2015-2016
M1 INFORMATIQUE

GRAPHE ET ONTOLOGIES

FOURMOND Jérôme

Sous la direction de
Dr. RICHER Jean-Michel
Dr. AIT EL MEKKI Touria

Soutenu publiquement le :
Jeudi 23 Juin 2016

université angers

REMERCIEMENTS

TABLE DES MATIERES

Introduction

1. Le sujet

On désire créer une application en **JAVA** pour la navigation dans un document électronique à l'aide d'un graphe d'ontologie afin d'en faciliter l'étude.

2. Les objectifs

Il est demandé dans un premier temps de développer un modèle pour la gestion des graphes ainsi que la partie visualisation (affichage des sommets, des arcs, zoom, clic sur un sommet, sur un arc…).

Cette première partie fait abstraction des données stockées au sein des sommets et des arcs et elle doit pouvoir être adaptée en fonction du sujet à traiter.

Dans un second temps, on désire mettre en œuvre ce modèle en l'appliquant aux graphes d'ontologie :

- Elaborer un graphe à partir d'une ontologie (**XML**)
- Faire le lien entre le texte (le document, fichier **XML DocBook**) et les nœuds du graphe.

3. Les prérequis

Il est nécessaire de savoir manipuler **Java**, la libraire **Swing** de ce dernier, et les fichiers **XML**.

4. N.B

Le développement du projet a pu être suivi sur **GitHub** au lien suivant :

https://github.com/jfourmond/Graphe-Et-Ontologies

Il est à noter que la librairie **Java FX** pour les interfaces graphiques a été utilisée en remplacement de la libraire **Swing** dont le ressenti et le visuel n'était pas suffisant pour une application qui devait s'avérer agréable pour l'utilisateur.

Dans le cadre de la lecture et de l'écriture de fichier **XML**, la libraire externe **JDOM** a été utilisée. Fournissant des outils simples d'accès et rapides, elle semblait adéquate au projet.

I.Vision Une – Sans Ontologie

1. Le Framework

La première partie de développement a débuté par la mise en place d'un framework pour la gestion de graphe. Ce dernier s'est donc tout d'abord composé de trois principales classes :

- ***Tree***, représentant un arbre / graphe non orienté, composée de deux listes, l'une de sommets, l'autre d'arcs
- ***Vertex***, représentant une interface qui devait être implémenté pour être utilisé dans l'arbre
- ***Edge***, représentant un arc, composée de deux sommets et d'une valeur (le libellé de l'arc).

2. La Visualisation

Le développement de l'interface graphique lors de cette première vision s'est effectuée à l'aide de Swing, quatre classes ont été produites dans ce but :

- ***VertexView*** : un **JComponent** dessinant un sommet du graphe sous la forme d'un cercle, uniquement.
- ***EdgeView*** : un **JComponent** dessinant un arc entre deux **VertexView**
- ***TreeView*** : un **JPanel** associant les sommets du **Tree** avec les **VertexView** et les arcs avec les **EdgeView**, et attribuant également différents listeners.
- ***Window*** : la fenêtre contenant **TreeView**

3. Conclusion

La réalisation a rapidement été faîte que l'interface ainsi produite manquait de confort et d'intérêt pour l'utilisateur. **Java FX**, permettant de réaliser des interfaces graphiques évoluées et modernes, s'est avéré être un candidat plus qu'acceptable au remplacement de la librairie **Swing**. Cette perturbation a donc mené à une découverte de la librairie et à de nombreux essais.

L'interface n'était pas le seul changement à opérer. Après avoir produit le framework, il s'est avéré qu'il ne sciait pas à l'idée du projet. Ce dernier était générique, le développeur pouvait manipuler des entiers comme des chaînes de caractères, mais le rôle de remplacement des ontologies n'était pas appliqué, elles devaient pouvoir se lire dans un graphe. Il devait donc devenir plus *relationnel*.

II. Vision Deux – Avec Ontologie

1. XML, DTD et XSD

1.1. XML

Le fichier XML d'une ontologie se présente ainsi :

```
<?xml version="1.0" encoding="UTF-8"?>
<IndexSource>
    <ENTREE id="1" nom="Pays-De-La-Loire">
        <RELATION nom="appartient à la région" />
        <RELATION nom="appartient au département du" />
    </ENTREE>
    <ENTREE id="2" nom="Maine-Et-Loire">
        <RELATION nom="appartient à la région">
            <LIEN>1</LIEN>
        </RELATION>
        <RELATION nom="appartient au département du" />
    </ENTREE>
    <ENTREE id="3" nom="Loire-Atlantique">
        <RELATION nom="appartient à la région">
            <LIEN>1</LIEN>
        </RELATION>
        <RELATION nom="appartient au département du" />
        </ENTREE>
</IndexSource>
```

La déduction étant que la définition d'un sommet s'effectue par la balise ENTREE. La balise RELATION représente une relation et peut contenir une ou plusieurs balises LIEN qui effectuent un lien/un arc vers un second sommet dont l'identifiant est détaillé.

Par exemple, correspondant au fichier précédent :

Deux relations :

- *appartient à la région*
- *appartient au département du*

Trois sommets :

- *Pays-De-La-Loire*, portant l'identifiant 1
- *Maine-Et-Loire*, portant l'identifiant 2
- *Loire-Atlantique*, portant l'identifiant 3

Des arcs/liens :

- *Maine-Et-Loire appartient à la région Pays-De-La-Loire (1)*
- *Loire-Atlantique appartient à la région Pays-De-La-Loire (1)*

1.2. DTD et XSD

Pour une utilisation aisée et une validation du fichier **XML** dans l'application, il était nécessaire de créer un ou des documents permettant de décrire un modèle à respecter.

Une **Document Type Definition** [1]a donc été produite à cet effet :

```
<!ELEMENT IndexSource (ENTREE*) >
<!ELEMENT ENTREE (ATTRIBUT*, RELATION*, RENVOIS?) >
```

[1] **DTD** dans la suite du document

```
<!ELEMENT RELATION (LIEN*) >
<!ELEMENT ATTRIBUT EMPTY>
<!ELEMENT LIEN (#PCDATA) >
<!ELEMENT RENVOIS (LIEN*) >


<!ATTLIST IndexSource            corpus          CDATA #IMPLIED >
<!ATTLIST ENTREE                 id              CDATA #REQUIRED
                                 nom             CDATA #REQUIRED >
<!ATTLIST ATTRIBUT               nom             CDATA #REQUIRED
                                 valeur          CDATA #REQUIRED >
<!ATTLIST RELATION               nom             CDATA #REQUIRED >
```

Tout fichier **XML** sera validé sur cette **DTD** avant de pouvoir être modélisé par l'application. **JDOM** ne bénéficiant pas encore d'une fonctionnalité de validation à l'exécution sur **DTD**, il a fallu convertir ce fichier sous la forme d'un **XML Schema**[2].

2. Le nouveau Framework

La notion de relation a modifié l'utilité du framework. Un graphe (**Tree**) ne contenait plus une liste d'arcs mais désormais une liste de **Relation**. Le code a donc été profondément modifié.

2.1. Les sommets

L'interface **Vertex** est devenu une classe à part entière. Un sommet est désormais composé d'un identifiant devant être unique une fois ajouté au graphe, d'un nom et d'une collection associant clé et valeur (**Map**).

```
Vertex sommet = new Vertex("1"); // Création d'un sommet avec l'ID "1"
sommet.add("Nom");               // Création d'un attribut "Nom"
sommet.set("Nom", "Jerome");     // Edition de la valeur de l'attribut "Nom"
String nom = sommet.get("Nom");  // Récupération de la valeur de l'attribut "Nom"
```

2.2. Les relations

La classe **Relation** est une nouveauté de cette seconde vision de l'application. Elle décrit une relation. Elle est définie par un nom, devant être unique une fois ajouté au graphe, et d'une liste de paire de **Vertex**.

A cet effet une classe générique **Pair** a été écrite.

Une paire de sommet décrit un arc de cette relation.

```
Vertex v1 = new Vertex("1");
Vertex v2 = new Vertex("2");
Relation relation = new Relation("est voisin de");// Création d'une relation
Pair<Vertex, Vertex> pair = new Pair<>(v1, v2);
relation.add(pair);                               // Ajout de la paire à la relation
```

2.3. L'arbre

La nouvelle modélisation de **Tree** comporte désormais une liste de **Vertex**, une liste de **Relation** ainsi qu'un fichier nécessaire pour le chargement et la sauvegarde.

---

[2] **XSD** dans la suite du document

L’ajout de sommet :

```
Tree tree = new Tree();           // Création de l'arbre
tree.createVertex("1");           // Création d'un sommet portant l'identifiant "1" dans
l'arbre
Vertex vertex = new Vertex("2");
tree.createVertex(vertex);        // Ajout d'un sommet dans l'arbre
```

L’ajout d’un arc, après création d’une relation :

```
tree.createRelation("est voisin de");             // Création d'une relation dans
l'arbre
tree.addPair("est voisin de", "1", "2");          // Ajout d'une paire dans la relation
"est voisin de", entre le sommet portant l'identifiant "1" et le sommet portant
l'identifiant "2"

Relation relation = new Relation("est parent de");
Vertex p1 = new Vertex("3");
Vertex p2 = new Vertex("4");
Pair<Vertex, Vertex> pair = new Pair<>(p1, p2);
relation.add(pair);
tree.createRelation(relation);                    // Création d'une relation dans
l'arbre
```

3. Graphe

4. Graphe & Ontologies

III.L’Application

IV.Conclusion

V.Bibliographie

Android : http://developer.android.com/reference/packages.html

Scripts : http://developer.android.com/tools/help/adb.html

Qt : http://doc.qt.io/qt-4.8/ et Qt Assistant

Présidence de l'université
40 rue de rennes – BP 73532
49035 Angers cedex
Tél. 02 41 96 23 23 | Fax 02 41 96 23 00

université angers

ENGAGEMENT DE NON PLAGIAT

Nous, soussignons **FOURMOND Jérôme** et **NOEL Florentin**
déclarons être pleinement conscients que le plagiat de documents ou d'une partie d'un document publiée sur toutes formes de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.
En conséquence, nous nous engageons à citer toutes les sources que nous avons utilisées pour écrire ce rapport.

signé par les étudiants le . . / . . / . . . .

Cet engagement de non plagiat doit être signé et joint à tous les rapports, dossiers, mémoires.

Présidence de l'université
40 rue de rennes – BP 73532
49035 Angers cedex
Tél. 02 41 96 23 23 | Fax 02 41 96 23 00

université angers