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

Mes remerciements s'adressent en particulier à mes encadrants et tuteurs, Jean-Michel Richer et Touria Ait El Mekki, qui ont su se montrer présents et intéressés, et m'ont soutenu dans ce projet.

Merci à la faculté des Sciences de l'Université d'Angers qui nous a, à mes collègues d'autres projets et moi-même, autorisés à utiliser leurs locaux.

Je n'oublie pas mes compagnons de promotion, qui ont su se montrer distrayant durant des périodes nécessaires mais tout aussi sérieux et inspirant lorsque le besoin se faisait sentir.

Merci.

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 librairie **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.

Le projet a été proposé l'année passée (2014 – 2015). La possibilité de s'en inspirer était proposée mais la vision de l'application était complétement différente, et il a été préféré de s'engager sur une nouvelle voie.

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 le graphe, pour permettre une personnalisation assez importante à l'utilisateur
  - ***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

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 :

```
<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.

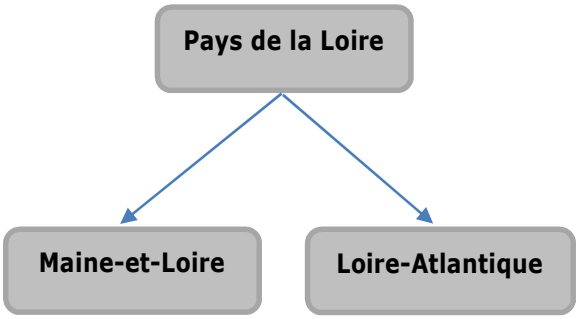
Par exemple, correspondant au fichier précédent :

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.

Des exceptions ont également été créées pour la gestion de chacune des classes principales.

2.1. Les sommets

L'interface **Vertex** est devenue 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 pairs de **Vertex**.

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

Une paire de sommets 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] **XSD** dans la suite du document

2.3. Le graphe

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.

L'ajout de sommet :

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

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

```
tree.createRelation("est voisin de");            // Création d'une relation dans le
grpahe
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 le
graphe
```

2.4. Lecture & Ecriture

JDOM permet une lecture et une écriture aisée des fichiers **XML**. Deux classes, chacune dans un traitement précis, respectivement **TreeLoader** et **TreeSaver**, sont utilisées. Elles héritent de la classe **Task** de la librairie **Java FX**, ce qui permet de les utiliser tel des threads ou d'envoyer des *mises à jour* aux composants graphiques.

2.4.1. Lecture

La procédure de lecture d'un fichier **XML** se découpe en plusieurs étapes :

1. Validation du fichier sur le schéma
2. Lecture de toutes les balises ***ENTREE***
    1. Construction des **Vertex** en récupérant les attributs ***id*** et ***nom***
    2. Lecture de toutes les balises ***ATTRIBUT*** pour ajout dans le sommet courant
3. Nouvelle lecture de toutes les balises ***ENTREE***
    1. Lecture de toutes les balises ***RELATION*** pour création des relations
        a. Construction des Relation en récupérant l'attribut ***nom***
        b. Lecture de toutes les balises LIEN et construction des arcs en récupérant le contenu de la balise

2.4.2. Ecriture

La procédure d'écriture du graphe dans un fichier **XML** s'effectue par une écriture de tous les **Vertex** et des relations.

Pour chaque sommet, une balise ***ENTREE*** est créée. Les attributs de cette dernière, ***id*** et ***nom***, sont remplis à partir des propriétés ***id*** et ***nom*** du sommet. A l'intérieur de cette balise, on ajoutera pour chaque paire clé/valeur de la **Map** d'attributs, une balise ***ATTRIBUT*** ayant pour attributs :

- ***nom*** : la clé
- ***valeur*** : la valeur

L'écriture de la balise ***ENTREE*** se conclut par l'insertion de toutes les balises ***RELATION*** correspondantes aux relations du graphe. Pour chacune de ses relations, une vérification et une écriture est effectuée quant à la présence de paires dont l'origine est le sommet actuel sous la forme de balise ***LIEN***.

3. La visualisation

Nouvelle librairie dit nouvelle gestion de la visualisation. Les classes produites lors de la première vision grâce à la librairie **Swing** ont donc été profondément modifiées, seules ces dernières seront traitées et détaillées

3.1. VertexView

Le modèle visuel d'un sommet est construit dans la classe ***VertexView***. La classe hérite de ***Group***, et est composé d'un cercle et de deux labels, respectivement pour l'identifiant et le nom du **Vertex** (affichables à tour de rôle).

Le **Vertex** ainsi modélisé est lié à son visuel par un attribut.

3.2. EdgeView

EdgeView hérite également de ***Group***. La classe est composée d'une ligne et d'un label, construit sur la base de deux **VertexView** (origine et fin), et du nom de la relation associée. Une couleur peut également lui être attribuée, étant par défaut le noir.

3.3. TreeView

TreeView permet de faire le lien entre la visualisation complète et les composants graphiques du graphe. Héritant de ***BorderPane***, le panel contient le visuel de l'application : barre de menu, graphe, liste des sommets et liste des relations.

Les évènements pouvant être déclenchés y sont créés en son sein.

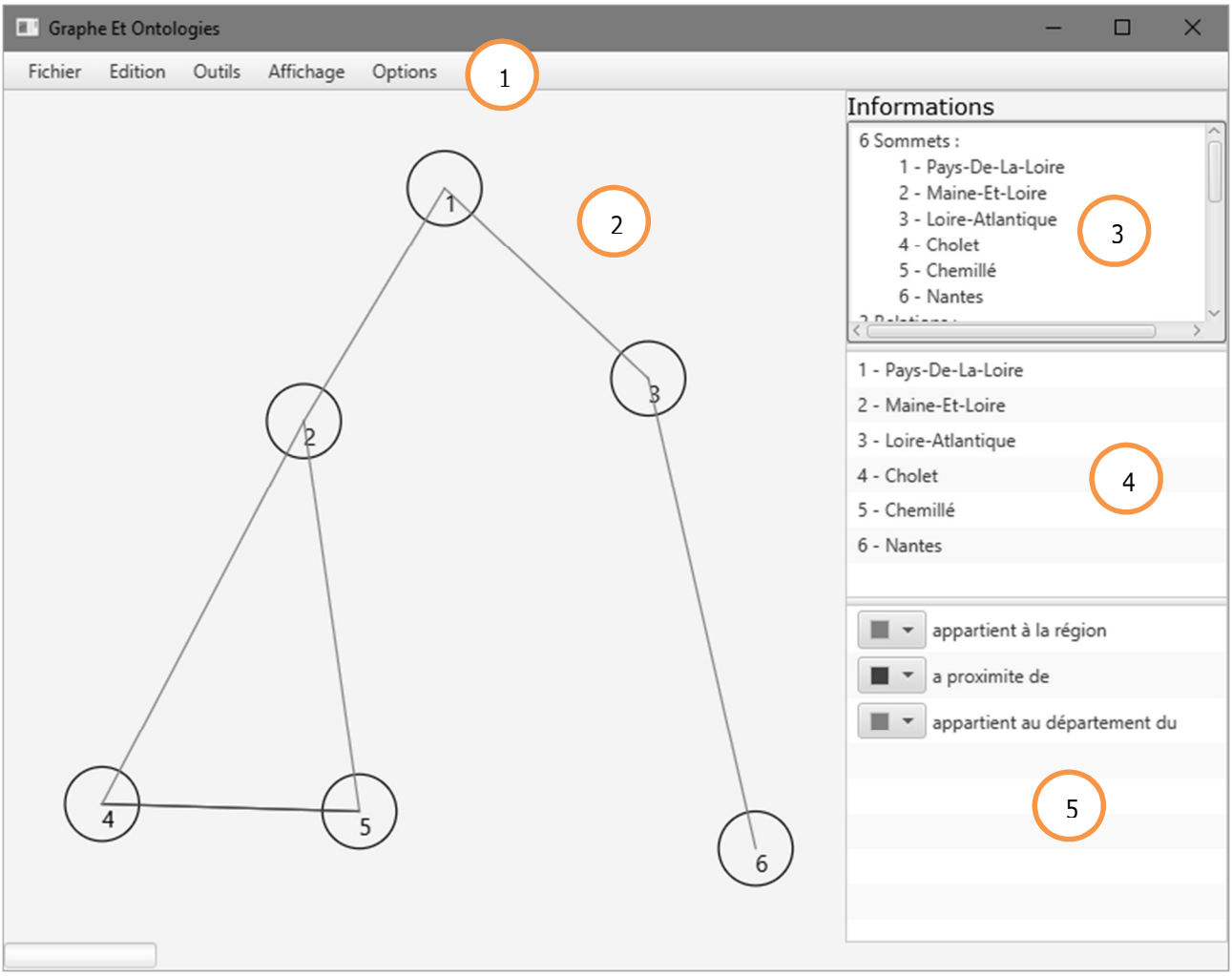
3.4. Autre

De nombreuses autres classes ont été mises en place.

Des interfaces pour :

- L'ajout de sommet
- L'ajout de relation
- L'ajout d'arcs…
- L'affichage d'un splash-screen lors du chargement de l'application
- L'affichage d'un splash screen lors de la fermeture de l'application

III. L'Application



L'application dispose de 3 zones principales.

1. Les zones du BorderPane

1.1. Panel Nord (1)

La première, la barre de menu, est composée de cinq menus :

- Fichier, contenant toutes les actions de création, d'ouverture et d'enregistrement du graphe
- Edition, contenant toutes les actions d'ajout de sommet, de relation et d'arcs
- Outils, permettant d'ouvrir une nouvelle fenêtre affichant une table des sommets du fichier
- Affichage, contenant des actions permettant la gestion de l'affichage des sommets, leurs libellées (identifiants, noms, ou néant), et des relations
- Options, contenant des actions permettant de modifier quelques options de l'application

Quelques raccourcis sont associés aux actions qui peuvent être effectuées le plus souvent pour faciliter leurs utilisations.

1.2. Panel Centre (2)

Le centre de l'application sert d'affichage au graphe par des cercles, représentant les sommets, et des lignes, représentant des arcs.

Un zoom / dézoom et un déplacement du panel ont également été implémentés.

Chaque composant graphique dispose d'actions influençant les autres zones de l'application.

1.3. Panel Est

La zone est de l'application est elle-même composé de trois composants graphiques :

1.3.1. Zone d'information (3)

La zone d'information est le composant graphique affichant des détails sur le graphe à l'utilisateur. Elle est mise à jour en fonction de ses actions (ajout, suppression, édition).

Elle peut également fournir des informations plus détaillées sur un élément du graphe.

Par exemple, un clic sur un sommet affichera son identifiant, son nom, et la totalité de ses attributs.

1.3.2. Liste des sommets (4)

La liste des sommets affiche, comme son nom l'indique, la liste des sommets de l'application. Elle est mise à jour en fonction des actions effectuées sur les sommets du graphe (ajout, suppression, édition).

Le sommet sélectionné sur le graphe est mis en évidence dans la liste et réciproquement.

1.3.3. Liste des relations (5)

La liste des relations, affiche comme son nom l'indique, la liste des relations de l'application et leurs couleurs associées. Elle est mise à jour en fonction des actions effectuées sur les relations du graphe (ajout, suppression, édition, changement de couleur).

La relation sélectionné dans la liste provoquera une mise en évidence de toutes ses arcs sur le graphe.

IV. Conclusion & Perspectives

1. Le projet

Le but du programme était de pouvoir rendre une ontologie sous la forme d'un fichier **XML** plus lisible sous la forme d'un graphe.

L'objectif, dans le cas d'une création est atteint avec quelques contraintes : créer personnellement son ontologie et placer les sommets, changer les couleurs, selon le besoin. Dès lors l'ontologie est lisible pour le *propriétaire*. Il serait nécessaire d'envoyer le fichier de position associé pour toute utilisation délocalisée.

Dans la situation d'un fichier dont la création a été effectuée manuellement, il est nécessaire d'effectuer une analyse, parfois poussée, du graphe.

2. L'apport personnel

Le projet m'a permis d'appréhender de nouvelles librairies **Java**.

JDOM m'a autorisé la découverte de méthodes pour une manipulation simplifiée de fichier **XML**.

Java FX est un outil puissant et facile d'accès pour la création d'interface graphique. La comparaison avec **Android** peut être effectuée : il est possible de créer des interface *statiques* grâce à des fichiers **XML**, sous **Java FX** appelé **FXML**. Cette séparation autorise la mise en place de l'interface graphique par un tiers.

Le framework des graphes m'a permis d'appréhender plus largement les classes génériques, bien que cette notion ne soit pas inutilisée par la suite.

3. Perspectives

Le fichier XML proposé pour présenter le projet affichait la notion de *renvois*. Elle permet d'envoyer des informations sur le sommet à l'utilisateur.

Il pourrait donc être intéressant de l'implémenter.

L'application manque globalement d'identité. Outre le logo réalisé simplement et personnellement, le nom de l'application, GandO, rien ne permet de la différencier d'une autre.

Il serait donc bon de styliser l'application et d'utiliser un nom plus parlant permettant d'identifier rapidement les technologies utilisées : OntoGraph, OntoloGraphX, OntologiX…

V.**Sitographie**

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

Java Doc : https://docs.oracle.com/javase/8/docs/api/

Java FX Doc : http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

JDOM : http://www.jdom.org/

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

ENGAGEMENT DE NON PLAGIAT

Je, soussigne **FOURMOND Jérôme**
déclare être pleinement consciente 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, je m'engage à citer toutes les sources que j'ai utilisé pour écrire ce rapport.

signé par l'étudiant 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