Model Driven Engineering : projet SmallUML

Léo CASSIAU Geoffrey DESBROSSES

16 décembre 2016

1 Introduction

Ce document résume le travail effectué au cours du projet de Service de 2e année du master ALMA 2016 - 2017. Ce projet est mené par des étudiants de la faculté des sciences et techniques de Nantes, et le sujet est proposé par M. Gerson [1].

L'objectif de ce projet est de comprendre comment fonctionne la génération de modèles et les techniques de modélisation et de génération de code à travers plusieurs outils. Pour cela, nous avons dû apprendre à utiliser des outils comme EMF, Xtext ou encore ATL pour ensuite aborder le sujet. A partir d'un métamodèle UML qui nous a été fournis, nous avons dû créer un métamodèle smallUML, qui reprend les principes de base de UML, pour ensuite pouvoir faire une transformation smallUML vers UML.

Ce document commence par expliquer comment nous avons créer le métamodèle smallUML, puis expliquer notre grammaire écrite grâce à Xtext, pour enfin expliquer notre transformation ATL smallUML2UML.

2 Travail effectué

2.1 Construction d'un métamodèle

La première partie de ce projet a été de créer le métamodèle de smallUML grâce à l'outil EMF. Nous avons essayé de nous rapprocher le plus possible du métamodèle UML pour les principes sélectionnés, pour que la future transformation soit plus simple et homogène.

Une classe abstraite principal appelé NamedElement permet d'identifier toutes les entités par un nom et une classe Root qui possède une liste de classe et de types primitifs permet de servir de racine aux modèles. Nous avons repris les types primitifs d'UML : string, real, integer, boolean et unlimitedNatural. Ces cinq types ont une classe mère Type, qui est également pour fils la classe (EClass) Class. Ainsi, une opération a un type de retour qui peut être soit un type primitif soit une classe définit dans le modèle.

La classe Class a quatre attributs : son nom, l'attribut superClass qui permet de spécifier une ou plusieurs classe mère, l'attribut ownedOperations est la

1. http ://sunye.free.fr/

liste d'opérations de la classe et enfin l'attribut ownedProperties est la liste des propriétés (attributs) de la classe.

Une opération a trois attributs : son nom, resultType qui est le type de retour de l'opération et params qui contient les types des paramètres de la classe. Une propriété quand à elle a quatre attributs : son nom, son type, et la cardinalité représenté par la borne inférieur et supérieur.

2.2 Ecriture d'une grammaire

Notre métamodèle smallUML étant créé, il nous a fallu écrire la grammaire qui nous permet de créer un modèle à partir de notre métamodèle. Pour celà nous avons utilisé l'outil Xtext. Nous avons choisi une grammaire qui se rapproche de l'écriture d'une interface en Java ou en C++ par exemple. Un exemple de modèle de parking ci-dessous :

```
primitiveType Boolean
primitiveType Integer
primitiveType String

Class Parking {
        properties {
                0..50 Vehicule vehiculesGares
        }
        operations {
                nbVehiculesGares() : ^Integer
        }
}
Class Vehicule {
        properties {
                1..1 ^String modele
                1..1 ^String couleur
        }
        operations {
                changerConducteur(^String)
                sonConducteur() : ^String
                demarrer() : ^Boolean
                nouveauPassager(^String) : ^Boolean
                nouveauPassager(^String, ^String) : ^Boolean
        }
}
Class Voiture : Vehicule {
        properties {
                1..1 ^String immatriculation
        }
        operations {
                demarrer() : ^Boolean
                nouveauPassager(^String) : ^Boolean
                nouveauPassager(^String, ^String) : ^Boolean
        }
}
```

2.3 Transformation vers UML

A partir de ce point, nous avons nos deux entrées : le métamodèle smallUML et un modèle respectant notre grammaire. Nous avons également le métamodèle de sortie (UML) qui nous a été fournit. Il nous a donc fallu écrire nos règles de transformation afin d'obtenir un modèle UML à partir de notre modèle smallUML. Pour cela, nous avons utilisé l'outil ATL.

La plupart des éléments de notre méta-modèle SmallUML avaient une correspondance assez claire avec les éléments d'UML, comme les types. L'élément Root devient l'élément Package, les classes, opérations et paramètres ont des correspondances direct. Le type de retour d'une opération a posé problème, mais après plusieurs recherches il s'avère qu'en UML le type de retour est en faites un paramètre avec une direction particulière. Les autres attributs avaient une correspondance direct.

3 Conclusion

Nous avons réussi à obtenir un modèle UML à partir de notre modèle d'entrée. Ce modèle de sortie respecte bien la syntaxe et la sémantique définies dans le métamodèle UML qui nous a été fournit. La transformation a donc fonctionnée correctement.

Lors de ce projet, nous avons vu les principes de la génération et de la transformation de modèles. Il existe beaucoup d'outils permettant ce genre de chose, ici nous n'avons vu qu'une petite partie.

Il est assez simple d'écrire des règles pour générer un modèle à partir de son métamodèle ou encore de transformer un modèle en un autre. Ces règles sont assez longues à écrire pour que la génération se fasse sans erreur et que les modèles respectent leurs métamodèles.

Cependant, une fois toutes les règles en place, il est plus aisé d'écrire un programme ou encore de migrer d'une technologie à une autre. Certaines sociétés ont compris les avantages de la génération à partir de modèles et ont développé leurs propres outils tels que Mia Studio.