Rapport du projet SmallUMl

Réalisé par

Hiba BENYAHIA, Abdeldjallil FERCHOUHE

Table des matières

Table des figures

I. 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, etc.

Concepts UML à retenir : Classes, Associations, Généralisation/Spécialisation, Attributs, Opérations, DataTypes, Énumérations, Multiplicités.

II. Objectif

L'objectif de ce projet est 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 spécifié en se basant sur UML 2.5: les concepts de SmallUML seront traduits en des concepts UML grâce au langage de transformation ATL.

III. Plan du travail

Nous avons commencé le travail par la réalisation du Méta Model qui correspond au langage SmallUML, par la suite nous avons spécifier la syntaxe concerte de ce langage en se basant sur le fichier Ecore du Framework EMF, puis après nous avons généré la syntaxe concrète du ce langage à partir du fichier Ecore créer dans la phase précédente, et on a terminé le travail par faire la transformation du modèle avec le Framework ATL.

1. Syntaxe abstraite

La figure ci-dessous décrit le Méta Model utilisé pour le langage SmallUML.

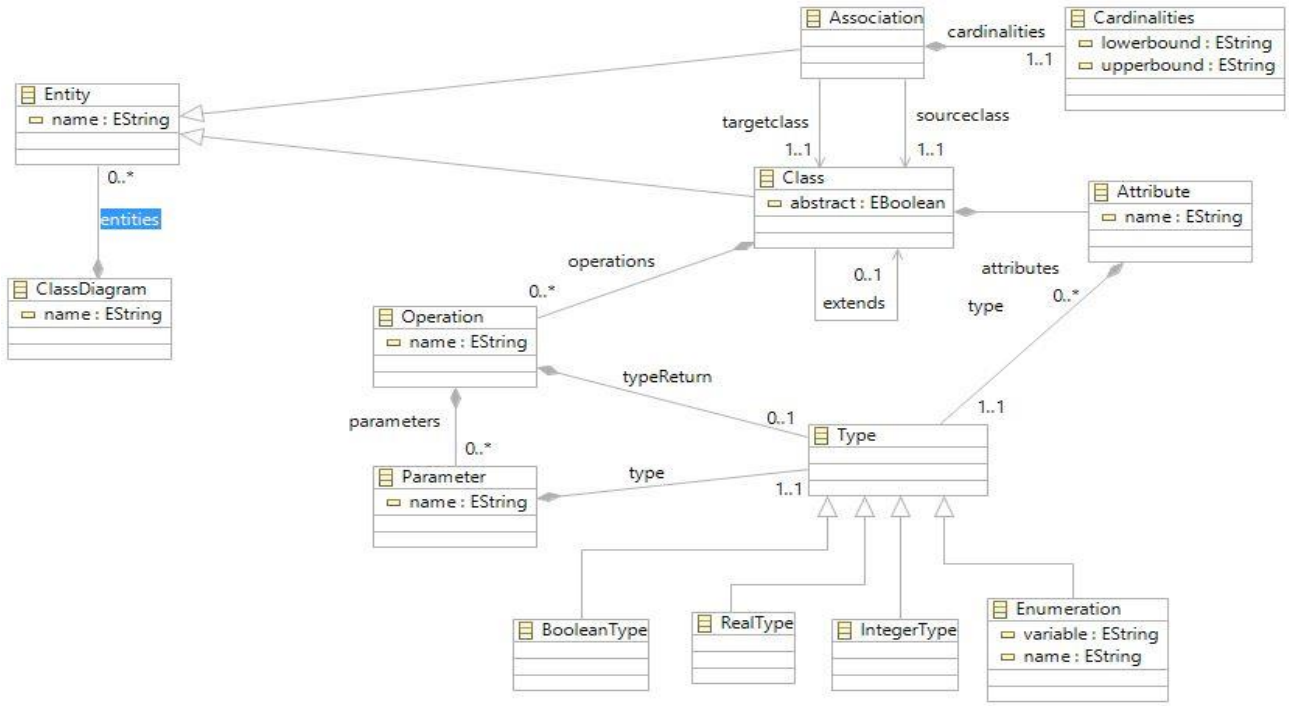

Figure III-1: MétaModel SmallUML

Par la suite nous avons généré notre fichier Ecore à partir du ce modèle afin de spécifier la syntaxe abstraite du langage.

2. Syntaxe concrète

Après avoir créé notre syntaxe abstrait du langage SmallUML par EMF, nous avons passé à l'étape de la création du syntaxe concrète du langage, la grammaire a été générée dans un premier temps à partir du fichier EMF qui contient la syntaxe abstrait, puis nous avons introduit quelques modifications sur la grammaire générée afin qu'elle corresponde exactement à la syntaxe abstraite créer dans la première phase du projet.

Le fichier Xtext qui contient la grammaire est illustré dans la figure suivante.

```
ClassDiagram returns ClassDiagram:
    {ClassDiagram}
    'ClassDiagram'
    name=EString
    '{'
        (entities+=Entity)*
    '}'
    ;
Entity returns Entity:
    Class | Association
    ;
EString returns ecore::EString:
    STRING | ID
    ;
Class returns Class:
    {Class}
    (abstract?='abstract')?
    'Class'
    name=EString ('extends' extends=[Class|EString])?
    '{'
        ('attributes' '(' attributes+=Attribute ( "," attributes+=Attribute)* ')' )?
        ('operations' '(' operations+=Operation ( "," operations+=Operation)* ')' )?
    '}'
    ;
Association returns Association:
    {Association}
    'Association'
    name=EString
    '{'
        'cardinalities' cardinalities=Cardinalities ','
        'targetclass' targetclass=[Class|EString] ','
        'sourceclass' sourceclass=[Class|EString]
    '}'
    ;
```

```
Attribute returns Attribute:
    {Attribute}
    name=EString ':' type=Type
    ;
Operation returns Operation:
    {Operation}
    (typeReturn=Type)? name=EString
        '(' (parameters+=Parameter)?( "," parameters+=Parameter)* ')'
    ;
Type returns Type:
    BooleanType | RealType | IntegerType | Enumeration
    ;
BooleanType returns BooleanType:
    {BooleanType}
    'Boolean'
    ;

RealType returns RealType:
    {RealType}
    'Real'
    ;

IntegerType returns IntegerType:
    {IntegerType}
    'Integer'
    ;

Enumeration returns Enumeration:
    {Enumeration}
    'Enumeration'
    name=EString
    '{'
        variable+=EString ( "," variable+=EString)*
    '}'
    ;

Parameter returns Parameter:
    {Parameter}
    name=EString ':' type=Type
    ;

Cardinalities returns Cardinalities:
    {Cardinalities}
    '{'
        lowerbound=EInt ','
        upperbound=EInt
    '}'
    ;

EInt returns ecore::EInt:
        '-'?INT
;
```

Figure III-2 Grammaire SmallUMl avec Xtext

3. Transformation ATL

Avant de faire la transformation du modèle créer, nous avons préparé les données qui seront utilisées par ATL pour faire la transformation, nous avons commencé par créer un exemple du diagramme de classe basé sur la grammaire Xtext (syntaxe concrète).

Puis dans une seconde étape nous avons converti ce fichier vers un fichier XMI qui sera donné en entré pour ATL et il sera par la suite transformer du SmallUML vers UML, pour faire cette conversion nous avons créé un programme (cette classe est dans le dossier SmallUML\alma.mde.project\src\alma\mde\project\ConvertToXMI.java).

Exemple utilisé pour la transformation est illustré dans la figure ci-dessous.

```
ClassDiagram new {

    Class Feuille {
        attributes (premier : Boolean, deuxieme : Real)
    }

    Class Livre extends LivreAbst {
        attributes (troisieme : Integer)
        operations (
            Integer oper1(par1 : Boolean, par2 : Enumeration gt {v, r, b}),
            Integer oper2(par1 : Boolean, par2 : Integer)
                    )
    }
    abstract Class LivreAbst {
        attributes (abstr1 : Boolean, abstr2 : Real)
    }

    Association livrefeuille{
        cardinalities {1, 1},
        targetclass Livre,
        sourceclass Feuille
    }
}
```

Figure III-3 Exemple utilisé pour tester la transformation

Les différentes règles utilisées dans la transformation sont définies dans le fichier Small2UML.atl sont illustrées dans la figure ci-dessous

```
-- @path MM=/SmallUML/model/smalluml.ecore
-- @path MM1=/UML/model/uml.ecore
module Small2UML;
create OUT: MM1 from IN: MM;

rule RClass {
    from
        e: MM!Class
    to
        out: MM1!Class (
            name <- e.name,
            superClass <- e.extends,
            isAbstract <- e.abstract,
            ownedAttribute <- e.attributes->collect(iterator | iterator),
            ownedOperation <- e.operations
        )
}

rule RAttribute {
    from
        e : MM!Attribute
    to
        out : MM1!Property (
            name <- e.name,
            type <- e.type
        )
}

rule RParameter {
    from
        e : MM!Parameter
    to
        out : MM1!Parameter (
            name <- e.name,
            type <- e.type
            )
}


rule ROperation {
    from
        e : MM!Operation
    to
        out : MM1!Operation (
            name <- e.name,
            type <- e.typeReturn,
            ownedParameter <- e.parameters
        )
}

rule RCardinalities {
    from
        e : MM!Cardinalities
    to
        out : MM1!Property (
            lower <- e.lowerbound,
            upper <- e.upperbound
        )
}

rule REnumeration {
    from
        e : MM!Enumeration
    to
        out : MM1!Enumeration (
            name <- e.name
        )
}
```

Figure III-4 Les règles ATL