

UNIVERSITÉ DU QUÉBEC À MONTRÉAL

COMPOSITION ASSISTÉE DE REQUÊTES SUR DES DONNÉES LIÉES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
IMEN SARRAY

JUIN 2019

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.07-2011). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

REMERCIEMENTS

Au terme de cette maîtrise, je tiens à remercier vivement mon directeur de recherche, Dr Aziz Salah, pour sa patience sans égale, ses nombreuses suggestions et son appui inconditionnel tout au long la réalisation de ce projet de recherche. Qu'il trouve ici toute l'expression de ma gratitude et de ma profonde reconnaissance.

Je désirerai aussi remercier les professeurs qui liront ce mémoire. Je leur suis reconnaissante du temps qu'ils ont accordé pour lire ce travail.

Ce travail résulte également d'un grand soutien familial. Un immense merci à mes chers parents, à mon mari et mes frères, pour leur soutien inconditionnel et sans limites, leur confiance en moi et leurs encouragements sans faille.

J'ai sûrement oublié de citer certains d'entre vous qui m'avez aidé dans mon travail, même indirectement, du fond du cœur je vous dis MERCI.

TABLES DES MATIÈRES

LISTE DES FIGURES	vi
LISTE DES TABLEAUX.....	x
LISTE DES ABRÉVIATIONS, DES SIGLES ET DES ACRONYMES	xi
RÉSUMÉ	xii
INTRODUCTION	1
CHAPITRE I REVUE DE LITTÉRATURE	6
1.1 Introduction	6
1.2 Le Web sémantique : notions et principales composantes (Daly, Forgue, & Hirakawa, 2005).....	6
1.2.1 La représentation des données	7
1.2.2 Les ontologies (Hitzler, Krotzsch, & Rudolph, 2011).....	11
1.2.3 SPARQL (Hitzler, Krotzsch, & Rudolph, 2011)	14
1.3 Construction du schéma RDF	17
1.3.1 Schéma RDF à base d'arborescence	17
1.3.2 Schéma RDF à base de graphe.....	19
1.3.3 Schéma RDF à base d'UML	22
1.4 Interrogation des données RDF	24
1.4.1 Interrogation des données à base de navigateurs sémantiques	24
1.4.2 Interrogation des données à base de formulaires	28
1.4.3 Interrogation des données à base de graphe.....	30
1.5 Conclusion.....	32

CHAPITRE II CONSTRUCTION DU SCHÉMA DE DONNÉES RDF	34
2.1 Introduction	34
2.2 Problématique et objectif.....	36
2.3 Regroupement des instances avec type	38
2.4 Regroupement des instances sans type.....	39
2.5 Modélisation du schéma RDF	43
2.5.1 Visualisation avec le diagramme de classes	43
2.5.2 Ajout des statistiques au diagramme de classes.....	46
2.6 Un exemple illustratif.....	48
2.7 Conclusion.....	48
CHAPITRE III INTERROGATION DES DONNÉES RDF	51
3.1 Introduction	51
3.2 Chargement initial du jeu de données	52
3.3 Recherche par des mots-clés	55
3.3.1 Suggestion de complétion.....	55
3.3.2 Voisinage d'un élément du graphe RDF.....	56
3.4 Requête SPARQL.....	59
3.5 Conclusion.....	60
CHAPITRE IV TEST ET ÉVALUATION	62
4.1 Introduction	62
4.2 Prototype QUERY-RDF.....	63
4.2.1 Charger les données RDF	64
4.2.2 Visualiser le schéma RDF.....	64
4.2.3 Interroger les données RDF	65
4.3 Outils utilisés.....	68
4.3.1 Jena	68
4.3.2 Java 6	69
4.3.3 GRAPHVIZ	69
4.4 Évaluation du schéma RDF construit.....	69

4.4.1	Évaluation de la qualité des résultats	69
4.4.2	Évaluation sur des données RDF synthétiques	73
4.5	Évaluation de l'approche d'interrogation des données RDF	79
4.6	Évaluation de la complémentarité de l'interrogation des données RDF et de l'exploration du schéma RDF	82
4.6.1	Modélisation du schéma : Évaluer le besoin d'interroger les données RDF	82
4.6.2	Interrogation des données RDF : Évaluer le besoin d'un schéma RDF ...	83
4.7	Conclusion.....	97
CONCLUSION.....		99
BIBLIOGRAPHIE.....		103

LISTE DES FIGURES

Figure	Page
Figure1.1 Les données RDF sous forme d'un graphe orienté (Schreiber & Raimond, RDF 1.1 Primer, 2014).....	10
Figure1.2 Les données RDF avec la syntaxe RDF/XML (Schreiber & Raimond, 2014)	10
Figure 1.3 Déclaration d'une propriété	13
Figure 1.4 Spécification du domaine et du co-domaine d'une propriété	13
Figure 1.5 Structure d'une requête SPARQL	15
Figure 1.6 Visualisation récapitulative du schéma avec LDT	18
Figure 1.7 Visualisation récapitulative du schéma de la source de données « Clean Energy » (Hitzler, Krotzsch, & Rudolph, 2011)	20
Figure 1.8 Visualiser les données de Jamendo avec un diagramme UML (Li & Zhang, 2013)	23
Figure 1.9 Navigateur gFacet (Heim, Ziegler, & Lohmann, 2008)	25

Figure 1.10 Tabulator-composition d'une requête SPARQL (Tim Berners-Lee & Sheets, 2006).....	27
Figure 1.11 Graph pattern builder (Auer, et al., 2007)	29
Figure 1.12 RDF-GL (Hogenboom, Milea, Frasincar, & Uzay, 2010)	31
Figure 2.1 Extrait de la source de données Jamendo (Raimond, 2016).....	36
Figure 2.2 SPARQL-Les groupements des instances avec type et leurs propriétés ...	39
Figure 2.3 Exemple de graphe avec des instances sans type	41
Figure 2.4 Source de données Jamendo : Diagramme de classes	49
Figure 3.1 Chargement initial du jeu de données	54
Figure 3.2 Suggestion de complétion.....	56
Figure 3.3 Voisinage d'un élément du graphe RDF	58
Figure 3.4 Requête SPARQL.....	60
Figure 4.1 Prototype QUERY-RDF.....	63
Figure 4.2 QUERY-RDF : Choix d'un fichier RDF	64
Figure 4.3 Schéma de données RDF	66
Figure 4.4 Interface interrogation des données RDF	67

Figure 4.5 Visualiser les données de Jamendo avec le diagramme UML (Li & Zhang, 2013)	71
Figure 4.6 Visualiser les données de Jamendo avec QUERY-RDF	72
Figure 4.7 Extrait du jeu de données Jamendo (Raimond, 2016)	73
Figure 4.8 Le diagramme physique des données relationnelles généré par la méthode reverse engineer de MySQL Workbensh.....	75
Figure 4.9 Diagramme de classes à partir de la source de données Film.rdf.....	76
Figure 4.10 Film.RDF avec des instances sans type : diagramme de classes.....	78
Figure 4.11 Explorer les instances sans type de Jamendo	84
Figure 4.12 Association Film et Langage	85
Figure 4.13 Association Film et Catégorie	86
Figure 4.14 Interface Exploration des données.....	88
Figure 4.15 Interface Exploration des données : créer manuellement un diagramme de classes approximatif.....	89
Figure 4.16 Extrait du diagramme de classes Film.....	92
Figure 4.17 Explorer Catégorie Film	92
Figure 4.18 Association Film et Catégorie	94

Figure 4.19 Classe Category	94
Figure 4.20 Déterminer la langue de film	95
Figure 4.21 Explorer les catégories de Film de type Drama.....	96

LISTE DES TABLEAUX

Tableau	Page
Tableau 1.1 Résultat d'une requête SPARQL	17
Tableau 2.1 Initialiser la matrice de similarité.....	41
Tableau 2.2 Les règles de conversion pour la construction du diagramme de classes UML à partir des données RDF	44
Tableau 4.1 Évaluation de l'approche d'interrogation des données RDF	80
Tableau 4.2 Durée de la construction manuelle du diagramme de classes en seconde	90

LISTE DES ABRÉVIATIONS, DES SIGLES ET DES ACRONYMES

HTTP HYPERTEXT TRANSFER PROTOCOL

IRI INTERNATIONALIZED RESOURCE IDENTIFIER

OWL WEB ONTOLOGY LANGUAGE

RDF RESOURCE DESCRIPTION FRAMEWORK

RDFS RESOURCE DESCRIPTION FRAMEWORK SCHEMA

URI UNIVERSAL RESOURCE IDENTIFIER

XML EXTENSIBLE MARKUP LANGUAGE

RÉSUMÉ

À travers cette recherche, nous visons à concevoir une approche qui permet une composition assistée de requêtes sur des données liées. L'exploitation pertinente de ces sources constitue un défi : l'interrogation des sources est en effet limitée d'abord, car elle suppose la maîtrise d'un langage de requêtes tel que *SPARQL*, mais surtout, car elle suppose une certaine connaissance de la source de données afin de cibler des ressources en particulier. Connaître le schéma de données *RDF* : les différentes classes, les propriétés et les relations avec les autres classes constituent une carte qui facilite la détermination des différents chemins possibles et ce qui permet de guider l'utilisateur tout au long l'exploration des données. Par conséquent, explorer les données *RDF* en absence du schéma constitue un vrai défi à l'utilisateur qui pourrait être perdu à cause de la voluminosité des données.

De nombreuses recherches visent à faciliter l'interrogation des données *RDF* alors que d'autres visent plutôt à faciliter la compréhension du schéma *RDF*. Cependant, il n'existe pas de technique efficace jusqu'ici qui regroupe ces deux objectifs complémentaires. Ce travail est un effort dans cette direction. Nous proposons une approche qui permet une composition assistée de requêtes sur des données *RDF*. Cette approche facilite l'interrogation des données *RDF* qui sera accompagnée par un schéma de données explicite qui peut aider à obtenir un meilleur aperçu de la structure des données avant l'interrogation. Le schéma de données *RDF* se base sur le regroupement des instances selon la présence et l'absence de la propriété *rdf:type* et selon les propriétés des ressources. L'interrogation des données *RDF* consiste à faciliter la recherche par des mots-clés, la suggestion de complétion, la restriction de voisinage d'un élément de graphe et la construction assistée de la requête *SPARQL*. L'approche proposée sera évaluée sur différents types de sources *RDF*. Nous illustrerons également la complémentarité et la pertinence des deux stratégies.

Mots-clés : Les données liées, RDF, requêtes SPARQL, construction du schéma des données, regroupement des ressources, interface graphique de requête

INTRODUCTION

Avec l'émergence du Web sémantique, les données liées accessibles et de plus en plus nombreuses sur la toile, sont exprimées en RDF. Les informations représentées en RDF sont normalement de grandes tailles (par exemple : 47 millions triplets RDF dbpedia, 700 millions de triplets UniProt, etc.) (Seema, Medha, kolovski, & Wu, 2010), fortement interconnectées et proviennent de différents domaines de la connaissance, ouvrant la voie ainsi à de nombreuses analyses et explorations.

Aujourd'hui, une grande partie des données RDF est interrogée et explorée à l'aide du langage de requêtes SPARQL. Cependant, la formulation des requêtes SPARQL est une tâche complexe. En effet, l'utilisateur devrait connaître la syntaxe du SPARQL et nécessite des connaissances techniques et une certaine compréhension de RDF, RDFS et IRIs, entre autres. Par contre, les données RDF sont non seulement destinées à la communauté du Web sémantique, mais aussi aux utilisateurs non experts et aux experts de différents autres domaines qui ne connaissent pas bien les différentes technologies utilisées. Par la suite, ça devient très difficile pour ces utilisateurs d'interroger et d'explorer les données avec SPARQL.

Avant de pouvoir composer une requête, l'utilisateur doit comprendre les données et doit disposer certaines informations sur le schéma du jeu de données RDF. Cette étape permet à l'utilisateur de localiser les informations pertinentes pour leurs besoins spécifiques et permet de déterminer si ces données peuvent être facilement réutilisées. Ainsi, cette étape permet à l'utilisateur d'avoir des réponses aux questions qu'il peut

se poser : « De quoi est-il question dans cette source ? », « Est-elle pertinente par rapport à mon besoin ? », etc.

Contrairement à une base de données relationnelle où le schéma est fixe et peut aider à comprendre la structure des données avant la formulation des requêtes, les données RDF ne possèdent pas un schéma explicite et ne doivent pas nécessairement se conformer à un schéma contraignant. Il peut donc être difficile pour les utilisateurs d'obtenir une vue d'ensemble lors de la manipulation d'un ensemble de données RDF volumineux et complexe.

Ainsi, l'absence d'un schéma qui représente bien les données RDF peut limiter l'interrogation de ces dernières : par exemple, écrire une requête sans connaître les classes existantes et leurs propriétés (connues sous le nom de prédicats) n'est pas simple. En effet, l'utilisateur doit d'abord soumettre plusieurs requêtes et parcourir manuellement les résultats afin de collecter toutes les classes et propriétés pertinentes, qui seront utilisées pour formuler la requête principale qui fournira le résultat final.

Par conséquent, l'interrogation des données RDF dans l'absence d'un schéma RDF explicite qui donne la structure des données RDF, s'avère une tâche difficile. L'utilisateur essaie de découvrir des bouts du schéma en utilisant des requêtes élémentaires ou bien des explorateurs de documents RDF.

Ces dernières années, quelques travaux ont été menés pour offrir à l'utilisateur un schéma de données RDF [(Tong, Zhang, & Cheng, 2014), (Knublauch & Kontokostas, 2017), (Benedetti, Bergamaschi, & Po, 2014), (Li & Zhang, 2013)]. Ainsi, l'utilisateur aura une vision globale de la source et pourrait déterminer les classes et les propriétés pertinentes pour lui. Cependant, l'utilisateur devra utiliser un langage de requêtes comme SPARQL ou des outils d'interrogation des données pour avoir plus de détails sur les données.

Il existe aussi des travaux qui proposent d'interroger les sources de données RDF à base de formulaire, de navigateur sémantique ou de graphe [(Tim Berners-Lee & Sheets, 2006), (Jethro, 2006), (Ambrus, Möller, & Handschuh, 2010), (Hogenboom, Milea, Frasincar, & Uzay, 2010). Ces outils aident les utilisateurs à comprendre les données RDF et à construire des requêtes automatiquement sans avoir besoin de connaître des langages des requêtes. Pour interroger ces données RDF, l'utilisateur a besoin d'un minimum de connaissance sur la structure des données. Cependant, ce dernier point n'est pas considéré par les approches existantes.

À notre connaissance, aucune approche permet d'un côté la construction du schéma de données et l'interrogation de données de l'autre côté, bien qu'ils soient complémentaires et dépendent l'une de l'autre. Dans ce contexte, nous proposons QUERY-RDF, un outil d'aide à la composition des requêtes SPARQL. L'interrogation de données sera accompagné d'un schéma de données qui peut aider à obtenir un meilleur aperçu de la structure des données et peut constituer un point de départ utile pour des interrogations et des analyses plus poussées. Les deux approches ensemble permettent à un utilisateur de comprendre le contenu d'une source de données RDF le plus rapidement possible et de répondre ainsi au mieux à ses besoins.

La suite de ce manuscrit est organisée comme suit. Le chapitre 2 présente un état de l'art relatif à notre problématique. Nous présentons les travaux récents de la littérature sur les deux approches proposées. Pour l'approche de construction du schéma de données RDF, les travaux présentés permettent la visualisation et l'identification de différentes classes ainsi que leurs attributs et leurs propriétés. Trois types de modélisation du schéma sont présentés : schéma sous forme d'arborescence, schéma sous forme de graphe et schéma sous forme d'un modèle UML. Nous présentons également des travaux qui s'intéressent à faciliter l'interrogation des données à base de

navigateur, de formulaires et de graphes. Dans ce chapitre, nous discutons plusieurs limites des approches présentées.

Dans le chapitre 2, à partir d'une source de données RDF, nous souhaitons construire une approximation du schéma des données. Cette approximation ne vise pas à reproduire le schéma réel, mais elle vise plutôt à construire un schéma approximatif qui résume l'ensemble des données RDF et dont l'objectif est d'assister à la formulation des requêtes. Les classes du schéma approximatif seront composées de groupement d'instances avec et sans type défini. Il faut tenir compte du fait que le graphe RDF est hétérogène, c'est-à-dire qu'une ressource RDF puisse avoir aucun type, un ou plusieurs types et qu'elle peut ou non être liée à une autre ressource. De plus, deux ressources du même type ne possèdent pas les mêmes propriétés et ne suivent pas la même structure, car les propriétés sont optionnelles. Cette approximation sera modélisée par un diagramme de classes en appliquant des règles de conversion. Nous ajoutons à ce diagramme des statistiques qui permettent de fournir un profil de la source de données tel que le nombre d'instances par groupement. Cette représentation permet aux utilisateurs de comprendre et d'avoir une meilleure idée de la structure de la source de données RDF. Par conséquent, les utilisateurs pourront cibler des groupements pertinents pour leurs besoins. Ainsi, ils peuvent définir les éléments de motifs de triplet de leur requête.

Le chapitre 3 décrit l'approche d'interrogation des données en détail avec la recherche par des mots-clés et la suggestion de complétion, une façon simple et intuitive d'interroger les sources de données. Nous expliquerons comment balayer le voisinage d'un élément du graphe RDF, une approche qui permet aux utilisateurs de cibler une ressource en particulier et de l'explorer en détail. Nous allons voir également comment l'utilisateur n'aura plus le besoin d'avoir des connaissances approfondies de SPARQL en utilisant l'interface de notre outil.

Le chapitre 4 présente d'abord l'outil que nous avons développé pour la validation de notre approche. Puis, nous présentons les expérimentations que nous avons menées pour évaluer les deux stratégies proposées. La première partie d'expérimentations concerne l'approche de construction du schéma de données RDF. La deuxième partie concerne l'approche de composition assistée des requêtes SPARQL. Pour les deux parties, nous décrivons les sources de données et les scénarios utilisés pour l'évaluation. Puis, nous présentons les métriques et les résultats des évaluations obtenues. Nous montrerons également la complémentarité et la pertinence des deux approches proposées.

Enfin, nous concluons ce mémoire par un bilan des travaux effectués et nous présentons quelques perspectives de recherche dans le chapitre 5.

Comme le schéma est important pour comprendre les données RDF, l'idée est de construire un schéma résumant l'ensemble de données RDF. Cependant, il faut tenir compte du fait que le graphe RDF est hétérogène, c'est-à-dire qu'une ressource RDF puisse avoir aucun type, qu'un ou plusieurs types et qu'elle peut ou non être liée à une autre ressource. De plus, deux ressources du même type ne possèdent pas les mêmes propriétés et ne suivent pas la même structure, car les propriétés sont optionnelles.

En effet, pour construire la requête et explorer les données d'une façon incrémentale, l'utilisateur doit connaître le schéma et les classes dont traitent les données sinon il pourrait être perdu à cause de la voluminosité des données RDF. Connaître les classes, leurs propriétés et leurs liens avec les autres classes constitue une carte descriptive partielle du schéma, ce qui permet de guider l'utilisateur tout au long l'exploration de données.

CHAPITRE I

REVUE DE LITTÉRATURE

1.1 Introduction

Avant d'introduire notre approche de la composition assistée de requêtes sur des données RDF qui sera décrite dans les chapitres II et III, nous réaliserons une revue de la littérature. À travers ce chapitre, nous tracerons un survol historique du Web sémantique : les notions de base et les principales composantes. La section 1.3 regroupe les approches de la construction du schéma *RDF* tandis que la section 1.4 regroupe les approches de la construction des requêtes. Enfin, la section 1.5 conclut le chapitre.

1.2 Le Web sémantique : notions et principales composantes (Daly, Forgue, & Hirakawa, 2005)

Le Web sémantique a été proposé en se basant sur les critiques adressées au Web actuel : (i) certes *HTML* a permis de tisser tout un réseau d'informations par ses liens hypertextes entre les pages Web, mais il n'a donné aucune sémantique à ces liens, ce qui les rendent pratiquement inexploitable par les machines (ii) les métadonnées utilisées sont non structurées et limitées dans leurs usages, et (iii) il est difficile de faire des inférences et des raisonnements sur les connaissances décrites dans les documents publiés sur le Web vu l'absence de modèles permettant la représentation sémantique de ces connaissances.

Tous ces problèmes ont fait l'objet de différents travaux de recherche qui ont convergé vers plusieurs solutions parmi lesquelles nous présentons celles qui nous semblent les plus essentielles :

- Proposer des langages et des formalismes de représentation et de structuration des connaissances. Ces langages permettent de modéliser et de représenter le contenu sémantique des ressources du Web (section 1.2.2) ;
- Rendre disponibles des ressources conceptuelles (des modèles) représentées dans ces langages modélisant les connaissances et facilitant leur accès et leur partage : les ontologies et RDFS (section 1.2.3) ;
- Proposer du langage de requêtes SPARQL permettant l'interrogation de données (section 1.2.4).

1.2.1 La représentation des données

Le fonctionnement du Web sémantique est fondé sur le fait que les machines puissent accéder à l'ensemble des informations éparpillées sur le Web. Le *W3C* ainsi que les chercheurs travaillant dans le domaine de l'intelligence artificielle et de bases de données ont beaucoup travaillé sur ce point et ont proposé plusieurs langages de représentation des connaissances afin de faciliter cet accès.

1.2.1.1 L'URI et l'IRI (Daly, Forgue, & Hirakawa, 2005)

L'*URI* a une fonction analogue, mais plus large puisqu'il permet d'identifier n'importe quelle ressource, même abstraite. L'*IRI* est une version internationalisée (utilisant Unicode) de l'*URI*. Par exemple, <http://www.irit.fr/> est un URL qui identifie la page d'accueil du site Web. Ce lien est également un *URI* et un *IRI*.

<http://www.irit.fr/ontologies/cinema#JeanDujardin> est un *URI* (et donc un *IRI*) qui identifie l'acteur Jean Dujardin, alors que <http://www.irit.fr/ontologies/cinema>

[#BéréniceBejo](#) est uniquement un *IRI*, car elle contient des caractères non compris dans le jeu de caractères *ASCII*.

1.2.1.2 Le modèle de graphe RDF (Resource Description Framework) (Schreiber & Raimond, 2014)

RDF est une recommandation du *W3C*. Il s'agit d'un modèle de graphes destiné à décrire de façon formelle les ressources Web et leurs métadonnées. Un document structuré en *RDF* est un ensemble de triplets. Un triplet *RDF* est une association (sujet ; prédicat ; objet) tel que :

- Ressource (Sujet) : représente la ressource à décrire qui peut être une page Web, ou un objet qui n'est pas directement accessible via le Web ;
- Propriété (prédicat) : l'attribut ou la relation utilisé(e) pour décrire une ressource ;
- Valeur (objet) : représente un littéral (une donnée) ou une ressource : c'est la valeur de la propriété. Cette valeur peut être une chaîne de caractères, un nombre, etc. ou une ressource. Celle-ci peut avoir aussi ses propres propriétés dans le graphe RDF.

Le sujet et l'objet de type ressource sont des *IRI(s)* ou des nœuds anonymes. Un nœud anonyme exprime l'existence d'une ressource sans l'identifier. Le prédicat est nécessairement un *IRI*.

Un document *RDF* correspond à un graphe orienté et étiqueté (lien). Les sujets et les objets des triplets RDF sont les nœuds du graphe, alors que les prédicats des triplets RDF représentent les arcs étiquetés et dirigés du graphe.

La syntaxe RDF/XML (*Schreiber & Raimond, 2014*)

RDF dispose de plusieurs syntaxes qui représentent ses modèles pour stocker les instances de ces derniers dans des fichiers lisibles pour la machine tels que *XML*, *Turtle*, *N-Triples* et *N3*.

RDF/XML est la syntaxe la plus connue, elle permet la représentation cohérente de la sémantique et il est le seul format standardisé dans la recommandation *RDF* 1.0. Il est fondé sur *XML*, ce qui le rend verbeux et lourd pour une utilisation manuelle. Par contre, comme *HTML*, cette syntaxe *RDF/XML* est interprétable automatiquement et peut relier, en utilisant des adresses *URI*, des bouts d'information à travers le Web.

Cependant, une représentation commune et naturelle pour les données *RDF* est un graphe orienté étiqueté. Les figures 1.1 et 1.2 montrent un exemple de document *RDF* représenté d'une part sous forme d'un fichier *XML* et de l'autre part sous forme d'un graphe orienté. En utilisant la syntaxe *RDF/XML*, il est possible de modéliser le fait que :

- Bob s'intéresse à la Mona Lisa ;
- Mona Lisa est une peinture de Leonard de Vinci.

Les conteneurs et les collections

Les conteneurs sont des groupes de ressources ou de littéraux ouverts, c'est-à-dire qu'il n'est pas possible d'arrêter l'énumération de leur contenu : d'autres membres d'un conteneur peuvent toujours être déclarés. Il existe trois types de conteneurs :

- Le sac (*rdf:Bag*) est un groupe non ordonné qui peut contenir des doublons ;
- La séquence (*rdf:Seq*) est un groupe ordonné qui peut contenir des doublons ;
- L'alternative (*rdf:Alt*) est un groupe dont les éléments représentent chacun un choix possible d'une alternative.

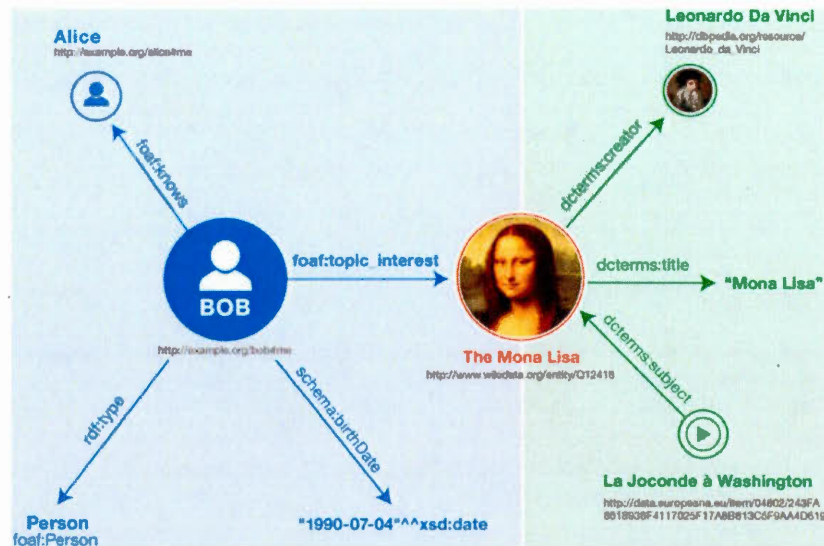


Figure1.1 Les données RDF sous forme d'un graphe orienté (Schreiber & Raimond, RDF 1.1 Primer, 2014)

EXAMPLE 15 RDF/XML

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <rdf:RDF
03     xmlns:dcterms="http://purl.org/dc/terms/"
04     xmlns:foaf="http://xmlns.com/foaf/0.1/"
05     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
06     xmlns:schema="http://schema.org/">
07     <rdf:Description rdf:about="http://example.org/bob#me">
08         <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
09         <schema:birthDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1990-07-04</sc
10         <foaf:knows rdf:resource="http://example.org/alice#me"/>
11         <foaf:topic_interest rdf:resource="http://www.wikidata.org/entity/Q12418"/>
12     </rdf:Description>
13     <rdf:Description rdf:about="http://www.wikidata.org/entity/Q12418">
14         <dcterms:title>Mona Lisa</dcterms:title>
15         <dcterms:creator rdf:resource="http://dbpedia.org/resource/Leonardo_da_Vinci"/>
16     </rdf:Description>
17     <rdf:Description rdf:about="http://data.europeana.eu/item/04802/243FA8618938F4117025F17A
18         <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
19     </rdf:Description>
20 </rdf:RDF>
```

Les collections sont des groupes de ressources ou de littéraux fermés, c'est-à-dire que le contenu d'une collection est fini et intégralement déterminé. Une collection se présente sous la forme d'une liste chaînée : une liste chaînée est une ressource (souvent un nœud anonyme) de type *rdf:List*.

Domaines d'application

RDF est considéré comme le premier pilier du Web sémantique. Dans ce cas, les utilisateurs ont intérêt à mieux décrire le contenu et les relations de contenu disponibles sur le Web (Pages Web, bibliothèque numérique, etc.) pour permettre :

- La découverte des ressources : Les moteurs de recherche utilisent *RDF* pour découvrir plus facilement les ressources sur le Web ;
- L'échange et le partage des connaissances utilisées par les agents logiciels dans le but de filtrer et de fusionner des données ;
- La gestion des droits de propriété intellectuelle : *RDF* permet aux utilisateurs de faire respecter les droits de propriété intellectuelle de sites Web.

1.2.2 Les ontologies (Hitzler, Krotzsch, & Rudolph, 2011)

RDF Schéma et *OWL* sont les formalismes proposés par le *W3C* permettant de définir des ontologies intégrées au cadre du Web sémantique, aussi appelées schémas ou ontologies légères lorsqu'elles sont définies en *RDFS* et par conséquent faiblement expressives. Nous présentons ici brièvement *RDFS* et *OWL*, et nous expliquons plus en détail les principales notions de ces formalismes, en nous focalisant sur les axiomes que nous exploitons dans la suite de ce travail, qui sont également les axiomes les plus répandus dans les bases de connaissances actuelles. Une description complète de *RDFS*, d'*OWL* et de leurs sémantiques est donnée dans (Hitzler, Krotzsch, & Rudolph, 2011).

RDF Schéma (Brian, 2004)

RDF Schéma est un méta modèle exprimé en RDF, recommandé par le *W3C* et permettant la définition de vocabulaire à utiliser dans une source de données RDF. *RDF Schéma* offre le moyen de définir un modèle de méta donné qui permet de :

- Définir les types de ressources (personne, livre, etc.) ;
- Donner du sens aux propriétés associées à une ressource (diplôme, titre, auteur, etc.) ;
- Formuler des contraintes sur les valeurs associées à une propriété afin de lui assurer aussi une signification.

Classes et instances

RDF fournit déjà un mécanisme permettant de typer des ressources, c'est-à-dire de les marquer comme appartenant à un groupe de ressources défini que l'on appelle une classe. Chaque ressource de la classe est une instance. La propriété *rdf:type* permet d'exprimer l'appartenance d'une ressource à une classe. Par exemple, le triplet de la figure 1.1 exprime que la ressource :*BOB* est une instance de la classe *Personne*.

Propriétés

RDF Shéma permet également de définir des propriétés, c'est-à-dire des entités qui sont utilisées comme deuxième membre d'un triplet RDF. La classe *rdf:Property* est prévue à cet effet : une fois déclarée comme appartenant à cette classe, une ressource est alors considérée comme une propriété. Le triplet de la figure 1.3 exprime que la ressource *foaf:name* est une propriété.

De plus, RDFS offre les propriétés *rdfs:domain* et *rdfs:range* qui permettent de définir respectivement le domaine (*domain*) et le co-domaine (*range*) d'une propriété.



Figure 1.3 Déclaration d'une propriété

Le domaine (resp. co-domaine) d'une propriété désigne la classe à laquelle appartient toute ressource utilisée comme sujet (resp. objet) dans un triplet dans lequel cette propriété a la place du prédicat. Le graphe de la figure 1.4 (a) exprime que le domaine de la propriété *foaf:topic_interest* est *foaf:person* et que son co-domaine est *foaf:image*. Avec ces connaissances supplémentaires, nous pourrions inférer du triplet de la figure 1.2 les types des ressources *:Bob* et *:MonaLisa*, comme illustré dans la figure 1.4 (b).

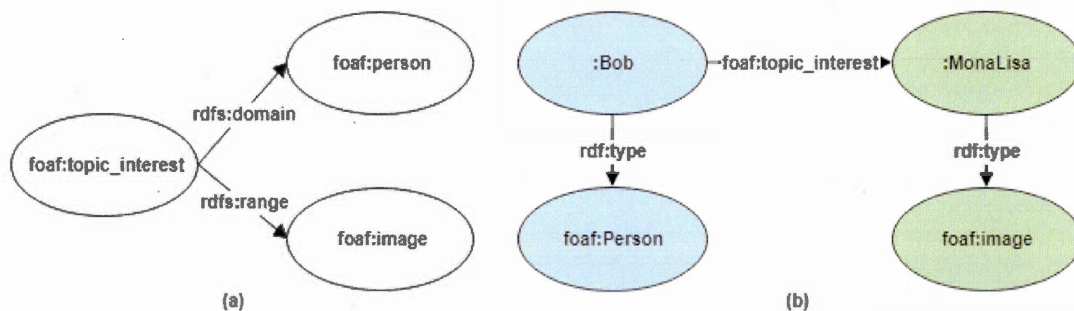


Figure 1.4 Spécification du domaine et du co-domaine d'une propriété

Hierarchie de classes et de propriétés

La propriété *rdfs:subClassOf* est une instance de la classe *rdf:Property*, utilisée pour déclarer qu'une classe est une sous-classe d'une autre.

La propriété *rdfs:subPropertyOf* est une instance de la classe *rdf:Property*, utilisée pour déclarer que toutes les ressources reliées par une propriété sont également reliées par une autre.

Les propriétés prédéfinies tel que *rdf:type*, *rdfs:domain*, *rdfs:range*, *rdfs:subClassOf*, etc permettent de bien décrire les ressources. Cependant, ces propriétés sont optionnelles et souvent absentes ce qui complique la compréhension des données RDF.

1.2.3 SPARQL (Hitzler, Krotzsch, & Rudolph, 2011)

SPARQL est le langage de requêtes recommandé par le W3C pour ajouter, modifier, supprimer, mais surtout rechercher des données RDF disponibles. SPARQL a une syntaxe proche de celle du SQL, mais adaptée à la structure spécifique des graphes RDF. SPARQL repose sur la correspondance des motifs de graphe appelé Basic Graph Pattern (BGP). Un BGP est un ensemble de motifs de triplet. Un motif de triplet ressemble à un triplet RDF sauf que son sujet, son prédicat et son objet peut être une variable. SPARQL offre quatre formes d'interrogation (query forms) qui sont :

- SELECT : Retourne toutes les variables liées, ou un sous-ensemble de celles-ci, dans une comparaison de motifs d'interrogation (query pattern match) ;
- CONSTRUCT : Retourne un graphe RDF construit en substituant des variables dans un ensemble de gabarits de triplet (triple templates) ;
- ASK : Retourne un booléen indiquant si un motif d'interrogation correspond ou non ;
- DESCRIBE : Retourne un graphe RDF décrivant les ressources trouvées.

Le langage SPARQL/Update 1,0 (Kjernsmo & Passant, 2009) est une extension de langage SPARQL permettant d'exprimer les mises à jour d'un graphe RDF ou d'un triple store. Le langage SPARQL/Update 1,0 fournit des fonctionnalités telles que insérer ou supprimer des triplets dans un graphe RDF, effectuer un groupe d'opérations de mise à jour en une seule action et créer ou supprimer un graphe RDF dans un graphe store.

Dans notre mémoire, nous nous intéressons aux requêtes de type SELECT. Une requête SPARQL accepte dans une clause WHERE des motifs de graphe obligatoires et optionnels, ainsi que leurs conjonctions (jointure) et leurs disjonctions (union). Une requête SPARQL comprend, dans cet ordre (Figure 1.5) :

- Déclaration de préfixes (Ligne 1) : pour l'abréviation des IRIs ;
- Une clause de résultat (Ligne 3) : identifie les informations à retourner par la requête ;
- Le modèle de graphe de base BGP (Ligne 6-9) : précise ce qu'il faut interroger à partir de l'ensemble de données sous-jacentes. De différentes options peuvent être utilisées telles que :
 - *UNION* : groupes de modèles de graphes alternatifs ;
 - *OPTIONAL* (Ligne 7) : groupes de modèles de graphes optionnels et non requis ;
 - *FILTER* (Ligne 8) : ce motif permet de rajouter des conditions à satisfaire notamment pour les littéraux.
- Les modificateurs de requête (Ligne 10) : *ORDER BY*, *Limit* et *OFFSET*.

```
#prefix declarations
PREFIX
1 rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
...
# result clause
3 SELECT ...
4 # dataset definition
5 FROM ...
# query pattern
6 WHERE {
...
7     OPTIONAL(...)
8     FILTER(...)
9 }
# query modifiers
10 ORDER BY ...
```

Figure 1.5 Structure d'une requête SPARQL

Voici un exemple de requête SPARQL retournant une liste de personnes et leurs intérêts à partir de la base de connaissances de la figure 1.1 :

```

1 PREFIX rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX foaf : <http://xmlns.com/foaf/0.1/>
3 PREFIX dcterms : <http://purl.org/dc/terms/>
4 SELECT DISTINCT ? nom ? intérêt
5 WHERE {
6     ?personne rdf:type foaf:Person.
7     ?personne foaf:name ? nom.
8     OPTIONAL{
9         ?personne foaf:topic_interest ?topic_interet.
10        ?topic_interet dcterms:title ?intérêt.
11    }
12} ORDER BY ?nom ASC

```

La déclaration des espaces de noms est en début (Ligne1-3), suivis de la requête proprement dite (Ligne 4-13). Les noms des variables (*?nom* et *?intérêt*) sont précédés d'un point d'interrogation. La clause SELECT (ligne4) permet de sélectionner l'ensemble des tuples, ou lignes de variables correspondant aux contraintes de la clause WHERE (Ligne 5). La première ligne de la clause WHERE s'interprète comme suit : la variable *personne* est de type *Person* au sens de l'ontologie *FoaF* (Ligne 6). La seconde ligne permet de définir la variable *nom* en tant qu'objet de la propriété *name* de la variable *personne* (Ligne 7). Les lignes 9-10 présentent deux motifs de triplet qui permettent de sélectionner l'intérêt de chaque personne, le cas échéant. Ces deux motifs de triplet sont optionnels grâce à la clause OPTIONAL (Ligne 8). Le résultat sera dans l'ordre croissant du nom de la personne *? nom*. Cette affirmation est définie grâce à l'opérateur ORDER BY (ligne 12). Le résultat est présenté à travers le tableau 1.1.

Tableau 1.1 Résultat d'une requête SPARQL

Nom	Intérêt
Alice	
Bob	Mona Lisa

1.3 Construction du schéma RDF

Une quantité énorme de données liées a été publiée ces dernières années et est prête à être consommée. Une grande partie de ces données est disponible au format *RDF* et peut être interrogée à l'aide du langage d'interrogation normalisé *SPARQL*. Les données ne suivent souvent pas un schéma fixe, mais généralement des ontologies et des vocabulaires différents sont utilisés pour les décrire de manière flexible. D'une part, cette flexibilité est une caractéristique et un avantage importants des données liées. De l'autre part, il peut être difficile de se faire une idée des données réellement disponibles pour *SPARQL*. Un schéma de données RDF peut aider à obtenir un meilleur aperçu de la structure des données et peut constituer un point de départ utile pour des interrogations et des analyses plus poussées. Il existe principalement trois différentes méthodes de modélisation du schéma à partir de données liées : schéma à base d'arborescence, schéma à base de graphe et schéma à base d'un modèle *UML* (Daly, Forgue, & Hirakawa, 2005).

1.3.1 Schéma RDF à base d'arborescence

Une arborescence de la hiérarchie des concepts est utilisée pour afficher les différentes classes et les propriétés contenues dans les données. Un navigateur hiérarchique offre à l'utilisateur un aperçu des principales classes et des propriétés d'un ensemble de données. Il permet également à l'utilisateur de déplier les classes et les propriétés spécifiques qui l'intéressent, afin de révéler leurs sous-classes et leurs sous-propriétés. La navigation hiérarchique permet de mieux comprendre un schéma de concept.

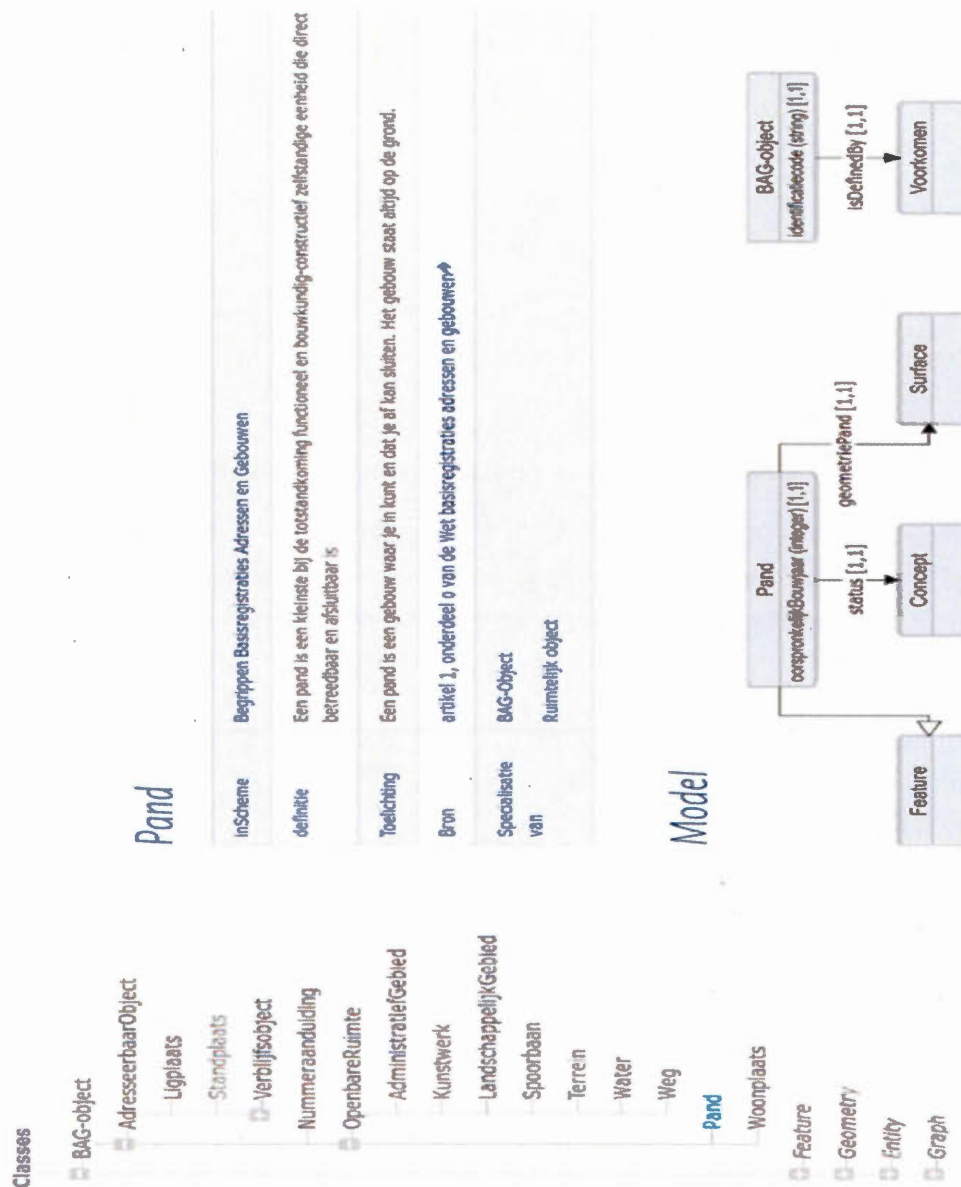


Figure 1.6 Visualisation récapitulative du schéma avec LDT

La figure 1.6 illustre le navigateur hiérarchique de *LDT*¹ (Linked Data Theater). La partie gauche montre une arborescence qui représente la hiérarchie de sous-classes de la classe *BAG*. Dans cette arborescence, l'utilisateur a sélectionné la classe *Pand*. La partie droite montre une description tabulaire et schématique de cette classe.

1.3.2 Schéma RDF à base de graphe

La visualisation des ontologies a pour objectif de produire une vision intuitive de la source de données. Quelques outils pour la visualisation de schéma de données *RDF* sous la forme de graphes ont été proposés dont les plus connus sont : *LD-VOWL* (Weise & al, 2016), *LODSight* (Dudáš, Svátek, & Mynarz, 2015) et *LODeX* (Benedetti, Bergamaschi, & Po, 2014). *LODeX* (Benedetti, Bergamaschi, & Po, 2014) est l'un des outils proposés récemment pour la visualisation des graphes *RDF*. Il offre un résumé du graphe qui se base sur le schéma et sur des statistiques de l'ensemble de données. Les données *RDF* sont explorées à travers l'interface Web où le résumé est représenté sous la forme d'un graphe contenant des classes connectées. Un arc non orienté sans étiquette existe entre deux classes si au moins une propriété existe entre les instances de ces deux classes dans le graphe initial. En plus du graphe résumé, un ensemble de données quantitatives est fourni à l'utilisateur tel que : la liste d'espaces de noms utilisés, le nombre de triplets formés avec chaque propriété pour chaque classe.

En effet, lorsque l'utilisateur sélectionne une classe dans le graphe résumé, une liste de ses propriétés et le nombre de ses instances sont indiqués.

Par conséquent, l'outil fournit une vue globale du schéma de la source de données avec des informations sur les instances. Ainsi, l'utilisateur comprendra plus facilement le contenu de la source de données. Avec cet outil, il y a aussi la possibilité de construire des requêtes *SPARQL*. La figure 1.7 donne une vue sur la source de données *Reegle*² à travers l'outil *LODeX*. La source de données *Reegle* contient des informations liées à

¹ http://www.pilod.nl/wiki/Linked_Data_Theatre

² <http://data.reegle.info>

la production d'énergies renouvelables. Les statistiques sur les données sont affichées. La liste de propriétés de la classe *Organization* est également affichée à gauche, car l'utilisateur a sélectionné cette dernière. Pour cette classe, nous pourrions voir le nombre d'instances qui est 1869 instances (Figure 1.7, B), les différentes propriétés, ainsi les classes avec lesquelles ces propriétés sont liées (Figure 1.7, C). Exemple : la propriété *depiction* permet d'avoir un lien entre la classe *Organization* et la classe *Image*. Les attributs sont également affichés (Figure 1.7, D) tel que *name*, *abbreviation*, etc. Chaque fois qu'une propriété est choisie, la classe connectée est ajoutée à la requête visuelle.

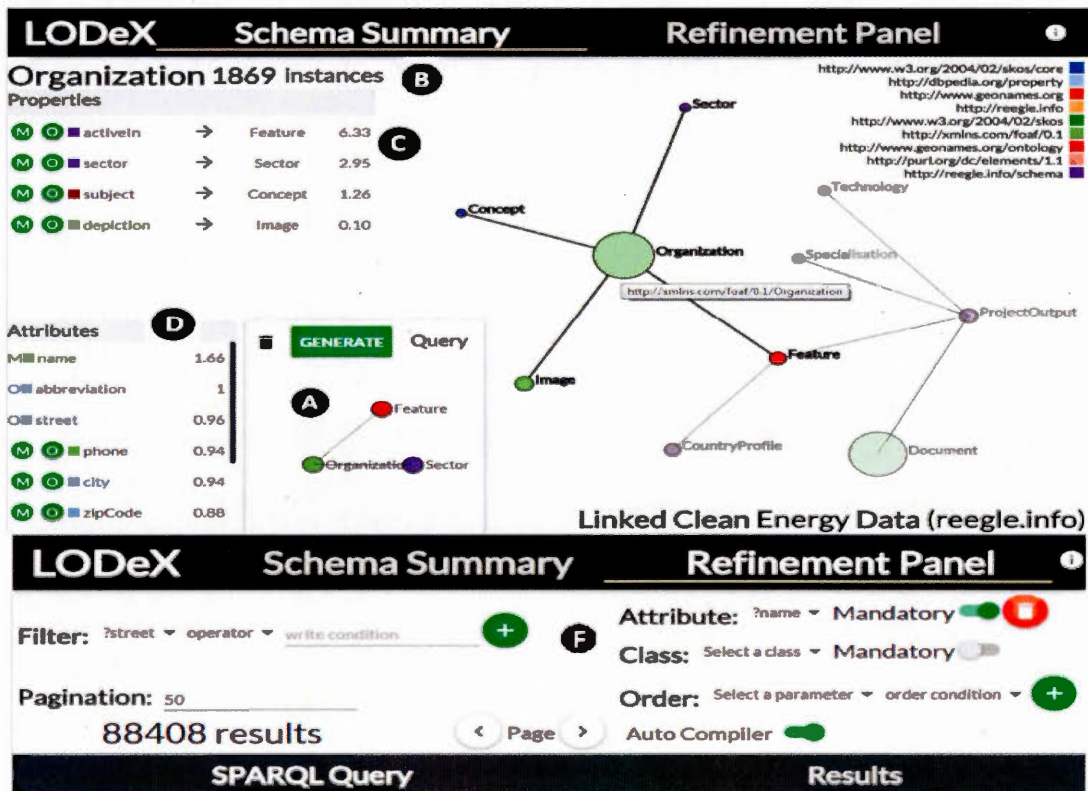


Figure 1.7 Visualisation récapitulative du schéma de la source de données « Clean Energy » (Hitzler, Krotzsch, & Rudolph, 2011)

En répétant ce processus, l'utilisateur peut étendre et modifier la requête en ajoutant ou en supprimant des conditions de filtre, en modifiant la liste d'attributs ou en triant les enregistrements (Figure 1.7, F). Après toute modification, la requête *SPARQL* est compilée et les résultats sont mis à jour.

Les données sources proviennent de *SPARQL endpoints*. Les *SPARQL endpoints* sont l'un des éléments clés de l'infrastructure actuelle du Web sémantique. L'idée de ces points de terminaison est de permettre aux utilisateurs d'extraire des informations d'un jeu de données *RDF* à distance. Le jeu de données est mis à disposition pour être interrogé par le protocole http. Les informations doivent être obtenues à l'aide de requêtes *SPARQL*, conformément à la dernière spécification *SPARQL*.

Nous notons que l'outil se base principalement sur le schéma RDFS. Par conséquent, son utilisation se limite à des sources de données possédant un schéma RDFS. En plus, la source de données parfois ne suit pas le schéma et dans ce cas, l'outil ne représente pas vraiment les données RDF. Ainsi, nous ne pourrions pas voir les propriétés qui lient deux classes directement à partir du graphe, il faut sélectionner une classe pour voir les différentes propriétés qui permettent le lien avec les autres classes.

LODSight, *LD-VOWL* et *LODeX* utilisent un ensemble de requêtes *SPARQL* pour rechercher les classes, les propriétés et les *datatypes*, etc. Ces outils ne traitent pas le cas des instances qui n'ont pas de classe. Par la suite, la représentation est une représentation partielle et non complète. Ensuite, ces outils visualisent uniquement les informations de schéma ce qui permet d'obtenir une meilleure compréhension de la structure de données. Cependant, ils ne permettent pas d'explorer les données *RDF*. Enfin, à part *LODeX*, la plupart de ces outils n'offrent pas la possibilité de créer des requêtes *SPARQL*.

1.3.3 Schéma RDF à base d'UML

Il existe étonnamment peu de travaux concernant l'extraction et la visualisation d'informations de schéma à partir de données liées sous forme de diagramme *UML* principalement (Li & Zhang, 2013) et (Jin-Sung & Mi-Kyung, 2005). Ce type de modélisation fonctionne bien pour avoir un aperçu de la relation entre certains concepts clés. Ceci ressemble à la modélisation des schémas de base de données dans le langage de modélisation *UML*, bien qu'il existe des différences entre *UML* et les formats *RDF* (S) et *OWL*.

(Li & Zhang, 2013) propose un outil basé sur *SPARQL* pour construire le schéma réel, rassembler les statistiques correspondantes et présenter une visualisation basée sur *UML* à partir des sources de données *RDF* telles que *SPARQL endpoints* et les *RDF dump*³. L'approche proposée aide les utilisateurs à comprendre la structure de données. La figure 1.8 montre le profil de données de *Jamendo RDF* (Raimond, 2016) (une ontologie musicale) sous forme d'un diagramme de classes avec des statistiques qui permettent aux utilisateurs de comprendre mieux les données *RDF* tel que le nombre d'instances par classe et le nombre d'instances pour chaque propriété. Ce type de modélisation permet de présenter le schéma d'une façon visuelle que nous jugeons que c'est la meilleure. C'est une vue concise et plus lisible par la plupart des utilisateurs. Cependant, cette approche n'offre pas le moyen d'interroger et d'explorer les données *RDF* de plus en détail. Les auteurs ont exposé le problème de représentation des instances qui n'ont pas de classe, cependant la solution n'a pas été détaillée.

³ <http://d2rq.org/dump-rdf>

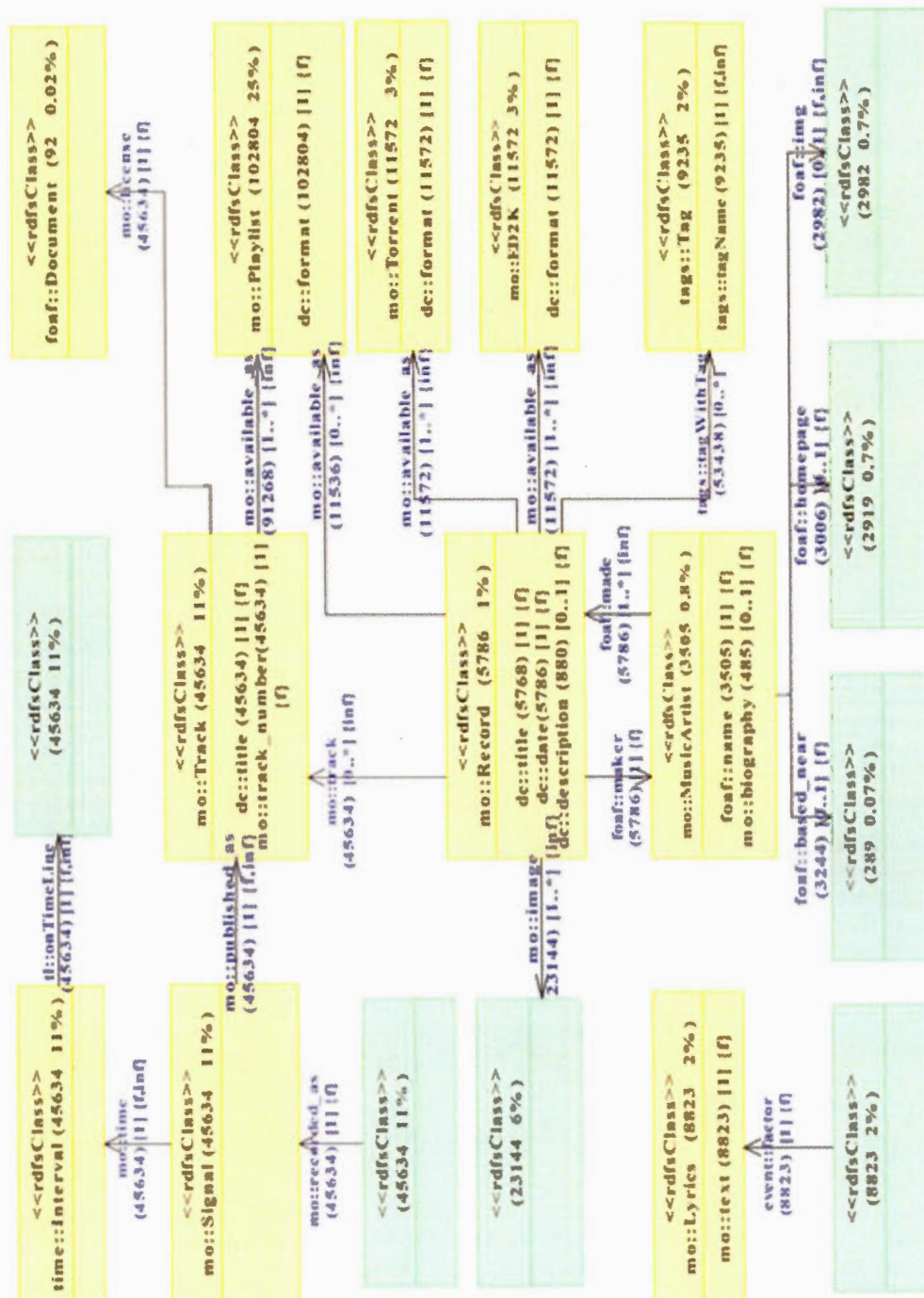


Figure 1.8 Visualiser les données de Jamendo avec un diagramme UML (Li & Zhang, 2013)

1.4 Interrogation des données RDF

Compte tenu de la croissance explosive de la taille et de la complexité du Web de données, il existe maintenant plus que jamais un besoin croissant de développer des méthodes et des outils afin de faciliter la compréhension, l'interrogation et l'exploration des données RDF. Nous pouvons classer ces outils en trois groupes principaux, chacun utilise une approche différente. Le premier groupe utilise une interrogation des données RDF basée sur des navigateurs sémantiques. Le deuxième groupe permet de construire des requêtes à base des formulaires. Le dernier groupe permet de construire des requêtes graphiques qui visualisent explicitement la structure des jeux de données *RDF* par nœuds et arêtes.

1.4.1 Interrogation des données à base de navigateurs sémantiques

Cette approche offre des interfaces visuelles qui créent implicitement les requêtes dans le processus de navigation grâce aux navigateurs sémantiques. Ces derniers utilisent des structures textuelles telles que les tables et les listes pour présenter les données liées, les entités, les propriétés et les relations. Certains navigateurs utilisent également des fonctionnalités avancées telles que la navigation à facettes. Avec ce type d'exploration, les données sont partitionnées à l'aide de dimensions conceptuelles orthogonales. Une des dimensions sert de jeu de résultats et les autres sont utilisées comme facettes pour filtrer le jeu de résultats selon des différents attributs pouvant être sélectionnés indépendamment les uns des autres. Un exemple populaire est iTunes (iTunes, 2018) qui utilise l'approche de filtrage par facettes pour permettre à l'utilisateur d'accéder à la musique en fonction de différents attributs, tels que l'artiste, l'album ou le genre. Pour les données RDF, nous pouvons citer l'exemple du navigateur *GFacet* (Heim, Ziegler, & Lohmann, 2008). Ce type de navigation transforme les propriétés apparaissant dans un ensemble de données en widgets pouvant être définis par l'utilisateur afin de formuler un ensemble de filtres conjonctifs à travers l'ensemble des

instances.

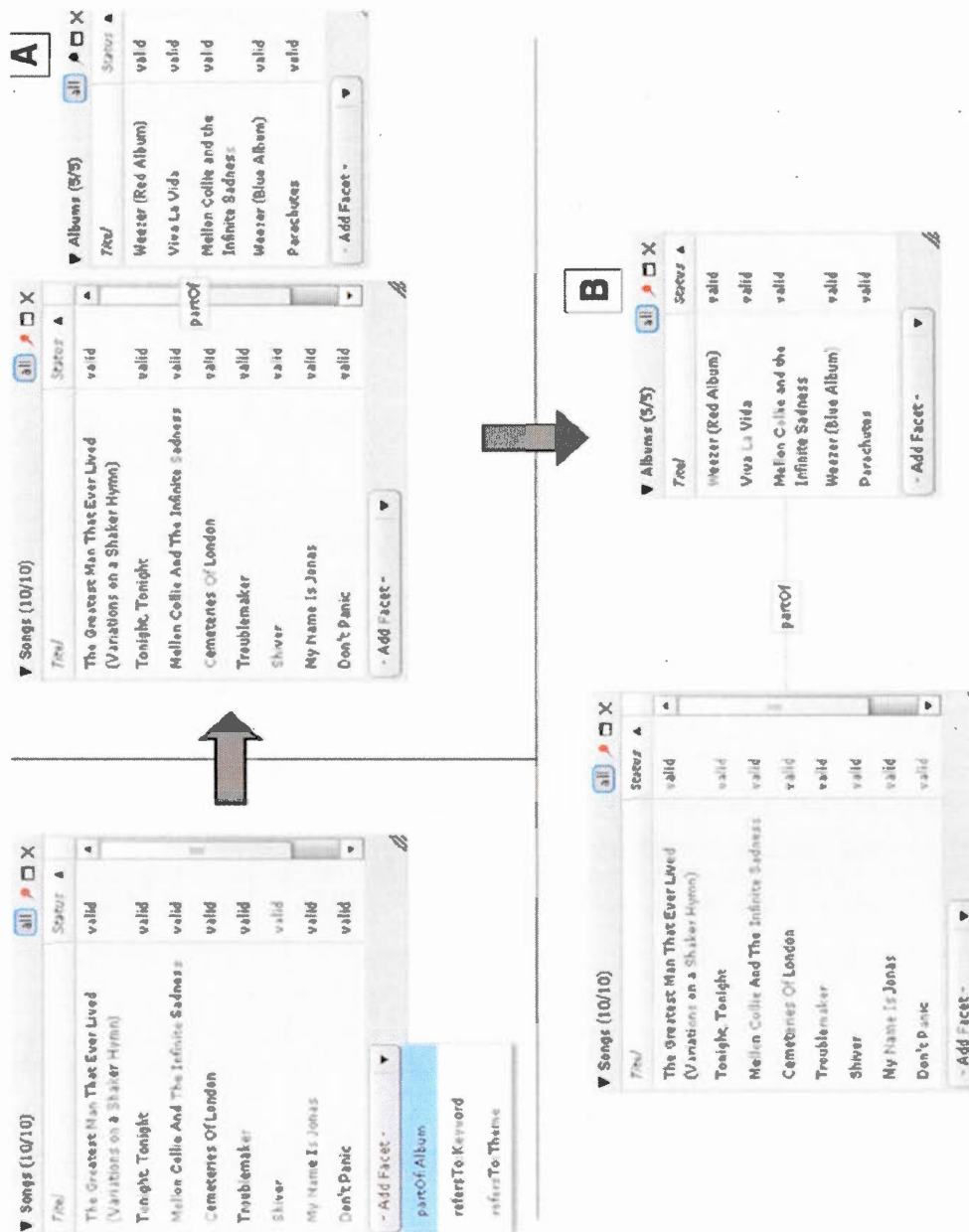


Figure 1.9 Navigateur gFacet (Heim, Ziegler, & Lohmann, 2008)

La figure 1.9 montre un exemple du navigateur *GFacet*. L'utilisateur ajoute de nouvelles facettes au graphique en les sélectionnant dans un menu situé sous la liste

(Figure 1.9, A). Les facettes sont automatiquement arrangées selon une disposition dirigée par la force. Ces facettes peuvent être ajoutées en tant que nouveaux nœuds qui sont connectés par des arêtes étiquetées (Figure 1.9, B).

Tabulator (Tim Berners-Lee & Sheets, 2006) est un autre type de navigateur sémantique, mais à base d'arborescence. Ce dernier utilise des concepts référencés (tels que *FOAF*) pour interpréter le contenu des données liées, récupérées par des *IRIs* absolus qui pointent vers des documents RDF. *Tabulator* se base sur une hiérarchie imbriquée (arborescence) d'étiquettes de texte pour afficher des niveaux croissants de raffinement tant que l'utilisateur navigue à travers cette arborescence. Il offre une composition assistée des requêtes SPARQL. Il supporte à la fois des requêtes simples (par sélection / concept exemple) et des requêtes complexes.

Dans *Tabulator* (Figure 1.10, A), une liste d'IRI (par exemple, *TheTabulatorProject*) est disponible avec la possibilité de les étendre. En mettant en évidence un champ, l'utilisateur spécifie un modèle de requête en se basant sur une série d'arêtes reliant ce champ au nœud racine. Par exemple, la navigation à partir du projet *TheTabulatorProject*, l'extension de « Adam Lerer » sous « developers » et la mise en évidence de la propriété *AIM chat ID* (en vert) peut constituer la base d'une requête *RDF* composée automatiquement (Figure 1.10, B) et qui va permettre de trouver tous les id de chat des développeurs *Tabulator*. Des contraintes simples peuvent également être ajoutées à la requête. Enfin, plusieurs champs peuvent être mis en évidence pour créer des requêtes plus complexes. Pour permettre aux utilisateurs experts à définir des requêtes plus complexes, *Tabulator* permet également la visualisation et la modification de requêtes *SPARQL* elle-même. Le résultat de la requête est également disponible (Figure 1.10, C).

▼ The Tabulator Project

- Data on location of libraries mostly in the UK
- Semantic Web Coordination Group plan includes calendar info
- TAGmobile road trip BOS→Amers photo locations
- Some flight itinerary options, with times and places

testDataset

▼ Adam Lerer

sparqlEndpoint <http://jena.hpl.hp.com:3040/backstage>

type ► **Person**

AIM Chat ID **adamlerer**

based near

(A)

(B)

Select queries to display: **Query #1**

SPARQL **Debug**

```

SELECT ?v0 ?v1
WHERE {
  <http://dig.csail.mit.edu/2005/ajar/ajaw/data#Tabulator>
  <http://usefulinc.com/ns/doap#developer> ?v0 .
  ?v0 <http://xmlns.com/foaf/0.1/aimChatID> ?v1 .
}

```

Submit SPARQL

(C)

View and Save Current Query **Save Current Query** **Help** **About** **File type...** **Export**

Select queries to display: **Query #1**

Table **Map** **Calendar** **SPARQL** **Debug**

The Tabulator Project developer AIM chat ID	
Adam Lerer	adamlerer
David Sheets	silentOpen

View and Save Current Query **Save Current Query** **Help** **About** **File type...** **Export**

Figure 1.10 Tabulator-composition d'une requête SPARQL (Tim Berners-Lee & Sheets, 2006)

L'interrogation des données à base de navigation est conçue pour être aussi facile et rapide que possible pour un nouvel utilisateur et pour les développeurs qui voudraient étendre leurs propres idées. Cependant, avant de réussir à interroger les données RDF, l'utilisateur a besoin de comprendre la structure de ces données et de déterminer les différents liens existants entre ces données.

Pour ce faire, l'utilisateur a besoin d'explorer toute une arborescence pour comprendre les différents liens entre les données, ce qui peut être une tâche fastidieuse et couteuse en temps et effort. De l'autre côté, si l'utilisateur a besoin d'une propriété/ressource précise, il aura besoin de parcourir chaque fois toute l'arborescence des données afin de sélectionner ce dont il a besoin ; une tâche non simple vu que les données *RDF* peuvent être volumineuses et par la suite le chemin d'accès peut être trop long.

1.4.2 Interrogation des données à base de formulaires

Cette approche permet d'interroger les données RDF et de construire des requêtes à travers des formulaires. C'est une approche populaire où les expressions de requêtes sont créées à partir des éléments d'un formulaire tel que les champs texte, les listes déroulantes, etc. Des exemples de cette approche sont *SPARQL Viz* (Jethro, 2006), *Konduit VQB* (Ambrus, Möller, & Handschuh, 2010) et *Graph Pattern Builder* du projet DBpedia (Auer, et al., 2007).

Graph Pattern builder a été développé spécifiquement pour interroger les données Wikipédia. La figure 1.11 montre une capture d'écran du *Graph Pattern builder*. Les utilisateurs interrogent la base de connaissances grâce à un modèle de graphe qui est constitué d'un ensemble de motifs de triplets. Pour chaque motif de triplet, trois champs du formulaire sont disponibles (Figure 1.11, A). Ces champs représentent le sujet, le prédicat et l'objet d'un motif de triplet. Ils peuvent avoir comme valeur soit des variables, soit des identifiants ou des filtres. Alors que les utilisateurs tapent des mots

clès dans l'un des champs de formulaire, une recherche d'anticipation propose les options appropriées (Figure 1.11, B).

UNIVERSITÄT LEIPZIG pedia

Query Wikipedia

This semantic database contains over 10 million statements extracted from the English Wikipedia.

search for queries | [Most popular](#) | [Upcoming](#)

[Tennis players from Moscow](#)

[Sitcoms set in NYC](#)

[Soccer player with tricot nr. 11, playing for a club having a stadium with >40.000 seats, born in a country with >10M inhabitants](#)

[People influenced by Friedrich Nietzsche](#)

[Films longer than 5 hours](#)

[Space Missions](#)

[Film music composer born 1965](#)

[People being 1.80m tall](#)

[List of Web browser software](#)

[Mayors of US cities higher than 1000m](#)

[Pictures of American guitarists](#)

[Battles in Saxony](#)

[What connects Innsbruck and Leipzig](#)

[Hip hop CDs from Texas Artists](#)

[Scientists and their doctoral advisors](#)

<< 1 >>

Soccer player with tricot nr. 11, playing for a club having a stadium with >40.000 seats, born in a country with >10M inhabitants

Subject	Predicate	Object
?player	currentclub	?club
?player	clubnumber	11
?player	countryofbirth	?country
?club	capacity	>40000
?country		>10000000

[GDP_PPP \(11\)](#)

population_estimate (10)

population_census (9)

established_date2 (8)

established_date1 (6)

established_date3 (5)

GDP_nominal (3)

accessionEUdate (2)

Keen magnifying glass

Click on a column head to sort this page. Results: 10

10 results found in 0.00

Nr.	?player	?country	>40000	>10000000
1	Cicinho	Brazil	80354	187560000
2	Gonzalo Fierro	Chile	62000	16432674
3	Lukas Podolski	Poland	69901	38536869
4	Mark González	South Africa	45362	47432000
5	Michael Thurk	Germany	52000	82438000
6	Ramón Morales	Mexico	72480	107784179
7	Robin van Persie	Netherlands	60432	16336346
8	Stefano Mauri	Italy	82656	58751711

Figure 1.11 Graph pattern builder (Auer, et al., 2007)

Ceux-ci sont obtenus non seulement par la recherche d'identifiants correspondants, mais en exécutant la requête actuellement construite en utilisant une variable pour l'identifiant en cours d'édition et en filtrant les résultats renvoyés pour cette variable pour les correspondances commençant par la chaîne de recherche fournie par l'utilisateur. Cette méthode garantit que l'identificateur proposé soit vraiment utilisé en conjonction avec le motif de graphe en cours de construction et que la requête renvoie effectivement des résultats. En outre, les résultats de la recherche sont classés par la fréquence d'utilisations, montrant les identificateurs couramment utilisés en premier. Tout cela est exécuté en arrière-plan en utilisant la technologie Web 2,0 *AJAX* et est totalement transparent pour l'utilisateur.

Cette méthode d'interrogation est aussi efficace pour l'exploration rapide des données et la création de requêtes *SPARQL*. Cependant, l'utilisateur a besoin dans une première étape de comprendre la structure des données *RDF* : les types et les propriétés possibles de chaque type. Cette information est généralement absente avec ces méthodes d'interrogation. Par conséquent, l'utilisateur sera obligé d'essayer de comprendre la structure des données manuellement, une tâche non facile et non évidente pour l'utilisateur vu que le schéma n'est pas fixe et change d'une source à l'autre.

1.4.3 Interrogation des données à base de graphe

La construction des requêtes à base de graphe permet généralement d'avoir plus de flexibilité que celle à base de formulaire, grâce à des éléments graphiques qui sont généralement reliés par des arcs pour former de simples graphes de type nœud-lien et qui décrivent les expressions de requête. Des exemples de cette approche incluent *NITELIGHT* (Smart, Russell, & Braines, 2008), *RDF-GL* (Hogenboom, Milea, Frasincar, & Uzay, 2010) et *LUPOSDATE* (Groppe, Groppe, & Schleifer, 2011).

RDF-GL (Hogenboom, Milea, Frasincar, & Uzay, 2010) est le premier langage visuel de requête graphique basée sur *SPARQL*, conçu pour les données *RDF*. Ce langage aide l'utilisateur à dessiner une requête graphique en utilisant des symboles, des zones textes et des menus. Les éléments de la requête *SPARQL* sont présentés à travers trois types de formes : les boîtes, les cercles et les flèches. La sémantique des symboles dépend de la forme et des couleurs. La figure 1.12 illustre la traduction d'une requête *SPARQL SELECT* utilisant tous les modificateurs de séquences avec l'outil *RDF-GL*. Le triplet BGP (*?Country j.1:elevation ?elevation.*) est dessiné à l'aide de deux boîtes roses (sujet/objet) liées avec la propriété *j.1:elevation*.

```

PREFIX j.1: <http://www.daml.org/2003/09/factbook/factbook-ont#>
SELECT DISTINCT ?elevation
WHERE
{
    ?country j.1:elevation ?elevation .
}
ORDER BY DESC(?elevation) OFFSET 5 LIMIT 4

```

(a) SPARQL

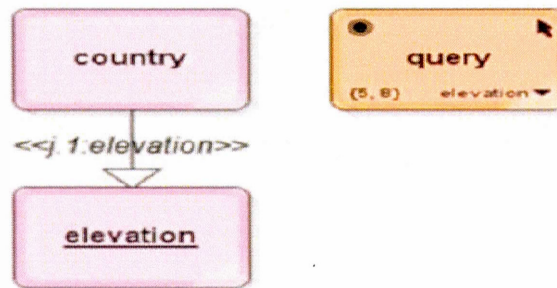


Figure 1.12 RDF-GL (Hogenboom, Milea, Frasinca, & Uzay, 2010)

Ces boîtes permettant d'identifier le sujet et l'objet d'un motif de triplet dans une requête *SPARQL* et elles représentent les variables *? Country* et *? elevation*. Seules des informations sur l'altitude seront retournées dans le jeu de résultats, ce qui est indiqué par le soulignement du nom de la variable *? elevation*. La boîte orange représente les résultats. Les côtés de cette boîte sont utilisés pour afficher les modificateurs de séquence. Le modificateur de séquence *ORDER BY* a été indiqué dans le côté droit de la boîte et précise que le résultat devrait être dans l'ordre décroissant de la valeur de la variable *? elevation*. Dans le côté gauche de la boîte, le modificateur de séquence *OFFSET* a été précisé pour avoir les résultats du 5 à 8. Ainsi, la requête affichée peut être lue comme suit : avoir les différentes altitudes des pays dans le *CIAWorld Factbook*. Les résultats 5, 6, 7 et 8 sont retournés dans l'ordre décroissant.

Cette approche permet de couvrir autant que possible l'expressivité de *SPARQL* tout en maintenant une simplicité intuitive. Cependant, le même problème persiste, vu que l'utilisateur n'a aucune idée du schéma ni des liens entre les différentes classes, il sera obligé dans un premier temps à comprendre la structure des données RDF en formulant des requêtes *SPARQL*. Une tâche fastidieuse et répétitive.

1.5 Conclusion

Ce mémoire vise à répondre aux objectifs suivants : (i) la construction du schéma de données RDF et (ii) l'interrogation des données RDF . Dans ce chapitre, nous avons présenté une analyse des approches de la littérature relative à notre travail. Nous avons regroupé ces approches selon deux axes.

Le premier axe concerne les approches liées à la construction de schéma. Il existe de différentes approches dont l'objectif principal est de donner à l'utilisateur une idée de la structure d'une source de données RDF. Les travaux présentés dans ce contexte sont des travaux sur la visualisation et l'identification de différentes classes, les attributs et les propriétés. En utilisant les outils existants, l'utilisateur peut explorer le schéma de données RDF ; ce qui lui permet d'avoir une idée à propos de la structure des données RDF. Toutefois, certains utilisateurs n'ont pas de requêtes concrètes, mais s'intéressent de manière générique à une certaine collecte de données. Dans de tels cas, une exploration des données en profondeur est appropriée. Cela permet à l'utilisateur de se plonger dans des parties du jeu de données arbitrairement éloignées de son point d'entrée et qu'elle n'a pas anticipé de visiter au début. De telles fonctionnalités ne sont pas offertes par ces approches et se limitent juste à explorer le schéma des données RDF.

Le deuxième axe concerne les approches qui s'intéressent à l'interrogation des données à base de navigateur, de formulaires ou de graphes. Nous avons commencé par présenter les approches les plus importantes sur ce sujet. Ces approches sont efficaces pour la création des requêtes SPARQL ; ce qui permet d'avoir une idée sur le contenu de la source de données. Toutefois, avoir un outil utilisable lorsque l'utilisateur ne possède aucune connaissance sur la structure de la source de données demeure un défi,

plus particulièrement lorsque les informations sur le schéma facilitent beaucoup la composition des requêtes SPARQL.

Durant ce chapitre, nous avons soulevé plusieurs limites des approches présentées, mais la limite principale et commune à toutes ces approches, que ce soit sur construction de schémas ou sur l'interrogation des données, est le fait qu'il n'y a aucune approche complète qui permet à la fois la construction du schéma de données RDF et l'interrogation des données bien qu'elles soient deux fonctionnalités complémentaires et dépendantes l'une de l'autre.

CHAPITRE II

CONSTRUCTION DU SCHÉMA DE DONNÉES RDF

2.1 Introduction

L'utilisation d'une source de données RDF nécessite de disposer de connaissances sur son contenu, afin de pouvoir déterminer quelles parties de la source sont pertinentes pour une requête ou une application donnée. Une description de la source peut être utile pour cibler des ressources ou des propriétés pertinentes lors de l'écriture d'une requête SPARQL. Comme le schéma est important pour comprendre les données RDF, l'idée est de construire une approximation du schéma des données. Cette approximation ne vise pas à reproduire le schéma réel, mais plutôt à construire un schéma approximatif résumant l'ensemble de données RDF dont l'objectif est d'assister à la formulation des requêtes. Il faut tenir compte du fait que le graphe RDF est hétérogène, c'est-à-dire qu'une ressource RDF peut n'avoir aucun type, qu'un ou plusieurs types et qu'elle puisse ou non être liée à une autre ressource. De plus, deux ressources du même type ne possèdent pas les mêmes propriétés et ne suivent pas la même structure, car les propriétés sont optionnelles.

Dans ce contexte, les classes du schéma approximatif seront composées de groupement d'instances avec et sans type défini. Chaque groupement représente un ensemble de ressources qui partagent des propriétés communes. Nous enrichirons cette

approximation du schéma avec des statistiques qui permettent de fournir un profil de la source de données tel que le nombre d'instances par groupement. Toutes ces informations sont déduites à partir du jeu de données RDF traité et elles sont de nature à faciliter l'exploration, l'interrogation de la source de données et la composition assistée des requêtes SPARQL. En plus, avoir une vue globale de la structure de données permet de cibler des groupements pertinents pour un besoin particulier et de limiter l'exploration à ces groupements.

Pour illustrer notre problématique, nous considérons tout au long de ce chapitre un exemple de graphe RDF extrait d'une source de données Jamendo (Raimond, 2016), présenté à travers la figure 2.1. Jamendo est un grand référentiel de musique sous licence Creative Commons, basé en France. Le RDF correspondant représente des informations relatives à un ensemble d'artistes et les différentes caractéristiques d'enregistrements des chansons. L'extrait de données RDF de la figure 2.1 fournit des informations sur deux artistes de la musique, leurs noms, les enregistrements qu'ils ont réalisés, les titres et les images des enregistrements s'il existe. Un si petit extrait utilise quatre ontologies, à savoir l'ontologie FOAF (avec le préfixe label *foaf*), l'ontologie Dublin Core (avec le préfixe label *dc*), l'ontologie musicale (avec le préfixe label *mo*) et l'ontologie événement (avec le préfixe label *event*).

Les utilisateurs ne peuvent pas facilement comprendre les données si le jeu de données RDF est volumineux. Les méthodes et les outils de collecte des informations communs supposent un point de départ de données relationnelles avec un schéma spécifique au domaine. Cette hypothèse ne tient pas pour les données du web sémantique.

Notre objectif consiste, étant donné un graphe RDF, à identifier les classes relatives à ce graphe, en d'autres termes, à répondre à la question « de quoi est-il question dans ce graphe? » Déterminer les classes n'est pas simple, car l'identification d'une classe

dépend de la présence de la propriété *rdf:type*. Cependant, l'absence de type de certaines ressources rend difficile l'identification de leurs classes.

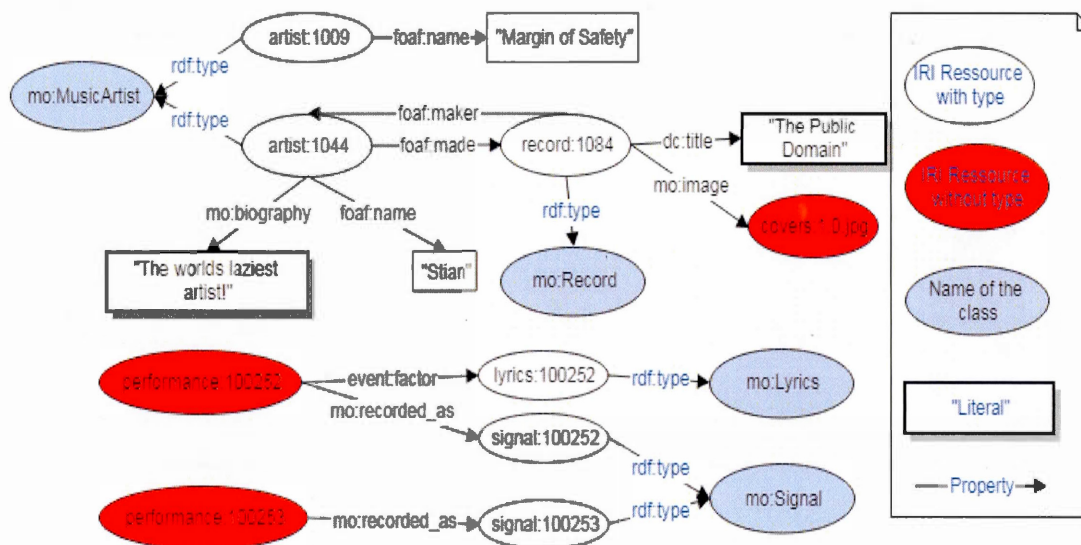


Figure 2.1 Extrait de la source de données Jamendo (Raimond, 2016)

Dans la section suivante, nous allons définir, plus formellement, la problématique traitée. Par la suite, nous expliquons comment identifier les classes pour les ressources avec type. Nous expliquons également comment identifier les classes dans le cas des ressources sans type et nous terminons le chapitre par présenter notre méthode de modélisation de l'approximation du schéma et précisement de la structure des données RDF qui sera représentée à travers un diagramme de classes enrichi par des statistiques.

2.2 Problématique et objectif

Vu que dans le graphe RDF, il n'y a pas des classes qui permettent d'identifier les ressources sans type alors nous parlons plus d'un groupement de ressources qu'une classe afin de généraliser. Ainsi, nous pouvons formuler le problème de l'identification des groupements de ressources comme suit : étant donné un graphe RDF,

l'identification des groupements est la découverte des groupements de ressources, non nécessairement disjoints, chacun représentant un ensemble de ressources ayant au moins une propriété en commun. L'ensemble des propriétés des ressources d'un groupement définissent les propriétés du groupement. Les ressources peuvent avoir plus qu'un type et par la suite elles peuvent appartenir à plus qu'un groupement ce qui explique pourquoi les groupements ne sont pas nécessairement disjoints.

Deux ressources appartenant au même groupement peuvent ne pas avoir les mêmes propriétés. Prenons l'exemple de la figure 2.1 : nous pourrions remarquer qu'il y a deux types : *mo:MusicArtist* et *mo:Record* et qui permettent de représenter deux groupements *MusicArtist* et *Record*. Les ressources *mo:artist1* et *mo:artist2* appartiennent au même groupement *MusicArtist*. Ce dernier est défini par les propriétés des ressources *mo:artist1* et *mo:artist2* et qui sont : *foaf:name*, *foaf:maker*, *foaf:made* et *mo:biography*. Cependant, les ressources *artist:1009* et *artist:1044* n'ont pas les mêmes propriétés puisque *artist:1009* est défini juste par les propriétés *foaf:name* et *foaf:maker*. Par contre, la ressource *artist:1044* est définie par les propriétés *foaf:name*, *foaf:made* et *mo:biography*. Par conséquent, les propriétés des groupements sont optionnelles c'est-à-dire une instance appartenant à un groupement, n'a pas forcément toutes les propriétés de ce groupement.

Une ressource *i* de type non défini ou sans type est une ressource qui n'a pas la propriété *rdf:type*, c'est-à-dire il n'existe aucun triplet de la forme *i rdf:type r* où *r* est une classe.

Une ressource *i* de type défini ou avec type est une ressource qui a la propriété *rdf:type*, c'est-à-dire il existe au moins un triplet de la forme *i rdf:type r* où *r* est une classe.

En nous basant sur le caractère principal du groupement qui est les propriétés optionnelles, nous pouvons affirmer qu'il est probable que des ressources non définies soient liées à un même centre d'intérêt et qu'elles forment ainsi un groupement même

s'elles partagent au moins une propriété en commun. Le problème d'identification des groupements peut donc être ramené à la recherche des propriétés en commun.

2.3 Regroupement des instances avec type

Dans cette section, nous essayons de déterminer l'ensemble des groupements qui représentent les ressources avec type. Pour ces ressources, la propriété en commun qui va permettre d'identifier les différents groupements est *rdf:type*. Dans ce cas, chaque groupement de ressources représente une classe définie. Ainsi, l'identification des groupements de ressources avec type consiste tout simplement à chercher les différentes classes définies dans un graphe RDF. L'ensemble des propriétés des ressources d'un groupement définissent les propriétés du groupement. Par conséquent, les groupements et leurs propriétés peuvent être déterminés à partir de la requête SPARQL suivante.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT distinct ? classe ? prop
WHERE {
    ?ressource rdf:type ? classe.
    OPTIONAL { ?ressource ? prop ?objet.
                Filter (?prop != rdf:type ) }.
}
ORDER BY ?classe
```

La requête SPARQL permet de collecter tous les groupements de ressources qui représentent les différentes classes d'une source de données RDF. Les groupements sont identifiés par la variable *?classe*. Pour chaque groupement *?classe*, nous explorons toutes les ressources *?ressource* pour collecter leurs propriétés *?prop*.

Prenons l'exemple de la figure 2.1 et appliquons la requête SPARQL mentionnée ci-dessus, nous obtenons le résultat présenté par la figure 2.2 . Le résultat montre qu'il y a quatre classes et par la suite quatre groupements et qui sont *mo:MusicArtist*, *mo:Record*, *mo:lyrics* et *mo:signal*. Par exemple : le regroupement MusicArtist représente les ressources *artist:1009* et *artist:1044*. Les propriétés associées à ces ressources sont : *mo:biography*, *foaf:made* et *foaf:name*. Par conséquent, ces propriétés représentent les propriétés du groupement MusicArtist. Le regroupement *Record* représente la ressource *record:1084* et qui a comme propriétés : *dc:title*, *mo:image* et *foaf:maker*. Ces propriétés représentent les propriétés du groupement *Record*. Les regroupements *mo:lyrics* et *mo:signal* représentent des instances qui n'ont pas des propriétés.

classe	prop	
<http://purl.org/ontology/mo/Lyrics>		
<http://purl.org/ontology/mo/Signal>		
<http://purl.org/ontology/mo/Record>	<http://xmlns.com/foaf/0.1/maker>	
<http://purl.org/ontology/mo/Record>	<http://purl.org/dc/elements/1.1/title>	
<http://purl.org/ontology/mo/Record>	<http://purl.org/ontology/mo/image>	
<http://purl.org/ontology/mo/MusicArtist>	<http://xmlns.com/foaf/0.1/name>	
<http://purl.org/ontology/mo/MusicArtist>	<http://xmlns.com/foaf/0.1/made>	
<http://purl.org/ontology/mo/MusicArtist>	<http://purl.org/ontology/mo/biography>	

Figure 2.2 SPARQL-Les groupements des instances avec type et leurs propriétés

2.4 Regroupement des instances sans type

Certaines instances n'ont pas la propriété *rdf:type* et par la suite, le/les groupement(s) de ressources n'est/ne sont pas facile à le/les déterminer vu que nous ne pouvons pas affirmer que deux instances sans type appartiennent au même groupement ou à deux groupements différents. Pour résoudre ce problème, une solution intuitive consisterait

à regrouper toutes les instances sans type dans un seul groupement. Cependant, ces instances peuvent être complètement différentes et ne représentent pas le même groupement. Ensuite, cette représentation est très sommaire, ce qui n'aide pas l'utilisateur à comprendre la structure du graphe RDF. Par la suite, l'utilisateur sera obligé à construire de petites requêtes afin de comprendre mieux la structure des données. Une autre solution intuitive consiste à créer pour chaque propriété des instances sans type un groupement. Dans ce cas, une instance qui a plus qu'une propriété sera représentée par plus qu'un groupement dont nous ne pourrions pas les identifier, ce qui n'aide pas l'utilisateur à comprendre la structure des données RDF et donc ne facilite pas la formulation des requêtes. En plus, nous avons le risque d'explosion des groupements s'il y a un grand nombre de propriétés vu que le nombre de groupements déterminé à partir des instances sans type sera égal au nombre de propriétés et dans ce cas cette solution ne rassure pas une bonne lisibilité des groupements.

En évitant les inconvénients des deux propositions précédentes, la solution choisie consiste à regrouper ces instances en se basant sur les propriétés en commun. Rappelons que dans un RDF, deux instances qui ont le même type peuvent déjà ne pas avoir les mêmes propriétés puisque les propriétés sont optionnelles. En suivant le même principe, deux instances sans type appartiennent au même groupement si et seulement s'elles ont au moins une propriété en commun avec n'importe quelle propriété différente de l'ensemble des propriétés prédéfinies tel que *rdfs:range*, *rdfs:domain*, *rdf:type*, *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:label*, *rdfs:comment*, etc.

En utilisant une relation binaire *R*, nous formalisons l'extraction des groupements de ressources sans types dans un graphe RDF comme suit : Deux instances *i* et *j* sont liées par *R* si et seulement si $\text{prop}(i) \cap \text{prop}(j) \neq \emptyset$, où $\text{prop}(i)$ représente l'ensemble des propriétés de la ressource *i*. Une propriété *p* appartient à $\text{prop}(i)$ si et seulement s'il

existe dans l'ensemble de données un RDF un triple ayant i pour sujet et p comme propriété (c'est-à-dire un prédicat).

Prenons l'exemple de la figure 2.3, nous avons cinq instances qui n'ont pas de types $\{i1, i2, i3, i4, i5\}$.

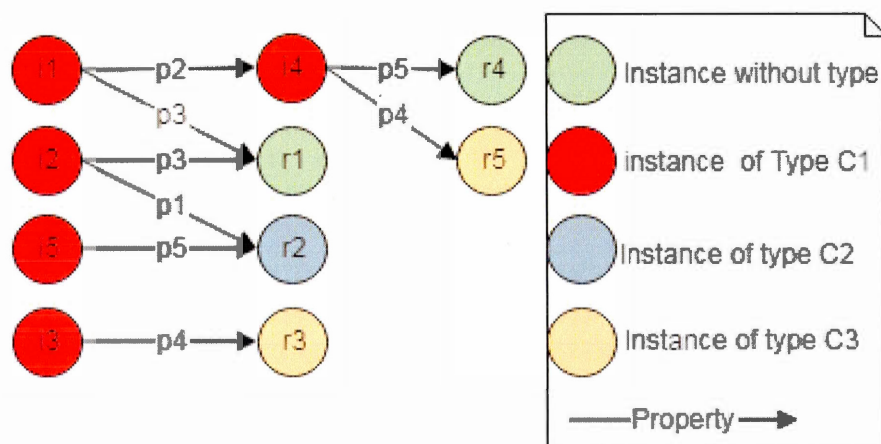


Figure 2.3 Exemple de graphe avec des instances sans type

Nous avons créé une matrice de similarité qui représente la relation binaire R (voir Tableau 2.1).

Tableau 2.1 Initialiser la matrice de similarité

Ressources	i1	i2	i3	i4	i5
i1	1	1	0	0	0
i2	1	1	0	0	0
i3	0	0	1	1	0
i4	0	0	1	1	1
i5	0	0	0	1	1

Cette matrice permet de déterminer s'il y a des propriétés en commun entre les différentes instances sans type. Dans cette matrice, les lignes et les colonnes représentent l'ensemble des instances sans type. Soit le couple (I, J) de la matrice de similarité avec I et J deux instances sans type, alors il y a deux valeurs possibles, soient :

- 0 : il n'existe aucune propriété commune entre I et J ;
- 1 : il existe au moins une propriété commune entre I et J .

Nous définissons la relation R^* comme étant la fermeture transitive de R . La fermeture transitive R^* est la plus petite relation binaire transitive contenant R . La relation binaire R^* est une relation d'équivalence par construction et ses classes d'équivalence sont exactement les groupes des instances sans types dans un graphe RDF et peuvent être obtenues en calculant la fermeture transitive de R .

Dans le cas de R dans le tableau 2.1, il existe deux classes d'équivalence $\{i1, i2\}$ et $\{i3, i4, i5\}$ dans R^* . Ces deux classes représentent les groupes d'instances sans types dans le graphe RDF. Comme les groupes sont des classes d'équivalence, ils sont nécessairement disjoints. Ce disjoint n'était pas ciblée, mais c'est une bonne caractéristique résultant de notre formalisation du problème du clustering. Nous caractérisons chaque groupe par les propriétés de leurs ressources sans types respectives.

Cette approche nous permet d'avoir des groupements maximaux vu que les propriétés d'un groupement représentent l'union de toutes les différentes propriétés des instances de ce groupement. Dans ce cas, ça se peut qu'un groupement ait une dizaine de propriétés, cependant, une instance i appartenant à ce groupement pourrait avoir juste une propriété. Par la suite, les propriétés de chaque groupement sont optionnelles. L'avantage de cette approche est que les propriétés de chaque instance sont incluses dans l'ensemble de propriétés de son groupement. Par conséquent, il n'y a aucun chemin du graphe RDF initial perdu avec notre approche.

Même si cette approche présente une approximation du schéma pour les instances sans types, notre objectif ne consiste pas à reproduire le schéma réel des données, mais plutôt à donner une idée de la structure des ressources ce qui facilite la composition assistée des requêtes SPARQL.

2.5 Modélisation du schéma RDF

2.5.1 Visualisation avec le diagramme de classes

Le regroupement des instances avec et sans type permet de donner une idée de la structure des données RDF. Cependant, l'utilisateur a besoin d'explorer cette structure d'une façon visuelle et simple. Le diagramme de classes d'UML⁴ représente un ensemble d'objets partageant les mêmes spécifications de propriétés, de contraintes et de sémantique. Il permet de constituer la vue logique du système : les classes (types d'objets), leurs attributs et les associations (relations) qui peuvent entretenir les objets du système. En plus, c'est une représentation fidèle et compréhensible par les utilisateurs. Pour toutes ces raisons, nous avons choisi le diagramme de classes pour présenter les différents groupements qui représentent l'approximation du schéma.

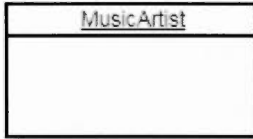
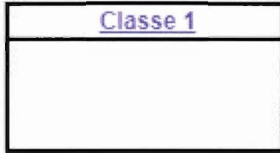
Pour que l'utilisateur puisse comprendre correctement le modèle, la visualisation devrait être :

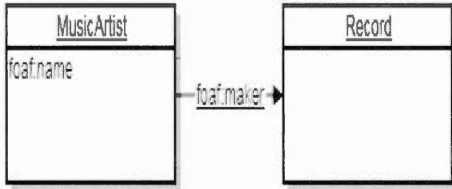
- Complète : tous les attributs et les propriétés qui figurent dans le graphe RDF sont présentés ;
- Précise : Chaque classe représente un groupement et devrait être présenter avec les attributs et les propriétés correspondants.

⁴ https://fr.wikipedia.org/wiki/Diagramme_de_classes

Le tableau 2.2 définit et illustre nos règles de conversion détaillées pour la construction du diagramme de classes UML à partir des données RDF.

Tableau 2.2 Les règles de conversion pour la construction du diagramme de classes UML à partir des données RDF

Scénarios	Correspondances UML
Classes UML : Groupement des instances avec type défini	Une classe dont le nom est le type indiqué dans la source RDF. Exemple : Artist:1044 <code>rdf:type mo:MusicArtist</code> . → <i>MusicArtist</i> est une classe UML. 
Classes UML : Groupement des instances sans types défini	Une classe dont le nom sera Classe I avec I un compteur qui augmente à fur et à mesure l'identification des groupements des instances sans type. Ces classes seront représentées avec une autre couleur afin de les distinguer des classes qui représentent les groupements des instances avec type. Exemple : <i>Performance:100252</i> <code>mo:recorded_as signal:100252</code> . → Supposons que <i>Performance:100252</i> appartient au groupement1 alors groupement1 sera présenté par Classe1. 
Attributs UML :	La propriété relative à cet objet sera un attribut de la classe. Exemple : Artist:1044 <code>rdf:type mo:MusicArtist</code> .

Scénarios	Correspondances UML
L'objet est de type littéral	<i>Artist:1044 foaf:name « Stian ».</i> →L'objet « Stian » est de type littéral alors la propriété <i>foaf:name</i> sera un attribut de la classe <i>MusicArtist</i>.  <pre> classDiagram class MusicArtist { foaf:name } </pre>
Association UML : L'objet est de type ressource	La propriété relative à cet objet sera l'association entre deux classes. Exemple : <i>Artist:1044 rdf:type mo:MusicArtist.</i> <i>Artist:1044 foaf:maker record:1084.</i> <i>record:1084 rdf:type mo:Record.</i> →L'objet <i>record:1084</i> est de type ressource alors la classe <i>MusicArtist</i> est associée à la classe <i>Record</i> grâce à la propriété <i>foaf:maker</i>.  <pre> classDiagram class MusicArtist { foaf:name } class Record MusicArtist --> Record : foaf:maker </pre>

Bien que notre diagramme garde les notations de base d'un diagramme de classes classique telles que les classes, les associations et les attributs, etc, il présente une approximation du schéma, précisément de la structure des données RDF et il est conçu d'une façon un peu différente. En effet, une instance peut appartenir à deux classes différentes vu qu'une instance peut avoir plus qu'un type, une telle spécification n'existe pas dans un diagramme de classes classique.

Ensuite, les attributs et les associations d'une classe sont optionnels. Une instance appartenant à une classe n'a pas forcément toutes les propriétés de cette classe et donc deux instances de la même classe peuvent ne pas avoir les mêmes propriétés. Par la suite, les classes des instances sans type sont représentées avec une autre couleur afin de les distinguer des classes des instances avec type. Enfin, nous avons ajouté des statistiques afin d'aider à comprendre mieux la structure des données RDF et par la suite à mieux écrire les requêtes SPARQL. Dans ce qui suit, nous allons expliquer plus en détail les statistiques ajoutées au diagramme de classes.

2.5.2 Ajout des statistiques au diagramme de classes

Nous avons ajouté des statistiques au diagramme de classes afin d'avoir une meilleure idée à propos de la structure des données RDF. Ces statistiques permettent de bien concevoir les requêtes SPARQL et de faciliter l'exploration des données. Les statistiques sont calculées comme suit :

- **Nombre d'instances de chaque groupement** : Pour chaque groupement C , nous déterminons le nombre des instances i tels que $i \text{ rdf:type } C$. La requête SPARQL est comme suit :

« *SELECT ? C, count (distinct ? i) as ? count*
WHERE {?i rdf:type ? C.} »

Prenons l'exemple de la figure 2.1, nous avons deux instances du groupement *MusicArtist* et une instance du groupement *Record*.

- **Nombre d'instances par propriété** : Pour chaque propriété p d'un groupement donné C , nous déterminons le nombre d'instances i distinctes qui ont cette propriété. Autrement dit, le nombre de triplets distincts (i,p,o) avec $i \text{ rdf:type } C$ et $o \text{ rdf:type } C1$. La requête SPARQL est comme suit :

« *SELECT ? C, ?p, count (distinct ? i) as ? count*

```

WHERE {?i p ? o.
       ?i rdf:type ? C.
       ?o rdf:type ? CI.}
GROUP BY ?C, ? p »

```

Prenons l'exemple du groupement *MusicArtist* de la figure 2.1, nous avons deux instances qui ont la propriété *name*. Par contre, nous avons juste une instance qui a les propriétés *biography* et *made*.

Ces statistiques permettent à l'utilisateur d'avoir une idée à propos des données RDF et vu que les instances du même groupement n'ont pas forcément les mêmes propriétés alors grâce à ces statistiques, l'utilisateur peut déterminer les propriétés optionnelles en comparant juste le nombre des instances par groupement avec le nombre d'instances par propriété. Si le nombre est le même, l'utilisateur peut conclure que toutes les instances d'un groupement donné ont la propriété en question sinon, l'utilisateur peut conclure que cette propriété est optionnelle et n'est pas présente pour toutes les instances de ce groupement et dans ce cas, l'utilisateur peut ajuster la requête en ajoutant par exemple le motif de graphe *OPTIONAL*⁵. Ce motif permet d'avoir des interrogations qui puissent ajouter des informations à la solution où l'information est disponible, mais qui ne rejettent pas la solution parce que des parties du motif d'interrogation ne s'apparient pas. Sans ce motif, la requête risque de retourner un ensemble vide comme résultat, par contre s'il est indiqué dans la requête, même si la partie optionnelle ne correspond pas, elle ne crée aucune liaison, mais elle n'élimine pas la solution.

⁵ <http://www.yoyodesign.org/doc/w3c/rdf-sparql-query/>

2.6 Un exemple illustratif

Notre approche de construction du schéma RDF a été appliquée sur la source de données Jamendo dont un extrait est illustré par la figure 2.1. Nous commençons par identifier les groupements des instances avec et sans type. Ensuite, ces groupements sont modélisés dans un diagramme de classes en appliquant des règles de conversion. Un tel diagramme UML fournit la structure de données et leurs statistiques (Figure 2.4).

Ce diagramme contient 17 classes dont 11 le type était déjà défini dans la source de données RDF et 6 classes ont été déterminées en appliquant la relation d'équivalence. Pour chaque classe, le nombre d'instances est fourni. Ainsi, le nombre d'instances est fourni pour chaque propriété de chaque classe. Ce diagramme est pratique pour les utilisateurs puisqu'il permet d'explorer et de comprendre la structure des données RDF d'une façon rapide, intuitive, efficace et visuelle.

2.7 Conclusion

Dans ce chapitre, nous avons proposé une approche pour construire le schéma de données RDF. Cette approche ne vise pas à reproduire le schéma réel des données, mais elle vise plutôt à construire un schéma approximatif dont l'objectif est d'assister à la formulation des requêtes.

Les classes du schéma approximatif seront composées de groupements d'instances avec et sans types défini. Ces groupements ont été identifiés directement à partir des sources de données RDF à l'aide des requêtes SPARQL. Pour les instances définies, les groupements d'instances sont identifiés à la base de la propriété *rdf:type*. Chaque groupement correspond à une classe. Deux instances appartiennent au même groupement s'elles ont le même type. Pour les instances non définies, l'approche

consiste à regrouper ces instances en se basant sur les propriétés communes autres que *rdf:type*. En effet, nous avons défini la relation suivante : une instance *i* appartient à un groupement si et seulement s'il existe au moins une instance *j* appartenant au même groupement avec $\text{prop}(i) \cap \text{prop}(j) \neq \emptyset$. Cette relation représente bel et bien une relation d'équivalence. En appliquant la fermeture transitive, nous pourrions déterminer les classes d'équivalence qui seront les différents groupements des instances sans type.

Le regroupement des instances avec et sans type permet de donner une idée de la structure des données RDF. Cependant, l'utilisateur a besoin d'explorer cette structure d'une façon visuelle et simple. C'est pourquoi nous avons choisi le diagramme de classes pour présenter les différents groupements de l'approximation du schéma RDF. Dans le diagramme de classes, les classes représentent les groupements des instances déterminés. Les attributs et les associations de chaque classe ont été déterminés à partir des propriétés de chaque groupement. Des statistiques ont été ajoutées également au diagramme de classes telles que le nombre d'instances par classe et le nombre d'instances par propriété. Ces statistiques permettent d'avoir une meilleure idée à propos de la structure des données RDF et permettent d'identifier les propriétés optionnelles.

CHAPITRE III

INTERROGATION DES DONNÉES RDF

3.1 Introduction

Dans le chapitre précédent, nous avons proposé une approche de construction du schéma de données RDF qui aide l'utilisateur à comprendre la structure des données RDF. Le schéma de données RDF sera accompagné par la possibilité d'explorer les données RDF, le but ultime de ce chapitre. Ainsi, notre approche consiste à faciliter l'exploration des données à travers la composition assistée de requête sur les données RDF.

La recherche mots-clés est un moyen facile à utiliser par l'utilisateur pour l'expression de sa requête. Ce type de recherche est plus utilisé dans le cas de documents textuels. Mais, plusieurs études proposent également des approches pour la recherche mots-clés dans les données structurées, à savoir des données relationnelles (Tong, Zhang, & Cheng, 2014), principalement en raison de sa facilité d'utilisation et parce que l'utilisateur n'a besoin d'aucune information supplémentaire pour exprimer son besoin à part un ensemble de mots-clés.

Dans ce contexte, notre approche consiste à fournir aux utilisateurs les moyens qui facilitent la composition assistée des requêtes SPARQL. L'approche se compose de trois parties : le chargement initial du jeu de données, la recherche du voisinage d'un élément du graphe RDF et la construction assistée des requêtes SPARQL.

Nous présentons en détail les différentes parties de notre approche dans les sections 3.2, 3.3 et 3.4 respectivement. Nous résumons nos contributions et concluons ce chapitre dans la section 3.5.

3.2 Chargement initial du jeu de données

Le chargement initial du jeu de données consiste à récupérer les valeurs possibles des ressources, des littéraux et des propriétés à partir d'une source de données RDF. Ces valeurs seront sauvegardées dans trois listes : List_All_Subject, List_All_Property et List_All_Object. Cette structure est inspirée de la structure d'un motif de triplet vu qu'une requête SPARQL est basée sur un ensemble de motif de triplet. La liste List_All_Subject contient tous les sujets possibles d'une source de données RDF déterminés à partir de la requête SPARQL suivante :

```
select distinct ? subject
where {?subject ?property ?object.}
```

La liste List_All_Property représente toutes les propriétés d'une source de données RDF. Elle contient l'ensemble de solutions de la requête SPARQL suivante :

```
select distinct ? property
where {?subject ?property ?object.}
```

La liste List_All_Object est la liste des objets possibles d'une source de données RDF. Cette liste est l'ensemble de solutions de la requête SPARQL suivante :

```
select distinct ? object
where {?subject ?property ?object.}
```

Pour chaque élément retourné, une description est fournie en affichant le label et le commentaire grâce aux propriétés *rdfs:label* et *rdfs:comment* s'ils existent et ce qui permet à l'utilisateur d'avoir plus de détail à propos de chaque élément retourné. Cette exploration est beaucoup plus simple et rapide parce que l'utilisateur n'a plus besoin de créer des requêtes SPARQL pour explorer le contenu d'une source de données RDF et qui pourrait nécessiter beaucoup de temps surtout pour un utilisateur non expert qui ne maîtrise pas la syntaxe SPARQL.

La figure 3.1 représente l'interface qui sert à construire des motifs de triplet. Les différentes fonctionnalités de cette interface seront expliquées plus en détail dans le chapitre 4. Ces motifs de triplet permettent de construire des requêtes SPARQL ce qui nous motive à présenter les données extraites du graphe RDF sous forme de trois listes. En effet, chaque liste représente les valeurs possibles d'un élément du motif de triplet. La figure 3.1 montre les différents éléments de l'extrait de la source de données Jamendo (Raimond, 2016) présentée par la figure 2.1. Nous pourrions avoir toutes les valeurs possibles des éléments de ce graphe, autrement dit, les sujets tel que *records1*, *artist1*, etc, les propriétés tel que *factor*, *title*, *recorded_as*, etc., et les différents objets que ce soit des littéraux tel que « Margin of Safety » ou des ressources telles que *signal1*, *image1*, etc.

Subject	Predicate	Object
http://dbtune.org/jamendo/artist/artist1	http://purl.org/NET/c4dm/event.owl#factor	2007-06-06 15:41:56
http://dbtune.org/jamendo/artist/artist2	http://purl.org/dc/elements/1.1/date	Margin of Safety
http://dbtune.org/jamendo/lyrics/lyrics1	http://purl.org/dc/elements/1.1/title	Stan
http://dbtune.org/jamendo/performance/performance1	http://purl.org/ontology/imoblog/imoblography	The worlds laziest artist!
http://dbtune.org/jamendo/performance/performance2	http://purl.org/ontology/imoblog/imoblogimage	Wide Open Kisscool
http://dbtune.org/jamendo/record/record1	http://purl.org/ontology/imoblog/imoblogrecorded_as	http://dbtune.org/jamendo/artist/artist1
http://dbtune.org/jamendo/signal/signal1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbtune.org/jamendo/lyrics/lyrics1
http://dbtune.org/jamendo/signal/signal2	http://xmlns.com/foaf/0.1/made	http://dbtune.org/jamendo/record/record1

Figure 3.1 Chargement initial du jeu de données

3.3 Recherche par des mots-clés

Nous avons implémenté une approche de recherche par mot-clé qui permet de trouver des ressources, des propriétés ou des littéraux. Par exemple, si l'utilisateur cherche un mot dans une ressource sujet alors la recherche s'effectue juste au niveau de la liste `List_All_Subject` définie dans la section 3.3. Pour chaque ressource de cette liste, nous examinons l'URI, le commentaire et le label associé à cette ressource. Si le mot saisi est trouvé dans l'un de ces trois endroits indiqués, alors la ressource s'ajoute au résultat qui sera retourné à l'utilisateur par la suite. Une fois l'utilisateur choisit une ressource, la liste des valeurs possibles des prédicats et des objets se restreint en appliquant des requêtes filtres. La recherche par des mots-clés est basée sur la suggestion de complétion et l'exploration du voisinage d'un élément du graphe.

3.3.1 Suggestion de complétion

Lorsque nous parlons de la complétion, nous nous référons aux suggestions qui apparaissent au fur et à mesure que nous tapons les premières lettres d'une nouvelle recherche. L'objectif est de suggérer ainsi des termes de recherche similaire à ce que nous cherchons, voir à nous faire gagner du temps en prévoyant la requête.

Notre approche de suggestion consiste à la composition de requêtes à partir des mots-clés saisis par l'utilisateur pour lui afficher des suggestions de recherche. La recherche peut être effectuée soit au niveau des ressources sujet, au niveau des propriétés ou au niveau des ressources/littéraux objet. Les suggestions de recherche proviennent respectivement de la liste `List_All_Subject` ou de la liste `List_All_Property` ou de la liste `List_All_Object` selon le cas. Au début, la liste correspondante sera parcourue au complet. A chaque fois, le mot saisi sera vérifié s'il existe en analysant la valeur des ressources/littéraux. Si c'est le cas, la ressource ou le littéral sera ajouté à la liste des suggestions de recherche. Sinon, la recherche sera effectuée au niveau des

commentaires et par la suite au niveau de la liste des labels dans le cas d'une ressource. Notons qu'une ressource peut avoir aucun ou plusieurs commentaires/labels. Enfin, la nouvelle liste de suggestion de recherche sera envoyée et mise à jour instantanément au fur et à mesure que l'utilisateur saisit des mots-clés. Reprenons l'exemple de l'extrait de la source de données Jamendo (Raimond, 2016) présentée par la figure 2.1, si l'utilisateur saisit des mots-clés comme « signal » au niveau du sujet tel que montré par la figure 3.2, la liste des ressources qui contiennent ce mot-clé sera retournée.

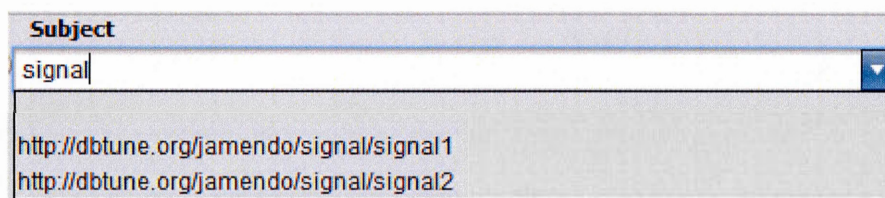


Figure 3.2 Suggestion de complétion

Cette approche aide l'utilisateur à trouver des informations plus rapidement. Ainsi, l'utilisateur peut cibler et affiner sa requête en temps réel. En plus, en saisissant un mot, au fur et à mesure, la liste de suggestion peut se restreindre jusqu'à ce qu'il n'y aurait aucune suggestion et ce qui veut dire que le mot saisi n'existe pas dans la source de données RDF.

Dans ce cas, l'utilisateur va se rendre compte que c'est une piste inutile à l'explorer et il va essayer d'explorer d'autres pistes de recherche puisqu'il a déjà une idée des ressources possibles à trouver dans la source de données RDF grâce à la liste des suggestions de recherche. Par conséquent, cette méthode reste plus efficace et plus rapide qu'écrire une requête SPARQL qui ne retourne en fin de compte aucun résultat.

3.3.2 Voisinage d'un élément du graphe RDF

Une fois que l'utilisateur a exploré la liste des ressources possibles, il se peut qu'il ait trouvé ce qu'il cherche et il n'a plus besoin d'explorer encore plus. Sinon, s'il veut aller

plus loin, il peut explorer le voisinage d'un élément du graphe RDF. En effet, les valeurs possibles des éléments du graphe non encore définies seront filtrées et déterminées en fonction des valeurs définies par l'utilisateur pour les autres éléments de graphe liés. Dans ce cas, un ensemble de requêtes SPARQL sera exécuté en arrière-plan afin de restreindre et limiter la zone de recherche. Tout cela est totalement transparent pour l'utilisateur qui n'a pas besoin d'avoir des connaissances à propos des requêtes SPARQL. Par la suite, cette approche aide l'utilisateur à explorer plus en détail le voisinage d'une ressource choisie d'une façon encore plus rapide et plus efficace sans besoin de formuler des requêtes manuellement.

Prenons l'exemple de la figure 3.3 qui prend en entrée l'extrait de la source de données présentée par la figure 2.1. Initialement, les éléments de graphe contiennent l'ensemble des données RDF. Autrement dit, la liste des sujets correspond à List_All_Subject, la liste des prédicats correspond à List_All_Property et la liste des objets correspond à List_All_Object. Supposons que dans le premier motif de triplet, l'utilisateur déclare une variable ? MusicArtist dont le type est *MusicArtist*. Si l'utilisateur choisit la même variable au niveau du sujet du deuxième motif de triplet, alors les valeurs des prédicats et des objets se limitent aux propriétés et aux objets des instances dont le type est *MusicArtist*. C'est-à-dire, il y aura juste les propriétés suivantes : *foaf:name*, *foaf:made*, *mo:biography* et *rdf:type*. Pour les objets, il y aura les littéraux suivants : « *Margin of Safety* », « *Stian* », « *The worlds laziest artist!* » et la ressource *eg:record1* (Figure 3.3, A). Si nous choisissons le prédicat *foaf:name*, alors nous cherchons les valeurs possibles des objets associés à la propriété *foaf:name* pour les instances de type *MusicArtist*. Ainsi, nous aurions deux suggestions possibles pour l'objet et qui sont « *Margin of Safety* », « *Stian* » (Figure 3.3, B). Nous choisissons la valeur « *Margin of Safety* ».

(A)

Subject	Predicate	Object
☐ ?MusicArtist	http://www.w3.org/1999/0222-rdf-syntax-ns#type	http://purl.org/ontology/moll/MusicArtist
☐ ?MusicArtist	http://purl.org/ontology/mobiography	Margin of Safety
	http://www.w3.org/1999/0222-rdf-syntax-ns#type	Slan
	http://xmlns.com/foaf/0.1/made	The worlds latest artist
	http://xmlns.com/foaf/0.1/name	http://dbtune.org/jamendorecord1
		http://purl.org/ontology/moll/MusicArtist

(B)

Subject	Predicate	Object
☐ ?MusicArtist	http://www.w3.org/1999/0222-rdf-syntax-ns#type	http://purl.org/ontology/moll/MusicArtist
☐ ?MusicArtist	http://xmlns.com/foaf/0.1/name	Margin of Safety
		Slan

(C)

Subject	Predicate	Object
☐ ?MusicArtist	http://www.w3.org/1999/0222-rdf-syntax-ns#type	http://purl.org/ontology/moll/MusicArtist
☐ ?MusicArtist	http://xmlns.com/foaf/0.1/name	Margin of Safety
☐ ?MusicArtist	http://www.w3.org/1999/0222-rdf-syntax-ns#type	Margin of Safety
	http://xmlns.com/foaf/0.1/name	http://purl.org/ontology/moll/MusicArtist

Figure 3.3 Voisinage d'un élément du graphe RDF

Dans le troisième motif de triplet, nous choisissons encore la même variable ? *MusicArtist*. La liste des propriétés et des objets se restreint et se limite maintenant au voisinage des artistes dont le nom est « *Margin of Safety* ». Bien qu'au début nous avons tous les propriétés et les attributs de la classe *MusicArtist*, nous avons eu juste les propriétés *foaf:name* et *rdf:type* et les valeurs « *Margin of Safety* » et *MusicArtist* pour l'objet pour cette instance (Figure 3.3, C). Nous pourrions déduire que cette instance n'a pas les propriétés *foaf:made* et *mo:biography*.

Cette approche permet d'avoir une idée à propos du voisinage d'un élément du graphe RDF. Le fait que les propriétés soient optionnelles au sein du même groupement, alors cette approche permet d'explorer le voisinage de chaque instance d'un groupement donné en particulier.

3.4 Requête SPARQL

La formulation des requêtes SPARQL est une tâche complexe. En effet, l'utilisateur devrait connaître la syntaxe du SPARQL et nécessite des connaissances techniques et une certaine compréhension de RDF, RDFS et IRIs, entre autres. Ainsi, la formulation de requête SPARQL pourrait nécessiter beaucoup de temps.

Une solution simple consiste à permettre la construction automatique des requêtes SPARQL. La clause *where* de la requête sera construite au fur et à mesure la définition des motifs de triplets. Si l'utilisateur indique qu'un motif de triplet est optionnel alors cette information sera ajoutée pour le motif en question. La clause *select* sera construite en se basant sur la liste des variables définies par l'utilisateur. L'option « distinct » sera ajoutée si l'utilisateur ne veut pas avoir des doublons dans l'ensemble de solutions. Les clauses *LIMIT*, *OFFSET* et *ORDER BY* seront ajoutés également à la requête à la demande de l'utilisateur. À la suite de l'exemple de la figure 3.3, C, l'utilisateur peut

ainsi définir la limite, offset, et l'ordre. Par conséquent, la requête SPARQL sera construite en arrière-plan (Figure 3.4).

The image shows a web-based SPARQL query interface. At the top, there are controls for the query execution: 'ORDER BY' is set to '?MusicArtist', with radio buttons for 'ASC' (selected) and 'DESC'. To the right, 'LIMIT' is set to '2' and 'OFFSET' is set to '1'. Below these controls is a 'Query' text area containing the following SPARQL query:

```
SELECT ?MusicArtist
WHERE {
  ?MusicArtist <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://purl.org/ontology/mo/MusicArtist>.
  ?MusicArtist <http://xmlns.com/foaf/0.1/name> "Margin of Safety".
}
ORDER BY ?MusicArtist LIMIT 2 OFFSET 1
```

Figure 3.4 Requête SPARQL

Cette approche offre une composition assistée des requêtes SPARQL au fur et à mesure l'exploration des données. En effet, l'utilisateur n'a pas besoin d'avoir de connaissances approfondies de la syntaxe SPARQL. Ainsi, il n'aura plus de soucis à créer des requêtes et il pourra juste explorer les données RDF et définir des motifs de triplets qu'il voudrait les avoir dans sa requête. Bien évidemment, cela n'empêche pas que l'utilisateur peut créer et mettre à jour des requêtes SPARQL manuellement.

3.5 Conclusion

Dans ce chapitre, nous avons présenté notre approche d'interrogation des données RDF. Cette approche permet à l'utilisateur d'explorer les différents éléments d'un graphe RDF à travers notre fonctionnalité du chargement initial du jeu de données (section 3.2). L'utilisateur peut ainsi faire des recherches ciblées de n'importe quel élément de graphe RDF. Nous avons offert la suggestion de complétion (section 3.3.1) qui permet de fournir une liste de suggestion selon des mots-clés saisis par l'utilisateur. Ce type de recherche est plus rapide que parcourir toute une arborescence tel que suggérée par l'article (Tim Berners-Lee & Sheets, 2006). Nous avons proposé également la possibilité de limiter la zone de recherche en permettant l'exploration de voisinage d'un élément du graphe RDF (section 3.3.2). L'exploration des données est accompagnée

par la composition automatique des requêtes SPARQL. Par conséquent, l'utilisateur n'aura plus de soucis à créer des requêtes, mais ça n'empêche pas l'utilisateur à mettre à jour la requête manuellement (section 3.4).

CHAPITRE IV

TEST ET ÉVALUATION

4.1 Introduction

Dans les deux chapitres précédents, nous avons proposé deux approches complémentaires : une approche de construction du schéma RDF et une approche d'interrogation des données RDF. Ces deux approches permettent une composition assistée des requêtes sur des données RDF. Dans ce chapitre, nous allons évaluer et montrer la pertinence de ces deux approches proposées.

Nous commençons par présenter, dans la section 4.2, le prototype QUERY-RDF que nous avons développé et qui intègre les deux stratégies. Nous présenterons les outils utilisés pour le développement dans la section 4.3. Par la suite, nous présenterons l'étude expérimentale que nous avons menée pour valider les deux approches. Comme les jeux de données et les méthodologies d'évaluation sont différents pour les deux approches, nous allons présenter les expérimentations séparément. La section 4.4 présente les expérimentations relatives à la construction du schéma RDF. Nous avons considéré deux types d'évaluations : une évaluation de notre résultat par rapport au travail mené par (Li & Zhang, 2013) et une évaluation sur des données synthétiques. Dans ce cas, nous avons évalué le résultat en ajoutant des instances sans type afin de voir l'impact et l'influence de ce dernier sur la qualité de résultat. Dans la sous-section

4.5, nous évaluons notre approche d'interrogation des données. Nous avons évalué la complémentarité de deux approches dans la section 4.6 à travers des exemples et des hypothèses. Nous concluons ce chapitre dans la section 4.7.

4.2 Prototype QUERY-RDF

Dans cette section, nous présentons l'outil QUERY-RDF que nous avons développé dans le but de valider et d'évaluer l'approche proposée dans notre mémoire. L'architecture générale du prototype QUERY-RDF est schématisée par la figure 4.1.

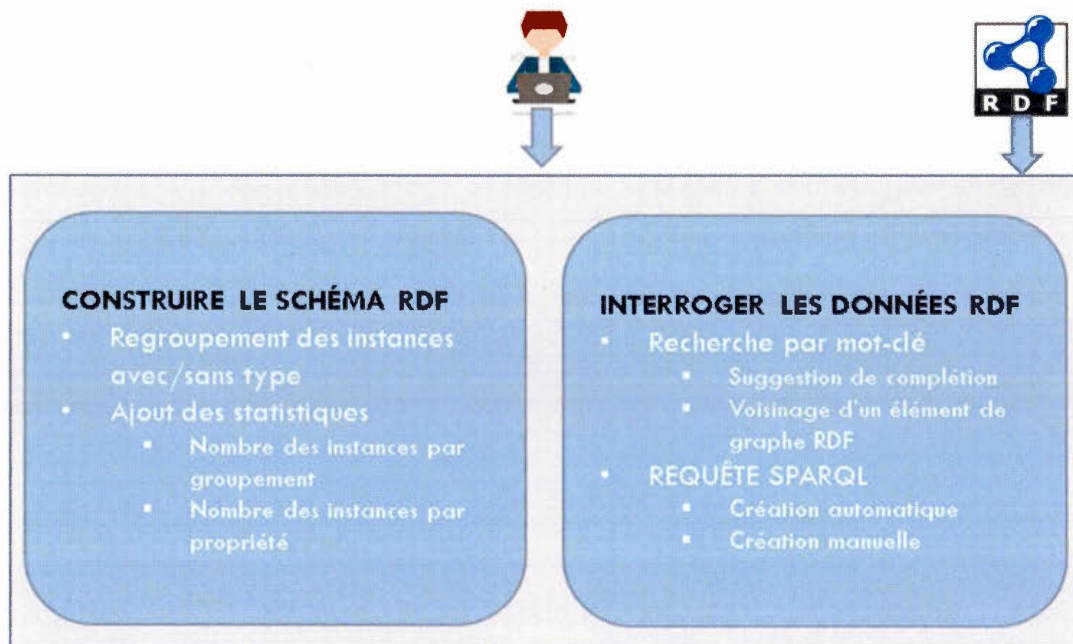


Figure 4.1 Prototype QUERY-RDF

Les modules principales de l'outil sont :

- Un chargement des données RDF
- Une implémentation du schéma de données RDF proposé dans le chapitre 2 ;

- Une implémentation du Framework d'interrogation des données RDF proposé dans le chapitre 3.

4.2.1 Charger les données RDF

Ce module identifie les triplets d'une source de données RDF. Il prend en entrée une source de données RDF choisie par l'utilisateur et génère des triplets qui seront sauvegardés pour une utilisation ultérieure par les modules qui seront détaillés dans les sections suivantes (4.2.2 et 4.2.3).

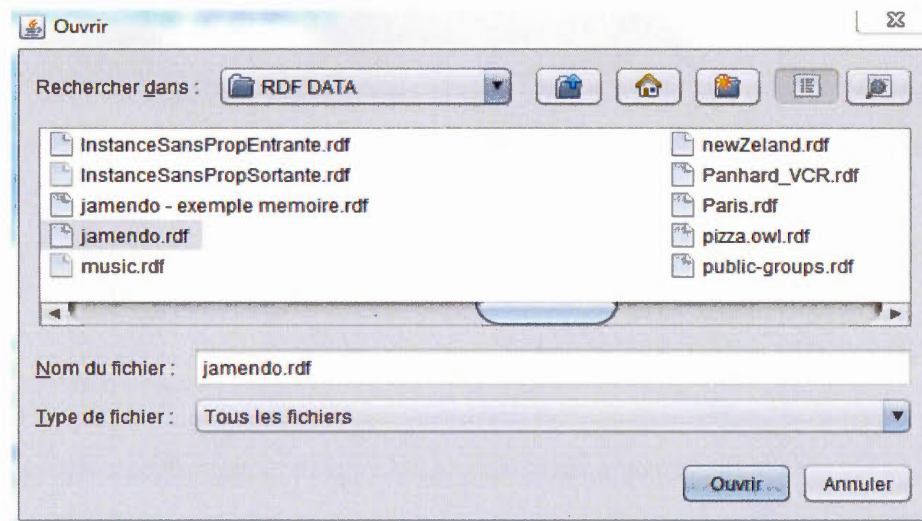


Figure 4.2 QUERY-RDF : Choix d'un fichier RDF

4.2.2 Visualiser le schéma RDF

Ce module prend en entrée les données RDF fournies par notre prototype QUERY-RDF pour générer un diagramme de classes qui représente une approximation du schéma de données RDF. Cette approximation ne vise pas à reproduire le schéma réel, mais plutôt à donner une idée de la structure des données RDF à partir des instances avec et sans type défini et qui sera à travers des groupements de ressources.

Pour les instances définies, les groupements d'instances sont identifiés à la base de la propriété *rdf:type*. Chaque groupement correspond à une classe. Deux instances appartiennent au même groupement s'elles ont le même type. Pour les instances non définies, l'approche consiste à regrouper ces instances en se basant sur les propriétés communes. En effet, nous avons défini la relation suivante : une instance *i* appartient à un groupement si et seulement s'il existe au moins une autre instance *j* appartenant au même groupement avec $\text{prop}(i) \cap \text{prop}(j) \neq \emptyset$ (voir le chapitre 2).

Dans le diagramme de classes, les classes représentent les groupements des instances déterminés. Les attributs et les associations de chaque classe ont été déterminés à partir des propriétés de chaque groupement. Des statistiques ont été ajoutées également au diagramme de classes tel que le nombre d'instances par classe et le nombre d'instances par propriété. La figure 4.3 montre une capture d'écran du diagramme de classes affiché par notre prototype QUERY-RDF.

4.2.3 Interroger les données RDF

Ce composant permet d'explorer les différents éléments de graphe RDF. La figure 4.4 montre une capture d'écran de différentes fonctionnalités offertes par ce composant. L'utilisateur peut construire des motifs de triplets afin d'explorer un élément de graphe en particulier. Ainsi, il peut faire des recherches, explorer et afficher le voisinage d'une ressource de son choix (Figure 4.4, B). La suggestion de complétion est offerte au niveau de chaque élément de motif de triplet. Les motifs de triplets peuvent être activés ou désactivés pour permettre l'exploration d'un voisinage dans le graphe RDF.

À chaque définition d'un motif de triplet, une requête SPARQL sera mise à jour. La clause WHERE de la requête sera construite au fur et à mesure la définition des motifs de triplets. Si l'utilisateur indique qu'un motif de triplet est optionnel alors cette information sera ajoutée pour le motif en question.

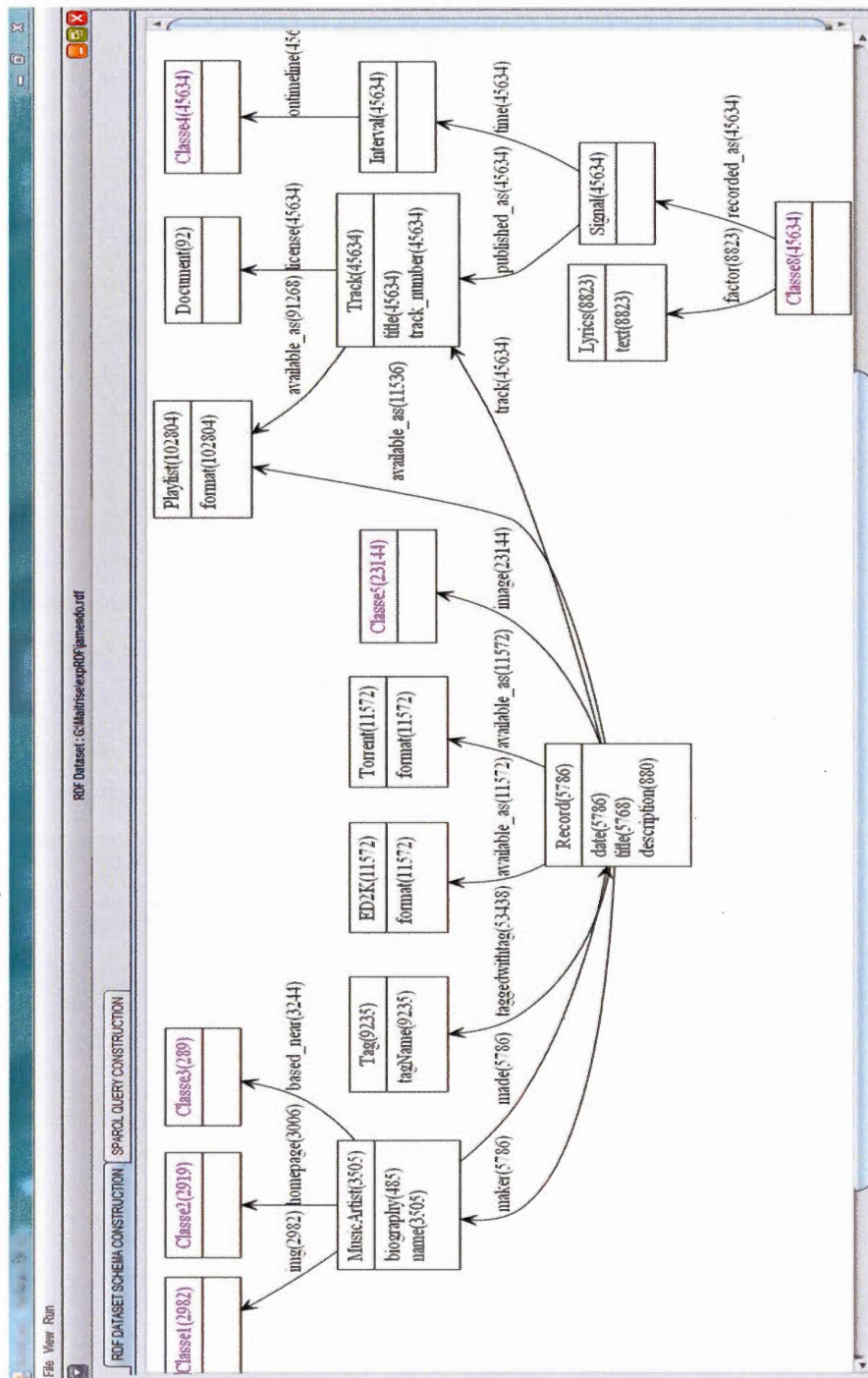


Figure 4.3 Schéma de données RDF

La clause *select* sera construite en se basant sur la liste des variables définies par l'utilisateur. L'option « distinct » peut être activée pour supprimer les doublons dans l'ensemble de solutions (Figure 4.4, A). Les clauses *LIMIT*, *OFFSET* et *ORDER BY* seront ajoutés également à la requête à la demande de l'utilisateur (Figure 4.4, C). L'utilisateur peut mettre à jour cette requête à l'aide du bouton *Update Query*. L'utilisateur peut aussi visualiser le résultat de la requête SPARQL (Figure 4.4, D) à l'aide du bouton *Run Query*. Le nombre d'enregistrements retourné est affiché.

4.3 Outils utilisés

Cette section présente les outils et les plateformes utilisés afin de mettre en œuvre notre approche.

4.3.1 Jena⁶

C'est un API Java open source. Jena a été construit en premier temps uniquement pour RDF, puis des nouvelles fonctionnalités ont été ajoutées pour l'utilisation du langage OWL (D.L & F, 2004). Jena a des classes d'objets pour représenter des graphes, des ressources et des littéraux. Dans Jena, un graphe est appelé un modèle et représenté par l'interface du modèle. Dans notre cas, nous avons utilisé Jena pour extraire les différents éléments du graphe RDF.

Le moteur de recherche ARQ du Jena permet l'intégration des données accessibles stockées dans un modèle en fournissant un langage de requêtes appelé SPARQL. Ce langage de requêtes standard est conçu pour être utilisé dans le modèle de type RDF.

⁶ <https://jena.apache.org/>

4.3.2 Java 6

C'est un langage de programmation orienté objet. Ce langage a été utilisé pour créer l'interface utilisateur qui permet d'explorer les données RDF ainsi le schéma RDF.

4.3.3 GRAPHVIZ⁷

Graphviz est un logiciel de visualisation graphique open source. Cet outil représente l'information structurelle sous forme de diagrammes abstraits et de réseaux. Il a d'importantes applications dans les réseaux, la bio-informatique, le génie logiciel, les bases de données et la conception de sites Web, l'apprentissage machine, et dans les interfaces visuelles pour d'autres domaines techniques. Grâce aux algorithmes utilisés par *Graphviz*, cet outil est capable de représenter des graphes denses avec un grand nombre de nœuds. Nous avons utilisé *Graphviz* afin de visualiser l'approximation du schéma RDF sous forme d'un diagramme de classes. Ce choix est basé sur la possibilité de personnaliser le rendu des graphes en choisissant les formes, les couleurs et les polices de caractères. De plus, *Graphviz* offre un format des fichiers d'entrée simple et des formats de sortie variés.

4.4 Évaluation du schéma RDF construit

4.4.1 Évaluation de la qualité des résultats

Dans notre approche, les regroupements de ressources peuvent être comparés aux classes du diagramme de classes présentées par l'article (Li & Zhang, 2013). L'article a présenté un diagramme de classes pour la source de données *Jamendo* tel que montré par la figure 4.5. Dans ce diagramme, il y a 18 classes dont 7 sont des classes anonymes

⁷ www.tonyballantyne.com/graphs.html

qui représentent les groupements des instances non définies. Avec notre prototype QUERY-RDF, nous avons identifié 17 classes (Figure 4.6). 16 classes correspondent parfaitement aux classes obtenues dans l'article : nous avons eu les mêmes classes, les mêmes attributs, les mêmes associations et les mêmes nombres d'instances au niveau des classes, au niveau des associations et au niveau des attributs. La seule différence était au niveau du groupement des instances non définies qui ont un lien avec des instances de type *Lyrics* et des instances de type *Signal* par les propriétés *factor* et *recorder_as* respectivement. L'écart entre les deux diagrammes est identifié en couleur rouge dans les figures 4.5 et 4.6. De notre part, nous avons présenté cet ensemble d'instances à travers juste un groupement représenté par *class0* en association avec les deux classes *Lyrics* et *Signal*. Cette présentation peut être interprétée comme suit : il y a 45634 instances sans type qui ont toutes la propriété *recorded_as* vu que le nombre d'instances au niveau de la classe est égal au nombre des instances au niveau de la propriété *recorded_as* (45634). Parmi ces 45634 instances, il y a 8823 instances qui ont aussi la propriété *factor*. Dans l'article, cet ensemble des instances est présenté par deux classes anonymes : un est associé à la classe *Lyrics* par la propriété *factor* et l'autre classe est associée à la classe *Signal* par la propriété *recorded_as*. Contrairement à notre représentation, la figure 4.5 pourrait être interprétée comme suit : Les instances sans type peuvent avoir soit la propriété *recorded_as* ou la propriété *factor*, mais aucune instance qui pourrait avoir les deux propriétés ensemble.

Afin de déterminer lequel des deux diagrammes de classes représente mieux les données, il y avait trois moyens de vérification soit à travers notre approche d'exploration des données, soit en fouillant directement dans la source de données pour un utilisateur qui ne maîtrise pas SPARQL ou bien en construisant des requêtes pour un utilisateur expert. Le premier et le troisième moyen nous les présentons dans la section 4.6.1 et qui permettent de prouver la complémentarité de nos deux approches d'exploration d'un jeu de données RDF.

Figure 4.5 Visualiser les données de Jamendo avec le diagramme UML (Li & Zhang, 2013)

Figure 4.6 Visualiser les données de Jamendo avec QUERY-RDF

En explorant directement la source de données Jamendo (Raimond, 2016) (Figure 4.7) pour chercher la propriété *recorded_as* et *factor*, nous avons effectivement trouvé qu'il y a des instances qui ont les deux propriétés (Ligne 475531-475534) et des instances qui ont juste la propriété *recorded_as* (Ligne 475536-475538). Toutes les instances qui ont la propriété *factor* ont la propriété *recorded_as*. De telles affirmations sortent bien explicites à travers notre diagramme de classes, mais pas dans celui de (Li & Zhang, 2013). Cette méthode de vérification nécessite beaucoup de temps. Cependant, nous pourrions affirmer que notre prototype permet de bien résumer la source de données même pour les instances sans type contrairement à l'approche de (Li & Zhang, 2013).

```

475531 <rdf:Description rdf:about="http://dbtune.org/jamendo/performance/100252">
475532   <event:factor rdf:resource="http://dbtune.org/jamendo/lyrics/100252"/>
475533   <mo:recorded_as rdf:resource="http://dbtune.org/jamendo/signal/100252"/>
475534 </rdf:Description>
475535
475536 <rdf:Description rdf:about="http://dbtune.org/jamendo/performance/100253">
475537   <mo:recorded_as rdf:resource="http://dbtune.org/jamendo/signal/100253"/>
475538 </rdf:Description>

```

Figure 4.7 Extrait du jeu de données Jamendo (Raimond, 2016)

4.4.2 Évaluation sur des données RDF synthétiques

Pour montrer encore plus la pertinence de notre approche, nous sommes allés plus loin et nous avons créé des données dont nous savons exactement le contenu de la source et la structure des données ce qui nous permet de mieux évaluer et de valider notre prototype. En effet, nous avons procédé comme suit :

Au début, nous avons créé une base de données relationnelle *Film_actors*. Elle contient 4 classes : les films, les acteurs, le langage et la catégorie de film. Nous avons inséré des données fictives dans chaque table. Par la suite, nous avons généré le diagramme

de classes. Ce dernier est généré par MySQL Workbench 5.2 CE grâce à la méthode *Reverse engineer*⁸ appliqué à la base de données *Films_actors*.

Ensuite, nous avons généré une source de données RDF avec l'outil *D2RQ* (Tsa, 2012) à partir de la base de données relationnelle en appliquant la mise en correspondance recommandée par *W3C*. *D2RQ* est un système d'accès aux bases de données relationnelles en tant que graphes RDF virtuels en lecture seule. Il supporte le *W3C Direct Mapping* et offre un accès basé sur RDF au contenu des bases de données relationnelles sans avoir à le répliquer dans un store RDF. Nous avons utilisé la commande suivante pour convertir le contenu de la base de données relationnelle entière dans un fichier RDF.(Films.rdf) :

dump-rdf -u root -f RDF/XML -o Films.rdf jdbcURL jdbc:mysql://localhost:3307/Films_actors où les options de la commande jouent les rôles suivants :

- **-u** utilisateur : le nom de l'utilisateur pour la connexion à la base de données ;
- **-f** format : La syntaxe RDF à utiliser pour la sortie. Les syntaxes prises en charge sont « TURTLE », « RDF / XML », « RDF / XML-ABBREV », « N3 » et « N-TRIPLE » (valeur par défaut). « N-TRIPLE » fonctionne mieux pour les grandes bases de données ;
- **-o** outfile : Nom du fichier de destination. Par défaut à la sortie standard.
- **jdbcURL** : URL de connexion JDBC pour la base de données.

Afin d'évaluer la qualité de résultat, dans un premier lieu, nous avons créé un fichier RDF dont toutes les instances sont définies. Dans un second lieu, nous avons inséré des

⁸ <https://dev.mysql.com/doc/workbench/en/wb-reverse-engineer-create-script.html>

instances sans type et nous avons observé l'impact sur le diagramme de classes. Ces deux cas sont présentés dans les sections suivantes.

4.4.2.1 Fichier RDF avec instances définies

Les figures 4.8 et 4.9 montrent qu'il y a des différences négligeables entre les deux diagrammes, par exemple dans le diagramme de classes généré par QUERY-RDF de la figure 4.9 : les noms des attributs de chaque classe sont précédés par le nom de la classe, le lien entre les différentes classes est composé des clés primaires des classes en relation. Nous pourrions l'expliquer par le fait que l'outil D2RQ génère le document RDF de cette façon selon les règles de mise en correspondance définie par W3C.

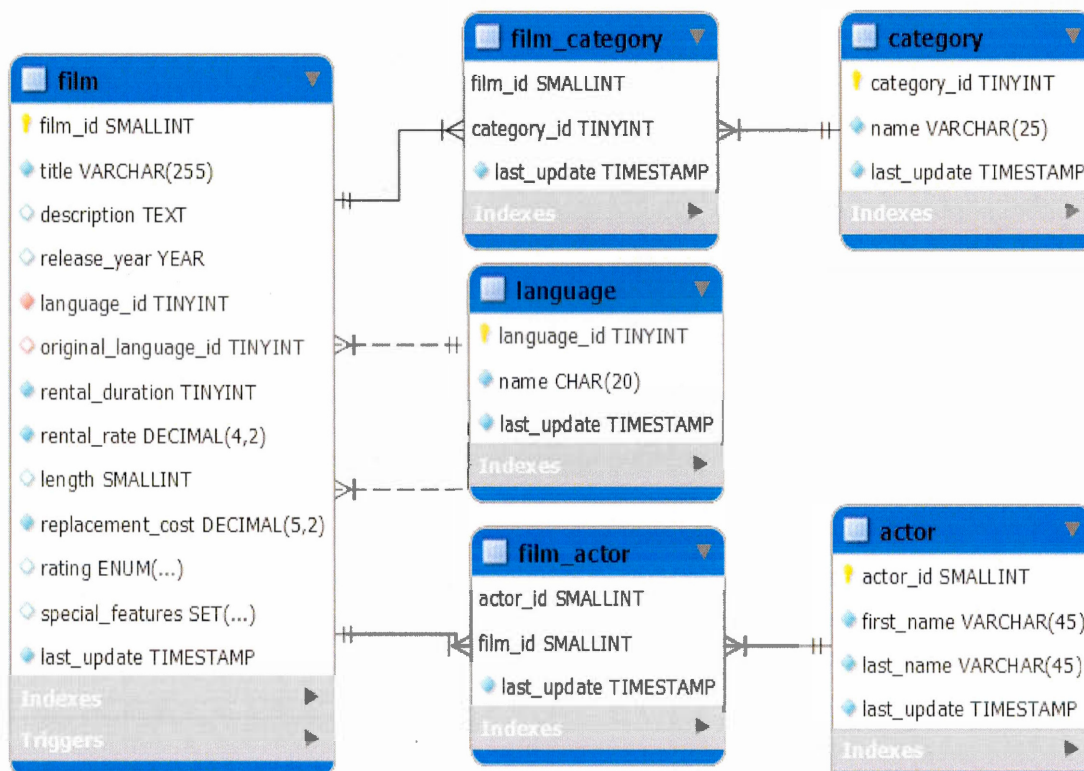


Figure 4.8 Le diagramme physique des données relationnelles généré par la méthode reverse engineer de MySQL Workbench

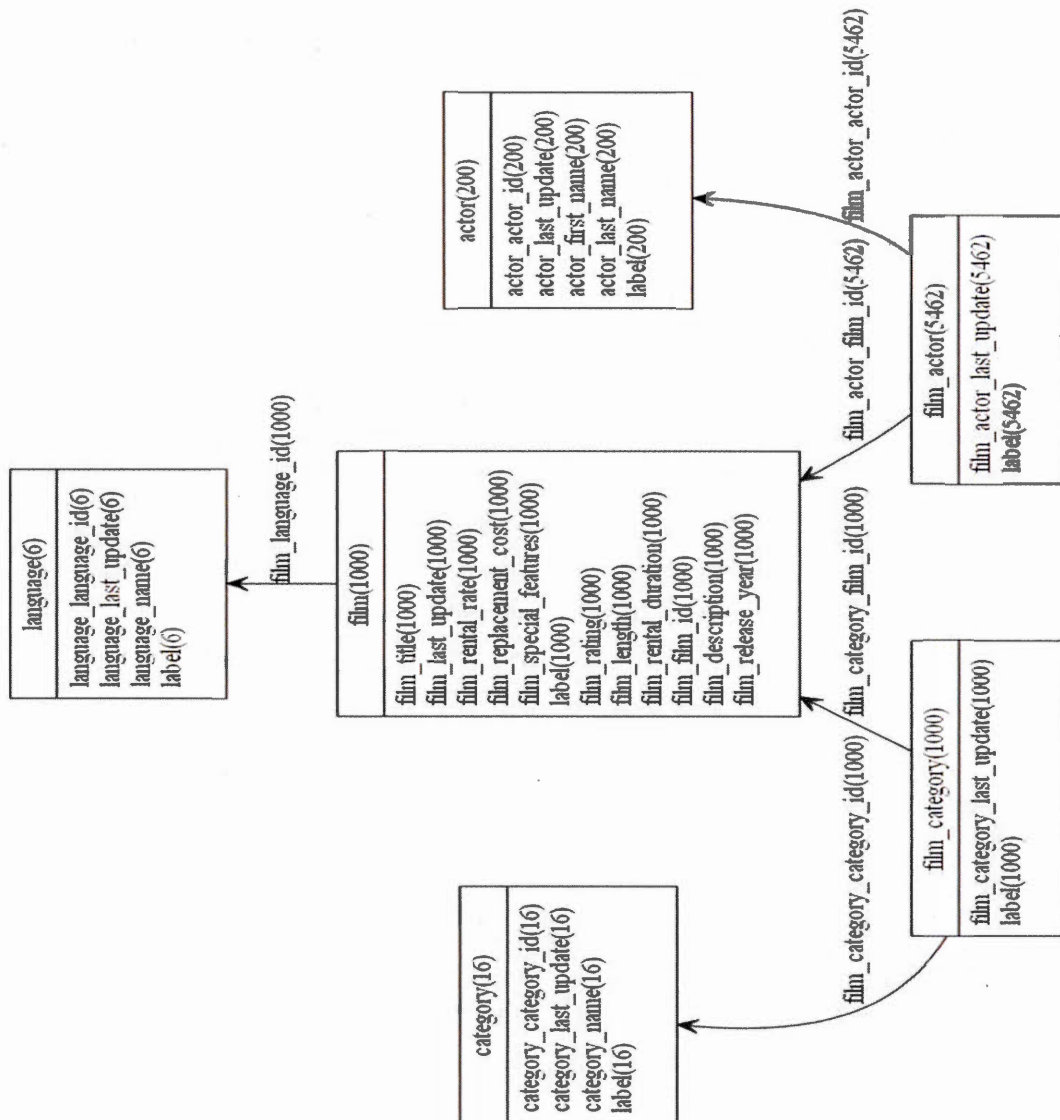


Figure 4.9 Diagramme de classes à partir de la source de données Film.rdf

Cependant, la structure générale est identique et les deux diagrammes contiennent les mêmes classes, les mêmes attributs et les mêmes associations ce qui montre la pertinence et l'exactitude de notre approche dans ce cas simple.

4.4.2.2 Fichier RDF avec des instances non définies

Dans ce cas, nous avons choisi de supprimer le type des instances Film et FilmActor afin d'évaluer l'impact sur notre diagramme de classe.

Pour ce faire, nous avons utilisé une requête SPARQL de type DELETE. Par exemple, pour les instances de type Film, la requête est comme suit :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX vocab : <file:///D:/Utilisateurs/gj091167/Downloads/d2rq-0.8.1/vocab/>
DELETE
{?ressource rdf:type vocab : film}
```

Nous avons lancé le module de la construction du schéma RDF et nous avons obtenu le diagramme présenté par la figure 4.10. Avec notre approche nous devrions nous assurer que même si les instances sont non définies, les classes anonymes devraient continuer à avoir les propriétés des instances sans type que ce soit sous forme d'association ou sous forme d'attribut.

Cette condition a été prouvée vu que nous avons obtenu deux classes anonymes : une classe avait les mêmes attributs et les mêmes associations que la classe Film, l'autre classe avait les mêmes attributs et les mêmes associations que la classe Actor. Par la suite, ces deux classes représentent bien les instances sans type. La seule différence est que ces classes non définies *classe1* et *classe2* sont anonymes.

Par conséquent, dans les meilleurs cas nous obtenons un diagramme qui représente parfaitement le graphe RDF initial dans le cas où les instances sont toutes définies et contiennent les mêmes propriétés par type. Dans les pires des cas, nous obtenons un diagramme approximatif maximal qui représente parfaitement les instances sans type et les instances qui n'ont pas toutes les propriétés d'un type donné et par la suite, le

graphe RDF initial est inclus ce qui permet de valider la pertinence de notre approche vue que nous ne perdons aucune information à propos du graphe RDF initial.

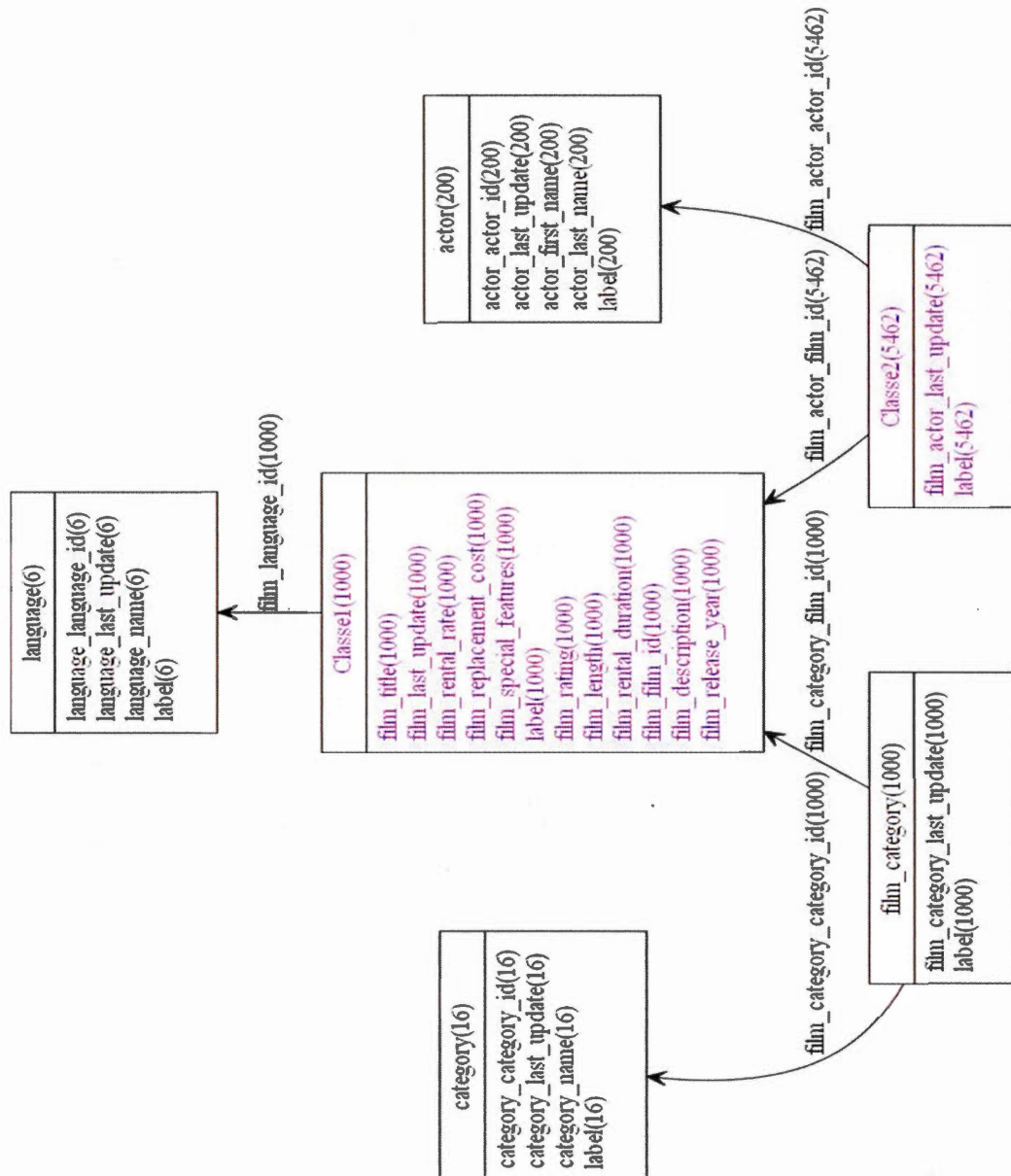


Figure 4.10 Film.RDF avec des instances sans type : diagramme de classes

4.5 Évaluation de l'approche d'interrogation des données RDF

Nous n'avons pas, à l'heure actuelle, évalué l'outil par rapport à des groupes d'utilisateurs. Nous pensons que l'outil pourrait être utile aux utilisateurs en train d'acquérir l'expertise de SPARQL, mais nous n'avons pas évalué cette affirmation dans un contexte d'études empiriques contrôlées.

Cependant et afin d'évaluer notre approche d'interrogation des données, nous allons essayer de faire des interrogations manuellement et de les comparer par rapport à notre approche mise en place. Le résultat de la comparaison est indiqué dans le tableau 4.1 et qui montre que la durée pour écrire n'importe quelle requête via notre outil restera toujours 50% inférieur par rapport à l'écriture manuelle. En effet, la comparaison a permis de montrer l'efficacité de l'option Voisinage d'un élément du graphe. Cette option permet à l'utilisateur d'explorer les différents éléments d'un graphe RDF sous forme des éléments de motif de triplet. L'utilisateur peut ainsi faire des recherches ciblées de n'importe quel élément du graphe RDF ce qui reste toujours rapide et efficace qu'écrire des requêtes manuellement.

La suggestion de complétion qui permet de fournir une liste de suggestion selon des mots-clés saisis par l'utilisateur permet de limiter la zone de recherche, ce qui reste plus rapide qu'écrire des requêtes manuellement. À noter que pour faire des recherches manuelles, il faut avoir un minimum de connaissance à propos des expressions régulières dans une requête SPARQL, une tel connaissance qui n'est pas forcément acquise par les utilisateurs. Ainsi, en utilisant l'interface de notre outil, l'utilisateur n'aura plus le besoin d'avoir des connaissances approfondies de SPARQL. Des options de SPARQL sont offertes telles que : le filtre, LIMIT, OFFSET et ORDER BY. Par conséquent, l'utilisateur n'aura plus de soucis à créer des requêtes et n'a plus besoin de

passer du temps à écrire des requêtes SPARQL, mais ça n'empêche pas l'utilisateur à mettre à jour la requête manuellement.

Tableau 4.1 Évaluation de l'approche d'interrogation des données RDF

	Approche manuelle	Approche d'interrogation
Explorer le contenu de la source de données RDF		
<i>Méthode</i>	Écrire les requêtes suivantes manuellement : Req1: <i>select distinct ? object</i> <i>where {?subject ?prop ?object.}</i> Req2: <i>select distinct ? prop</i> <i>where {?subject ?prop ?object.}</i> Req3: <i>select distinct ?objet</i> <i>where {?subject ?prop ?object}</i>	Notre approche permet le chargement initial des différents éléments de la source de données. Ainsi, il suffit juste d'explorer le contenu de chaque élément de motif de triplet de notre prototype Query-RDF
<i>Durée (s)</i>	90	3
Déterminer les différentes classes de la source de données RDF		
<i>Méthode</i>	Écrire et exécuter la requête suivante : <i>Select distinct ?objet</i> <i>where {?subject rdf:type ?object}</i>	Grâce à l'option du voisinage offerte par notre approche d'interrogation des données, nous pourrions choisir la propriété <i>rdf:type</i> dans la colonne <i>property</i> . Ainsi, il suffit juste d'explorer le contenu de de la colonne <i>objet</i> pour voir la liste des classes
<i>Durée (s)</i>	40	10
Déterminer les différentes propriétés de la classe Artist		

	Approche manuelle	Approche d'interrogation
Méthode	Écrire et exécuter la requête suivante : <pre>Select distinct ?prop where { ?Artist rdf:type mo:MusicArtist. ?Artist ?prop ?obj.}</pre>	Dans le premier triplet, saisir un artiste ?Artist de type mo:MusicArtist. Dans le second triplet, explorer directement le contenu de la colonne property pour voir les différentes propriétés du même artiste ?Artist.
Durée (s)	44	20
Chercher les artistes dont le nom commence par Jean		
Méthode	Écrire et exécuter la requête suivante : <pre>Select distinct ?Artist ?nom where { ?Artist rdf:type mo:MusicArtist. ?Artist foaf:name ?nom. FILTER REGEX (?nom, "^Jean", "i").}</pre>	Grâce à la suggestion de completion, nous pourrions juste saisir Jean dans la colonne object. Ainsi, nous pourrions voir toutes les noms qui commencent par Jean
Durée (s)	55	25
Afficher le résultat de la requête précédente dans l'ordre décroissant selon le nom et limiter le nombre de résultat à 5		
Méthode	Écrire et exécuter la requête suivante : <pre>Select distinct ?Artist ?nom where { ?Artist rdf:type mo:MusicArtist. ?Artist foaf:name ?nom. FILTER REGEX (?nom, "^Jean", "i").}</pre>	Grace aux options offertes. IL suffit juste de spécifier la colonne à mettre dans l'ordre, choisir l'ordre décroissant et spécifier la limit à 5

	Approche manuelle	Approche d'interrogation
	<i>ORDER BY ?nom DESC LIMIT 5</i>	
Durée (s)	70	30

4.6 Évaluation de la complémentarité de l'interrogation des données RDF et de l'exploration du schéma RDF

4.6.1 Modélisation du schéma : Évaluer le besoin d'interroger les données RDF

Principalement, l'objectif final d'un utilisateur consiste à formuler des requêtes sur les données RDF. Supposons qu'un outil permet simplement la visualisation du schéma RDF et qu'il ne permet pas l'interrogation des données RDF. Dans ce cas, l'utilisateur sera obligé à créer manuellement des requêtes, une tâche fastidieuse qui nécessite plus de temps et qui nécessite de l'expertise. Notre prototype permet la modélisation du schéma afin de comprendre la structure de données RDF et l'interrogation des données en permettant l'écriture assistée des requêtes SPARQL.

Comme discuté dans la section 3.4.2, l'interrogation de données aide même à vérifier la structure de données RDF et à valider le diagramme de classes vu qu'il est juste une approximation du schéma RDF. Le diagramme offre une vue approximative, cependant l'utilisateur a besoin d'explorer les données en détail par rapport aux attributs et aux associations optionnels d'une classe. Les statistiques que nous avons mises afin d'enrichir le diagramme de classes permettent de savoir les attributs et les associations optionnels. Cependant, l'exploration des données RDF aide à mieux déterminer exactement le profil des différentes instances. De l'autre côté, la représentation des instances sans type est tellement approximative que la meilleure façon pour déterminer les propriétés de chaque instance est l'exploration des données RDF tel que l'exemple de l'artiste « *Margin of Safety* » présenté dans la section 3.4.2.

Nous reprenons l'exemple des instances sans type de la source de données Jamendo pour vérifier si les instances sans type qui ont la propriété *recorded_as* pourraient avoir aussi la propriété *factor*. À partir de l'interface INTERROGATION DES DONNÉES RDF, nous avons créé trois motifs de triplet (Figure 4.11). Dans le premier triplet, nous avons déclaré une variable *instanceSansType* qui représente les instances sans type et qui sont associées à une variable *signal* par la propriété *recorded_as*. Dans le deuxième triplet nous avons imposé le type *Signal* pour la variable *signal*. L'idée ici est de chercher l'ensemble des propriétés possibles pour les instances sans type qui sont associés à la classe *Signal* par la propriété *recorded_as*.

Dans le troisième motif de triplet, il suffit d'explorer la liste des propriétés possibles de la variable *instanceSansType*. Nous remarquons que nous pourrions avoir les deux propriétés : *recorder_as* et *factor* (Figure 4.11). De telles affirmations sortent bien explicites à travers notre diagramme de classes, mais pas dans celui de (Li & Zhang, 2013). Cette validation n'a duré que 30 secondes grâce à l'approche d'interrogation des données RDF qui permet entre autre l'exploration des données RDF.

4.6.2 Interrogation des données RDF : Évaluer le besoin d'un schéma RDF

L'objectif principal d'un utilisateur est d'interroger les données RDF. Cependant, il a besoin d'avoir un guide qui lui permet de bien s'orienter et de cibler des ressources au lieu d'aller par ici et par là sans savoir exactement où et par quoi commencer.

Le schéma permet de représenter le contenu d'une source de données RDF en ce qui concerne les types et les liens. Le schéma est peut-être considéré comme une carte qui montre les différents chemins ainsi, il peut guider l'utilisateur à bien choisir le chemin à traverser et à être bien orienté dans le graphe RDF. Par exemple, pour interroger la source de données Films, il est utile de connaître les types existants et leurs propriétés.

Subject	Predicate	Object
?InstanceSansType	http://purl.org/ontology/imo/recorded_as	?Signal
?Signal	http://www.w3.org/1999/0222-rdf-syntax-ns#type	http://purl.org/ontology/imo/Signal
?InstanceSansType	http://purl.org/NET/ict4dm/levent.owl#actor	
	http://purl.org/ontology/imo/recorded_as	

Figure 4.11 Explorer les instances sans type de Jamendo

Supposons que nous voulons extraire les films de type *Drama* dont la langue originale est le français. La requête est formulée à l'aide du formulaire et elle est comme suit :

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX vocab : <file:///D:/Utilisateurs/gj091167/Downloads/d2rq-0.8.1/vocab/>

1. *SELECT* ?language ?FilmCategory ?category ?Film ?LanguageName

2. *WHERE* {

3. ?FilmCategory rdf:type vocab:film_category.

4. ?FilmCategory vocab:film_category_category_id ?category.

5. ?category vocab:category_name "Drama".

6. ?FilmCategory vocab:film_category_film_id ?Film.

7. ?Film vocab:film_language_id ?language.

8. ?language vocab:language_name ? LanguageName.

9. }

La formulation de cette requête est impossible sans savoir : qu'un film pourrait être lié à un langage par la propriété *film_language_id* (Figure 4.12 et la ligne 7 de la requête SPARQL).

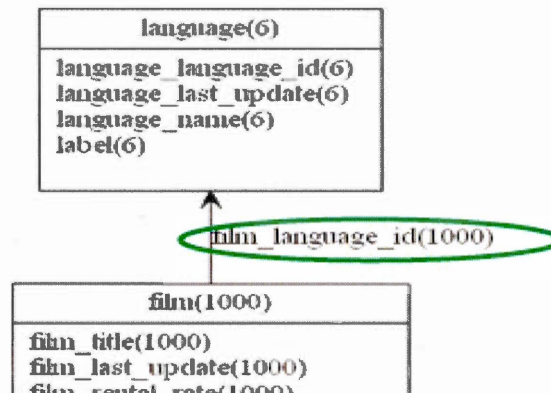


Figure 4.12 Association Film et Langage

Sans le schéma, nous n'aurions pu savoir que pour déterminer la catégorie du film il faut passer par la classe *Film_category*. Ainsi, le film est associé à la classe *Film_category* par la propriété *Filmcategory_film_id* (Figure 4.13, A et la ligne 6 de la

requête SPARQL) et que la classe *Film_category* est associée à la classe *Category* par la propriété *Filmcategory_category_id* (Figure 4.13, B et la ligne 4 de la requête SPARQL).

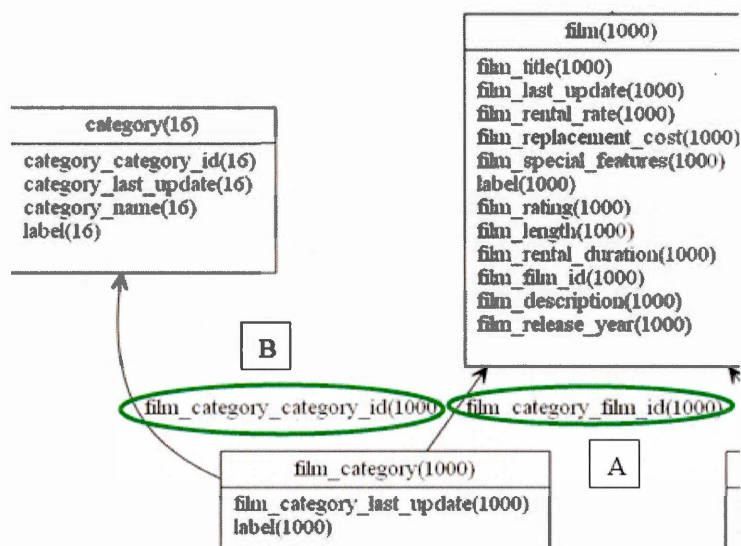


Figure 4.13 Association Film et Catégorie

Dans l'absence du schéma, l'utilisateur sera obligé dans un premier temps à explorer le schéma afin de comprendre la structure des données RDF. Pour motiver l'utilité de diagramme de classe, nous avons essayé de formuler des requêtes en présence et en absence du diagramme à l'aide de juste notre approche d'exploration des données.

4.6.2.1 Scénario A : Pas du schéma des données disponible

Dans ce cas, avant de formuler des requêtes et d'explorer la source de données pour des besoins spécifiques, l'utilisateur a besoin d'avoir une idée à propos du schéma de la source de données (types et liens entre types).

Dans l'absence d'un tel modèle, l'utilisateur sera obligé à se faire une idée du schéma manuellement. Une méthode simple consiste à explorer le schéma à travers l'approche d'exploration des données. Dans ce cas, les étapes seront comme suit :

1. Déterminer les types : avec la propriété `RDF:type`, nous pourrions déduire qu'il y a 6 types qui sont : `Actor`, `Category`, `Film`, `Film_actor`, `Film_category` et `langage`. (Figure 4.14, A) ;
2. Explorer le voisinage de chaque classe : un type présente une classe. Ainsi, pour chaque classe nous allons déterminer son voisinage : ses attributs ; ses relations avec les autres classes (associations). Prenons l'exemple de la classe, nous avons initialisé une variable ? `Film` de type `Film` afin de déterminer son voisinage. Selon la figure 4.14, B, les instances de type `Film` pourraient avoir les propriétés suivantes : *`film_film_id`, `film_title`, `film_length`, `film_description`, `film_rental_rate`, etc.* Les propriétés *`rdf:type`* et *`rdfs:label`* sont des propriétés prédéfinies dans RDF. Nous n'avons pas besoin de les explorer ;
3. Pour chaque propriété, nous allons déterminer si elle correspond à un attribut ou à une association avec une autre classe.
 - Si l'objet de la propriété en question est de type littéral alors cette propriété est considérée comme un attribut, tel que *`film_film_id`* de la figure 4.14, C ;
 - Sinon, si l'objet est une ressource alors nous allons déterminer son type s'il existe à l'aide de la propriété prédéfinie *`RDF:TYPE`* en utilisant une variable ? *`language`* qui représente cette ressource (Figure 4.14, D). Le type de cette variable est déterminé en créant un nouveau motif de triplet dont la propriété est *`rdf:type`*. Selon l'exemple de la figure 4.16, nous pourrions déduire que les ressources de type *`film`* sont associées aux ressources de type *`Language`* par la propriété *`film_language_id`*.
4. Construire le schéma des données RDF à partir du voisinage exploré : au fur et à mesure, en explorant les différentes classes, l'utilisateur pourrait créer le diagramme de classes afin d'avoir une vision globale des relations et des liens

entre les différents types des données RDF ce qui permet d'aider à créer des requêtes et explorer les données rapidement (Figure 4.15).

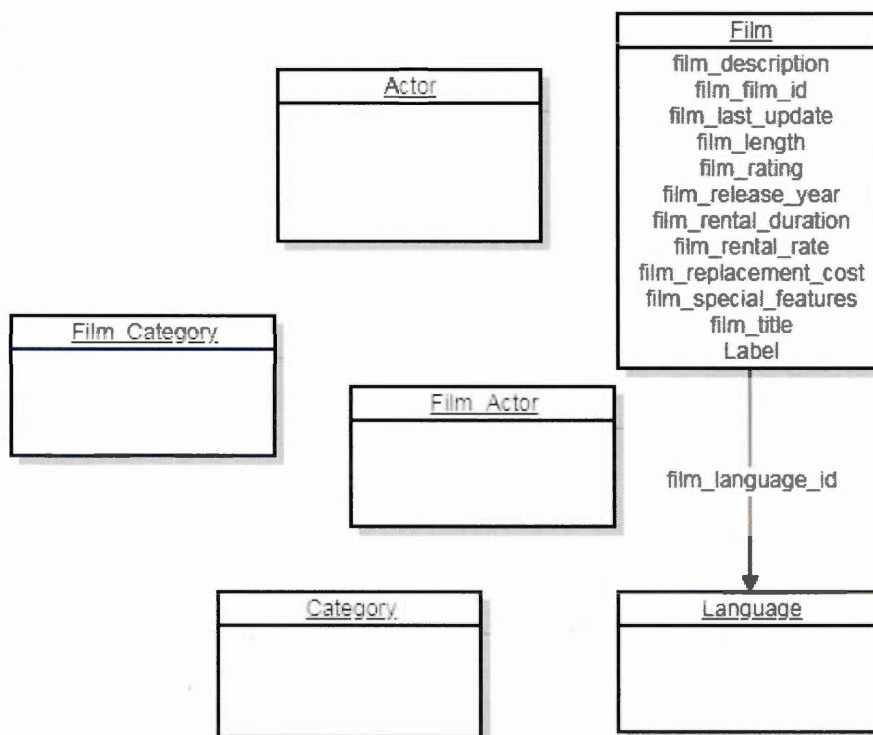


Figure 4.15 Interface Exploration des données : créer manuellement un diagramme de classes approximatif

Enfin, nous avons essayé d'explorer le voisinage de toutes les classes de la source de données Films.rdf. Ainsi, nous avons construit le diagramme de classes au fur et à mesure à l'aide de Gliffy⁹. L'exploration manuelle du schéma a pris environ 898 secondes (voir le tableau 4.2 pour plus de détail). Nous trouvons que la durée sera

⁹<https://chrome.google.com/webstore/detail/gliffy-diagrams/bhmicilclplefnflapjmnngmkkkpfad?hl=fr>

considérable surtout dans le cas où nous aurions plus de classes et plus de propriétés à déterminer.

Ainsi, faire le même exercice pour chaque nouvelle source de données RDF sera une tâche pénible pour l'utilisateur. C'est vrai que l'interface d'interrogation des données nous a aidés à déterminer la structure de schéma dans une durée qui sera toujours strictement inférieure à la durée que ça pourrait être passée dans le cas où la création des requêtes SPARQL est manuelle, cependant, faire le même exercice pour chaque source sera une tâche fastidieuse et qui nécessitera toujours du temps.

Tableau 4.2 Durée de la construction manuelle du diagramme de classes en seconde

Types de la source RDF	Durée d'exploration des propriétés (en s)	Durée de mettre à jour le diagramme de classes (en s)	Durée Total (en s)
Actor	30	40	70
Film	190	133	323
Film_Actor	120	87	207
Category	38	26	64
Film_category	72	87	159
Language	30	45	75

Types de la source RDF	Durée d'exploration des propriétés (en s)	Durée de mettre à jour le diagramme de classes (en s)	Durée Total (en s)
Durée Total	480	418	898

4.6.2.2 Scénario B : un schéma des données RDF disponible

La génération automatique de diagramme de classes grâce à notre approche de construction du schéma RDF pour la source de données Film a pris à peine trois secondes, une durée négligeable par rapport à la durée de l'approche manuelle détaillée dans la section 4.6.2.1 qui a nécessité 898 secondes.

Pour montrer encore plus la pertinence d'avoir un diagramme de classes pendant l'interrogation des données, nous allons présenter deux exemples de requêtes détaillées dans ce qui suit.

Exemple requête 1 : chercher les différentes catégories de Film.

L'utilisateur peut consulter le diagramme de classes afin de déterminer le chemin le plus court qui peut mieux répondre à notre requête. Selon la figure 4.16 qui montre une partie du diagramme de classes de la source RDF Film, les différentes catégories peuvent être extraites à partir de la propriété *category_name* de la classe *Category*. Ainsi, il ne reste plus qu'à explorer les différentes catégories de Film. Il y a deux méthodes permettant de déterminer les différentes catégories de Film soit :

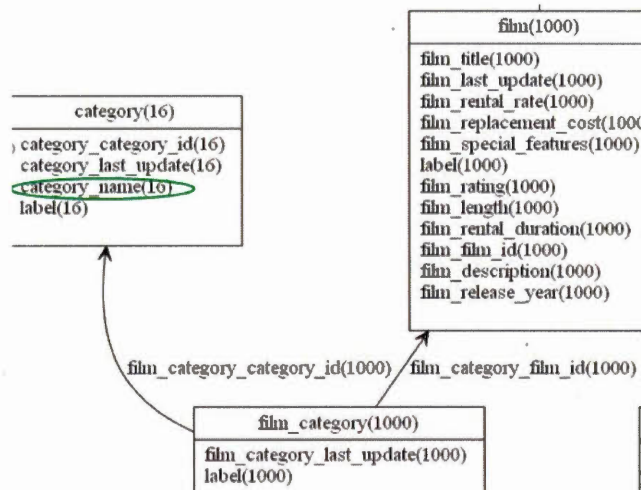


Figure 4.16 Extrait du diagramme de classes Film

- Explorer simplement la liste des objets : c'est une méthode d'exploration simple et rapide si l'utilisateur veut juste se renseigner et explorer à titre informatif (Figure 4.17). Dans ce cas, le processus de l'exploration au complet a pris à peine 20 secondes. C'est plus rapide que les approches de l'exploration traditionnelles vu que l'utilisateur connaît déjà le chemin à suivre grâce au diagramme de classes ;



Figure 4.17 Explorer Categorie Film

- Construire la requête : la composition de la requête assistée offerte par l'interface QUERY-RDF a pris à peine 30 secondes. Encore une fois, c'est plus rapide que les approches d'interrogation traditionnels.

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX vocab : <file:///D:/Utilisateurs/gj091167/Downloads/d2rq-0.8.1/vocab/>

SELECT distinct ?nomCategory

WHERE {

?category rdf:type vocab:category.

?category vocab:category_name ? nomCategory.

}

Exemple requête 2 : chercher les différents films Drama dont la langue d'origine est le français.

Selon le diagramme de classes de la figure 4.9, nous pourrions savoir qu'une instance de la classe *Film_Category* est liée à un film avec la propriété *Film_category_film_id*. La même classe *Film_Category* est liée à une catégorie par la propriété *Film_category_category_id*. Le film pourrait être lié à un langage par la propriété *film_language_id*. Vu que le chemin d'interrogation est identifié, les étapes de l'interrogation sont plus claires, faciles à les déterminer et pourront être comme suit :

1. Créer les variables : *?filmcategory* qui représente la classe *Film_Category*, *?Film* qui représente la classe *Film* et la variable *?Category* qui représente la

classe *Category*. Nous savons que ces trois classes sont liées grâce à l'extrait du diagramme de classes illustré par la figure 4.18.

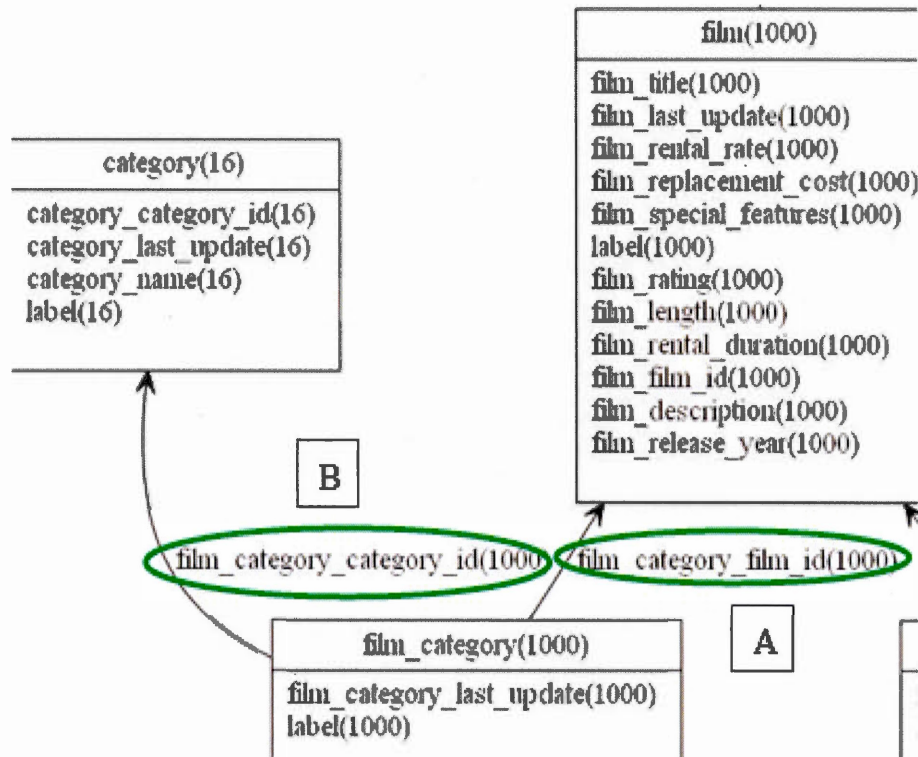


Figure 4.18 Association Film et Catégorie

2. Choisir la catégorie *Drama* qui sera à travers la propriété `category_name` de la classe *Category* selon la figure 4.19.

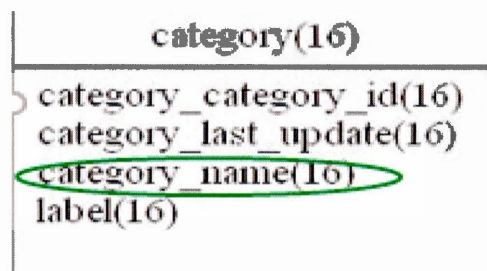


Figure 4.19 Classe Category

3. Déterminer la langue : à partir du diagramme de classes, la langue de Film est l'attribut *language_name* de la classe *language* qui est associée à la classe *Film* par la propriété *film_language_id* (Figure 4.20).



Figure 4.20 Déterminer la langue de film

4. En explorant la liste des langues des films *Drama* à l'aide de l'interface d'interrogation des données, nous remarquons qu'il y a juste la langue anglaise qui est associée aux films *Drama* (Figure 4.21). Dans ce cas, nous n'avons pas besoin d'aller plus loin vu qu'il n'y a pas des films *Drama* dont la langue est le français, un avantage d'utiliser notre prototype d'interrogation des données vu que ça nous évite de continuer à explorer un chemin qui ne

nous mène nulle part contrairement aux autres approches d'exploration des données.

Subject	Predicate	Object
☐ ?FilmCategory	http://www.w3.org/1999/0222-rdf-syntax-ns#type	file:///ID:/Utilisateurs/gj091167/Downloads/d2rq-0.8.1/vocabfilm_category
☐ ?FilmCategory	/ID:/Utilisateurs/gj091167/Downloads/d2rq-0.8.1/vocabfilm_category_film_id	?Film
☐ ?FilmCategory	/Utilisateurs/gj091167/Downloads/d2rq-0.8.1/vocabfilm_category_category_id	?Category
☐ ?Category	file:///ID:/Utilisateurs/gj091167/Downloads/d2rq-0.8.1/vocabcategory_name	Drama
☐ ?Film	file:///ID:/Utilisateurs/gj091167/Downloads/d2rq-0.8.1/vocabfilm_language_id	?language
☐ ?language	file:///ID:/Utilisateurs/gj091167/Downloads/d2rq-0.8.1/vocablanguage_name	English

Figure 4.21 Explorer les catégories de Film de type Drama

Nous avons bien illustré qu'au moment de composer la requête il est pertinent de se baser sur le diagramme de classes. Il donne une idée sur les chemins possibles à explorer et permet d'éviter de perdre le temps à explorer des chemins qui ne mènent nulle part. Parfois, les requêtes ne retournent rien. Cependant, l'approche d'interrogation des données permet déjà de détecter si une requête retourne un résultat ou non dès le début ce qui permet d'essayer d'explorer d'autres pistes au lieu de perdre le temps à construire une requête qui ne retourne aucun résultat, ce qui montre la complémentarité de deux approches.

4.7 Conclusion

Dans ce chapitre, nous avons commencé par présenter l'outil QUERY-RDF que nous avons développé et qui permet d'interroger des données RDF. QUERY-RDF est composé de deux principaux composants. Le premier correspond au schéma de données RDF. Le deuxième correspond à l'interrogation de données à travers soit la recherche par des mots-clés, soit la composition des requêtes SPARQL. Les outils utilisés pour le développement de notre outil ont été présentés dans la section 4.3.

Ensuite, nous avons présenté les expérimentations réalisées pour la validation de notre approche de construction du schéma de données RDF. Nous avons effectué deux méthodes d'évaluations. La première consiste à comparer notre résultat avec le résultat de (Li & Zhang, 2013). La deuxième méthode consiste à évaluer la qualité des classes identifiées avec des données synthétiques créées par nous-mêmes. Le fait que nous savons exactement le contenu de la source et la structure des données, ça nous a permis à mieux évaluer et valider notre résultat.

L'évaluation de la qualité de notre schéma des données était en ajoutant des instances sans type dans la source de données RDF pour voir l'impact sur la structure de schéma

construit par notre prototype. Les résultats des évaluations ont montré que les classes identifiées par l'approche résument parfaitement les données RDF.

L'évaluation de l'interrogation des données était accomplie en comparant le temps de la construction des requêtes manuelles par rapport aux fonctionnalités offertes par l'approche d'interrogation des données et ce qui a montré que notre approche permet de gagner beaucoup plus de temps.

Enfin, l'évaluation de la complémentarité de deux approches était comme suit : dans un premier lieu, nous avons expliqué les conséquences d'avoir juste le schéma des données RDF sans avoir la possibilité d'interroger les données RDF. Dans un second lieu, nous avons mesuré la durée de l'interrogation des données dans l'absence et la présence du schéma des données RDF. Nous avons montré que la durée de l'interrogation s'améliore d'une manière considérable lorsque l'utilisateur connaît la structure des données dès le début. De plus, nous avons montré l'importance d'avoir un schéma pendant la composition d'une requête.

CONCLUSION

Le Web sémantique qui a connu un essor ces dernières années prend une place de plus en plus importante. Une quantité toujours plus importante de connaissances, organisées selon des standards adaptés au traitement par des machines, est rendue accessible, mais le problème d'interroger ces connaissances n'est toujours pas résolu.

Plusieurs chercheurs ont tenté d'aider les utilisateurs à explorer les sources de données liées et à définir des requêtes complexes permettant une visualisation sophistiquée des résultats. Comme indiqué dans (Weise & al, 2016), la majorité des outils de visualisation des données obligent l'utilisateur à explorer manuellement le jeu de données et ne sont pas en mesure de lui fournir un schéma synthétique qui représente la structure des données RDF.

À travers ce mémoire, nous avons apporté une contribution à la résolution de ce problème. Nous avons présenté une approche qui permet une composition assistée de requêtes sur des données RDF. La principale contribution de notre travail est de jumeler deux approches : le schéma de données pour avoir une idée à propos de la structure de la source des données et l'interrogation des données RDF en permettant la recherche par des mots-clés, l'exploration du voisinage d'un élément de graphe et la construction assistée de la requête SPARQL.

Dans le chapitre 2, nous avons proposé une approche pour construire le schéma RDF. Cette approche ne vise pas à reproduire le schéma réel, mais elle vise plutôt à construire un schéma approximatif dont l'objectif est d'assister à la formulation des requêtes. Les

classes du schéma approximatif seront composées de groupement d'instances avec et sans type défini. L'identification des groupements est la découverte des groupements de ressources, non nécessairement disjoints, chacun représentant un ensemble de ressources ayant au moins une propriété en commun. L'ensemble des propriétés des ressources d'un groupement définissent les propriétés du groupement. Ces groupements ont été identifiés directement à partir des sources de données RDF à l'aide de requêtes SPARQL spécifiques. L'ensemble de groupements de ressources a été représenté par un diagramme de classes où les classes représentent les groupements des instances déterminés. Les attributs et les associations de chaque classe ont été déduits des propriétés de chaque groupement. Des statistiques telles que le nombre d'instances par classe et le nombre d'instances par propriété enrichissent le diagramme de classe. Enfin, le schéma proposée aide les utilisateurs à comprendre la structure de données, ce qui lui facilite une exploration des données ciblée et par la suite, la construction des requêtes SPARQL appropriées.

Dans le chapitre 3, nous avons présenté notre approche d'interrogation des données RDF. Cette approche est basée sur les motifs de triplets. Elle permet à l'utilisateur d'explorer les différents éléments d'une source de données RDF. L'utilisateur peut ainsi faire des recherches ciblées. La suggestion de complétion est offerte également et permet de fournir une liste de suggestion selon des mots-clés saisis par l'utilisateur. Nous avons proposé également la possibilité d'explorer le voisinage d'un élément du graphe RDF afin de limiter la zone de recherche. L'exploration est accompagnée par la composition assistée de la requête SPARQL. Par conséquent, l'utilisateur n'aura plus de soucis à créer des requêtes, mais ça n'empêche pas l'utilisateur à mettre à jour les requêtes. Notre objectif à travers cette approche consiste à fournir tous les outils possibles afin de faciliter l'interrogation des données RDF.

Dans le chapitre 4, nous avons commencé par présenter l'outil QUERY-RDF que nous avons développé et qui permet d'interroger des données RDF. QUERY-RDF est composé de deux principaux composants. Le premier correspond au schéma de données RDF présenté par le diagramme de classe. Le deuxième correspond à l'interrogation de données RDF en utilisant la recherche par mots-clés ou en construisant des requêtes SPARQL. Ensuite, nous avons présenté les expérimentations réalisées pour la validation de notre approche de construction du schéma de données RDF. Nous avons effectué deux méthodes d'évaluations. La première correspond à comparer notre résultat avec le résultat de (Li & Zhang, 2013). La deuxième méthode consiste à évaluer la qualité des classes identifiées avec des données synthétiques créées par nous-mêmes. Les résultats des évaluations ont montré que les classes identifiées par l'approche résument parfaitement les données RDF. L'évaluation de l'interrogation des données était accomplie en comparant le temps de la construction des requêtes manuelles par rapport aux fonctionnalités offertes par l'approche d'interrogation des données et ce qui a montré que notre approche permet de gagner beaucoup plus de temps. Enfin, l'évaluation de la complémentarité de deux approches était comme suit : dans un premier lieu, nous avons expliqué les conséquences d'avoir juste un schéma des données RDF sans avoir la possibilité d'interroger des données RDF. Dans un second lieu, nous avons mesuré la durée de l'interrogation des données dans l'absence et la présence du schéma des données. Nous avons montré que la durée de l'interrogation s'améliore d'une manière considérable lorsque l'utilisateur connaît la structure des données. Ainsi, nous avons montré l'importance d'avoir un schéma pendant la composition d'une requête.

Bien que les résultats témoignent de la pertinence et de l'efficacité de notre approche de composition assistée de requêtes sur des données RDF, il serait intéressant d'essayer d'améliorer ce travail, notamment :

- Implémentation basée sur le Web : À ce stade, pour utiliser QUERY-RDF, l'utilisateur doit installer GraphViz et certaines bibliothèques Java. Ce serait bien si QUERY-RDF pouvait être utilisé comme application web.
L'utilisateur peut utiliser la version Web du QUERY-RDF pour explorer la structure de schéma RDF, pour interroger les données RDF, pour créer et exécuter des requêtes SPARQL et afficher les résultats sans besoin d'installer quoi que ce soit sur sa machine ;
- Source de données de type SPARQL endpoint : Actuellement, nous limitons à des sources de données de type RDF. Cependant, ça sera intéressant de permettre d'avoir aussi la possibilité de traiter les sources de données de type SPARQL endpoint ;
- Faire un sondage : c'est intéressant d'inviter des utilisateurs afin qu'ils évaluent notre prototype et confirment la pertinence de notre approche ;
- Classement des suggestions : c'est possible de classer les suggestions de complétion selon la fréquence d'utilisation ou selon des critères de pertinence basés sur des métriques comme TF-IDF.

BIBLIOGRAPHIE

- Ambrus, O., Möller, K., & Handschuh, S. (2010). Konduit VQB: A visual query builder for SPARQL on the social semantic desktop. *Proceedings of the Workshop on Visual Interfaces to the Social and Semantic Web(VISSW 2010)*, (pp. 1-5).
- Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Zachary, I. (2007). DBpedia: A Nucleus for a Web of Open Data. *ISWC*, 4825, pp. 722–735.
- Benedetti, F., Bergamaschi, S., & Po, L. (2014). A Visual Summary for Linked Open Data sources. *In Proceedings of the ISWC 2014 Posters & Demonstrations*, 15(24), pp. 173-176.
- Brian, M. (2004, Février 25). *RDF Vocabulary Description Language 1.0: RDF*. Récupéré sur W3C Recommendation: <http://www.w3.org/TR/rdf-schema/>
- D.L, M., & F, V. H. (2004). *OWL Web Ontology Language Overview*. Récupéré sur W3C Recommendations: <http://www.w3.org/TR/owl-features/>
- Daly, J., Forgue, M.-C., & Hirakawa, Y. (2005, Janvier 26). *Le W3C soutient le standard URI et le standard proposé IRI*. Récupéré sur w3: <https://www.w3.org>
- Dudáš, M., Svátek, V., & Mynarz, J. (2015, 5 9). Dataset Summary Visualization with LODSight. *The Semantic Web: ESWC 2015 Satellite Events*, 15(20), pp. 36-40.

- Groppe, J., Groppe, S., & Schleifer, A. (2011). Visual Query System for Analyzing Social Semantic Web. *International World Wide Web Conference Committee*, pp. 217-222.
- Heim, P., Ziegler, J., & Lohmann, S. (2008). gfacet: A browser for the web of data. *Proceedings of the International Workshop on Interacting with Multimedia Content in the Social Semantic Web*, 417, pp. 49-58.
- Hitzler, P., Krotzsch, M., & Rudolph, S. (2011). *Foundations of semantic web technologies*. Chapman and Hall/CRC.
- Hogenboom, F., Milea, V., Frasincar, F., & Uzay, K. (2010). RDF-GL: A SPARQL-based graphical. *Emergent Web Intelligence: Advanced Information Retrieval*, pp. 87-116.
- iTunes*. (2018). Récupéré sur Apple: <http://www.apple.com/itunes>
- Jethro, B. (2006). *Graphical query composition and natural language processing in an RDF visualization*. Erasmus University Rotterdam.
- Jin-Sung, K., & Mi-Kyung, L. (2005). Object Modeling of RDF Schema for Converting UML Class Diagram. In *Proceedings of the International Conference on Computational Science and Its Applications*, pp. 31-41.
- Kjernsmo, K., & Passant, A. (2009, Juillet 02). *SPARQL New Features and Rationale*. Récupéré sur W3C: <https://www.w3.org/TR/sparql-features/#Update>
- Knublauch, H., & Kontokostas, D. (2017, Juillet 20). *Shapes Constraint Language (SHACL)*. Récupéré sur W3: <https://www.w3.org/TR/shacl/>

- Li, H., & Zhang, X. (2013, Mai 2). Visualizing RDF Data Profile with UML Diagram. *Semantic Web and Web Science*, pp. 273-285.
- Raimond, Y. (2016, Juillet 30). *DBTune.org Jamendo RDF Server*. Récupéré sur Datahub: <https://old.datahub.io/dataset/jamendo-dbtune>
- Schreiber, G., & Raimond, Y. (2014, Juin 24). *RDF 1.1 Primer*. Récupéré sur W3C: <https://www.w3.org/TR/rdf11-primer/>
- Seema, S., Medha, A., kolovski, V., & Wu, Z. (2010, Mars 01). Visualizing large-scale RDF data using Subsets, Summaries, and Sampling in Oracle. *International Conference on Data Engineering*, 10(2), pp. 1048-1059.
- Smart, P., Russell, A., & Braines, D. (2008). A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer. *EKAW 2008*, pp. 275 – 291.
- Tim Berners-Lee, Y. C., & Sheets, D. (2006). Tabulator: Exploring and Analyzing linked data on the Semantic Web. *In 3rd Int. Semantic Web User*.
- Tong, Q., Zhang, F., & Cheng, J. (2014). Construction of RDF(S) from UML Class Diagrams. *Computing and Information Technology (CIT)*, 22(4), pp. 237-250.
- Tsa, I. (2012, Juillet 4). *D2RQ*. Récupéré sur W3C: <https://www.w3.org/2001/sw/wiki/D2RQ>
- Weise, M., & al. (2016). LD-VOWL: Extracting and Visualizing Schema Information for Linked Data. *International Conference on Knowledge Engineering and Knowledge Management*, pp. 120-127.