UNIVERSITÉ DE BOURGOGNE

UFR Sciences et Techniques

Développement d'une interface graphique de visualisation d'une ontologie

Cours: Logiques de description / Ontologies et Web sémantique

Présenté par: Elie Matta et al.

Table des Matières

Introduction

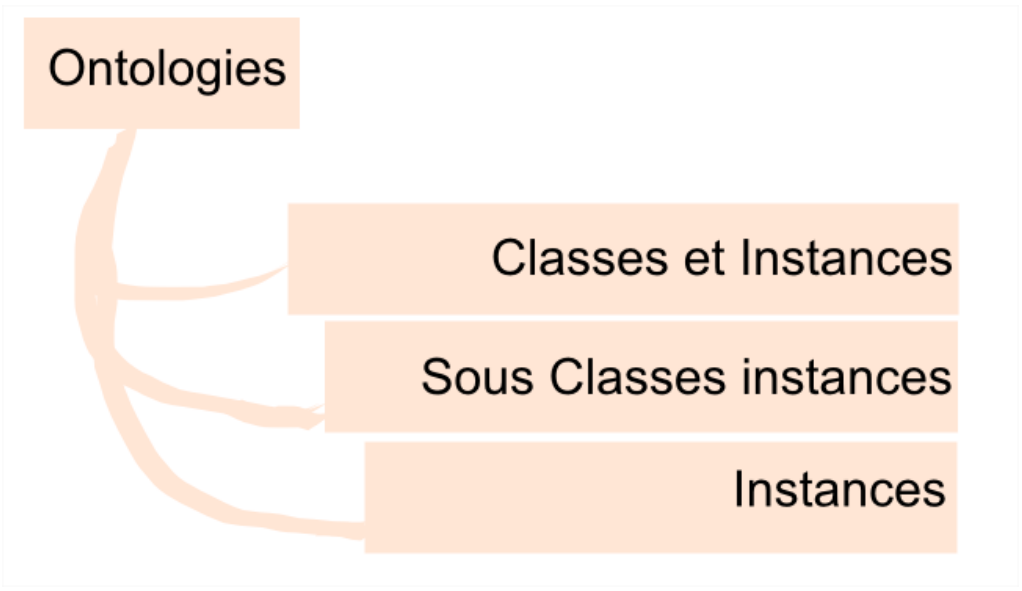
Nous sommes actuellement en Master II BDIA. Durant notre cours d'ontologie du WEB nous avons étudiés les ontologies. Cependant nous avons aussi pu remarquer combien cette technologie est jeune notamment en termes d'interface graphique et de convivialité.
En effet les outils étudiés en cours ne sont pas très facile d'accès hormis Protégé® qui possède une interface graphique riche mais demande un temps d'adaptation pour le maitriser. Nous avons aussi étudié la bibliothèque Jena qui elle permet à l'aide du langage Java d'effectuer les manipulations nécessaires sur les ontologies.
Dans le cadre de notre cours il nous a été demandé de réaliser un projet avec cette bibliothèque. Le sujet consiste à créer une visionneuse de Fichier Protégé (extension OWL). Dans ce document nous présenterons d' une part la phase d'analyse ainsi que les fonctionnalités que nous avons choisis d'implémenter et nous conclurons par une présentation de notre solution.

Analyse

Travaux Préliminaires

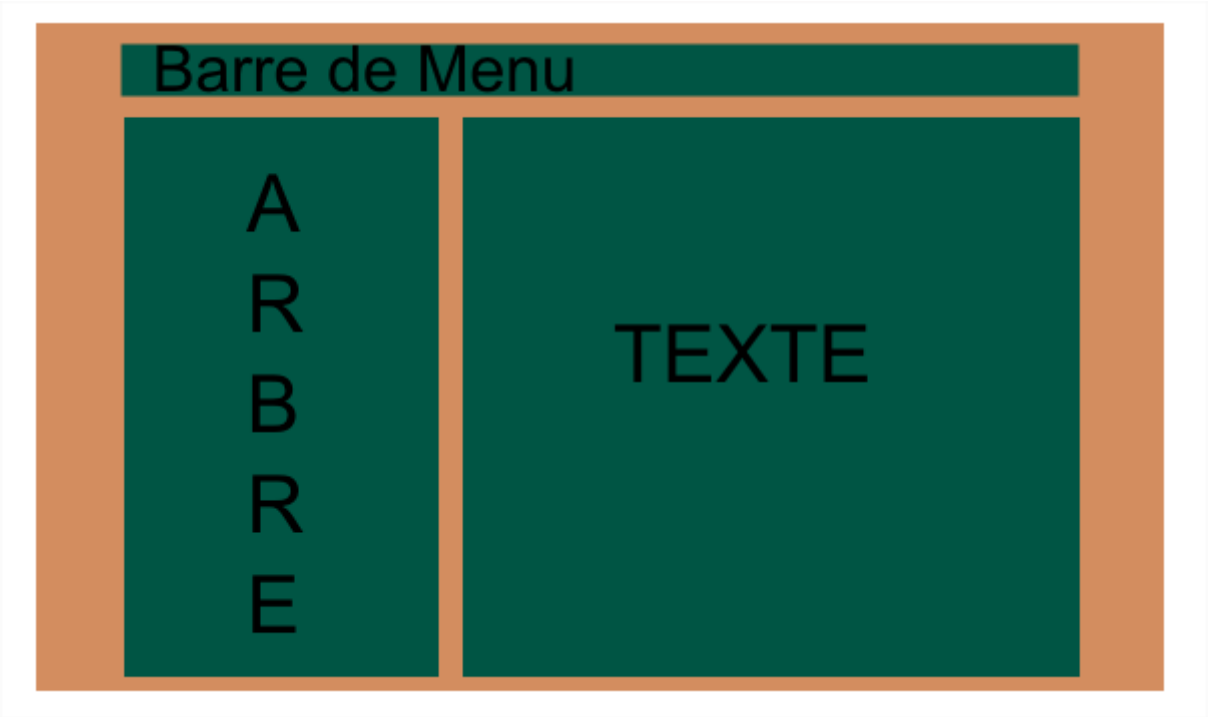
Tous d'abord nous avons effectué un travail de recherche sur les fichiers OWL. Nous en avons téléchargé plusieurs afin de bien analyser leur syntaxe. Il nous est très vite apparu qu'il existe une multitude d'ontologie toutes implémentées de manières différentes. De plus son formalisme basé sur les balises renforce sa complexité de compréhension.

Nous avons donc dans un premier temps extrait les composants fondamentaux. L'ontologie elle-même les Classes, sous-classes et les instances. Afin de représenter correctement ses informations et au vue de leurs caractères hiérarchiques nous avons choisi de les représenter sous forme d'un Arbre (Voir schéma ci-dessus)



Nous Partons donc de l'ontologie pour descendre de plus en plus en détail. Lorsque nous disposons de ces informations nous pouvons à l'aide des différentes fonctions de la bibliothèque Jena les extraire, comme par exemple les attributs d'une classe ou encore ses relations avec les autres classes mais aussi les valeurs de ses différentes propriétés.

Ébauche d'interface



Le but du sujet est de concevoir une interface permettant de représenter graphiquement toutes les informations d'une ontologie. Dès le début du projet nous avons décidé de créer 'un arbre avec une profondeur de vue modulable. Pour le reste des informations elles seront affichées dans un éditeur de texte.

Implémentation du Logiciel

- IDE Eclipse
- Plug In Jigloo pour bâtir des interfaces graphiques en Java
- La Bibliothèque Jena Version 2.6.4

Structure de notre Application

Notre Application se Base sur trois Classes :

- La Classe Menugeneral (Interface graphique)
- La Classe ActionSpeciale (Chargé d'implémenter les méthodes appelées par l'interface Graphique)
- La Classe Ontologie qui elle fait le lien avec la Bibliothèque Jena

La Classe Ontologie

Cette classe est chargée de faire l'interface entre les différentes classes de la bibliothèque Jena et notre application.

Attribut de la Classe

- Model de type OntModel : Modèle Jena
- URIOntologieselectionee : attribut mis en place pour gérer une multi Ontologie. permet de savoir avec quelle ontologie on travaille
- D'autres attributs essentiellement des ArrayList me permettant de charger les classes sous-classes et instances en mémoire afin de les afficher dans l'arbre (Voir code ci-dessous)

```
      public int code ;
      public String Errormessage;
      public OntModel Model = null;
    public String URIOntologieselectionee       = null;
    public ArrayList<String> ontologieName            = new
ArrayList<String>();
    public ArrayList<String> ontologieURI       = new ArrayList<String>();
    public ArrayList<String> ClassesList        = new ArrayList<String>();
    public ArrayList<String> SubClassesList     = new ArrayList<String>();
    public ArrayList<String> Instances                = new
ArrayList<String>();
```



```
    public ArrayList<String> ListAttribut       = new ArrayList<String>();
    public ArrayList<String> ListRelation       = new ArrayList<String>();
    public ArrayList<Individual>ListIndividual  = new
ArrayList<Individual>();
    public            ArrayList<Restriction>ListRestrictions=           new
ArrayList<Restriction>();
```

Méthodes de la Classe

La première des méthodes utilisées est la méthode GénérerModel(). Celle ci prend en paramètre une chaine de caractère correspondant au chemin du fichier OWL. Ensuite à l'aide de la méthode ***createOntologyModel()*** on va créer le contenant du modèle en lui passant quelque paramètres . Ici dans notre application nous avons choisi de créer un fichier OWL chargé totalement en mémoire sans possibilité d'utilisation d'un prouveur puisque notre application ne va écrire dans l'ontologie . Nous remplissons le modèle avec une lecture du fichier passer en paramètres .Pour finir nous mettons en place une gestion des exceptions pour capturer les divers problèmes qui peuvent se survenir à la lecture d'un fichier (fichier bloqué, fichier illisible...).
L'appel de cette Procédure s'effectue au moyen de l'interface Graphique avec l'utilisation d'un JfileChooser auquel on a appliqué des filtres afin de ne sélectionner que les fichiers OWL.

```
public void  GenererModel(String CheminOwl) {
            FileReader ModelReader  =null ;

            try {
                  ModelReader             = new FileReader(CheminOwl) ;
                  Model                   =
ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM);
                  Model.read(ModelReader,null);
                  code=00;
                  Errormessage="Pas d'erreur";
            }
            //Fichier Non trouvÃ©
            catch (FileNotFoundException e) {
                  code=01 ;
                  Errormessage            = "Fichier Non trouvé";
            }
            //Erreur de Fichier
            catch (IOError e) {
                  code=10 ;
                  Errormessage            = "Erreur d'entrée/sortie sur le
fichier";
            }
            catch (Exception e ){
                  code=99;
                  Errormessage            = "Erreur Inconnue";
            }
      }
```


Récupération des informations pour peupler l'arbre

Les Ontologies

Une fois le modèle créé on doit maintenant rechercher les ontologies qui le composent. En effet d'après les études que nous avons menées sur des fichiers OWL il s'avère qu'un fichier OWL peut contenir plusieurs ontologies (voir ontologie Wine.owl). Afin de bien prendre en compte tous cela Nous avons créé une fonction ***Listeontologie()*** qui elle est chargé de lister toutes les ontologies charger dans le Modèle . Cette Fonction remplit une Arraylist contenant les noms de l'ontologie (le nom de l'ontologie peut-être récupérer avec la fonction ***getLocalName().***

```
public void Listeontologie() {
            Iterator<Ontology> iter = Model.listOntologies();
            int i=0;
            while (iter.hasNext()) {
                  OntologyImpl ontimpl= (OntologyImpl) iter.next();
                  if
(ontologieURI.contains(ontimpl.getURI().toString())==false){
                        if
(ontologieName.contains(ontimpl.getLocalName().toString())==false) {
                        ontologieURI.add(ontimpl.getURI().toString());
                        ontologieName.add(ontimpl.getLocalName());
                        i++;
                        }
                  }
            }
      }
```

Les Classes : ListClasses

Même Principe que pour l'ontologie excepté que dans la liste des classes prises en comptes nous enlevons toutes les classes définies par restrictions. En effet ces classes sont dites abstraites et sont basées sur les propriétés d'une relation ou d'un attribut.

```
public void ListeClasse() {
            Iterator<OntClass> iter = Model.listClasses();
            while (iter.hasNext()) {
                  OntClass classes= (OntClass) iter.next();
                  if (classes.getLocalName()!=null){
                  if (classes.isRestriction()!=true){ //Il faut Enlever les
Classes Restrictions
                        ClassesList.add(classes.getLocalName());
                        }
                  }
            }
      }
```

Les Sous-Classes : ListSousClasses

```
public void ListeSubClasse(String URI ,String NomClasse) {
            SubClassesList.clear();
            if (NomClasse!=null && URI!=null) {
                  OntClass classes= Model.getOntClass(URI+ '#'+ NomClasse);
                  Iterator<OntClass>subclasseiter=classes.listSubClasses();
                  while (subclasseiter.hasNext()){
                        OntClass subclass= (OntClass) subclasseiter.next();
```



```
                        SubClassesList.add(subclass.getLocalName());
                    }
                }
            }
```

Idem que la methode **ListeClasses**() retourne simplement les sous-classes d'une classe identifié par son URI et son nom local . En effet Jena à partir de L'URI (Adresse Interne) et du nom de la classe permet de retourner l'objet classe à l'aide de la fonction ***GetOntClass***().
Une fois que l'on possède la classe il suffit d'appeler la méthodes ***GetSubClasses()*** pour obtenir la liste des classes enfants dans un Itérateur .

Les Instances de classes ou sous-classes

Les instances en Jena sont des variables de type Individuals il suffit donc d'appliquer la méthode ***GetIndividuals***() du model Jena afin de récupérer la Liste des instances d'une classe.

```
public void ListeInstances(String URI ,String NomClasse){
            Instances.clear();
            ListIndividual.clear();
            Iterator<Individual> Instance;

                  Instance    = Model.listIndividuals();

            while (Instance.hasNext()) {
                  Individual Exemple                        =
Instance.next();

                  if
(Exemple.getURI().toString().equalsIgnoreCase(URI)==false){ // A Ajouter
pour le multi ONTOLOGIE
                        if (ListIndividual.contains(Exemple)==false) {
                              ListIndividual.add(Exemple);
                              String NomExemple       =
Exemple.getLocalName();
                              String Classe                 =
ClasseInstance(Exemple);
                              if (NomClasse.equalsIgnoreCase("#"+Classe)){
                                    if
(Instances.contains(NomExemple)==false){
                                          Instances.add(NomExemple);
                                    }
                              }
                        }
                  }
            }
      }
```

A Partir d'ici nous avons tous les composants pour remplir notre arbre. En Java l'arbre est représenté par un contrôle Jtree. Il est très hiérarchique et donc impose une méthode adapté à son remplissage. Notre arbre est ainsi remplit par la méthode ***PeuplerArbre()*** de la Classe ***ActionSpéciale***.Voici son mode opératoire.

Nous partons de la racine de l'arbre appelée ontologies. Les ontologies presentes dans le modèle seront directement insérés au niveau en dessous . La Logique voudrait que nous insérions ensuite les classes puis les sous-classes. Lors de notre premier remplissage il est apparu un problème avec les instances. En effet il y a des instances qui ne sont dépendante d'aucune classe elle sont simplement gréffées à la super classe OWL-Thing ce qui constitue une rupture dans notre hiérarchie. De plus cette rupture engendre aussi un autre problème celui de l'identification d'un nœud. En fait à partir du moment ou il existe des instances au mêmes niveau que les classes il est clair que l'on ne peut plus basé notre identification de nœud sur la profondeur de l'arbre . Afin de bien identifier le type des nœuds de l'arbre nous avons pris la décision de regrouper toutes les instances sous un nœud appelé Instances.

Revenons maintenant sur la fonction de population de notre arbre une fois le problème des instances réglés nous déroulons l'algorithme normalement à savoir Classes Sous-Classes Instances et ceci pour toute les ontologies du Modèles.

Pour terminer nous avons choisi d'implémenter la notion de profondeur de vue dans notre arbre. En effet l'utilisateur va pouvoir choisir le niveau de détails qu'il souhaite. Par défaut le niveau de détail le plus élevé est affiché

Cette fonctionnalité s'active à l'aide d'un Popup Menu . En fonction du choix un attribut profondeur est fixé 3 pour le maximum et 1 pour le minimum de détails (uniquement les classes)

A ce stade nous avons notre arbre de remplit avec les différents concepts que l'on a vu précédemment cependant maintenant il va falloir se pencher sur le problème de la sélection.

```
public void PeuplerArbre (Ontologie ont) {
            DefaultMutableTreeNode Noeud=null;
            DefaultMutableTreeNode Branche=null;
            DefaultMutableTreeNode Feuille=null;
            DefaultMutableTreeNode subrep=null;
            DefaultMutableTreeNode inst=null;
            DefaultMutableTreeNode rac= Menugeneral.getRacine();
            int b=0 ;
            ont.Listeontologie();

            //Creation ontologie
            for (int i=0;i<ont.ontologieName.size();i++){
                  Noeud                          = new
DefaultMutableTreeNode(ont.ontologieName.get(i));
                  rac.add(Noeud);
                  if (Vision==3){
                        ont.ListeInstances(ont.ontologieURI.get(i)
,"#"+"Thing");

                        for (int a=0;a<ont.Instances.size();a++){
                              if (b==0){
                                    subrep                          = new
DefaultMutableTreeNode("Instances");
                                    Noeud.add(subrep);
```



```
                                    b=1;
                                }
                                else
                                {
                                    inst                    = new
DefaultMutableTreeNode(ont.Instances.get(a));
                                    subrep.add(inst);
                                }
                            }
                        }
                    }

                    ont.ListeClasse();
                    ont.ClasseAnonymme();

                        //Creation des classes Racines
                    for (int z=0 ;z<ont.ontologieURI.size();z++)
                        for (int j=0;j<ont.ClassesList.size();j++){
                            String NomClasse
                    = ont.ClassesList.get(j).toString();
                            String URI
                        = ont.ontologieURI.get(z);
                            //Appel de la Fonction pour verifier si c'est une
Classe racine
                            if (NomClasse!=null){
                                Boolean result=ont.Estracine(URI,NomClasse);
                                if (result==true){
                                    Branche
            = new DefaultMutableTreeNode(ont.ClassesList.get(j));
                                    Noeud.add(Branche);
                                    //Ajout des Instances de la classe

                                    if (Vision==3) {

            ont.ListeInstances(URI,"#"+NomClasse);
                                        b=0;
                                        for (int
a=0;a<ont.Instances.size();a++){
                                            if (b==0){
                                                subrep
            = new DefaultMutableTreeNode("Instances");
                                                Branche.add(subrep);
                                                b=1;
                                            }
                                            inst                    =
new DefaultMutableTreeNode(ont.Instances.get(a));
                                            subrep.add(inst);

                                        }
                                    }
                                    //Creation des enfants
                                    if (Vision>=2) {

            ont.ListeSubClasse(URI,NomClasse);
                                        for(int
k=0;k<ont.SubClassesList.size();k++){
                                            String NomsubClasse
            = ont.SubClassesList.get(k).toString();
                                            if (NomsubClasse!=null) {
                                            Feuille
                    = new DefaultMutableTreeNode(ont.SubClassesList.get(k));
```