Examen MDA

Mardi 25 mai 2010 – 2h – Les documents électroniques autorisés

Cours (4 pt)

Q1 : Un langage de modélisation peut-il disposer de plusieurs syntaxes concrètes ? Quel est l'intérêt ?

Réponse **:**
Oui, dans ce cours, nous avons précisé qu'un langage de modélisation correspondait à une syntaxe abstraite (métamodèle ou profile) et qu'à une syntaxe abstraite pouvait correspondre plusieurs syntaxes concrètes.
L'intérêt est d'avoir par exemple une syntaxe concrète textuelle ainsi qu'une syntaxe concrète graphique. UML par exemple, dispose d'une syntaxe concrète graphique (diagramme) et d'une syntaxe concrète textuelle (XMI).

Q2 : Un langage de modélisation doit-il absolument disposer d'une sémantique formelle ? Quel est l'intérêt ?

Réponse **:**
Non, dans ce cours, nous avons précisé qu'un langage de modélisation correspondait à une syntaxe abstraite (métamodèle ou profile) et qu'à une syntaxe abstraite pouvait correspondre (pas obligatoirement) une sémantique formelle. Cette sémantique formelle peut être opérationnelle ou dénotationelle. L'intérêt d'une sémantique formelle est de pouvoir faire des opérations de validation et de faire en sorte que ces opérations de validations soient « prouvée » grâce à une base mathématique.

UML Simplifié (4 pt)

Nous considérons dans cet examen que le métamodèle UML est celui présenté en Figure 1 :

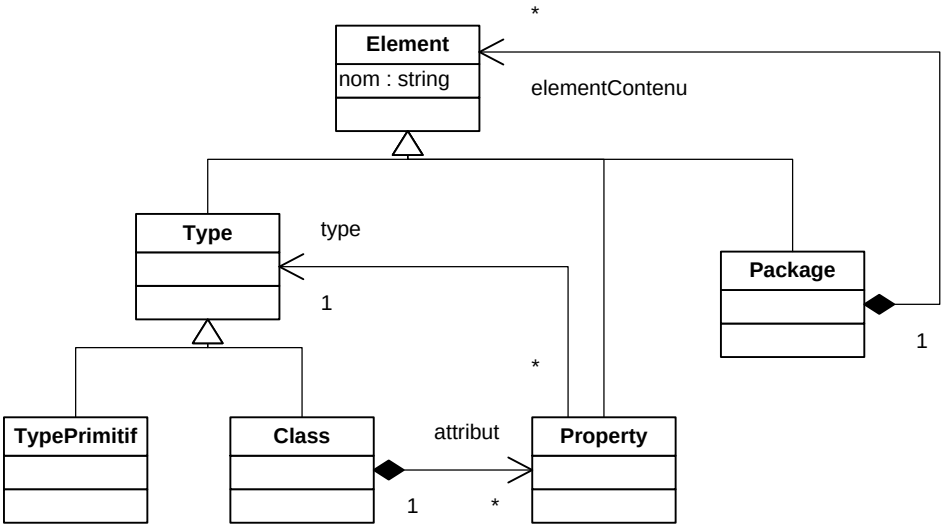


Figure 1 : métamodèle UML

Il permet de modéliser des diagrammes de classes relativement simples :

- les classes contiennent des attributs,
- les attributs peuvent être de type primitif (string, boolean, integer, énumération, etc.) ou de type classe (ce sont alors des références).
- les packages contiennent des classes.

La figure suivante (Figure 2) présente un modèle UML.

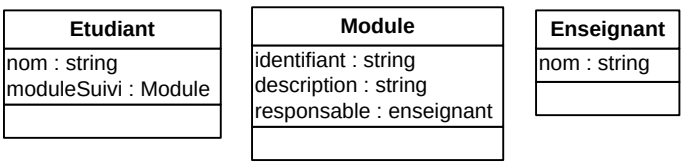


Figure 2 : un modèle UML

Q1 : Précisez pourquoi le modèle UML (donné en Figure 2) est conforme au métamodèle UML (donné en Figure 1) ?
Réponse :
Le modèle est composé de 3 classes (instance de la métaclasse Class). Ces classes possèdent (association attribut) des attributs (instances de la métaclasse Property). Tous les attributs de ce modèle sont typés (association type). Les types sont soit primitifs soit des classes du modèle.

Q2 : Le métamodèle UML (Figure 1) s'auto-décrit-il ?
Réponse :
Non, car ce métamodèle ne défini pas l'héritage ni la multiplicité des associations alors qu'il l'utilise. Par exemple Package hérite de Element

Script (6pt)

Le métamodèle suivant (Figure 3) permet de modéliser des scripts d'appel d'opérations sur des objets (dont les classes sont contenues dans des packages importés par le script) :

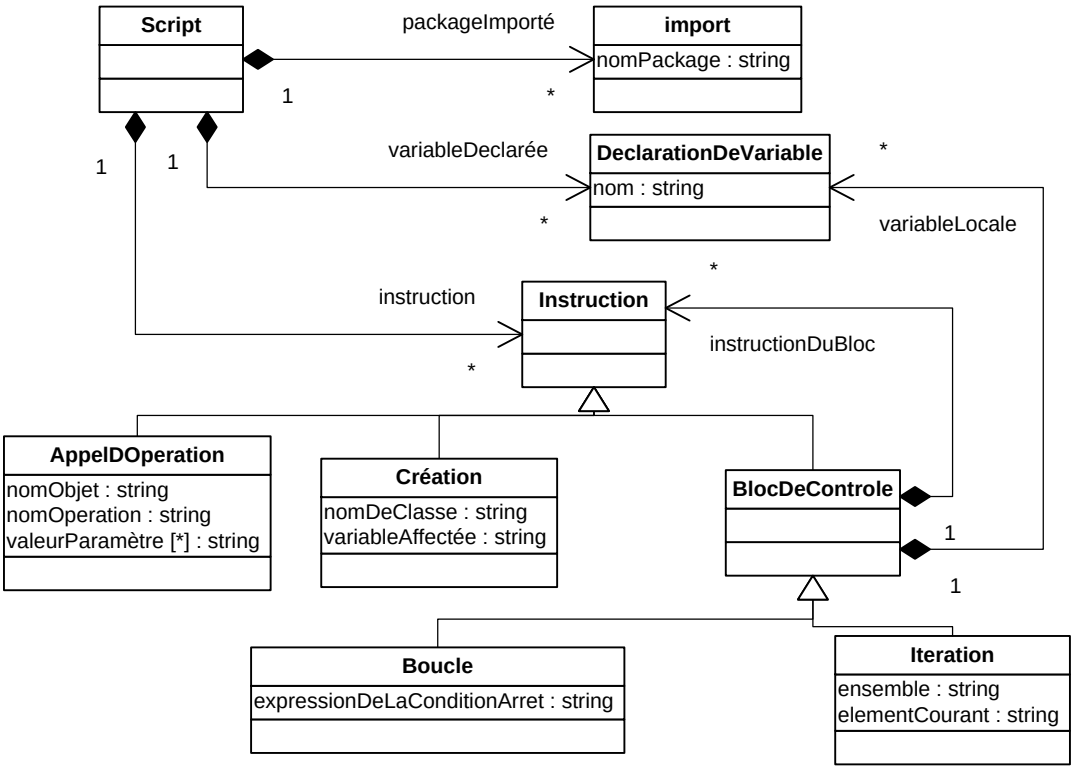


Figure 3 : métamodèle de script

Un script contient :

- Plusieurs importations de package. Les classes de ces packages seront utilisées par le script.
- Plusieurs déclarations de variables (le typage est dynamique)
- Plusieurs instructions ordonnées

Une instruction peut être :

- un appel d'opération sur un objet
  - L'objet appelé doit être référencé par une variable,
  - L'opération doit appartenir à la classe de l'objet (la classe doit appartenir à un package importé, cette vérification ne se fait pas grâce au métamodèle)
  - Les paramètres de l'appel peuvent être des variables ou des valeurs littérales (booléen, entier, chaîne de caractères, …)
- une création d'objet
  - La classe de l'objet doit appartenir à un package importé
  - Après création l'objet est référencé par une variable
- Un Bloc de contrôle qui contient lui-même des déclarations de variables locales et des instructions.
  - Une boucle répète les instructions jusqu'à ce que l'expression d'arrêt soit vraie
  - Une itération répète les instructions pour tous les éléments d'un ensemble. L'ensemble est identifié par la variable « ensemble » et, pour chaque itération, l'élément courant est identifié par la variable « elementCourant ».

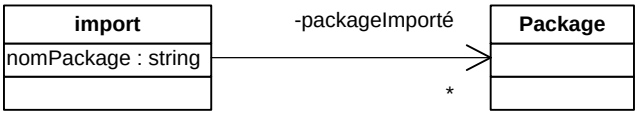
Q3 : Précisez, pour chaque ligne du script suivant (exprimé en pseudo langage) le lien avec le métamodèle (comme cela est fait pour les première et deuxième lignes). On considérera que le package IO contient la classe File offrant les opérations de manipulation de fichier open et write.

Script ; // métaclasse Script
packageImporté IO ; //métaclasse Import, nomPackage=IO
variableDeclarée f ;
f = creation(File) ;
f.open() ;
f.write("Bravo");

Réponse :
Script ; // métaclasse Script
packageImporté IO ; //métaclasse Import, nomPackage=IO
variableDeclarée f ; //métaclasse DeclarationDeVariable, nom=f
f = creation(File) ; //metaclasse Creation, nomDeClasse=File, variableAffectée=f
f.open() ; //métaclasse AppelDOperation, nomObjet=f, nomOperation=open
f.write("Bravo"); //métaclasse AppelDOperation, nomObjet=f, nomOperation=write, valeurParamètre=« bravo »

Q4 : Modifiez le métamodèle de script pour faire en sorte que les packages importé soient uniquement des packages UML (instance de la métaclasse Package du métamodèle UML)

Réponse :
Il existe plusieurs réponses à cette question. Une réponse simple consiste à associer la metaclasse import à la classe UML::Package

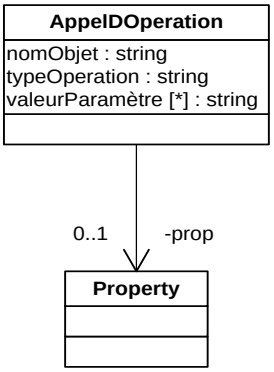


L'attribut nomPackage ne sert alors à rien.

Quelque soit la solution, il faut établir un lien entre l'import et le package UML.

Q5 : Modifiez le métamodèle de script pour faire en sorte que les appels d'opération ne puissent être que des « getter » ou des « setter » sur les attributs des classes UML contenues dans les packages importés. Pour les questions suivantes, nous considérerons que le métamodèle de script contient vos modifications.

Réponse :
Il y a encore plusieurs solutions.
Néanmoins, il faut voir apparaître les deux types d'opération (getter et setter) et voir apparaître le lien vers la métaclasse UML::Property (et non pas UML::Operation)



Transformation (8pts)

Le métamodèle suivant (Figure 4) permet de modéliser des transformations de modèles. Les transformations modélisées sont des transformations binaires (une source et une cible) et monodirectionnelle (de la source vers la cible). Une transformation est modélisée par un arbre de règles de correspondance de classes (avec une racine).

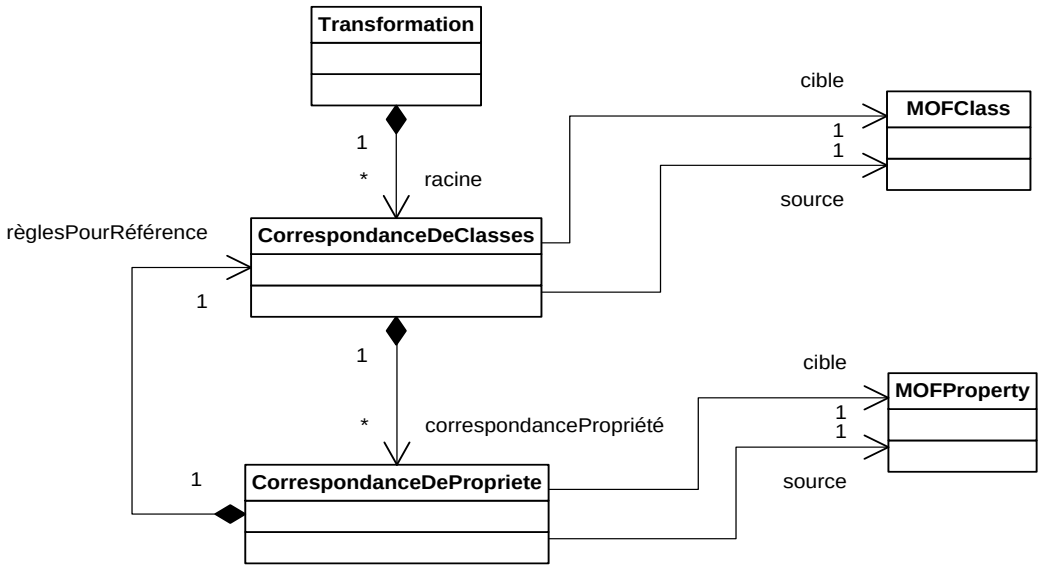


Figure 4 : métamodèle de transformation de modèles

Chaque règle de correspondance de classes a pour cible et pour source une seule métaclasse d'un métamodèle existant (d'où la métaclasse MOFClass dans le métamodèle).

Une règle de correspondance peut contenir plusieurs règles de correspondance de propriétés. Si le type de la propriété est primitif alors la valeur de la propriété cible est égale à la valeur de la propriété source. Si le type de la propriété est une métaclasse MOF alors la règle pour la référence est appelée pour remplir la valeur de la propriété cible.

En considérant le métamodèle UML et le métamodèle de base de données suivant (Figure 5), nous pouvons alors modéliser la transformation de modèles UML vers modèles de base de données.

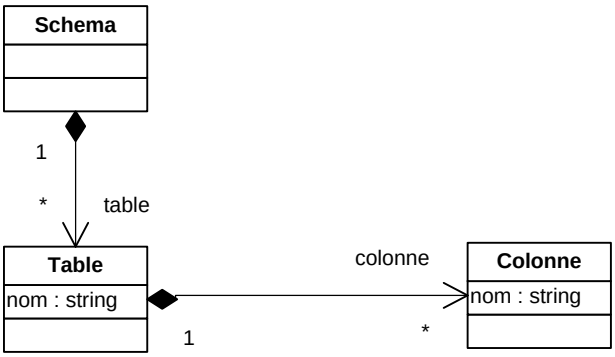


Figure 5 : métamodèle de base de données

La transformation est alors la suivante :

Transformation

racine (source=Package) (cible=Schema)
  correspondacePropriété (source=elementContenu) (cible=table)
    règlePourRéférence (source=Class) (cible=Table)
      correspondacePropriété (source=nom) (cible=nom)
      correspondacePropriété (source=attribut) (cible=colonne)
        règlePourRéférence (source=Property) (cible=Colonne)
          correspondacePropriété (source=nom) (cible=nom)

Q6 : Décrivez en quelques mots ce que fait cette transformation. Précisez de plus, pour chaque ligne, le lien avec le métamodèle de transformation.

Réponse :
Cette transformation construit un schéma à partir d'un package.
Pour toutes les elementContenu du package, une table est ajoutée au schéma
Si l'élément contenu est une classe, il faut alors créer une table
Le nom de la table correspond au nom de la classe
Pour tous les attributs de la classe, une colonne est ajoutée
Si l'attribut est une propriété, il faut alors créer une colonne
Le nom de la colonne correspond au nom de la propriété

Q7 : Modélisez la transformation précédente sous forme d'un modèle de script instance du métamodèle de Script.

Réponse :
Ca commence à être difficile.

Il faut voir apparaître l'itération sur les éléments contenus du packages (10%)
Il faut voir apparaître une création d'une classe pour chaque table (10%)
Il faut voir apparaître l'affectation du nom de la table (10%)
Il faut voir apparaître l'itération sur les attributs de la classe (10%)
Il faut voir apparaître une création de colonne pour chaque propriété (10%)
Il faut voir apparaître l'affectation du nom de la propriété (10%)
Il faut bien voir apparaître les liens vers les éléments du métamodèle
Import (10%)

Creation (10%)
Appel d'opérations pour naviguer (10%)
En principe, il manque les variables d'entrée au script, il faut donc les rajouter (10%)

Q8 : Proposez une façon de transformer automatiquement un modèle de transformation vers un modèle de script (si vous avez besoin de modifier les métamodèles de script ou de transformation précisez vos modifications).

Réponse :
Ca commence à être vraiment difficile.

Rule 1 : Pour une transformation il faut construire un script.
Rule 2 : Ce script importe les packages correspondant aux méta-modèles.
Rule 3 : Pour chaque règle de correspondance de classe, ce script doit disposer d'une opération permettant de lire l'élément source de la correspondance et doit disposer d'une opération permettant de construire l'élément cible de la correspondance.
Rule 4 : Pour chaque règles de correspondance de propriété, ce script doit disposer d'une opération permettant de lire l'élément source de la correspondance et doit disposer d'une opération permettant d'écrire l'élément cible de la correspondance. Si la correspondance de propriété est une correspondance de classe, il faut alors appeler la Rule 3.

Q9 : Proposez un moyen de modéliser la transformation que vous avez proposée en question 8. Pour ce faire, vous devez élaborer un modèle instance du métamodèle de transformation. Quel est l'intérêt d'un tel modèle ?

Réponse :
C'est quasiment impossible à faire.

Il faudrait exprimer les Rule1 à 4 de la question 8 sous forme d'un modèle instance du méta-modèle de transformation de modèle.