelle à un métamodèle pour pouvoir écrire les modèles sous forme de texte. La technique que nous avons vue ici n'est pas très adaptée pour la génération de code. Des outils plus adéquates existent autour de Kermeta (Sintaks – http ://www.kermeta.org/sintaks/ et KET – http ://www.kermeta.org/mdk/ket) et d'EMF (JET). Nous ne les étudierons pas ici.