Ebrard Alexis et Ben Mustapha Khaled
GIS 5 POLYTECH LILLE

30/12/2017

Rapport détaillé d'Ingénierie Logicielle

Tuteur Pédagogue :

Monsieur Olivier Caron

Table des matières

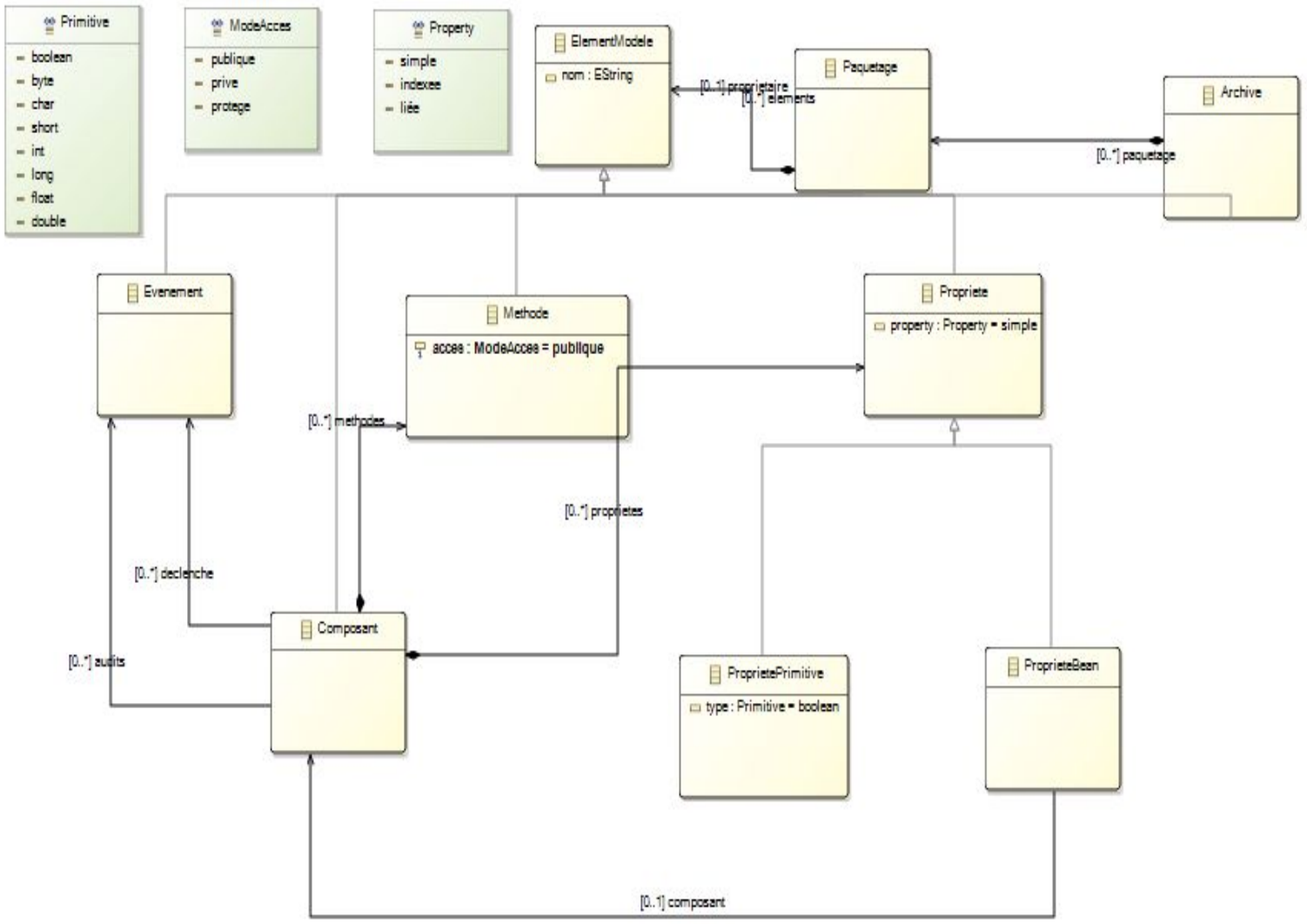
Introduction

Dans le cadre de la formation Génie Informatique et Statistique de l'école POLYTECH LILLE, il est demandé en cinquième année de réaliser un projet d'ingénierie logicielle.
Afin de réaliser ce projet et dans le but de maîtriser certains concepts nous sommes amenés à étudier la Méta-modélisation, les designs patterns de développement ainsi que les modèles EMF.
L'objectif est ici de créer un projet EMF capable de spécifier les méta-modèles dans le but de générer des composants Java Beans.

A travers ce rapport nous aborderons différents points tels que l'analyse du méta-modèle, les contraintes mises en place mais également l'utilisation des designs patterns.

Méta-Modèle

Description du schéma



Sur ce schéma on observe trois classes d'énumérations, une classe d'énumération « ModeAcces » énumérant le type d'accès à une méthode, une classe « Property » énumérant le type de propriété et enfin une classe « Primitive » énumérant le type de propriété primitive.

On dénombre un total de 9 classes, 5 d'entres elles héritent de la classe ElementModel et 2 autres héritent de la classe Propriété (elle-même héritant de ElementModel).
Ainsi toutes les classes héritant de la classe abstraite ElementModel sont caractérisées par un Nom. En effet, la propriété est héritée car ElementModel possède Nom de type EString. Ce qui nous permet d'éviter d'attribuer manuellement à chaque classe un attribut nom qui lui est propre.

Ici, la seule classe n'héritant pas d'ElementModele est la classe Paquetage.

Les différentes liaisons entre les classes :

- Une Archive est composé d'un ou plusieurs Paquetages. (Composition)
- Les Paquetages sont composés d'ElementModeles. (Composition)
- Les ElementModeles peuvent se décliner sous la forme de Méthodes, Propriétés, Composants, Événements ou encore Archive. (Héritage)
- Les propriétés peuvent être de type ProprieteBean ou encore de type ProprietePrimitive. (Héritage)
- Un composant est composé de Méthodes ou de Propriétés et peut déclencher ou auditer des Evenements. (Composition)
- Une ProprieteBean peut être un composant. (Association)

TP 2 : PROGRAMMATION JAVA DE MODELE EMF

Voici ci-dessous le code commenté du TP2 :

```
public class Main {

    public static void main(String[] args) {
        JavaBeansFactory fabrique = JavaBeansFactory.eINSTANCE ;

        //Charger le modèle de l'archive
        Archive demoJar=chargerModele("../../runtime-EclipseApplication/testJavaBean/testJavaBeans.javabeans") ;

        //Initialisation des variables
        int nbPropriete;
        int nbProprieteLiee;
        int nbProprieteContrainte;

        //On parcours la collection des paquetages
        for (Paquetage paquetage: demoJar.getPaquetage()){

            for(ElementModele elt: paquetage.getElements()){

                //Si instance de composant alors on affiche
                if(elt instanceof Composant && elt!=null){
                    System.out.println("*) composant "+paquetage.getNom()+"."+ elt.getNom()+" :");
                    nbPropriete=0;
                    nbProprieteLiee=0;
                    nbProprieteContrainte=0;
                    //On récupere la propriété et le type de propriété
                    //On incrémente selon le type de propriété
                    for(Propriete prop: ((Composant) elt).getProprietes()){

                        if(prop.getProperty()!=null) nbPropriete++;

                        if(prop.getProperty().getValue()==Property.CONTRAINTE_VALUE){
                            nbProprieteContrainte++;
                        }
```

```
            if(prop.getProperty().getValue()==Property.LIÃ‰E_VALUE){
                nbProprieteLiee++;
            }
        }
        //Affichage du message
        System.out.println("- nb .propriÃ©tÃ©s: "+nbPropriete+"(nb. liÃ©es:"+nbProprieteLiee+", nb.contraintes:"+nbProprieteContrainte+")");
        //Si l'élement Déclenche un evenement
        if( ((Composant) elt).getDeclenche().size()>0 ){
            for(Evenement evnt: ((Composant) elt).getDeclenche()){
                //Affichage du message pour tous les evenements qui peuvent être déclenchés
                System.out.println("- peut dÃ©clencher: "+paquetage.getNom()+"."+evnt.getNom());
            }
        }else{
            System.out.println("- ne dÃ©clenche pas d'Ã©vÃ©nements");
        }

        //Si l'element ecoute un évenement
        if( ((Composant) elt).getEcoute().size()>0 ){
            for(Evenement evntAudited: ((Composant) elt).getEcoute()){
                //Affichage du message pour tous les evenements pouvant être écoutés
                System.out.println("- est auditeur de: "+paquetage.getNom()+"."+evntAudited.getNom());
            }
        }else{
            System.out.println("- n'est pas auditeur d'Ã©vÃ©nements");
        }



    }
  }
}
```

Design Pattern

Nouveau Design Pattern : UniqueName

Dans cette section nous allons tenter de définir un nouveau design pattern, celui-ci porte le nom UniqueName.

Le problème dans lequel il intervient est le suivant : Dans une collection, deux éléments lui appartenant ne peuvent porter le même nom. Ces éléments doivent posséder un nom qui est unique.

L'objectif de ce Design Pattern est donc d'éviter les conflits dû au fait que deux éléments aient un nom identique.
Son intention est en conséquence de rendre chaque élément unique, ces éléments uniques doivent être identifiables par leurs noms.
On utilisera donc ce Design Pattern dès que l'on va créer un nouvel objet afin d'éviter tout conflit sur le système.
Un des moyens d'empêcher que les éléments ne portent le même nom serait d'avoir un stockage de ces éléments dans une collection. On vérifiera alors dans cette collection que lorsqu'un objet est créé on ne l'instancie pas avec un nom déjà existant dans notre collection en question.

Contraintes sur le Méta-Modèle

Langage naturel et langage OCL :

Voici une liste non exhaustive de contraintes auxquelles nous avons pensé pour ce projet avec leurs écritures en langage naturel et en langage OCL.

Classe Evenement et Propriété :

```
class Evenement extends ElementModele
{
    invariant conventionNommage: self.nom.substring(1,1) =  self.nom.substring(1,1).toLower();
}
```

Chaque nom d'évènement et chaque nom de propriété doit commencer par une minuscule.

Classe ElementModele :

```
property proprietaire#elements : Paquetage[?];
invariant motsReserves: self.nom <> 'class' and self.nom <> 'paquetage'
and self.nom <> 'composant' and self.nom <> 'proprietePrimitive'
and self.nom <> 'propriete' and self.nom <> 'archive' and self.nom <> 'methode';
```

Nos différentes classes sont pour la plupart des extensions de la classe ElementModele, ces classes ne peuvent donc pas porter le nom de « composant, class, paquetage, propriete, proprietePrimitive, archive et méthode ».

Classe Composant

```
invariant uniciteNomProprietes: self.proprietes->forAll( e1,e2 : Propriete |
    e1.nom = e2.nom implies e1=e2);
invariant uniciteNomMethodes: self.methodes->forAll( e1,e2 : Methode|
    e1.nom = e2.nom implies e1=e2);
```

Un composant est unique ! Chaque composant se nomme avec une minuscule en première lettre. Ses propriétés et ses méthodes sont également uniques, il ne peut y avoir de doublons.

Classe Paquetage :

```
invariant uniciteNoms: self.elements->forAll( e1, e2 : ElementModele |
                    e1.nom = e2.nom implies e1=e2);
invariant conventionNommage: self.nom.substring(1,1) =  self.nom.substring(1,1).toLower();
```

Le nom d'un paquetage est unique et chaque paquetage doit commencer par une minuscule.
Enfin les méthodes doivent également commencer par une minuscule et font référence à la contrainte « conventionNommage ».

Code MTL

Ci-dessous le code du fichier MTL permettant de générer les composants JAVA beans.

```
[comment encoding = UTF-8 /]
[module generate('http:///javaBeans.ecore')]

[template public generateJavaBeans(anArchive : Archive)]
[comment @main/]
[descriptionPaquetage(anArchive)/]

[/template]

[template public descriptionPaquetage(anArchive:Archive)]
[for (aPaquetage: Paquetage | anArchive.paquetage)]
    [descriptionElementModele(aPaquetage)/]
[/for]
[/template]

[template public descriptionElementModele(aPaquetage:Paquetage)]
[for (aElement: ElementModele|aPaquetage.elements)]
    [descriptionElementModele(aElement)/]
[/for]
[/template]


[template public descriptionElementModele(aElement: ElementModele)/]
```

```
[template public descriptionComposantDeclenche(aEvenement: Evenement)]
public void add[aEvenement.nom.toUpperFirst()/]Listener([aEvenement.proprietaire.nom/].[aEvenement.nom.toUpperFirst()/]Listener   listener) { ... };
public void remove[aEvenement.nom.toUpperFirst()/]Listener([aEvenement.proprietaire.nom/].[aEvenement.nom.toUpperFirst()/]Listener  listener) { ... };
[/template]

[template public descriptionPropriete(aPropriete: Propriete)]
[/template]

[template public descriptionPropriete(aPropriete: ProprietePrimitive)]
public [aPropriete.primitive/] get[aPropriete.nom.toUpperFirst()/]() { ... }
public void set[aPropriete.nom.toUpperFirst()/]([aPropriete.primitive/] value) [if (aPropriete.property=Property::contrainte) ] throws java.beans.Proper
[if (aPropriete.property=Property::liÃ©e) ]
public void addPropertyChangeListener(java.beans.PropertyChangeListener l) { ... }
public void removePropertyChangeListener(java.beans.PropertyChangeListener l) { ... }
[/if]
[if (aPropriete.property=Property::contrainte) ]
public void addPropertyVetoListener(java.beans.PropertyVetoListener l) { ... }
public void removePropertyVetoListener(java.beans.PropertyVetoListener l) { ... }
[/if]
[/template]

[template public descriptionPropriete(aPropriete: ProprieteBean)]
public  [aPropriete.composant.proprietaire.nom/].[aPropriete.composant.nom.toUpperFirst()/] get[aPropriete.nom.toUpperFirst()/]() { ... }
public  void Set[aPropriete.nom.toUpperFirst()/]([aPropriete.composant.proprietaire.nom/].[aPropriete.composant.nom.toUpperFirst()/] value) { ... }
[/template]
```

Conclusion

Le but de cette application était de créer un projet EMF capable de spécifier les méta-modèles afin de générer des composants JAVA beans (simplifiés). En l'état actuel, notre projet respecte toutes les demandes de chacuns des 4 TP et fonctionne comme attendu. Ce projet nous aura permis de développer notre compréhension des méta-modèles et des designs patterns ainsi qu'un autre aspect de l'ingénierie dirigée par les modèles.

Nous n'avons pas rencontré de problèmes majeurs durant ce TP, les consignes étaient claires et les charges de travail bien réparties.