Rapport de stage

INTERFACE DE RECHERCHE DE LA DÉFINITION D'UN TERME,
VISUALISATION DE LA SIMILITUDE DES RÉSULTATS TROUVÉS

Réalisé par PIERRE PORTAL

Sous la direction de Dr. HÉLÈNE DE RIBAUPIERRE

Dans le cadre de l'obtention d'un DUT INFORMATIQUE

I - Introduction **2**

II - Revue de littérature **3**

1. Introduction de la revue de littérature 3
2. Présentation du web sémantique 4
3. Architecture du web sémantique 4
4. Présentation du concept de Semantic Publishing 5
5. Utopia Documents, un cas concret de publication sémantique 6
6. Le modèle SciAnnotDoc, un autre cas concret de Semantic Publishing 7
7. La problématique de la visualisation des données du web sémantique 7

III - Présentation du sujet de stage **8**

1. Introduction au sujet de stage 8
2. Le choix des critères de comparaison des définitions 8
3. L'importance de la visualisation 9

IV - Rapport technique **9**

1. Introduction du rapport technique 9
2. Algorithme 10
3. Obtention des termes similaires 11
4. Requête SPARQL, obtention des définitions et de leurs métadonnées 13
5. Extraction des définitions depuis le fichier JSON 17
6. Comparaison sémantique des définitions 19
7. Utilisation du Tagger, étiquetage grammatical des mots de chaque définition 21
8. Extraction des mots pertinents et des nom propres 22
9. Comparaison des facteurs secondaires 23
10. Calcul du score général de similarité 24
11. Traçage du graphe 26
12. Résultats 27

V - Rapport d'activité **29**

1.Organisation du travail 29
2. Méthodes de travail 29

VI - Bibliographie **30**

I - Introduction

Dans le cadre du travail de recherche universitaire, un scientifique se doit d'avoir de bons outils pour s'informer du travail des autres. Il lui est primordial de rester actualisé des dernières découvertes et innovations liées à son domaine d'étude. Il doit aussi avoir un moyen efficace de s'informer sur tout type de connaissances scientifiques. Avec la multiplication du nombre de publications scientifiques depuis plusieurs années, il semble plus que jamais nécessaire d'offrir aux chercheurs des outils de recherche élégants et efficaces au moyen, par exemple, de requêtes complexes.

Aujourd'hui, les scientifiques utilisent principalement des moteurs de recherche académiques tel que Google Scholar pour effectuer ce travail de recherche. Ces moteurs de recherche indexent les documents scientifiques par leurs métadonnées (auteur, date, …) ainsi que par les mots contenus dans le texte. Les requêtes se font par mots-clés et les résultats sont affichés en fonction d'un algorithme utilisant plusieurs paramètres comme, par exemple, le nombre de citations. Le moteur de recherche ne fait qu'indexer les documents et ceux-ci sont disponibles au téléchargement sur le site web des éditeurs scientifiques ou d'autres plateformes qui hébergent ce type de documents.

Ces documents sont disponibles au format PDF, celui-ci permet une excellente portabilité entre les machines cependant il est critiqué pour son manque de fonctionnalités. Le contenu d'un document PDF est statique, on ne peut pas interagir avec son contenu et il n'offre aucunes informations sur sa structure (introduction, titres, citation, …) ce qui le rend difficilement interprétable par un programme informatique.

On peut juger que ce modèle n'est pas suffisant et qu'il peut être améliorer pour offrir aux scientifiques de meilleurs outils de recherche et des publications plus interactives qu'un simple fichier PDF. C'est dans ce contexte actuel que de nombreux universitaires travaillent dans le domaine du "Semantic Publishing". Ce champs d'étude a débuté en 2001 avec *"The Semantic Web"* de *Tim Berners-Lee* et les problématiques qui lui sont liés ont fortement évoluées depuis.

En 2009, D.Shotton [1], donne une définition large du Semantic Publishing comme *"anything that enhances the meaning of a published journal article, facilitates its automated discovery, enables its linking to semantically related articles, provides access to data within the article in actionable form, or facilitates integration of data between papers"* [1]. C'est dans ce domaine là que travaille Dr. Hélène de Ribaupierre, avec qui j'ai eu la chance de travailler pendant 3 mois à l'Université de Cardiff.

[1] *"tout ce qui améliore le sens d'un article publié dans un journal, qui facilite sa découverte automatisée, qui permet de le lier avec des articles parlant de la même chose, qui fournit un accès aux données dans l'article sous une forme exploitable ou qui facilite l'intégration des données entre les articles"*

Après avoir présenté le domaine d'étude en détails au moyen d'une revue de littérature, en expliquant les différents enjeux et en dégageant les différentes problématiques liées au sujet, je vais par la suite expliquer le sujet de stage que j'ai choisi en présentant son utilité et les défis principaux que je vais rencontrer. Je vais enfin présenter en détails l'ensemble des démarches que j'ai du accomplir tout en expliquant le fonctionnement de la solution développée ainsi que les résultats que j'ai pu obtenir.

II - Revue de littérature

1. Introduction de la revue de littérature

Le premier mois de stage a consisté à effectuer un travail de recherche. Il était nécessaire que je m'approprie le domaine de recherche et que je comprenne toutes les facettes de celui-ci. C'est pour cela que ma tutrice m'a demandé d'effectuer une revue de littérature. Ce travail fait partie intégrante du rôle de chercheur, celui-ci doit avoir une solide compréhension de son domaine d'étude et il doit être au courant de toutes les dernières avancées sur le sujet.

J'ai d'abord étudié les méthodologies à utiliser pour permettre d'accomplir efficacement cette première tâche. J'ai utilisé différents outils comme Google Scholar, moteur de recherche pour les documents scientifiques. En tant que membre de l'université de Cardiff j'ai eu accès à leur bibliothèque numérique ce qui m'a permis d'accéder à la quasi totalité des documents que je voulais. Google Scholar permet avant tout de trouver les documents les plus populaires dans le domaine, il renseigne aussi les documents qui en citent d'autres, cela permet de naviguer facilement entre les publications scientifiques du même domaine et donc de découvrir de nouvelles connaissances. J'ai aussi utilisé le logiciel Zotero qui m'a permis de sauvegarder tous les documents intéressants que j'ai pu trouver..

En addition, j'ai utilisé un carnet dans lequel je faisais un résumé pour chaque document que je trouvais intéressant. Je renseignais l'auteur, la date de publication, les domaines abordés (disciplines), la question de recherche, les thèses avancées et mes remarques personnelles. Cela m'a permis d'avoir une vue d'ensemble sur les différents documents étudiés et ainsi me donner une idée du plan de ma revue de littérature.

Tout cela m'a permis de dégager de nombreuses problématiques liées au domaine de recherche, il m'a été alors beaucoup plus simple de trouver un objectif spécifique pour mon projet. Je vais maintenant expliquer en détail mon travail de recherche et montrer, pas à pas, par cette revue de littérature, pourquoi j'ai choisi ce sujet de stage et pourquoi cela peut être utile.

2. Présentation du web sémantique

En 2001, T. Berners-Lee et al. [2], introduit pour la première fois la notion de web sémantique. Dans l'introduction, il explique à l'aide d'une histoire concrète l'utilité de ce qu'il va présenter avant de dégager la problématique principale : la plupart du contenu sur le web est avant tout conçu pour qu'il soit compréhensible et lisible par l'homme mais il n'est pas du tout conçu pour être manipulable par les programmes informatiques. Les balises HTML ne servent qu'à décrire le contenant mais pas le contenu. On peut donner comme exemple cette ligne : <h1>Introduction</h1>. La balise <h1> permet seulement de comprendre que ce mot est un titre de catégorie 1 mais il ne donne aucune information sur la sémantique du mot qu'il contient. L'objectif de la recherche est de discuter des moyens fiables pour permettre aux programmes informatiques de traiter les données du web (données sémantiques) de la façon la plus efficace possible pour leur permettre d'opérer des tâches sophistiqués utiles à l'utilisateur. C'est là que le web sémantique entre en jeu. Celui-ci est une surcouche que l'on applique au web classique et qui permettra aux applications informatiques de manipuler facilement les données du web. Berners-Lee relève que ces données devront être structurées par des règles logiques afin de leur donner un sens et de permettre aux machines d'effectuer des raisonnements sur celles-ci. Pour cela il apparaît le besoin d'un modèle unique pour représenter les données sémantiques sur le web.

3. Architecture du web sémantique

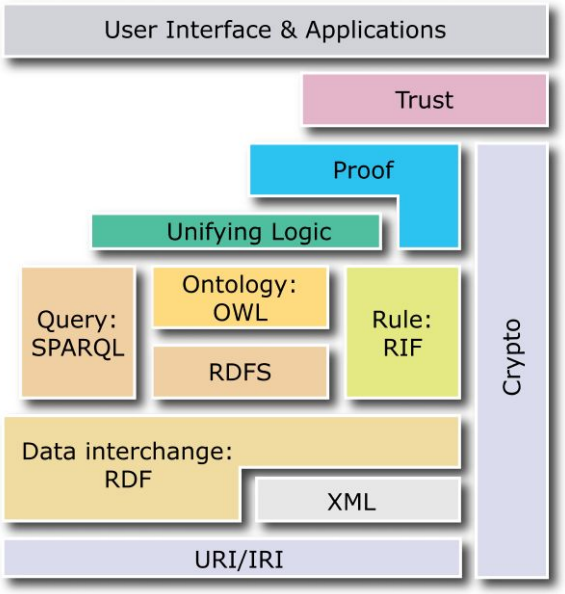


Figure 1 : Architecture du web sémantique

Le web sémantique est composé de multiples couches. Un des langages principaux est RDF, il permet de décrire les données sous forme de triplets. Un triplet est composé d'un

sujet, d'un prédicat et d'un objet. Le prédicat est obligatoirement une ressource représentée par un IRI et qui décrit la relation entre l'objet et le sujet. L'objet et le sujet sont aussi des ressources qui peuvent être représentées par un IRI ou alors par de la donnée (texte, nombre, etc …). Un IRI (International Resource Identifier) est une chaîne de caractères qui permet de décrire une ressource sur le web. Un document RDF peut s'écrire sous forme de balises XML mais d'autres formats existent comme Turtle. Ce modèle RDF/XML/IRI permet ainsi de décrire des relations simples entre les données et celles-ci peuvent être récupérées sous la forme de requêtes à l'aide du langage SPARQL. Cependant, le langage RDF n'est pas suffisant si on veut décrire des concepts de représentation de donnée plus complexes (classes, hiérarchie de classes, propriétés, etc ...). Pour cela on utilise RDFS qui est un langage qui permet la représentation de ces concepts. On utilise par dessus le langage OWL qui vient ajouter de nombreux autres concepts de description de connaissances (classes équivalentes, propriétés équivalentes, égalité, différences, contraire, symétrie, cardinalité, etc …) et qui permettent de décrire une ontologie. En 2009, T. Gruber [7] définit une ontologie comme "*the specification of a conceptualization. [...] A conceptualization is an abstract and simplified view of the domain we want to represent*"[2]. Une ontologie permet de modéliser un ensemble de connaissances dans un domaine précis ce qui sert à uniformiser la description des concepts de ce domaine.

4. Présentation du concept de Semantic Publishing

En 2001, T. Berners-Lee et al. [3] discute de l'impact du développement du web sémantique sur la publication scientifique. Il présente la nécessité de rendre les documents scientifiques compréhensibles par les machines afin d'incorporer les connaissances scientifiques au web sémantique. Cela demande l'annotation sémantique des différentes parties d'un document scientifique mais cette tâche n'est pas simple à mettre en place. Elle demanderait le développement d'outil pour permettre aux scientifiques d'accomplir cela manuellement mais cela demanderait tout de même beaucoup d'efforts. Une autre piste serait l'annotation automatique des documents scientifiques aux moyens d'un programme informatique.
Il décrit ensuite les nombreux avantages du Semantic Publishing. Celui-ci permettrait d'améliorer considérablement l'efficacité des moteurs de recherche en permettant de réaliser des requêtes beaucoup plus complexes qu'une simple recherche par mot-clé. Elle permettrait aussi de simplifier le travail des chercheurs en leur offrant des outils de partage et de modification de leurs recherches ce qui rendrait obsolète le modèle traditionnel de publication par le biais d'un éditeur scientifique en ligne. Elle rendrait la communication entre les chercheurs de domaines différents beaucoup plus simple par le développement d'outils de formalisation des données techniques. Enfin, on pourrait lier tous les concepts scientifiques entre eux pour créer un réseau de connaissance universel (Universal Web of Knowledge).

En 2009, D.Shotton [1] dresse un état des lieux depuis l'article de Berners-Lee. Il y présente les avancées concrètes qui ont été achevées ainsi que les nouvelles problématiques qui

[2] "la spécification d'une conceptualisation. […] Une conceptualisation est une vue abstraite et simplifiée du monde que l'on veut représenter"

sont apparues. Pour finir il discute des différents champs à explorer pour faire avancer le domaine. Il y définit le concept de Semantic Publishing comme *"anything that enhances the meaning of a published journal article, facilitates its automated discovery, enables its linking to semantically related articles, provides access to data within the article in actionable form, or facilitates integration of data between papers"* [3]. Il s'accorde avec l'hypothèse de Berners-Lee, le Semantic Publishing a bien une utilité concrète et il offre de réels avantages, les éditeurs scientifiques ont eux-même commencer à améliorer sémantiquement leurs publications en mettant en place différentes solutions. Bien que de nombreux outils ont été développés pour permettre l'annotation manuelle des documents scientifiques, cette tâche demande encore trop d'efforts pour les chercheurs. Il y présente la nécessité de concevoir une approche incrémentale agile d'annotation manuelle afin de ne pas les décourager. Il donne aussi la priorité au développement de solutions d'annotation automatique au moyen du "text-mining" et par l'utilisation d'ontologies pour permettre la détection des termes scientifiques liés au domaine en question.

5. Utopia Documents, un cas concret de Semantic Publishing

En 2011, S. Pettifer et al. [4] aborde les avantages et inconvénients des différents formats de publication scientifique avant de présenter une solution concrète permettant d'améliorer l'interactivité d'un document.
Les documents scientifiques sont souvent de simples fichiers textes et ne sont pas suffisamment structurés pour permettre une classification claire des données qu'ils contiennent (annotation). C'est un problème car de plus en plus de documents scientifiques sont publiés et il faut permettre aux scientifiques d'avoir des outils rapides et efficaces pour rechercher des informations précises dans cette multitude de nouveaux documents. Bien souvent les fichiers sont au format PDF, le contenu est statique et il n'y a pas de renseignements sur la structure du document ce qui rend compliqué l'interprétation du document par un programme informatique.
On peut penser que les auteurs eux-même devrait rendre leur publication "machine processable" mais cela demanderait trop d'efforts de leur part, surtout qu'une publication scientifique est "une histoire qui persuade avec de la donnée". Il est impossible d'enlever aux scientifiques cet outil essentiel qu'est la rédaction personnelle de leurs travaux..
C'est là qu'on se rend compte de la nécessité du Semantic Publishing, il est nécessaire de trouver des moyens d'analyser automatiquement ces documents afin de les annoter.
L'article pointe du doigt le format de fichier PDF, celui-ci est utilisé dans 80% des cas cependant il présente énormément d'inconvénients : il n'est pas suffisamment riche en fonctionnalités, il est très difficile d'y extraire et annoter des données, il présente des formats incohérents, etc …
L'article présente un nouvel outil : Utopia Documents. C'est un logiciel qui visualise un PDF tout en analysant son contenu. Quand des informations analysables sont trouvées (terme

[3] *"tout ce qui améliore le sens d'un article publié dans un journal, qui facilite sa découverte automatisée, qui permet de le lier avec des articles parlant de la même chose, qui fournit un accès aux données dans l'article sous une forme exploitable ou qui facilite l'intégration des données entre les articles"*

technique, citation, référence), celles-ci sont annotées automatiquement ce qui rend le contenu beaucoup plus interactif. A ces annotations on lie des définition, des citations et des références, ce qui aide grandement le lecteur de la publication. D'autres fonctionnalités sont aussi implémentées pour permettre de rendre le fichier PDF le plus interactif possible.

6. Le modèle SciAnnotDoc, un autre cas concret de Semantic Publishing

En 2017, H. de Ribaupierre et G. Falquet [5] adressent la problématique de la multiplicité des publications scientifiques et la nécessité d'offrir aux chercheurs des outils pour rechercher efficacement les informations contenues dans ces documents.
Pour commencer, ils présentent une ontologie qui décrit l'architecture d'un document scientifique (SciAnnotDoc). Celle-ci est utilisée pour classifier les différentes parties de documents scientifiques. Cette classification est faite au moyen d'un programme informatique qui annote automatiquement les différents éléments de discours dans les documents.
Ils ont dégagé 5 types d'éléments de discours dans l'ontologie : definition, finding, methodology, hypothesis et related work (si l'élément provient d'autres travaux). Ceux-ci ont été sélectionnés par rapport aux besoins spécifiques des chercheurs suite à deux études par questionnaire.
Les auteurs ont alors développé une interface de recherche qui est basé sur le modèle et qui permet aux scientifiques d'accomplir des requêtes plus complexes qu'un simple recherche par mot-clef. Au moyen d'expérimentations, ils ont démontré que leur interface était plus efficace qu'une interface de recherche classique.

7. La problématique de la visualisation des données du web sémantique

En 2011, A.-S. Dadzie et M. Rowe [6] présentent les différentes approches pour visualiser les données du web sémantique (linked data). Ils adressent d'abord les différentes problématiques qui expliquent la complexité de visualiser correctement ces données. Enfin ils présentent un état des lieux des solutions déjà existantes.
"Une image vaut mille mots", la visualisation des données est essentielle, elle améliore grandement la compréhension et facilite l'analyse et la découverte de connaissances. Cependant, il n'existe pas un moyen définitif de représenter les données du web sémantique, cette représentation doit se faire en fonction de l'utilisateur-type pour qu'elle soit le plus compréhensible possible par celui-ci. Il existe de nombreux types de visualisations : graphe, nuage de points, navigateur texte, etc … Et ces visualisations peuvent utiliser différents outils pour améliorer leur interactivité : exploration dynamique, requêtes complexes, etc ... Ainsi, il existe de nombreuses solutions concrètes mais celles-ci ne sont cantonnées qu'à un usage bien spécifique.

III - Présentation du sujet de stage

1. Introduction au sujet de stage

Comme vu précédemment, la solution développée par Hélène de Ribaupierre et Gilles Falquet [5] permet d'effectuer des requêtes en fonction des éléments de discours sélectionnés. Cependant, la requête ne prend pas en compte les termes similaires au terme recherché. Cela est problématique car le chercheur pourra passer à côté d'un résultat pertinent uniquement parce qu'un synonyme du terme recherché a été utilisé dans la publication. Un autre problème est que les résultats ne sont pas hiérarchisés. Ils sont affichés ligne par ligne, sans vraiment d'ordre, alors qu'il existe nécessairement des similarités et des différences entre les différents résultats obtenus. Il semble alors nécessaire de développer une visualisation qui prendra en compte différents critères afin de mettre en évidence les relations entre les différents résultats obtenus.

J'ai donc décidé de travailler sur ce sujet en me concentrant uniquement sur la visualisation d'un seul type d'élément de discours : les définitions. La tâche principal est de développer un modèle qui permettra de visualiser les relations entre plusieurs définitions au moyen de différents critères. Il me faudra ensuite lier le modèle à l'ontologie SciAnnotDoc pour que je puisse y extraire directement les définitions et leurs métadonnées à l'aide de requêtes SPARQL. Enfin, je pourrais développer une interface qui analysera d'abord le terme recherché pour trouver les termes similaires afin ensuite de visualiser l'ensemble des définitions trouvées.

2. Le choix des critères de comparaison des définitions

Tout le défi de ce travail est de rendre la visualisation pertinente. Pour cela, il faut que les critères que j'utilise pour comparer les définitions aient un sens. Le premier critère, celui qui semble le plus pertinent, est la comparaison de la sémantique des différentes définitions : si on trouve que deux définitions veulent dire la même chose, alors elles ont une forte similarité, il faudra donc placer ces définitions côte-à-côte dans la visualisation.

La comparaison sémantique de phrases, de mots, de documents, reste un domaine encore fortement étudié par de nombreux chercheurs. Il n'était pas dans le cadre de mon travail de développer à partir de rien un algorithme de comparaison mais d'étudier les différentes solutions qui ont été mises en place et de trouver celle qui me semble la plus appropriée pour mon modèle.

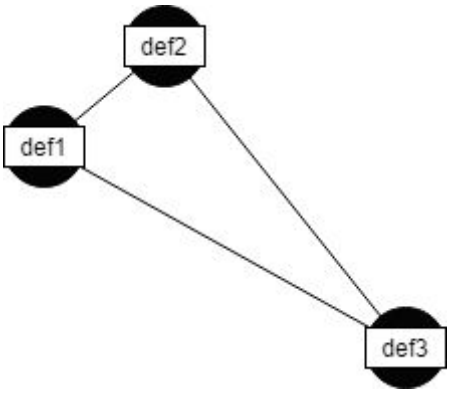
D'autres critères pertinents peuvent être utilisés pour comparer les définitions entre elles : les auteurs, les écoles de pensée, les dates de publication, les publications et les auteurs cités … On considère que ces critères sont secondaires par rapport à la comparaison

sémantique, c'est pourquoi on leur attribuera un poids moins important quand on viendra à calculer le score général de similarité entre deux définitions.

3. L'importance de la visualisation

Un autre facteur important pour rendre la visualisation pertinente est le choix du type de visualisation que l'on va utiliser. Comment visualiser la similitude de différentes définitions entre elles ? On sait d'abord qu'on aura une liste de plusieurs définitions, et qu'on aura calculé un score de similitude pour chaque paire de définitions.

L'idée est d'utiliser un graphe simple dans lequel on va créer un sommet pour chaque définition. On va ensuite lier tous ces sommets entre eux et on va attribuer un poids à chacune des arêtes créées. Le poids d'une arête sera égal au score de similitude des deux définitions qui sont représentées par les deux sommets de l'arête.



Visualisation de la similitude de 3 définitions

Plus le poids d'une arête est élevé, plus sa taille sera réduite, cela permettra donc de visualiser la similitude des différentes définitions selon la distance qu'elles auront entre elles dans le graphe. Si la distance entre deux définitions est grande, cela veut dire que leur score de similitude est faible.

Dans le diagramme ci-dessus, le score de similarité entre def1 et def2 est élevé et les scores de similarité entre def1 et def3, ainsi que def2 et def3, sont faibles.

IV - Rapport technique

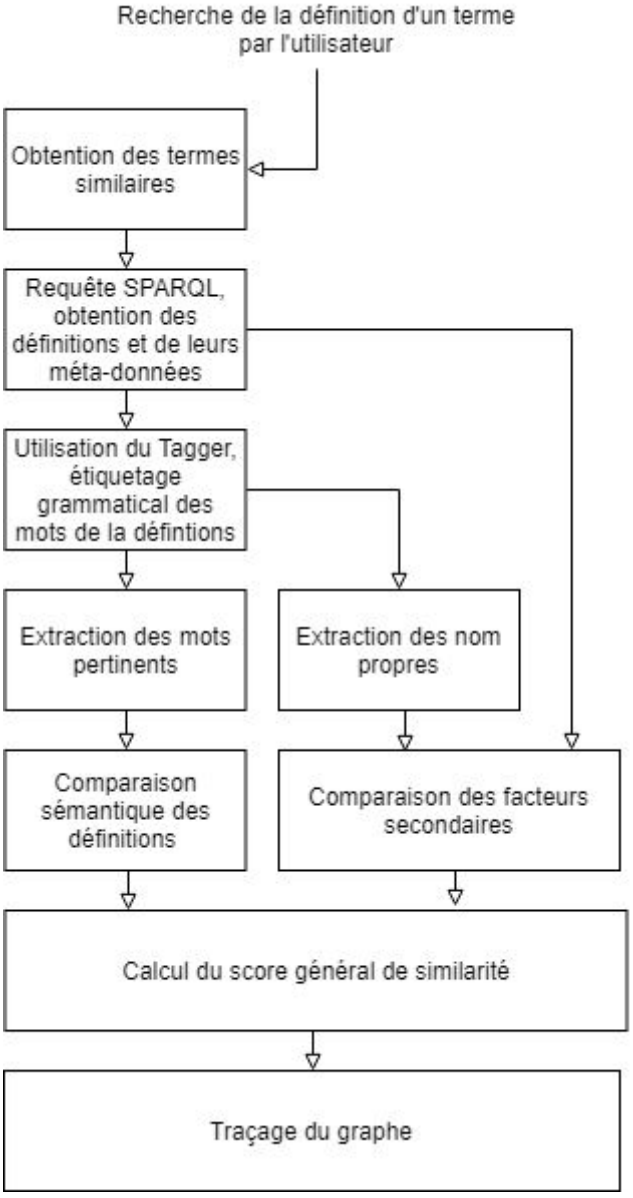
1. Introduction du rapport technique

Le développement de la solution a été réalisé avec le langage de programmation Java. D'abord parce que c'est le langage que j'ai le plus étudié mais aussi parce que c'est celui

par lequel j'ai pu trouver le plus de librairies adaptées au projet. J'ai utilisé l'environnement de développement Eclipse couplé avec Maven qui est un outil de gestion de projet. Il m'a permis, en particulier, d'installer très facilement de nombreuses librairies sans se soucier des dépendances. L'explication de l'utilisation des différentes librairies se fera tout au long du rapport technique.

La solution peut se diviser en plusieurs étapes. Bien que je vais les expliquer en détails dans les différentes parties de ce rapport technique, je vais d'abord présenter un résumé du fonctionnement de l'application tout en justifiant l'utilité des différentes étapes.

2. Algorithme



On part d'un ensemble de définitions extraites automatiquement depuis des documents scientifiques (PDF) utilisant l'application développée dans [5] et qui sont stockées dans une ontologie. On commence par la requête d'un terme recherché par l'utilisateur. La première étape consiste à obtenir les termes similaires au terme recherché, cela permettra d'obtenir plus de résultats lorsqu'on fera la requête.

On a maintenant une liste de termes. Il est de temps d'effectuer la requête SPARQL sur l'ontologie pour chaque terme. Cette étape permet d'obtenir la liste des définitions ainsi que leurs métadonnées (auteurs, date, auteurs et publications cités). On garde ces métadonnées pour les utiliser lors de la comparaison des facteurs secondaire.

On utilise alors un processus de traitement automatique du langage permettant d'extraire les mots de la définition et d'annoter leur type grammaticale correspondant (nom, verbe, adjectif, …). L'objectif est d'identifier les mots qui nous serviront lors de la comparaison sémantique ainsi que les nom propres.

Une fois qu'on a extrait les mots pertinents pour chaque définition, on peut commencer par les comparer avec ceux des autres définitions. On obtient un score de comparaison sémantique que l'on va coupler avec les scores de comparaison des facteurs secondaires. Ceux-ci sont calculés par la comparaison des noms propres contenus dans la définition et des métadonnées préalablement extraites.

On finit par tracer le graphe en utilisant le score obtenu pour chaque paire de définition en l'appliquant à l'arête correspondante.

3. Obtention des termes similaires

Pour rechercher les termes similaires à celui recherché par l'utilisateur, on utilise la base de données lexicale WordNet. Celle-ci répertorie l'ensemble des relations qui peuvent exister entre les mots de la langue anglaise. WordNet fonctionne avec des unités atomiques appelées les synsets ("sets de synonymes"). Chaque synset est une liste d'un ou plusieurs mots qui veulent dire la même choses, sachant qu'un mot peut appartenir à plusieurs synsets du fait qu'il existe plusieurs acceptions pour décrire un mot.

Par exemple, en recherchant le mot "family" sur WordNet, on obtient cette liste de synsets avec, pour chacun d'eux, la définition correspondante :

1. family, household, house, home, menage -- (a social unit living together)
2. family, family unit -- (primary social group; parents and children)
3. class, category, family -- (a collection of things sharing a common attribute)
4. family, family line, folk, kinfolk, kinsfolk, sept, phratry -- (people descended from a common ancestor)
5. kin, kinsperson, family -- (a person having kinship with another or others)
6. family -- ((biology) a taxonomic group containing one or more genera)
7. syndicate, crime syndicate, mob, family -- (a loose affiliation of gangsters in charge of organized criminal activities)
8. family, fellowship -- (an association of people who share common beliefs or activities)

Chaque synset est aussi lié à de nombreux autres synsets par le biais de relations sémantiques. Le nombre de ces relations est trop important pour toutes les décrire mais on peut citer par exemple :

- les antonymes, les concepts opposés (ex: chaud et froid)
- les hyperonymes, concepts dont le sens inclut celui d'autres mots plus spécifiques (ex: animal est l'hyperonyme de chien, chat, oiseau)

Nous utilisons WordNet dans la solution avec la librairie JWI (the MIT Java WordNet Interface) https://projects.csail.mit.edu/jwi/api/index.html [8], celle-ci offre une multitude de commandes permettant d'interagir efficacement avec la base de données. Avant de commencer, il est d'abord nécessaire de télécharger le dictionnaire WordNet en sélectionnant la version la plus récente car c'est celle la plus fiable.

Pour initialiser le dictionnaire dans l'environnement Java, on crée un objet *IDictionary* et on lui passe comme un argument l'URL du dictionnaire. On appelle ensuite la fonction *open*() de l'objet *IDictionary*.

```
String path = "C:\\Chemin\\Vers\\Le\\Dictionnaire";

URL url = null;

try{
    url = new URL("file", null, path);
}
catch(MalformedURLException e){
    e.printStackTrace();
}
if(url == null)
    return;

IDictionary dict = new Dictionary(url);
dict.open();
```

On peut maintenant utiliser la fonction *getTermesSimilaires()* à laquelle on passe comme argument l'objet *IDictionary* et le terme recherché.

```
public static List<String> getTermesSimilaires(IDictionary dict, String terme){

        List<String> ts = new ArrayList<String>();

        IIndexWord idxWord = dict.getIndexWord(terme, POS.NOUN);

        for(IWordID wordID: idxWord.getWordIDs()) {
                IWord word = dict.getWord(wordID);
                ISynset synset = word.getSynset();
                for (IWord w : synset.getWords()) {
                        String lemma = w.getLemma().replaceAll("_"," ");
                        if (!ts.contains(lemma)) {
                                ts.add(lemma);
                        }
                 }
        }
        return ts;
}
```

On initialise d'abord la liste de *String ts* qui contiendra tous les termes similaires qu'on aura trouvé. La fonction *dict.getIndexWord(terme, POS.NOUN)* permet d'obtenir l'objet *IIndexWord* lié au terme recherché, cet objet contient l'index de tous les mots dont l'intitulé est le même que celui du terme recherché. Le second argument *POS.NOUN* spécifie que le

terme recherché est un nom commun. A l'aide d'une boucle for, de la fonction *getWordIDs* et d'autres fonctions get, on parvient à obtenir tous les synsets de tous les mots qui étaient contenus dans l'objet *IIndexWord*. Avec la fonction *getWords()* que l'on appelle sur chaque synset on peut alors obtenir l'intitulé de chaque mot de chaque synset. Si un intitulé n'est pas présent dans la liste ts, alors on l'ajoute puis on renvoie la liste *ts*.

Au final, cette fonction permet de renvoyer l'ensemble des synonymes d'un mot pour toutes les acceptions qui existent pour ce mot. Cela peut être problématique car on va obtenir des termes similaires au terme recherché qui n'ont peut être rien à voir avec le concept recherché par l'utilisateur.

Par exemple, si on applique l'algorithme avec le terme "family", on obtient la liste de termes suivante :

family, household, house, home, menage, family unit, class, category, family line, folk, kinfolk, kinsfolk, sept, phratry, kin, kinsperson, syndicate, crime syndicate, mob, fellowship

Tout dépend de l'intention de l'utilisateur, mais il semble improbable que celui-ci cherche à connaître la définition de "crime syndicate" en recherchant le mot "family".

On pourrait explorer encore plus en détails cette partie et il existe quelques pistes qui permettrait de pallier à ce problème. Dans un premier temps, il serait intéressant de développer un algorithme qui permettrait de sélectionner le synset correspondant au concept recherché par l'utilisateur. Dans un deuxième temps, on pourrait étoffer le résultat en recherchant des termes similaires en utilisant la multitude de relations sémantiques que WordNet propose. Cependant, cette partie n'étant pas l'objectif principal du projet, j'ai décidé de la mettre de côté pour me concentrer sur des parties plus importantes.

4. Requête SPARQL, obtention des définitions et de leurs métadonnées

L'ontologie qu'on m'a mis à disposition pour travailler est hébergée sur une base de données triplestore qui tourne avec la plateforme Stardog. Celle-ci propose une API Java permettant d'interagir avec elle. L'ontologie est une version simplifiée du modèle SciAnnotDoc et qui contient uniquement des définitions.

Chaque définition possède :

- un prédicat "rdfs:comment" qui pointe vers le String de la définition
- un prédicat "#written_by" qui pointe vers le ou les auteurs qui ont écrit le document qui contient la définition. Chaque auteur possède aussi un prédicat "rdfs:comment" qui pointe sur le nom de l'auteur.
- un prédicat "#Year" qui pointe vers la date de publication du document

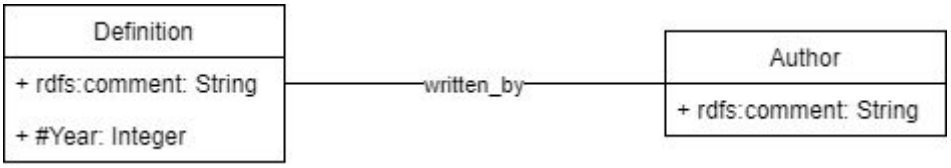
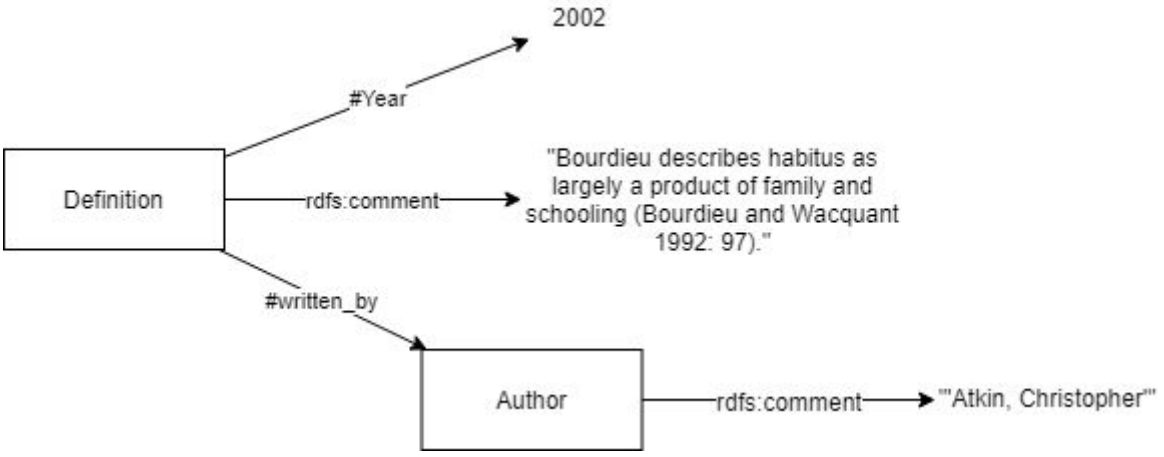


Diagramme de classe simplifié de la définition



Instance d'une des définitions du mot habitus

A partir de ces informations, on va construire une requête SPARQL qui va rechercher le terme dans toutes les définitions de l'ontologie puis qui va renvoyer, pour chaque définition trouvée : le *String* de la définition, le ou les auteurs, et l'année de publication.

```
select distinct ?s ?n ?d {
        ?f rdfs:comment ?s.
        ?s <http://jena.hpl.hp.com/ARQ/property#textMatch> 'terme recherché'.
        ?f <http://cui.unige.ch/~deribauh/annotDocuScientific#written_by> ?author.
        ?f rdf:type <http://cui.unige.ch/~deribauh/annotDocuScientific#Definition>.
        ?author rdfs:comment ?n.
        ?f <http://cui.unige.ch/~deribauh/annotDocuScientific#Year> ?d
}
```

On utilise le prédicat "#textMatch" pour rechercher le terme dans le String de la définition. Ainsi, chaque fois que le terme recherché est contenu dans une définition, la requête renvoie :

- ?s, le String de la définition
- ?n, le ou les nom du ou des auteurs
- ?d, l'année de publication

SPARQL Results (returned in 38 ms)

s	n	d
While the habitus is the product of early experiences received from family and school, Bourdieu argues that it is continually 're-structured by individuals' encounters with the outside world' (Di Maggio, 1979 cited in Reay et al, 2007:434).	'Adewumi, Barbara	2015
Bourdieu describes habitus as largely a product of family and schooling (Bourdieu and Wacquant 1992: 97).	'Atkin, Christopher'	2002
'Habitus' is used by Bourdieu to refer to durable patterns of thought and behaviour, resulting from the internalisation of culture or objective social structures, i.e. the norms and practices and dispositions of particular social classes or groups, created and shaped by the interaction between structures (education, family, class), 'fields' and personal histories (Bourdieu & Passeron, 1977).	'Clegg, S	2009
Their narratives remind us too that Bourdieu's concept of habitus is one that is subject to the myriad influences of individuals' family, friends and home.	'Davey, Gayna'	2009
This is a concept from bourdieu (1986) and refers to language use, skills, competencies, and orientations of perceptions (or habitus) that a child is endowed with by virtue of socialization in his or her family and community.	Lee, King Siong	2012
This is a concept from bourdieu (1986) and refers to language use, skills, competencies, and orientations of perceptions (or habitus) that a child is endowed with by virtue of socialization in his or her family and community.	Lee, Su Kim', "Azizah Ya'acob,"	2012
This is a concept from bourdieu (1986) and refers to language use, skills, competencies, and orientations of perceptions (or habitus) that a child is endowed with by virtue of socialization in his or her family and community.	'Wong, Fook Fei	2012

1 - 16 50 Per Page Page 1

Résultat de la requête précédente en utilisant l'interface Stardog et en recherchant le mot "family" dans l'ontologie

Une fois la requête SPARQL construite, il suffit de l'intégrer à l'environnement Java. Pour cela il faut d'abord se connecter au serveur en utilisant l'API Stardog :

```
Connection aConn = ConnectionConfiguration
                        .to("allSentences")
                        .server("http://10.72.99.109:5820/")
                        .credentials("username", "password")
                        .connect();
```

Ensuite on crée un objet *SelectQuery* qui contiendra la requête SPARQL, on considère que l'objet *t* est un *String* qui représente le terme recherché :

```
SelectQuery aQuery = aConn.select(
"select distinct ?s ?n ?d"
+ "{" +
"  ?f rdfs:comment ?s." +
" ?s <http://jena.hpl.hp.com/ARQ/property#textMatch> '\"" + t.replaceAll("""", "\\\\\"") + "\"'." +
"  ?f <http://cui.unige.ch/~deribauh/annotDocuScientific#written_by> ?author." +
"  ?f rdf:type <http://cui.unige.ch/~deribauh/annotDocuScientific#Definition>." +
"  ?author rdfs:comment ?n." +
"  ?f <http://cui.unige.ch/~deribauh/annotDocuScientific#Year> ?d" +
"}"
);
```

Puis on exécute la requête.

```
TupleQueryResult aResult = aQuery.execute();
```

On obtient un objet *TupleQueryResult* qui contient les résultats de la requête. On écrit ces résultats dans un fichier JSON qu'on intitule *t+".json"*. Les résultats sont convertis en JSON à l'aide de la fonction *QueryResultIO.writeTuple()*.

```
TupleQueryResult aResult = aQuery.execute();

try {
        File file = new File(t+".json");
        FileOutputStream fop = null;
        try {
                fop = new FileOutputStream(file);
        } catch (FileNotFoundException e1) { e1.printStackTrace(); }
        QueryResultIO.writeTuple(aResult, TupleQueryResultFormat.JSON, fop);
        fop.flush();
        fop.close();
}
catch (TupleQueryResultHandlerException e) { e.printStackTrace(); }
catch (QueryEvaluationException e) { e.printStackTrace();}
catch (UnsupportedQueryResultFormatException e) { e.printStackTrace(); }
catch (IOException e) { e.printStackTrace(); }
```

Le choix du JSON comme format de données s'explique par le fait qu'il n'existe pas de moyen de convertir un objet *TupleQueryResult* en un objet dont les données sont accessibles. Ainsi, pour rendre les résultats de la requêtes accessibles, on est obligé de les écrire dans un fichier JSON auquel on pourra accéder par la suite

Exemple d'un fichier JSON résultant d'une requête qui a obtenu un seul résultat :


```
{ "head" : { "vars" : ["s","n","d"]},
  "results" : {
    "bindings" : [
     {
       "s" : {
         "type" : "literal",
         "value" : "While the habitus is the product of early experiences received from family and school, Bourdieu argues that it is continually 're-structured by individuals' encounters with the outside world' (Di Maggio, 1979 cited in Reay et al, 2007:434)."
       },
       "n" : {
         "type" : "literal",
         "value" : "'Adewumi, Barbara"
       },
       "d" : {
         "type" : "literal",
         "value" : "2015"
       }
     }
    ]
   }
}
```


5. Extraction des définitions depuis le fichier JSON

On va maintenant accéder au fichier JSON et on va extraire ces données en utilisant la librairie JSON.simple. Pour cela on utilise un Parser JSON qui permet de convertir le texte brut du fichier JSON en objet compatible avec Java.

```
JSONParser parser = new JSONParser();
JSONArray a = null;
try {
        a = (JSONArray) parser.parse(new FileReader("fichier.json"));
} catch (FileNotFoundException e) { e.printStackTrace(); }
catch (IOException e) { e.printStackTrace(); }
catch (ParseException e) { e.printStackTrace(); }
```

On a maintenant un objet *JSONArray* (*a*) duquel on pourra accéder à toutes les données.
On commence par créer 3 listes : *definitions*, *authors*, *dates*.
On crée une boucle for qui permet d'itérer sur chaque résultat de requête obtenu (*binding*).

Pour chaque binding, on accède aux données qui lui sont liés (s, n, d) et on les attribue aux listes correspondantes (definitions, authors, dates).

Notons que l'objet *authors* est une liste de listes de *String* car il est possible qu'une définition ait plusieurs auteurs. Cela se traduit dans le fichier JSON par une duplication des bindings pour une définition avec seulement le nom de l'auteur qui change. Il est alors nécessaire dans la solution de détecter quand une définition a plusieurs auteurs. Pour se faire, on vérifie, pour chaque définition extraite, si celle-ci est déjà contenue dans la liste des définitions. Si c'est le cas, on ajoute uniquement l'auteur dans la liste de *String* de l'objet authors à l'index de la définition.

```
List<String> definitions = new ArrayList<String>();
List<ArrayList<String>> authors = new ArrayList<ArrayList<String>>();
List<String> dates = new ArrayList<String>();

JSONObject results = (JSONObject) a.get("results");
JSONArray bindings = (JSONArray) results.get("bindings");

for(int b = 0; b < a.size(); b++) {
	JSONObject JSONBinding = (JSONObject) bindings.get(b);
	JSONObject s = (JSONObject) JSONBinding.get("s");
	JSONObject n = (JSONObject) JSONBinding.get("n");
	JSONObject d = (JSONObject) JSONBinding.get("d");
	String value = (String) s.get("value");
	String name = (String) n.get("value");
	String date = (String) d.get("value");
	Boolean first = true;
	for(int c = 0; c < definitions.size(); c++) {
		if(value.equals(definitions.get(c))){
			authors.get(c).add(name);
			first = false;
		}
	}
	if(first) {
		definitions.add(value);
		ArrayList<String> auts = new ArrayList<String>();
		auts.add(name);
		authors.add(auts);
		dates.add(date);
	}
}
```

6. Comparaison sémantique des définitions

Il existe des API en ligne, tel que Cortical.io ou Dandelion, qui permettent de comparer la sémantique de deux phrases entre elles. Ces outils sont très puissants et proposent des résultats très convaincants cependant leur utilisation est limitée, ces solutions ne sont pas ouvertes et proposent des abonnements payants si on voulait utiliser leur service dans notre solution. À notre connaissance, il n'existe pas encore de librairie libre et gratuite permettant la comparaison sémantique de phrases.

Cependant, nous avons pu trouver de nombreuses librairies permettant la comparaison sémantique de mots. Nous avons utilisé WS4J (WordNet Similarity for Java) qui utilise WordNet, la base de donnée lexicale que nous avons vu précisément, et qui propose des outils de calcul de similarité entre les mots.

Puisqu'on ne peut pas comparer les phrases en tant que tel, on va plutôt comparer chaque mot de chaque phrase un par un. On va ensuite additionner les résultats que l'on va diviser par le nombre de comparaison effectuées (nombre de mot de la première définition * nombre de mots de la seconde définition).

Pour cela, il est nécessaire d'extraire les mots pertinents de chaque définition et de supprimer les "stop words". On ne veut pas que le calcul du résultat final soit biaisé par la comparaison de mots qui sont trop communs et qui ne sont pas nécessaire dans le calcul du score. Nous expliquerons ces étapes dans une autre partie, il est donc nécessaire de considérer que nous avons déjà préalablement extrait les mots pertinents des définitions.

WS4J propose une multitude d'algorithmes différents et qui n'offrent pas toujours les mêmes résultats. Nous avons mené des expérimentations pour chacun des ces algorithmes au moyen d'une matrice montrant le résultat obtenu pour chaque comparaison de mots. Au terme de ces expérimentations, nous avons sélectionné l'algorithme qui nous semblait le plus efficace.

Exemple d'une des expérimentations, comparaison des deux listes de mots suivantes :

- "bird kid car"
- "animal boy plane"

Matrice des résultats obtenus pour chaque algorithme :

WuPalmer :

0.84	0.67	0.42
0.78	0.8	0.43
0.48	0.43	0.69

Resnik :

4.74	1.82	1.37
1.82	1.9	1.37
1.37	1.37	5.5

JiangConrath :

0.46	0.1	0.83
0.13	0.11	0.09
0.11	0.09	0.26

Lin :

0.81	0.27	0.18
0.33	0.29	0.19
0.23	0.2	0.74

LeacockChodrow :

2.3	1.6	1.25
2.08	2.08	1.05
1.20	1.05	1.5

Path :

0.25	0.12	0.12
0.2	0.2	0.08
0.08	0.07	0.11

On commence par l'initialisation de la base de données WordNet puis on crée un objet *RelatednessCalculator* avec la fonction *JiangConrath*() qui désigne l'algorithme de comparaison que nous allons utiliser. Enfin on exécute la fonction *getSimilarityMatrix*() en passant en arguments les deux listes de mots des deux définitions que l'on veut comparer.

```
ILexicalDatabase db = new NictWordNet();
RelatednessCalculator rc = new JiangConrath(db);
String[] array1 = words_def1.toArray(new String[words_def1.size()]);
String[] array2 = words_def2.toArray(new String[words_def2.size()]);
double[][] matrice = getSimilarityMatrix(array1, array2, rc);
```

Cette fonction renvoie une matrice des résultats (result) obtenus par la comparaison de chaque mot de chaque définition. Pour se faire, on crée deux boucles imbriquées qui vont itérer l'index des mots de chaque liste (*i* et *j*). A chaque itération, on appelle la fonction *calcRelatednessOfWords*() avec en arguments les deux mots à comparer (*words1[i]* et *words2[j]*), elle renvoie un score de similarité (entre 0 et 1) qu'on ajoute à l'index (*i,j*) correspondant dans la matrice (*result[i][j]*).

```
double[][] getSimilarityMatrix(String[] words1, String[] words2, RelatednessCalculator rc)
{
   double[][] result = new double[words1.length][words2.length];
   for ( int i=0; i<words1.length; i++ ){
      for ( int j=0; j<words2.length; j++ ) {
         double score = rc.calcRelatednessOfWords(words1[i], words2[j]);
         result[i][j] = score;
       }
      }
   return result;
 }
```

Une fois qu'on obtient la matrice, on peut additionner tous ses termes pour enfin diviser ce résultat et obtenir une moyenne. Il faut noter que la fonction *calcRelatednessOfWords*() renvoie un nombre infini quand on compare deux mots de même orthographe, c'est pourquoi il faut rajouter une condition lors du calcul du résultat afin d'éviter d'ajouter un nombre infini à celui-ci. Dans ce cas, on choisi plutôt d'ajouter la valeur 2 au total.

```
double total= 0;
for(int ii = 0; ii < array1.length; ii++) {
        for(int yy = 0; yy < array2.length; yy++) {
                if(matrice[ii][yy] > 1) {
                        total += 2;
                }else {
                        total += matrice[ii][yy];
                }
        }
}
double resultat = total/(array1.length*array2.length)
```

7. Utilisation du Tagger, étiquetage grammatical des mots de chaque définition

Une fois qu'on a pu extraire les définitions et qu'on les a placées dans une liste de String, on utilise un Tagger qui est un processus de Traitement Automatique du Langage qui permet d'extraire les mots de la définition et d'annoter leur type grammaticale correspondant (nom, verbe, adjectif, …). Cela va permettre d'analyser les mots de la définition pour qu'on puisse isoler uniquement ceux qui nous intéressent. En effet, nous avons vu précédemment que l'algorithme de comparaison sémantique utilise uniquement les mots qui font sens dans la définition, il est alors nécessaire de supprimer ce qu'on appelle les "stop words". Ce sont des mots qui sont trop communs et qui ne sont pas discriminants, c'est à dire qu'ils ne permettent pas de distinguer les textes par rapport aux autres. On va uniquement extraire

les noms (NN), les verbes (V) et les adjectifs (JJ). Il est aussi nécessaire d'isoler les noms propres car ceux-ci devront être comparés séparément lorsqu'on s'occupera des facteurs secondaires de comparaison.

On utilise le Tagger présent dans l'API Stanford NLP [9], qui propose une multitude d'outils qui sont liés au domaine du Traitement Automatique du Langage qui vise à rendre interprétable le langage naturel par les machines.

On commence par initialiser le Tagger avec un objet *MaxentTagger* en mettant en argument le chemin vers un fichier .tagger qui est un modèle de Tagger qui a déjà était entraîné. On peut maintenant utiliser la fonction *tagString*() de l'objet *MaxentTagger* pour étiqueter chaque définition une par une.

```
MaxentTagger tagger = new MaxentTagger("taggers/english-left3words-distsim.tagger");

String raw_def = null;
String tagged_def = null;

List<String> tagged_defs = new ArrayList<String>();

for(int i = 0; i < definitions.size(); i++) {
        raw_def = definitions.get(i);
        tagged_def = tagger.tagString(raw_def);
        tagged_defs.add(tagged_def);
}
```

Résultat de l'utilisation du Tagger sur la définition suivante :
"Bourdieu describes habitus as largely a product of family and schooling (Bourdieu and Wacquant 1992: 97)."

```
"Bourdieu_NNP describes_VBZ habitus_NN as_IN largely_RB a_DT product_NN of_IN
family_NN and_CC schooling_NN -LRB-_-LRB- Bourdieu_NN and_CC Wacquant_JJ
1992_CD :_: 97_CD -RRB-_-RRB- ._. "
```

8. Extraction des mots pertinents et des nom propres

On a maintenant un objet *tagged_defs* qui est une liste de toutes les définitions et dont chaque mot a été étiqueté. Comme il est dit précédemment, il est nécessaire d'extraire uniquement les noms (NN), les verbes (V) et les adjectifs (JJ) dans une liste (*words_defs_lists*). Et dans une autre liste (*NNP_defs_lists*), il est nécessaire d'extraire uniquement les noms propres (NNP).

L'étiquetage a consisté à accoler à chaque mot le caractère '_' et le symbole qui caractérise le type grammatical du mot (ex: NN, JJ, V, ...). Pour extraire les mots que l'on veut, on va

analyser caractère par caractère le contenu de chaque définition et détecter quand le motif apparaît.

Le partie de code qui correspond à cette étape est volumineuse mais pourtant plutôt facile à expliquer, c'est pourquoi nous utiliserons du pseudo-code pour décrire le mécanisme :

```
Pour chaque définition_taggée dans tagged_defs :
        Pour chaque mot dans définition_taggée :
                Si mot contient "_NNP" :
                Alors : ajouter mot à liste_des_noms_propres
                Sinon si mot contient "_NN" OU "_V" OU "_JJ" :
                Alors : ajouter mot à liste_des_mots_pertinents
        Ajouter liste_des_noms_propres à NNP_defs_lists
        Ajouter liste_des_mots_pertinents à words_defs_lists
```

On a maintenant deux listes :

- *NNP_defs_lists* qui contient les listes de noms propres de chaque définition
- *words_defs_lists* qui contient les listes des mots pertinents de chaque définition

9. Comparaison des facteurs secondaires

Les facteurs secondaires regroupent les métadonnées que l'on a préalablement extraites depuis l'ontologie grâce à la requête SPARQL ainsi que les noms propres présents dans les définitions. Pour l'instant, la solution propose seulement la comparaison deux facteurs secondaires, les noms propres et les auteurs de la définition.

Pour chaque comparaison de définitions, on va calculer un *NNPScore* (score de nom propre). Chaque fois qu'un nom propre est présent dans les deux définitions, on incrémente de 1 le *NNPScore*.

```
int calculerNNPScore(ArrayList<String> NNP_def1, ArrayList<String> NNP_def2){
        int NNPScore = 0;
        for(String nnp1 : NNP_def1) {
                for(String nnp2 : NNP_def2) {
                        if(nnp1.contains(nnp2) || nnp2.contains(nnp1)) {
                                NNPScore += 1;
                        }
                }
        }
        return NNPScore;
}
```

On fait la même chose pour calculer un *AuthorScore*. A chaque fois qu'un des auteurs est similaire, on incrémente de 1 le *AuthorScore*.

```
int calculerAuthorScore(ArrayList<String> authors1, ArrayList<String> authors2){
        int AuthorScore = 0;
        for(String author1: authors.get(i)) {
                for(String author2: authors.get(y)) {
                        if(author1.equals(author2)) {
                                authorsScore += 1;
                        }
                }
        }
        return AuthorScore;
}
```

10. Calcul du score général de similarité

Le score général de similarité se calcule en couplant le score de comparaison sémantique avec les scores de comparaison des facteurs secondaires. On a vu que le score de comparaison sémantique est la donnée la plus importante, c'est donc celle qui doit avoir le plus de poids dans le résultat final.

On a 3 scores :

- le score de comparaison sémantique, qui a une valeur entre environ 0.001 et 0.5
- le *NNPScore*, un entier de 0 ou plus
- le *AuthorScore*, un entier de 0 ou plus

Pour coupler ces 3 scores il faut normaliser chaque score pour qu'il soit égal à une valeur entre 0 et 1. Ensuite il suffira de choisir le poids de chaque score en le multipliant par un pourcentage. Enfin, on pourra additionner ces résultats pour obtenir un résultat final. Pour normaliser, il est d'abord nécessaire de calculer les valeurs maximales et minimales pour chaque liste de scores.

```
int calcMax (ArrayList<Integer> scores){
        int max = 0;
        for(int score: scores){
                if(score > max){
                        max = score;
                }
        }
        return max;
}
```

```
int calcMin(ArrayList<Integer> scores){
        int min= scores.get(0);
        for(int score: scores){
                if(score < min){
                        min = score;
                }
        }
        return min;
}
```

La fonction qui permet de normaliser une valeur entre 0 et 1 est une fonction affine qui respecte ces conditions :

- f(Xa = min) = (Ya = 0)
- f(Xb = max) = (Yb = 1)

On cherche a avec la formule :

a = (Yb - Ya) / (Xb - Xa)
= (1 - 0) / (max - min)
= 1 / (max - min)

Puis on cherche b avec la formule :

b = Ya - a * Xa
= 0 - 1 / (max - min) * min
= - 1 / (max - min) * min

Ainsi, pour normaliser une valeur entre 0 et 1, on lui applique la fonction :

f(x) = 1 / (max - min) * x - 1 / (max - min) * min

```
double normaliser(int val, int min, int max){
        return 1 / (max - min) * val - 1 / (max - min) * min
}
```

Il suffit maintenant d'additionner les scores, qu'on aura multiplié avec des pourcentages, pour obtenir un résultat final (score de similarité général). On obtient le code suivant :

```
//definition des pourcentages
double POIDS_AS = 0.10;
double POIDS_NNPS = 0.10;
double POIDS_R = 0.80;

//calcul max et min des AuthorScores
int as_max = calcMax(AuthorsScores);
int as_min = calcMin(AuthorsScores);

//calcul max et min des NNPScores
int npps_max = calcMax(NNPScores);
int nnps_min =   calcMin(NNPScores);

//calcul max et min des scores de comparaison sémantique
double r_max = calcMax(resultats);
double r_min = calcMin(resultats);

List<double> scores_finaux = new ArrayList<Double>();

for (int h = 0; h < resultats.size(); h++){
        score_finaux.add(
                normaliser(AuthorsScores.get(h), as_min, as_max) * POIDS_AS
                + normaliser(NNPScores.get(h), nnps_min, nnps_max) * POIDS_NNPS
                + normaliser(resultats.get(h), r_min, r_max) * POIDS_R); }
```

11. Traçage du graphe

On utilise la librairie GraphStream pour tracer le graphe, son fonctionnement est plutôt simple et elle propose un affichage dynamique du graphe, ce qui veut dire qu'on peut interagir avec celui-ci dans la fenêtre d'affichage.

On commence par initialiser le graphe, et on crée un sommet pour chaque définition :

```
System.setProperty("org.graphstream.ui.renderer",
"org.graphstream.ui.j2dviewer.J2DGraphRenderer");
Graph graph = new SingleGraph("DefGraph");

for(int i = 1; i < definitions.size() + 1; i++) {
        Node d = graph.addNode("definition "+i);
        d.addAttribute("ui.label", "definition "+i);
}
```

Il faut maintenant créer des arêtes pour lier chaque définitions entre elles. GraphStream propose la fonctionnalité d'ajouter un poids (*layout.weight*) à chaque arête, ce qui permet de déterminer sa taille.

```
Edge e = graph.addEdge("edge1","definition1","definition2");
e.setAttribute("layout.weight", 2);
```

On sait qu'on va appliquer le score général de similarité au poids de chaque arête correspondant à la comparaison des deux définitions représentées par les sommets de l'arête.

Plus la valeur du poids est élevé, plus la taille de l'arête sera grande. On dira que 5 est la valeur la plus haute possible pour le poids d'une arête. On appliquera 5 (ou des valeurs qui s'y rapprochent) aux arêtes dont le score général de similarité correspondant est bas.

On fera l'inverse pour les arêtes dont le score correspondant est élevé et on dira que 0.1 est la valeur la plus basse possible pour le poids d'une arête.

Pour obtenir le poids d'une arête à partir du score général de similarité, il faut de nouveau utiliser une fonction affine qui respecte ces conditions :

- f(Xa = min) = 5
- f(Xb = max) = 0.1

Avec min et max qui sont les valeurs maximales et minimales dans la liste des scores (*scores_finaux*).

On cherche a avec la formule :

a = (Yb - Ya) / (Xb - Xa)
= (0.1 - 5) / (max - min)
= -4.9 / (max - min)

Puis on cherche b avec la formule :

b = Ya - a * Xa
= 5 - (-4.9 / (max - min) * min)

On obtient la fonction :

f(x) = -4.9 / (max - min) * x + 5 - (-4.9 / (max - min) * min)

Et on a le code suivant :

```
double min = calcMin(scores_finaux);
double max = calcMax(scores_finaux);

int c = 0;

for(int b = 1; b <= definitions.size(); b++) {
        for(int z = b + 1; z <= definitions.size(); z++) {
                Edge e = graph.addEdge(""+b+z,"definition "+b,"definition "+z);
                double result = score_finaux.get(c);
                result =  -4.9/(max-min) * result + 5 - (-4.9/(max-min) * min);
                e.setAttribute("layout.weight", result);
                c++;
        }
}
```

Il suffit maintenant d'afficher le graphe.

```
graph.addAttribute("ui.quality");
graph.addAttribute("ui.antialias");
Viewer viewer = graph.display();
```

12. Résultats

Au final, on obtient une solution qui fonctionne. En faisant quelques expérimentations on peut observer que la visualisation est correcte. Si on compare des définitions qui n'ont rien à voir entre elles, alors elle seront éloignés dans la visualisation. En revanche, si ces définitions sont similaires, on les verra proches dans la visualisation.

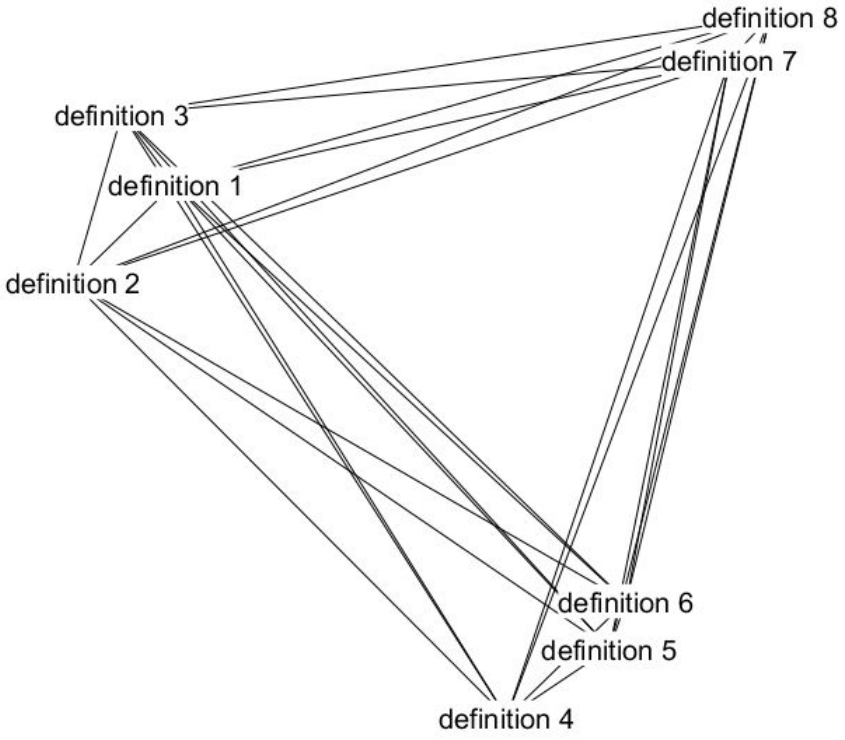
Exemple d’une expérimentation, comparaison de cette liste de définitions :

```
{
        // DÉFINITIONS DU MOT NIHILISM
        "d1" : "the rejection of all religious and moral principles, in the belief that life is meaningless",
        "d2" : "the philosophical viewpoint that suggests the denial or lack of belief towards the reputedly meaningful aspects of life",
        "d3" : "belief that all values are baseless and that nothing can be known or communicated",

        // DEFINITIONS DU MOT BOTTLE
        "d4" : "a narrow-necked container as compared with a jar",
        "d5" : "a container for liquids, usually made of glass or plastic, with a narrow neck",
        "d6" : "a rigid or semirigid container typically of glass or plastic having a comparatively narrow neck or mouth and usually no handle",

        // DEFINITIONS DU MOT TENNIS
        "d7" : "game played by two or four players on a rectangular court. The players use an oval racket with strings across it to hit a ball over a net across the middle of the court.",
        "d8" : "game in which two opposing players (singles) or pairs of players (doubles) use tautly strung rackets to hit a ball of specified size, weight, and bounce over a net on a rectangular court."
}
```

En faisant tourner l’algorithme de comparaison, on obtient le graphe suivant :



Comme on pourrait s’y attendre, les définitions qui définissent le même concept se retrouvent côtes-à-côtes et elles sont éloignées de celles qui sont différentes.

V - Rapport d’activité

1.Organisation du travail

Ce stage se déroule dans l’enceinte de l’Université de Cardiff. En tant que stagiaire, on m’a fait part d’une carte universitaire pour que je puisse accéder aux salles d’étude.

Je travaille dans deux bâtiments :

- le Queen’s Building dans lequel se trouve la School of Computer Science, je m’y rends pour travailler dans la salle commune et dans laquelle sont mis à disposition des ordinateurs.
- la Friary House dans lequel se trouve le Crime and Security Research Institute qui me permet d’accéder à des bureaux en open-space et qui offrent un cadre plus spacieux et confortable que la salle commune du Queen’s Building.

Avec ma tutrice, Dr Hélène de Ribaupierre, nous nous sommes mis d’accord au début du stage sur l’organisation de celui-ci. Nous avons décidé de se voir une fois par semaine, et il m’était demandé de rédiger un rapport mensuel sur mes travaux. Nous gardions un contact permanent par email en cas de questionnement sur le travail que j’effectuais.

A chaque rendez-vous, je lui faisais part de mon avancement et elle m’aidait sur les interrogations que je pouvais avoir. Souvent, nous fixions un objectif pour la semaine afin d’être sûre que le projet avance.

2. Méthodes de travail

On peut opposer les méthodes de travail utilisées dans ce stage avec celles du cadre industriel.. En effet, ce stage s’est effectué dans le cadre d’un travail de recherche. Au début du développement, notre idée de la solution était encore floue. Ils nous a fallu expérimenter, essayer, comparer, divaguer, changer de direction, pour obtenir un résultat.

Il m’a été accordé une grande autonomie pour ce stage ce qui m’a permis d’être libre d’expérimenter et de découvrir de nombreux autre domaines. D’un autre côté il fallait que je me fixe des objectifs par moi-même et que je ne perde pas le fil de l’objectif principal.

Ce qui est aussi différent avec le cadre industriel, c’est que j’ai dû m’approprier intégralement le domaine d’étude. Il m’a fallu comprendre parfaitement le contexte avant que je ne commence à choisir un sujet de stage. Ce travail s’est fait au moyen d’une revue de littérature et il m’a fallu apprendre différentes méthodologies de recherche. Cette tâche m’a permis de me rendre compte de la multiplicité des problématiques et des différents champs d’étude à explorer. Aussi, il m’a fallu apprendre de nouveaux concepts et de nouveaux langages comme OWL, RDF ou SPARQL.

VI - Bibliographie

[1] D. Shotton, « Semantic publishing: the coming revolution in scientific journal publishing », *Learned Publishing*, vol. 22, n° 2, p. 85‑94, avr. 2009.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” *Scientific American*, vol. 284, no. 5, pp. 34--43, 2001.

[3] T. Berners-Lee et J. Hendler, « Publishing on the semantic web », *Nature*, vol. 410, p. 1023, avr. 2001.

[4] S. Pettifer, P. McDERMOTT, J. Marsh, D. Thorne, A. Villeger, et T. K. Attwood, « Ceci n’est pas un hamburger: modelling and representing the scholarly article », *Learned Publishing*, vol. 24, n° 3, p. 207‑220, juill. 2011.

[5] H. de Ribaupierre et G. Falquet, « Extracting discourse elements and annotating scientific documents using the SciAnnotDoc model: a use case in gender documents », *International Journal on Digital Libraries*, août 2017.

[6] A.-S. Dadzie et M. Rowe, « Approaches to Visualising Linked Data: A Survey », p. 34.

[7] T. Gruber, « Ontology », in *Encyclopedia of Database Systems*, L. LIU et M. T. ÖZSU, Éd. Boston, MA: Springer US, 2009, p. 1963‑1965.

[8] M. A. Finlayson, « Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation », p. 8.

[9] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, et D. McClosky, « The Stanford CoreNLP Natural Language Processing Toolkit », in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, p. 55–60.

Résumé :

Développement d’une interface de recherche de définition prenant en compte les termes similaires au terme recherché. Obtention des définitions et de leurs métadonnées depuis une ontologie au moyen d’une requête SPARQL. Visualisation d’un graphe représentant la comparaison de la similarité entre tous les résultats.

Mots-clés : web sémantique, semantic publishing, ontologie, RDF, OWL, SPARQL, NLP, POS Tagger, JSON, similarité sémantique, visualisation, WordNet

Summary:

Development of a definition search interface taking into account related terms to the one being searched. Get the definitions and their metadata from an ontology using a SPARQL request. Visualization with a graph representing the comparison of the similarity between all the results.

Keywords : semantic web, semantic publishing, ontology, RDF, OWL, SPARQL, NLP, POS Tagger, JSON, semantic similarity, visualization, WordNet