**L'ingénierie des langages de modélisation :**

L'ingénierie des langages de modélisation dans MDE comprend au moins deux composants [24].

Premièrement, la syntaxe abstraite d'un langage doit être définie par un métamodèle, c'est-à-dire un modèle définissant la grammaire du langage. Deuxièmement, pour rendre un langage plus utilisable, une mise en correspondance des éléments de syntaxe abstraite avec des éléments de syntaxe concrets (tels que des rectangles, des arêtes et des étiquettes) doit être fournie. Dans ce qui suit, une description basée sur des exemples de la définition d'un langage de modélisation est donnée.



**Association**

assoc 0..1

role

\*

type

0..1

attribute 0..1 class

\*

**(a)**

**(b)**

p1:Property

name:”name” lower: 1

upper: 1 type: String

name:”examines”

name:”name” lower: 1

upper: 1 type: String

c1:Class name:”examiner” lower: 1

name:”Professor” upper: 1

isAbstract: false type: Undefined

name:”examinee” lower: 1

upper: -1

type: Undefined

c2:Class

name:”Student” isAbstract: false

isAbstract:Boolean

**Class**

type:String lower:Integer upper:Integer

**Property**

name:String

**NamedElement**

p4:Property

p3:Property

examines:Assoc

p2:Property

Fig. 2. (Meta)modeling: (a) UML Metamodel Kernel, (b) Example Model in AS, and  
(c) Example Model in CS

**Syntaxe abstraite**. Semblables aux grammaires basées sur EBNF [18] pour les langages de programmation, les métamodèles représentent les concepts et leurs interrelations des langages de modélisation. Le formalisme le plus largement utilisé pour définir des métamodèles dans MDE est le Meta Object Facility [33] (MOF), qui est un langage standardisé pour définir des langages de modélisation basés sur les concepts de base des diagrammes de classes UML (classes, attributs et références). Dans la partie supérieure de la Fig. 2, un extrait du noyau du métamodèle UML est représenté en termes de concepts MOF.

L'objectif de la métamodélisation réside principalement dans la définition des langages de modélisation d'une manière orientée objet, conduisant également à des implémentations de référentiel efficaces pour stocker et récupérer des modèles. Cela signifie que dans un métamodèle, tous les concepts de modélisation ne sont pas nécessairement représentés comme des citoyens de première classe. Au lieu de cela, les concepts sont souvent cachés dans des attributs et des références. Nous appelons ce concept de phénomène masquage (cf. [20] pour une discussion approfondie).

En instanciant des métamodèles, des modèles sont créés. Une instanciation est représentée par un diagramme d'objets UML comprenant des objets en tant qu'instances de classes, des valeurs en tant qu'instances d'attributs et des liens en tant qu'instances de références, comme par exemple illustré dans la partie centrale de la Fig. 2. Il convient de noter qu'en revanche aux métamodèles de grammaires basés sur EBNF ne définissent pas la syntaxe concrète des langages. Ainsi, seuls des graphes d'objets génériques tels que représentés dans la partie médiane de la figure 2 peuvent être créés. La syntaxe concrète doit être définie en plus du métamodèle qui est expliqué ensuite.

**Syntaxe concrète.** La syntaxe concrète des langages de modélisation [2] comprend dans la plupart des cas des éléments graphiques tels que l'ellipse, l'étiquette et le rectangle, qui peuvent ensuite être combinés à des formes plus complexes. La notation proprement dite des concepts de modélisation est définie par une mise en correspondance d'éléments de syntaxe abstraite avec des éléments de syntaxe concrets. Les mappages peuvent être exprimés en triplets de la forme suivante :

**T riple :=< as\_E, cs\_E, const(as\_E)? > (1)**

La première partie comme as\_E représente un élément de la syntaxe abstraite, la seconde cs\_E pour un élément de la syntaxe concrète, et la dernière const(as\_E) représente une contrainte optionnelle, principalement définie dans le langage Object Constraint (OCL) [36], qui définit sous quelles conditions, c'est-à-dire les liens et les valeurs d'attribut d'un élément as\_E, cet élément est représenté par un élément cs\_E. Dans le cas où aucune contrainte n'est définie, il existe une correspondance un à un entre un élément de syntaxe abstraite et un élément de syntaxe concrète, c'est-à-dire que le concept défini dans le métamodèle est directement représenté par un élément de notation concret. Cependant, l'autre cas est le plus intéressant dans le contexte du MTBE. La présence d'une contrainte définit un nouveau concept pour la couche de notation, qui n'est pas représenté explicitement par l'une des classes du métamodèle. Par conséquent, lors de la définition de transformations de modèles basées sur la syntaxe abstraite, les contraintes de ces concepts doivent être définies par l'utilisateur. Il s'agit d'une tâche fastidieuse et sujette aux erreurs qui nécessite une excellente connaissance du métamodèle.

Lorsque nous considérons notre exemple en cours d'exécution, par exemple, le concept Class est mappé sur Rectangle et Class.name est mappé sur Label. En utilisant de tels mappages, le diagramme d'objets UML représenté dans la partie médiane de la figure 2 peut être rendu comme représenté dans la partie inférieure de la figure 2 par des éditeurs de modélisation graphique.

**Langages de transformation de modèles**

Diverses approches de transformation de modèles ont été proposées au cours de la dernière décennie, principalement basées soit sur un mélange de concepts déclaratifs et impératifs, tels que ATL [19], ETL [25] et RubyTL [12], soit sur des transformations de graphes, telles que AGG [44] et Fujaba [32], ou sur des relations, comme MTF2 et TGG [1]. De plus, l'Object Management Group (OMG) a publié la norme de transformation de modèle QVT [35] qui n'est actuellement que partiellement adoptée par l'industrie. En résumé, toutes les approches décrivent des transformations de modèle par des règles utilisant des éléments de métamodèle, tandis que les règles sont exécutées sur la couche de modèle pour transformer un modèle source en un modèle cible.

Les règles comprennent des in-patterns et des out-patterns. Le in-patterns définit quand une règle est réellement applicable c.a.d récupère les éléments nécessaires du modèle d'entrée pour calculer le résultat de la règle. Le out-patterns décrits l'effet d'une règle, par exemple quels éléments sont créés, mis à jour et supprimés. Toutes les approches mentionnées sont basées sur la syntaxe abstraite des langages de modélisation uniquement, et la notation du langage de modélisation est totalement négligée.

Définir des transformations de modèle en utilisant la syntaxe abstraite des langages de modélisation présente d'une part l'avantage de l'applicabilité générique. D'autre part, la création de telles transformations est souvent compliquée et leur lisibilité est beaucoup plus faible par rapport au travail avec la syntaxe concrète [3,28,41,45,46]. Par conséquent, des approches MTBE ont été proposées pour utiliser la syntaxe concrète des langages de modélisation pour définir des transformations de modèle.