Stéphane INTHAVIXAY
Sébastien RABERE
Adrien SANTUS

M2 MIAGE EVRY

PROJET
DOCUMENTS STRUCTURES

Représentation SVG de graphes RDF, RDFS, OWL

Professeurs enseignants : Henry BOCCON-GIBOD **février 2007**
Antoine RISK

SOMMAIRE

INTRODUCTION

A l'heure où se diversifient les technologies XML, il est intéressant de parler des langages RDF, RDFS et OWL. Ceux-ci permettent de décrire des ressources selon des concepts de classe, individu et relation. La force de ces langages d'ontologie est qu'ils peuvent être représentés sous la forme de graphes orientés. Ainsi, grâce à l'aspect visuel, ils peuvent être aisés à comprendre pour un utilisateur.

Le langage SVG est un standard du W3C pour les graphiques sur le Web et utilise la représentation vectorielle. Notre projet est de représenter les graphes exprimés dans des fichiers RDF, RDFS, OWL en utilisant SVG comme un outil graphique.

Nous avons donc décidé pour répondre à ce sujet d'accentuer le développement de notre outil sur la simplicité. En effet, l'essentiel est d'aider et d'accompagner l'utilisateur à comprendre le contenu de ces fichiers.

Nous allons donc présenter une à une, les différentes grandes parties notre solution.

1. Rappel du cahier des charges

1.1. Rappel du sujet

Le projet consiste à représenter graphiquement, grâce au langage SVG, des entités et des liens décrits en RDF, RDFS ou OWL, de façon la plus ergonomique possible. L'utilisation de formes graphiques et couleurs permet d'analyser et trier les données dans le but d'extraire facilement les informations pertinentes.

1.2. Définition des termes (RDF, RDFS, OWL et SVG)

- **RDF** (Resource Description Framework) est un langage XML de réseau sémantique. RDF permet de décrire des ressources applicables à n'importe quel domaine. C'est un langage basé sur un principe de description sujet/predicat/objet.

- **RDFS** qui signifie RDF schéma permet de structurer les ressources décrites par RDF. Il décrit notamment les classes d'un fichier RDF.

- **OWL** (Ontology Web Language) est un langage d'ontologie qui permet de compléter les deux précédents langages en insistant sur la description des classes, grâce à des concepts d'équivalence, d'héritage, etc… C'est une extension de RDF qui rajoute certains concepts et qui est donc plus expressif.

- **SVG** (Scalable Vector Graphic) est un langage de description graphique 2D en XML. Il permet de représenter plusieurs types d'objets graphiques, statiques ou animés et possède la portabilité du langage XML. Sa modularité et sa puissance d'intégration dans le développement Web en font un standard tout désigné.

1.3. Définition du besoin

Nous cherchons à lier les deux concepts suivants : la description de ressources par les formats spécifiés plus haut et la représentation graphique SVG. Notre fil conducteur est la maîtrise par l'utilisateur de l'information qui découle des fichiers RDF, RDFS et OWL. En effet, celui-ci attend un outil qui l'accompagnera dans la compréhension de la sémantique décrite dans les différents fichiers. L'aspect graphique doit être une solution en soi de par sa simplicité d'expression. Tout doit donc être pensé selon ce souci utilisateur: mise à disposition de l'information, maîtrise de celle-ci par l'utilisateur, interactivité et interface graphique (notamment par le formalisme des graphes), …

1.4. Problématique

Notre problématique est la suivante : Partir de fichiers descripteurs de ressources pour découvrir et comprendre les liens entre les entités au travers de leur représentation graphique. Ceci implique une représentation visuelle ordonnée des informations pour en faciliter la compréhension. Il faut donc réfléchir à la façon d'extraire les informations des différents fichiers, à la façon de les trier, de les mettre à disposition de l'utilisateur, de les lier en leur donnant un sens, de les représenter graphiquement de façon claire, …

1.5.Solution proposée

Nous avons donc développé une application qui prend en entrée des fichiers aux formats cités précédemment et qui génère les graphes représentatifs, définis en SVG, celle-ci se basant sur une interface graphique simple et intuitive.

Nous parsons donc les fichiers spécifiés afin d'en extraire les éléments et de les présenter sous la forme de trois listes à l'utilisateur. Celui-ci sélectionne les éléments qu'il juge pertinent de représenter, puis, nous établissons un ensemble d'éléments à dessiner que nous affichons alors à l'écran. Le graphe peut alors être exporté sous forme d'un fichier SVG.

2. CONCEPTION

2.1.Plan d'action

Pour rentrer un peu plus dans le détail, notre solution se décompose de la sorte :

- Etudier les différents formats de fichiers pour choisir une méthode de parsage qui fera ressortir les concepts clés à représenter.

- Proposer une interface graphique qui permette à l'utilisateur de sélectionner les données parmi celles extraites et triées des documents. Cela permet de minimiser la complexité du graphe en affichant uniquement les informations considérées pertinentes par l'utilisateur.

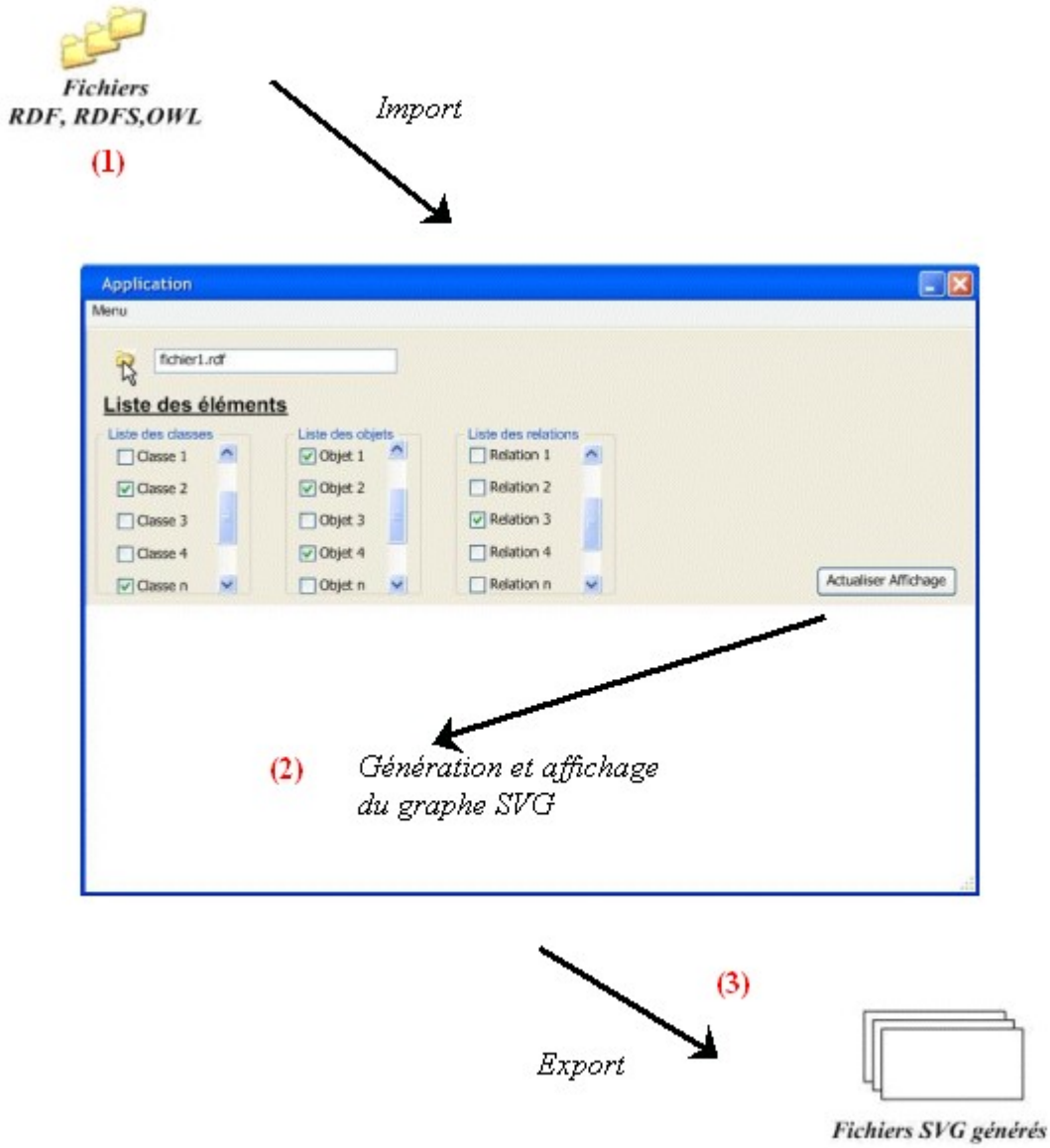
- Associer un formalisme à chaque type d'élément. Grâce à des formes simples et un système de couleurs basiques, nous offrons une vision concrète du graphe à l'utilisateur. Celle-ci fait bien la distinction entre les trois types d'éléments et le fait qu'ils aient été sélectionnés ou non.

- Elaborer un algorithme qui parcoure le graphe pour rechercher les éléments à afficher selon les choix utilisateurs. Nous utilisons un parcours en trois temps, avec des méthodes récursives, qui traite les éléments type par type et recherche les différents liens.

- Réaliser l'affichage du graphe au sein de notre interface et donner la possibilité de l'exporter sous la forme SVG. Pour cela, un outil complémentaire a été utilisé qui aide à la génération des graphiques.

2.2.Maquette

Voici la maquette de la solution que nous avons imaginée :



La phase d'analyse nous a permis de dégager notre plan d'action pour construire une solution au besoin énoncé. Voyons maintenant comment ces actions ont été mises en oeuvre afin de réaliser notre produit.

3. REALISATION

Voici maintenant la façon dont nous avons mis en œuvre notre solution proposée. Celle-ci se distingue en trois grandes parties :

- le parsage des différents fichiers d'entrée qui doit nous permettre d'extraire les données décrites et de les présenter à l'utilisateur
- la sélection par l'utilisateur des éléments à afficher afin de lui donner une certaine maîtrise de l'information et de rendre plus pertinent le graphe affiché
- l'affichage du graphe en lui-même selon un formalisme logique et intuitif, pour que de sa simple lecture se fasse la compréhension de l'information

3.1. Opérations de parsage et de tri

Comment extraire les données des fichiers ?
Il faut parser et cela implique une bonne connaissance de la structure de ces fichiers.

- Pour ce qui est du format RDF/RDFS, nous nous sommes aperçus de la complexité de gestion de ceux-ci. En effet, le formalisme diffère selon les auteurs et gérer ces différences aurait imposé de créer une structure dynamique qui aurait listé les balises pertinentes des fichiers.
  Nous nous sommes donc, faute de temps, consacré uniquement à la syntaxe abrégée de RDF. Dans celle-ci, la balise « description », qui peut faire référence à un identifiant d'élément ou à une URI, et qui rend ainsi compliqué le parsage, n'est pas utilisée. Ainsi nous avons considéré que l'identifiant d'élément était l'attribut « label », et ce pour toutes les classes, toutes les instances de classes ou de relations, toutes les propriétés, etc…

```
<kb:Personne
  rdf:about="http://www.MonVocabulaire#KB_155146_Instance_6"
  rdfs:label="Juliette"/>
```

- Pour ce qui est du format OWL, nous avons décidé de traiter les quatre balises suivantes :
  - Class qui identifie les classes
  - ObjectProperty qui permet de trouver les relations
  - SubclassOf qui définit une relation d'héritage entre deux classes
  - InverseOf qui définit l'inverse d'une relation existante, c'est à dire que la classe source devient la classe de destination et vise-versa.

```
<owl:Class rdf:about="#garcon">
   <owl:disjointWith>
     <owl:Class rdf:about="#fille"/>
   </owl:disjointWith>
   <rdfs:subClassOf>
```

```
    <owl:Class rdf:ID="personne"/>
   </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Objet_tragique"/>
  <owl:Class rdf:about="#personne">
   <rdfs:subClassOf rdf:resource="#Objet_tragique"/>
  </owl:Class>
  <owl:Class rdf:ID="famille">
   <rdfs:subClassOf rdf:resource="#Objet_tragique"/>
  </owl:Class>
……
<owl:ObjectProperty rdf:ID="habite">
    <rdfs:domain rdf:resource="#famille"/>
    <rdfs:range rdf:resource="#ville"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#a_pour_amant">
    <owl:inverseOf rdf:resource="#a_pour_amante"/>
    <rdfs:range rdf:resource="#garcon"/>
    <rdfs:domain rdf:resource="#fille"/>
  </owl:ObjectProperty>
```

Les identifiants d'élément, que ce soit le label pour RDF ou l'ID pour OWL, correspondent aux libellés qui apparaîtront sur notre interface.

Quels sont les outils et les techniques pour parser les fichiers ?

L'outil de parsage utilisé est l'API DOM (Document Object Model). C'est un analyseur syntaxique qui permet de parcourir un document XML et d'en extraire des informations. Il permet de créer une structure hiérarchique (arbre) du document en mémoire et fournit une liste de méthodes d'accès, de création, … de nœuds de l'arbre. Les fichiers en entrée sont considérés comme valides et bien formés.

Au début, nous utilisions les méthodes simples de DOM comme getChildren( ), firstChild( ), etc… Or, ces méthodes fonctionnent sur un parcours séquentiel de l'arbre, mais RDF et les autres formats décrits ne sont pas séquentiels (on peut faire une référence à une classe qui est définie plus tard dans le document). Pour pallier à ce problème, nous avons utilisé la classe Xpath qui permet de récupérer les éléments à partir d'une expression Xpath, et donc de parser selon le nom de l'élément ou par type.

De quel façon faut-il parser le ficher ?

- Pour RDF /RDFS :

Premièrement, nous parsons le fichier RDFS pour obtenir la liste des classes et leur description. A noter que les relations sont décrites de la même façon que les classes et il est à ce stade impossible de les différencier.

Lors du parsage du fichier RDF qui est la seconde étape: pour chaque classe trouvée, on vérifie l'existence de balises « rdf : from » et « rdf : to ». Si elles existent, cela signifie que l'entité courante est une relation sinon il s'agit d'un individu.

- Pour OWL :

Nous avons attribué à chaque classe une liste d'attributs. Pour alimenter cette liste en fonction du fichier OWL, nous récupérons la valeur de l'élément « domain » qui nous indique la classe à laquelle se réfère la propriété en question.
Pour les classes et les individus, nous avons utilisé une propriété « ressource ». Celle-ci est indispensable car elle permet de pallier au manque de séquentialité du fichier OWL : en effet, l'élément « resource » lors de notre parsage peut faire référence à une classe ou un individu qui n'est pas encore défini. Il nous faut donc stocker sa valeur en attendant de déclarer et de décrire l'élément ainsi référencé.

Exemple de l'utilisation de la balise « domain » :

```
<owl:ObjectProperty rdf:ID="habite">
  <rdfs:domain rdf:resource="#famille"/>
  <rdfs:range rdf:resource="#ville"/>
</owl:ObjectProperty>
```

Dans cet exemple, la classe « famille » possède la propriété « habite ».

Difficultés rencontrées
Trouver les balises pertinentes dont nous avions besoin a été la plus grosse difficulté. En effet, après avoir récupéré plusieurs fichiers d'exemple, nous nous sommes rendus compte qu'il pouvait y avoir plusieurs manières d'exprimer une contrainte.
Voici par exemple l'élément subClassOf : parfois cet élément a un attribut « resource » qui indique la classe mère :

```
<rdfs:subClassOf rdf:resource="#personne"/>
```

Mais parfois la classe mère est directement définie dans un élément Class fils :

```
<rdfs:subClassOf>
<owl:Class rdf:ID="personne"/>
</rdfs:subClassOf>
```

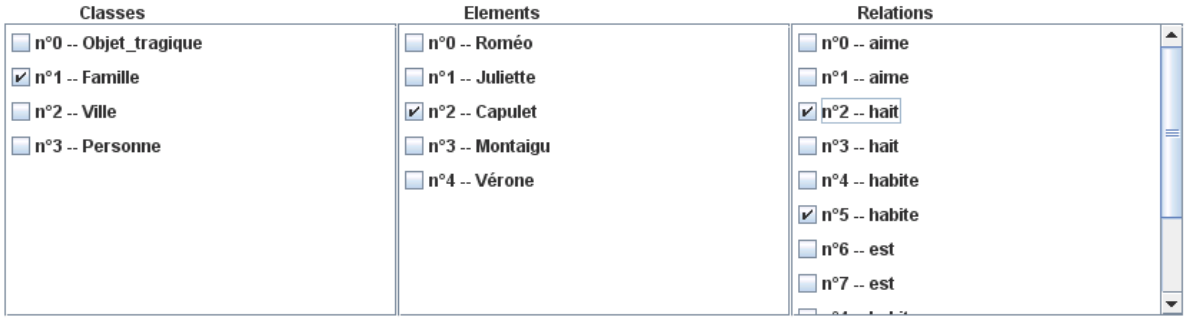
De ce fait, nous avons dû traiter énormément de cas différents lors de notre parsage, ce qui fut relativement fastidieux.

3.2.Sélection de l'information

Pourquoi sélectionner ?
Après le parsage, l'utilisateur dispose de la liste de tous les éléments répartie en trois catégories (classes, individus, relations). Les documents importés peuvent contenir peu d'information ou bien beaucoup. Une masse d'informations trop grande devient alors problématique dans la représentation et la compréhension globale du fichier.

C'est pourquoi l'application permet à l'utilisateur de sélectionner les données qui lui semblent pertinentes à représenter.

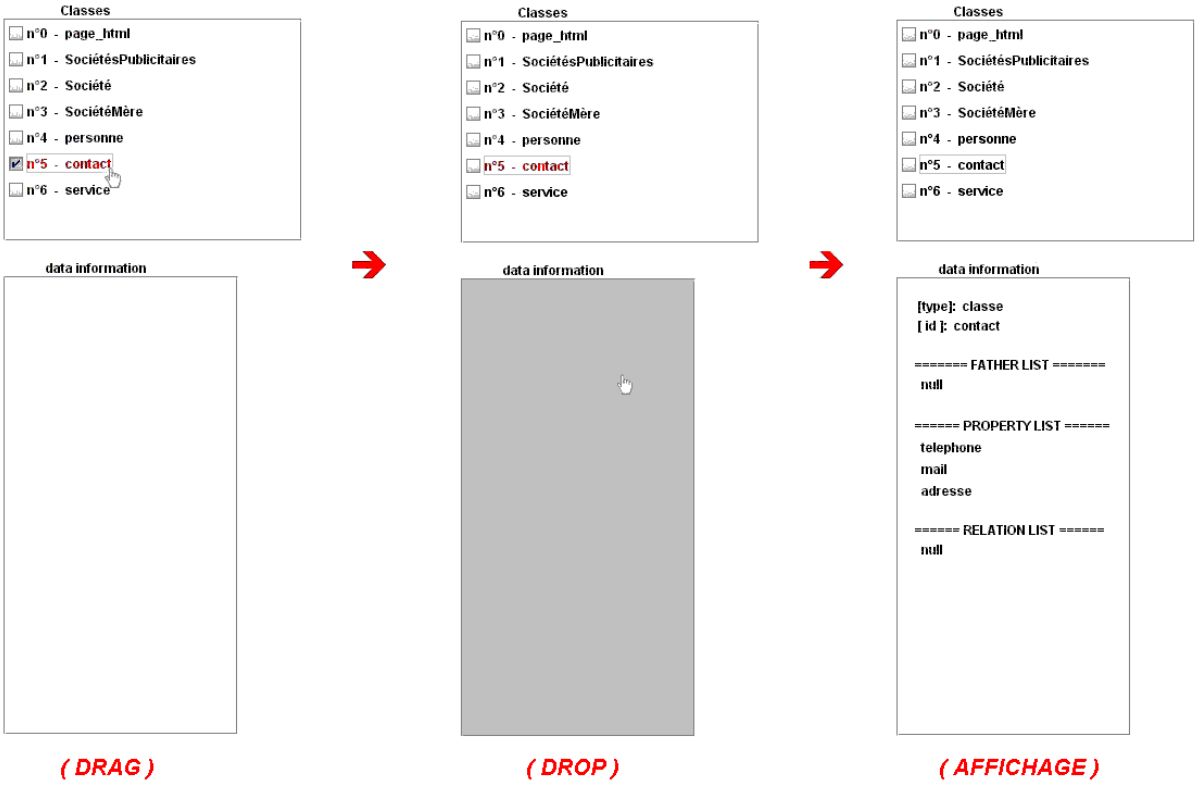
Comment permettre à l'utilisateur de sélectionner l'information ?
L'application affiche les trois listes d'éléments, une par catégorie. A chaque élément est rattaché un checkbox. La sélection d'un élément se fait par un simple coche de la case correspondant à l'élément.



Afin de guider l'utilisateur dans sa sélection, il dispose d'une boite d'information. Cette boite d'information s'utilise en faisant un drag & drop d'un élément dont il souhaite connaître le détail. Selon la catégorie de l'élément, la boite affichera des informations différentes :

- Pour une classes :
  . La liste des pères dont il a hérité de ses propriétés
  . La liste des attributs

- Pour un individu :
  . La liste des attributs avec les valeurs
  . La liste de ses voisins



L'utilisateur a aussi la possibilité d'effacer toute sa sélection et revenir au début. Pour valider sa sélection d'éléments, il lui faut appuyer sur le bouton"Edit graphic".

A cette étape du processus, l'utilisateur a effectué deux traitements sur l'ensemble des éléments. Une première opération de classification et de tri et une deuxième de filtre et sélection. La dernière étape consiste à donner une représentation visuelle à ces éléments sélectionnés.

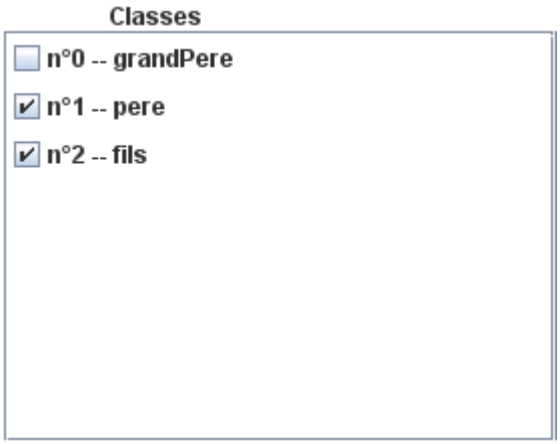
3.3. Affichage des graphes

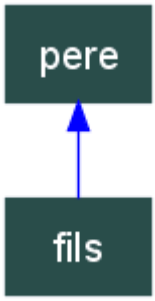
Quels types d'informations faut-il présenter à l'utilisateur ?
Après sélection par l'utilisateur d'un groupe d'éléments, il nous a semblé nécessaire de guider celui-ci en complétant la vision que le graphe allait lui offrir.

L'application affichera les relations directes existantes entre les individus sélectionnés. Elle affichera également les éléments implicites qui renforcent le sens du graphe.
Voici un récapitulatif des cas possibles :

Héritage direct entre classes

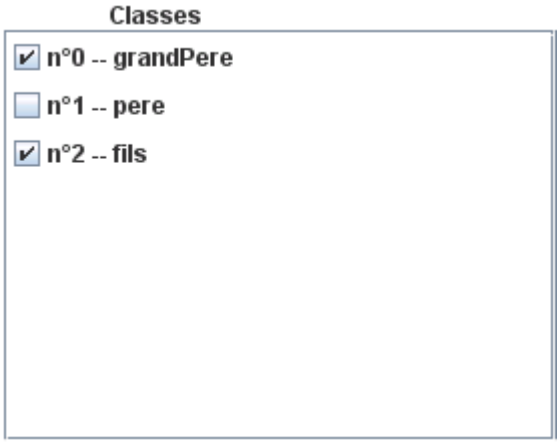
Il existe une relation d'héritage entre deux classes sélectionnées.

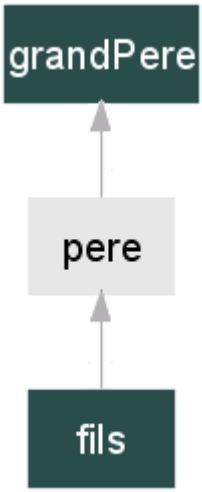




Héritage indirect entre classes

Il n'existe aucune relation d'héritage direct en deux classes sélectionnées. Mais il existe une ou plusieurs classes intermédiaires entre les deux.

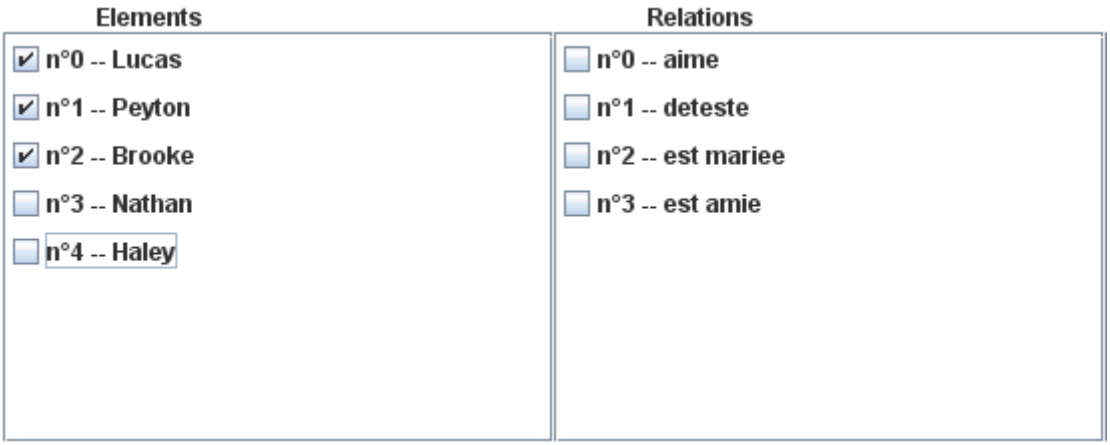


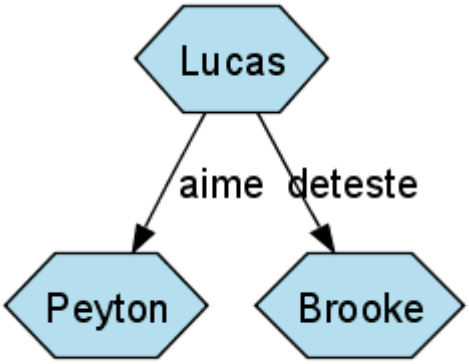


La classe intermédiaire est alors une information supplémentaire par rapport à la sélection de l'utilisateur.

Liens directs entre individus

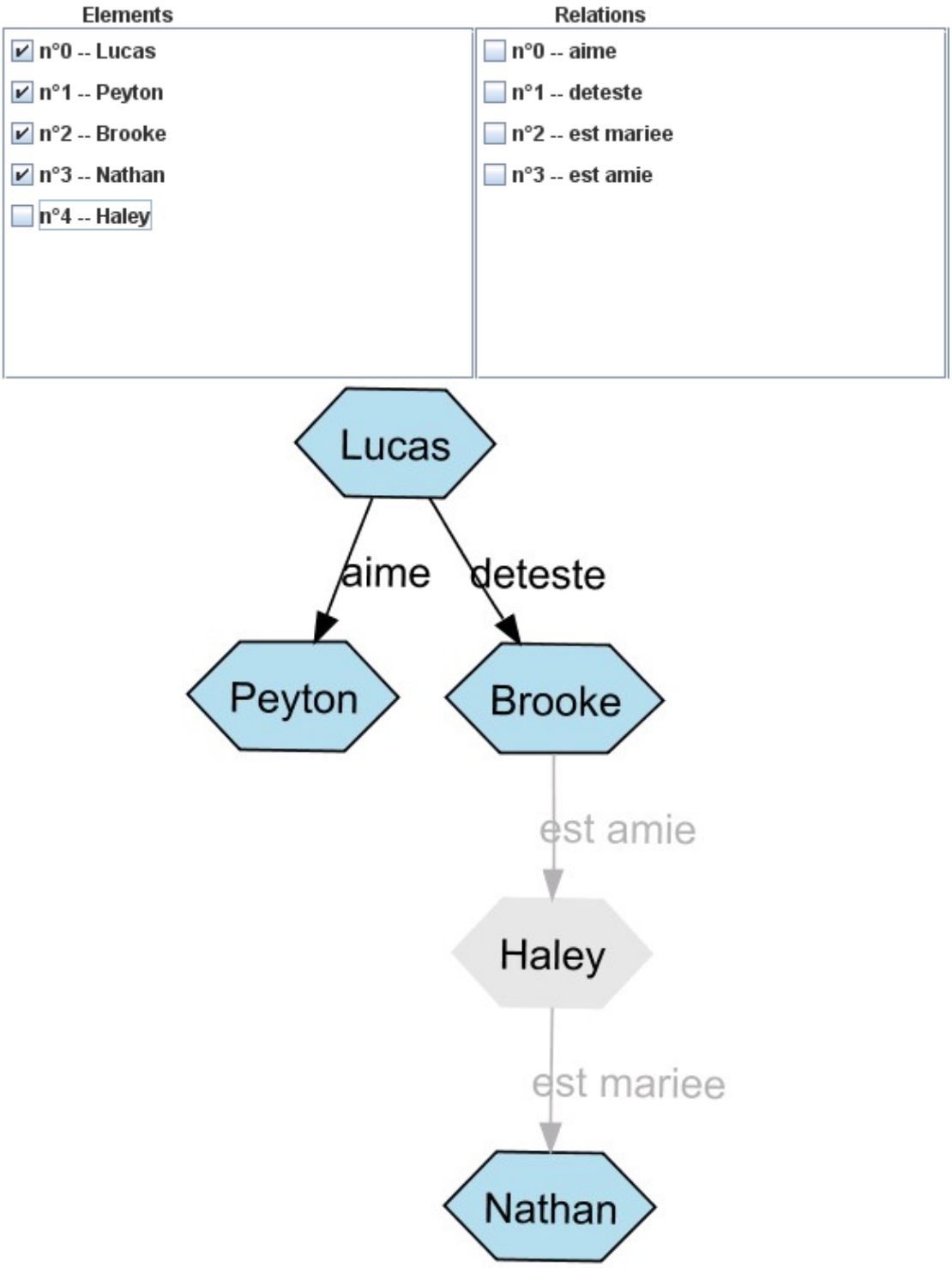
Il existe une relation entre deux classes sélectionnées.





Liens indirects entre individus

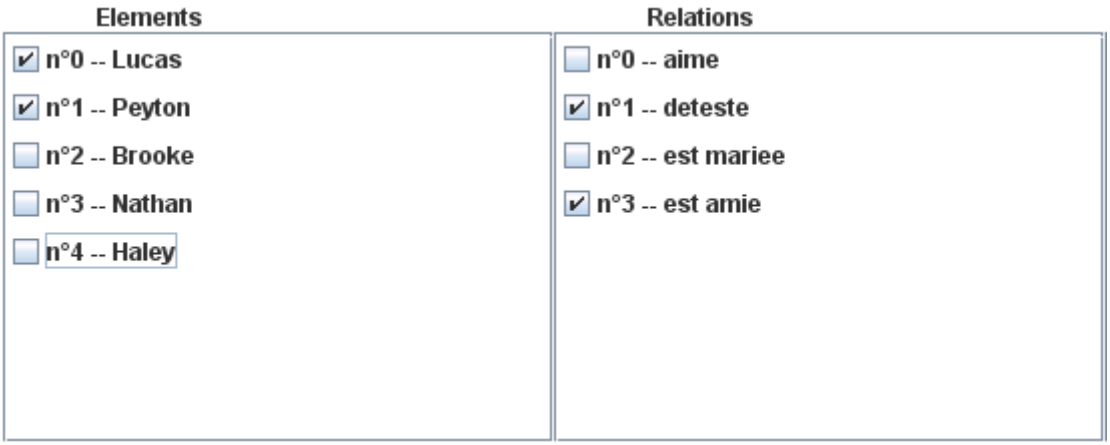
Il n'existe aucune relation directe entre deux individus sélectionnés. Mais il existe un individu potentiel entre les deux.

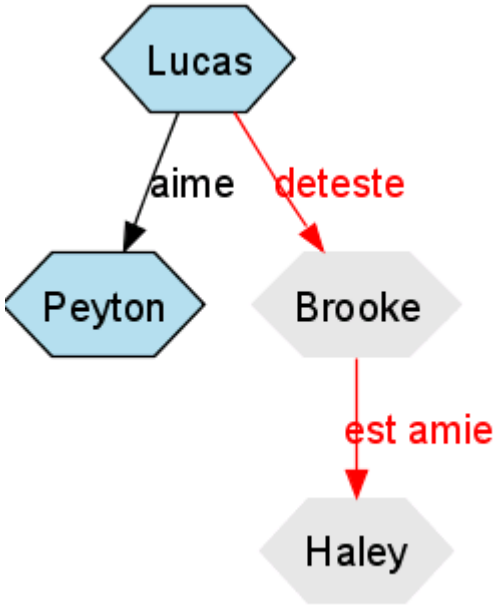


L'individu intermédiaire est alors une information supplémentaire par rapport à la sélection de l'utilisateur.

Relations complètes et partielles

Les relations sélectionnées font apparaître les individus intervenant dans le lien s'ils n'ont pas été sélectionnés.





Comment déterminer les éléments à afficher suivant la sélection de l'utilisateur?
Le contenu d'un fichier RDF/RDFS et OWL peut se schématiser sous formes de graphes. Déterminer quels sont les éléments sélectionnés par l'utilisateur et quels sont les éléments à enrichir revient à élaborer un algorithme de parcours d'arbre complexe.
Celui-ci se décompose en trois phases : il recherche d'abord les classes et leurs héritages de façon récursive, puis les individus, puis les relations en s'attardant notamment sur les sources et destinations de ces relations.
Au fur et à mesure du parcours de l'arbre, l'algorithme détermine pour chaque élément :

- s'il doit l'afficher
- si oui, quels sont les éléments potentiels qui ont un lien avec l'élément sélectionné ? Quelle représentation graphique lui attribuer selon son type ?

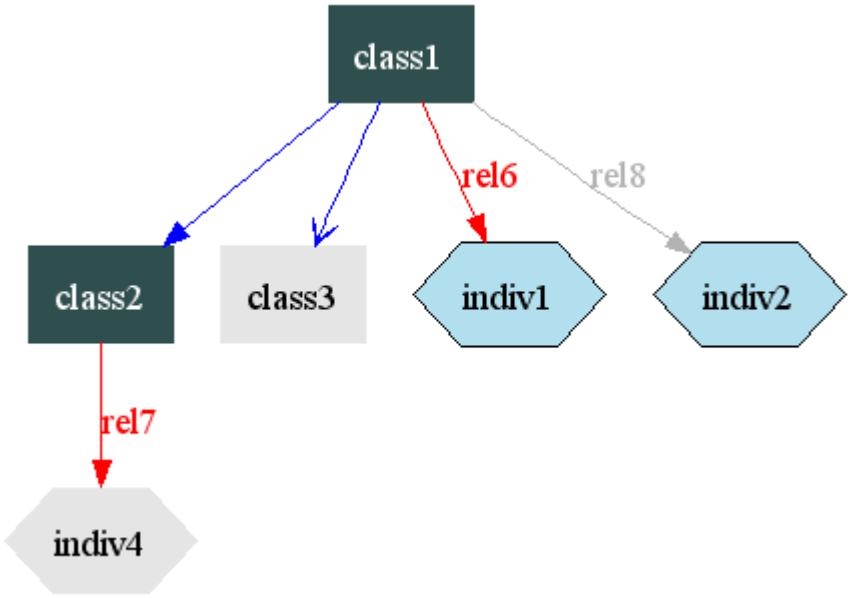
Quels sont les outils et les techniques pour organiser les graphes ?
Pour l'étape de l'affichage des graphes, nous avons d'abord pensé à dessiner dans notre interface chaque élément sélectionné, et ce, grâce à la bibliothèque d'objets graphiques Javax.Swing. Or cela s'est avéré trop difficile à gérer, notamment du point de vue de la disposition des éléments dans l'espace.

Nous nous sommes tournés vers le freeware GraphViz. Celui-ci est une solution conçue par une équipe des laboratoires de recherche de AT&T (American Telephone & Telegraph) et qui permet de représenter visuellement des graphes.
Cette solution convient à la représentation de graphes très denses comprenant un très grand nombre de nœuds, cela grâce des algorithmes très puissants. Elle est très rapide à l'exécution et le rendu est optimisé afin que les liens ne recouvrent pas les nœuds et qu'ils ne se croisent pas. De plus, ce produit permet de personnaliser le rendu des graphes par le choix des formes, couleurs et polices de caractères. Le format des fichiers d'entrée est simple et souple. Les formats de sortie sont très variés dont SVG.

Mise en œuvre : l'intégration de la solution GraphViz
L'étape suivante était de coupler GraphViz avec notre application et de générer un fichier d'entrée correct pour cet outil. En effet, GraphViz prend en entrée un format de fichier d'expression de graphes spécifique, appelé Dot, et qui est traité ensuite différemment selon le moteur de génération choisi dans l'application.
Nous avons donc étudié le formalisme de Dot pour générer nous-même un tel fichier à fournir à l'application GraphViz (cette dernière étant appelée par simple ligne de commande dans notre outil). Voici un exemple de code Dot tel que nous l'avons conçu :

```
digraph G {
//definition des noeuds classes
class1[shape=box, color=darkslategrey, fontcolor=white, style=filled];
class2[shape=box, color=darkslategrey, fontcolor=white, style=filled];
class3[shape=box, color=grey90, style=filled];
//definition des noeuds individus
indiv1[shape=polygon, sides=6, width=1, fillcolor=lightblue2, style=filled,
color=black];
indiv4[shape=polygon, sides=6, width=1, color=grey90, style=filled];
indiv2[shape=polygon, sides=6, width=1, fillcolor=lightblue2, style=filled,
color=black];
//definition des relations
class1->indiv1[label="rel6", fontcolor=red, color=red];
class2->indiv4[label="rel7", fontcolor=red, color=red];
class1->indiv2[label="rel8", fontcolor=grey70, color=grey70];
class1->class2[fontcolor=blue, color=blue];
class1->class3[fontcolor=blue, color=blue, arrowhead=open];
}
```

La syntaxe est assez intuitive et donne naissance au graphe suivant :



Exemple de graphe GraphViz

Quels sont les outils et les techniques pour afficher les graphes SVG ?

Notre projet consistait à représenter les ressources des fichiers RDF, RDFS, OWL en SVG. Or le choix d'un tel langage est facilement justifiable, puisque de type XML, il présente de nombreux avantages :

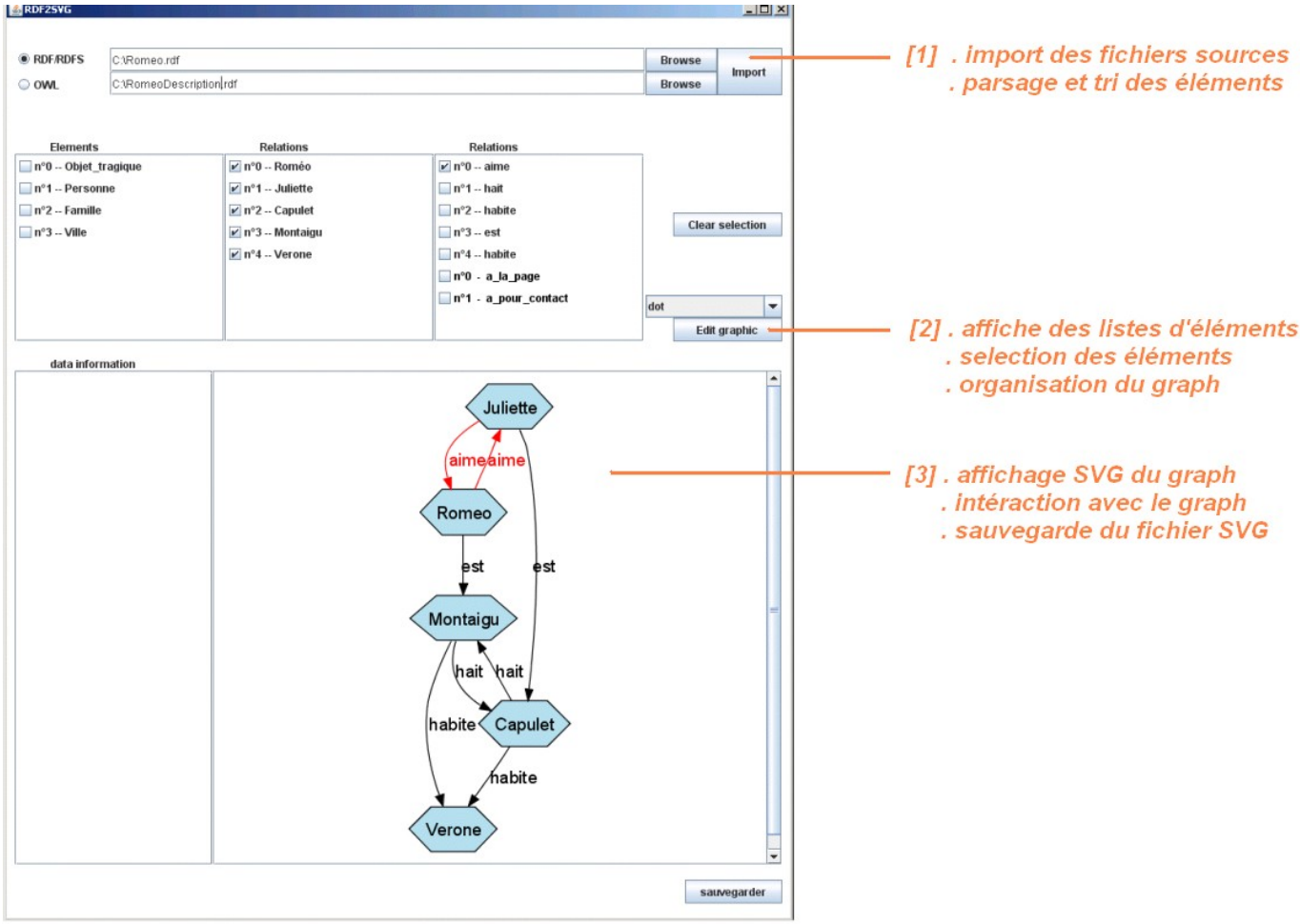
a. Représentation de type vectorielle
b. Portabilité du XML
c. Standardisé

Dans une première version de notre projet, nous devions développer nous même des méthodes permettant de transposer les éléments affichés en SVG, mais il s'est avéré plus logique

d'utiliser les fonctionnalités natives de GraphViz. Il est en effet capable de générer son graphe de sortie au format SVG, si le fichier DOT en entrée est bien formé.

Afin d'afficher un graphique SVG, il existe en java des modules de composants nommés « Batik Swing components ». Ce module fournit une classe java « JSVGCanvas » qui prend en entrée un fichier SVG (provenant d'une URI ou d'un arbre DOM) et génère en sortie un composant de type Canvas en java. De plus, en exploitant les propriétés interactives du format SVG, la classe permet de manipuler le graphique obtenu (zoom, rotation, translation, …).

3.4. Interface finale



3.5. Autres fonctionnalités

- **Interactivité**

Toujours selon un souci utilisateur, notre produit donne la possibilité d'interagir avec le graphe. Ces options sont déjà présentes grâce aux propriétés natives du graphe généré par GraphViz et aux fonctionnalités graphiques de Java. En effet, on peut par exemple zoomer ou effectuer une rotation du graphe, tout ceci facilitant sa lecture et donc sa compréhension.

- **Choix des fichiers**

Nous avons également guidé l'utilisateur dans le choix des différents fichiers et ce, pour limiter les erreurs de saisie. Nous utilisons en effet un composant java appelé JFileChooser, qui propose un fenêtre de sélection de fichier à ouvrir, afin que l'utilisateur n'ait pas à rentrer le chemin complet de son fichier RDF, RDFS ou OWL. De plus, des filtres permettent de ne faire apparaître dans ce menu que les fichiers d'extensions souhaitées et compatibles.

- **Sauvegarde SVG**

Le format SVG étant totalement portable, nous avons décidé qu'en plus de générer le graphe dans ce format, nous permettrions à l'utilisateur de sauvegarder le fichier correspondant, et ce à l'emplacement de son choix. Ainsi, il peut re-visualiser à volonté le graphe pour l'étudier ou l'exporter dans une application ou une présentation multimédia.

4. BILAN

4.1.Bilans personnels

Ce projet constituait pour nous un véritable challenge : en effet, nous devions comprendre la syntaxe et les notions des fichiers de type RDF, RDFS et OWL, mais aussi savoir générer un graphe avec le langage multimedia SVG. Or ces différents formats sont vraiment complexes et certains possèdent un haut formalisme.
L'étape importante suivante fut le développement car nous nous sommes rendus compte de la difficulté algorithmique du projet, notamment pour la disposition des éléments du graphe et pour leur affichage. Le déclic dans notre avancement fut la recherche d'outils sur le Web et le fait que nous ayons trouvé GraphViz. Cette application nous a permis de gérer la phase la plus difficile de notre outil et a en plus enrichi notre projet en ne nous limitant pas qu'au travail sur les différents formats du sujet. Sur le plan personnel, ce projet nous a fait comprendre l'importance du travail de recherche préalable à tout développement.

4.2.Améliorations possibles

- **Interaction avec le graphe**

La première amélioration qui nous vient tout de suite à l'esprit, c'est de rendre toujours plus interactif le graphe pour l'utilisateur. En plus des options déjà présentes, on peut en imaginer d'autres qui amèneraient de la sémantique et pas seulement de l'accompagnement visuel. L'idéal serait de permettre d'éditer, de supprimer, ou de déplacer les nœuds, ainsi l'utilisateur apporterait sa propre vision de l'information au graphe.

- **Choix des formes**

La deuxième amélioration possible est d'offrir à l'utilisateur le choix du formalisme utilisé pour son graphe. On peut imaginer que celui-ci puisse sélectionner les formes et les couleurs définissant chaque type d'élément (classe, individu, relation) et qui seront utilisées dans le graphe. Il suffirait de présenter un bagage de formes (et de couleurs) existantes dans le langage Dot.

- **Localisation directe d'un élément**

L'utilisateur n'aurait qu'à sélectionner l'élément de son choix pour que le graphe se positionne automatiquement dessus. Cela faciliterait la lisibilité et la recherche dans des graphes imposants.

CONCLUSION

Ce projet nous a permis d'appréhender et de manipuler RDF, RDFS, et OWL. Ces langages descripteurs de ressources sont de véritables modèles XML qui permettent à communauté d'utilisateurs d'utiliser les méta-données pour des ressources partagées. Ces syntaxes proposées par le W3C permettent de structurer l'information accessible sur le Web et de l'indexer efficacement.

De plus, outre le fait que nous ayons pu travailler sur les différents formats, nous a surtout permis de prendre conscience de complexité de la représentation des graphes. Ceci est un problème algorithmique ardu. La conception d'un programme offrant une telle fonctionnalité est une tâche de longue haleine qui requiert de fortes compétences en structures de données et algorithmique, et il est souvent utile de recourir à un programme externe à qui on délègue la génération des graphes.

Enfin, nous avons su développer un outil qui guide l'utilisateur tout au long de la création de son graphe et qui lui permet ainsi de comprendre le contenu des fichiers RDF, RDFS, et OWL. Notre produit lui offre un contrôle véritable sur l'information présente dans les différents fichiers. L'utilisateur est plus qu'un simple spectateur, mais un acteur de l'expression sémantique des ressources décrites. Grâce aux fonctionnalités proposées, nous ne lui donnons pas une solution mais sa propre solution, la réponse à son besoin.

ANNEXES

1/ pré-requis

- Une JDK égale ou supérieure à la version 6

http://java.sun.com/javase/downloads/index.jsp

- Les modules JDOM (pour le parsage)

http://www.jdom.org/dist/binary/

- Le logiciel libre GraphViz (pour la manipulation et la visualisation des graphes)

http://graphviz.org/Download.php

- Les modules « Batik Swing components » (pour l'affichage de fichier au format SVG)

http://xmlgraphics.apache.org/batik/download.cgi#distributon

2/ Installation de JDom

Pour les opérations de parsage, il est nécessaire de posséder une bibliothèque appelée JDom. La procédure d'installation spécifiée ci-dessous est décrite pour l'environnement de développement java Eclipse.

L'API JDom peut être téléchargée sur le lien suivant :
http://www.jdom.org/dist/binary/

Choisissez le fichier jdom-1.0.zip. Une fois que vous l'avez, allez sur dans les propriétés de votre projet correspondant (pour cela, effectuez un clic droit sur le nom du projet dans la fenêtre d'arborescence des classes). Puis, cliquez sur l'onglet « Java Build Path », puis sur « Add External JARs ». Sélectionnez le .jar de JDom et ajoutez-le. Par précaution, il est recommandé de relancer Eclipse juste après pour que celui-ci soit bien pris en compte.

3/ Installation de GraphViz

GraphViz est open source, gratuit et libre de droits. C'est cette application que nous utilisons pour la génération de nos graphes. Son installation est nécessaire au fonctionnement de notre produit, voici donc la procédure à suivre sous plate-forme Windows :

Téléchargez la source de GraphViz sur le site http://www.graphviz.org dans la section Download. Nous avons utilisé la version 2.12 de ce soft.
Une fois graphviz-2.12.exe récupéré, lancez l'installation.

L'installation est très simple à réaliser, il suffit de cliquer sur « Next » pour les différentes fenêtres proposées et GraphViz s'installe très rapidement dans C:\Program Files\ATT.