Rapport Projet SmallUML
Model Driven Engineering

Thomas Minier - Alicia Boucard
Master 2 ALMA

15 décembre 2016

Table des matières

1 Introduction

SmallUML est un langage de modélisation dont l'objectif est de permettre la modélisation du domaine métier d'un système d'information.

Il s'agit d'une version simplifiée du diagramme de classes UML, débarrassée des certains concepts liés à la conception de logiciels, comme la visibilité, les interfaces, les signaux, classes paramétrées, agrégations, dépendances, ...

Dans ce projet, il nous a été demandé de spécifier et d'outiller le langage de modélisation SmallUML. La spécification du langage sera réalisée en trois parties. Dans la première, la syntaxe abstraite du langage sera spécifiée en EMF. Dans la deuxième partie, la syntaxe concrète sera spécifiée en Xtext. La troisième partie, la sémantique du langage, sera spécifiée en se basant sur UML : les concepts de SmallUML seront traduits en des concepts UML grâce au langage de transformation ATL.

Dans ce rapport, nous détaillerons les trois étapes du projet dans l'ordre. Tout le code du projet est disponible sur Github à l'adresse suivante : `https://github.com/Liciax/PatouneSmallUML`.

2 Création du métamodèle

Nous avons modélisé le métamodèle de SmallUML avec le framework EMF en utilisant le format Ecore. Notre métamodèle, représenté par la figure 1, est constitué des éléments suivants :

- **NamedElement** : un élément nommé est un élément abstrait possédant un nom.
- **DiagramEntity** : une entité de diagramme représente n'importe quelle entité dans un diagramme SmallUML.
- **AbstractEntity** : une entité abstraite représente n'importe quelle entité pouvant être utilisée comme type par d'autres entités.
- **Diagram** : un diagramme est l'élément principal d'un diagramme SmallUML qui contient tous les éléments du diagramme.
- **Attribute** : un attribut représente un attribut d'une classe.
- **Parameter** : un paramètre représente un paramètre d'une méthode.
- **Type** : un type représente un type simple (exemples : int, boolean, ...)
- **Enumeration** : une énumération représente une énumération qui est un type avec un nombre fini de valeurs.
- **Class** : une classe représente une entité possédant des attributs et des opérations et pouvant être impliquée dans une ou plusieurs associa-

tions.

— **Operation** : une opération représente une méthode spécifiée par son nom, ses paramètres et un type de retour.
— **Association** : une association représente une relation entre au moins deux rôles.
— **Role** : un rôle représente le rôle d'une entité dans une relation.
— **Cardinality** : une cardinalité représente la borne inférieure et supérieure d'une relation.

Nous avons également ajouté de la documentation pour chaque concept de notre métamodèle avec l'annotation `http://www.eclipse.org/emf/2002/GenModel`. Cette fonctionnalité nous permet d'avoir un modèle documenté et la documentation sera présente dans le code généré par EMF.

3 Création d'un langage de modélisation avec XText

Nous avons utilisé XText pour définir une syntaxe concrète pour notre métamodèle. Cela nous permet de créer un langage qui permet d'exprimer naturellement des diagrammes SmallUML. Le listing 6 montre un exemple d'utilisation de notre syntaxe.

Pour générer la grammaire XText, nous avons commencé par la générer automatiquement depuis le modèle Ecore puis nous l'avons retravaillé pour l'adapter à nos spécifications et la rendre plus simple d'utilisation.

4 Transformation de SmallUML vers UML avec ATL

Pour effectuer la transformation, nous avons utilisé comme métamodèle cible celui de UML tel que fourni par la fondation Eclipse (`http://www.eclipse.org/uml2/5.0.0/UML`), plutôt que celui fourni avec le sujet. Ce choix se justifie par le fait que le métamodèle fournit présente de nombreux bugs et défauts qui nous empêchent de mener à bien la transformation. Nous avons choisi un métamodèle alternatif qui ne présente pas ces défauts.

Nous avons défini la correspondance SmallUML vers UML pour tous les éléments de notre métamodèle. La plupart des concepts (Class, Attribute, Operation, ...) sont les mêmes entre les deux métamodèles et se transforment assez facilement. Pour les relations entre les classes (associations et héritage),

nous avons utilisé les entités Association et Generalization de UML, qui ont nécessité un peu plus de travail.

Nous fournissons un exemple d'instance au format XMI, le fichier *SmallUMLExample.xmi* dans le dossier source de *SmallToUML*, qui peut être transformé par ATL de SmallUML vers UML.

5 Conclusion

En conclusion, ce projet nous a permis de mettre en pratique certaines des technologies liées au Model Driven Engineering que vous avons vu en cours. Nous avons pu voir qu'une fois le métamodèle défini, il est assez simple de le manipuler via les différents outils de l'écosystème Eclipse, qui s'intègrent relativement bien entre eux. Néanmoins, nous avons aussi pu constater que les outils générant du code sont assez sensibles et doivent être utilisés correctement.

6 Annexes

Listing 1 – Exemple d'utilisation de notre langage pour créer un diagramme SmallUML

```
Diagram OneDirection {
        Type String;
        Type Integer;
        Type Boolean;

        Enumeration InstrumentFamily {
                Wind, Percussion, Electronic
        }

        abstract Class Member {
                attributes(name: String, firstName: String)
        }

        Class Instrument {
                attributes(name : String, family : InstrumentFamily)
        }
```

```
        Class Singer extends Member {
                operations(String sing (String : musicSheet),
                        String twitterDrama())
        }

        Class Musician extends Member {
                attributes(instrument : Instrument)
                operations(
                    String play (Integer : soundVolume,
                        String : musicSheet)
                )
        }

        Class PurityRing {
                attributes(violated : Boolean)
        }

        Association SacredOath {
                role member ['1','1'] with Member,
                role ring ['1','1'] with PurityRing
        }

        Association MusicBand {
                role singer ['1','1'] with Singer,
                role musicians ['2','*'] with Musician
        }
}
```

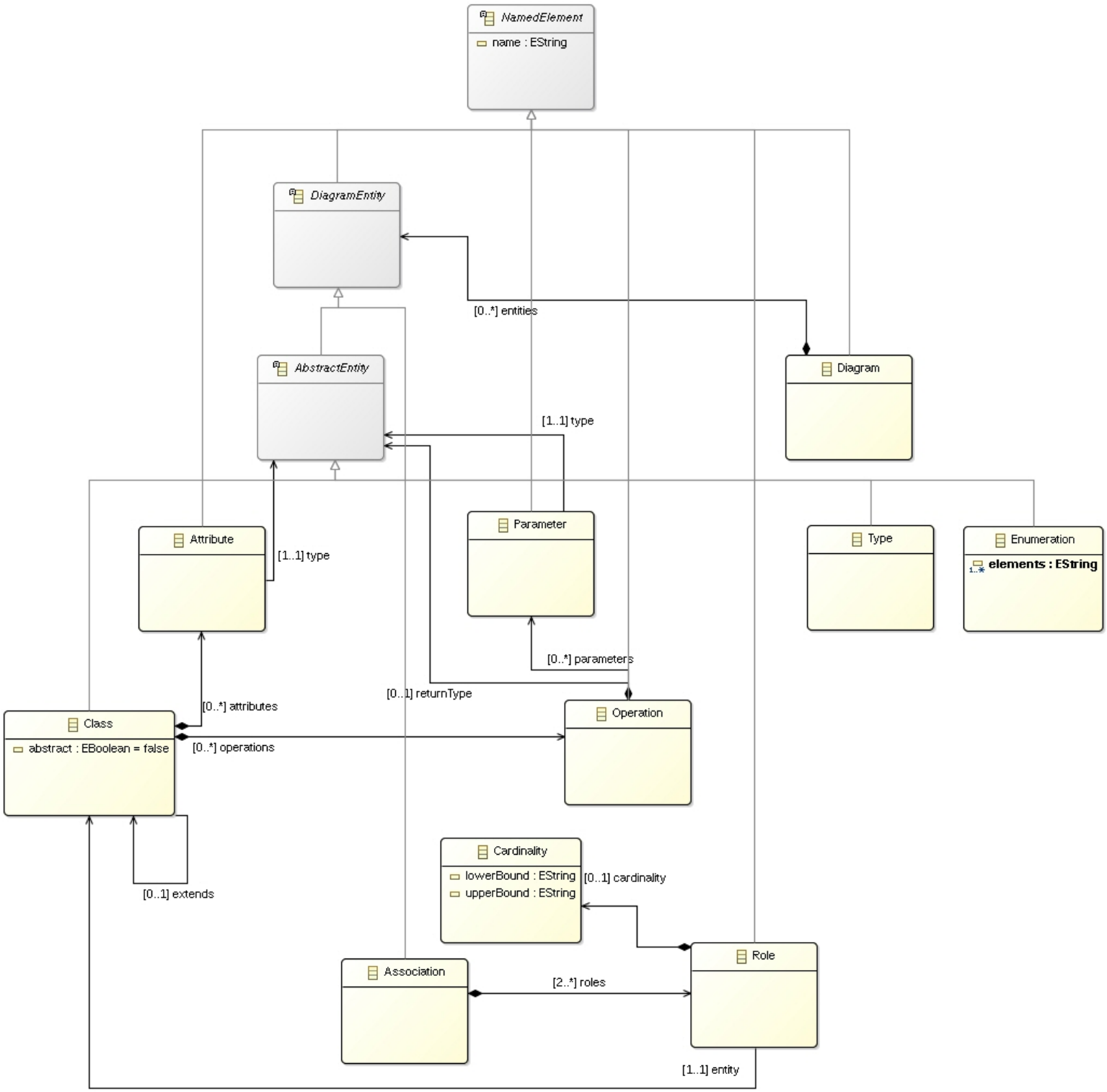

FIGURE 1 – Métamodèle crée pour le langage de modélisation SMALLUML