Nouvelles méthodes pour l'évaluation, l'évolution et l'interrogation des bases du Web des données

Pierre Maillot

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

université angers

Thèse de Doctorat

Pierre MAILLOT

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université d'Angers
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique, section CNU 27
Unité de recherche : Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA), UFR Sciences d'Angers
De Vinci Technology Lab, Ecole Supérieure d'Ingénieur Léonard de Vinci (ESILV)

Soutenue le 26 Novembre 2015

Nouvelles méthodes pour l'évaluation, l'évolution et l'interrogation des bases du Web des données

JURY

Président :	**M. Francis ROUSSEAUX**, Professeur, Université de Reims
Rapporteurs :	**M^me^ Juliette DIBIE**, Professeur, AgroParisTech/INRA
	M. Mohand-Saïd HACID, Professeur, LIRIS, Université Claude Bernard Lyon 1
Examinateurs :	**M. David GENEST**, Maître de conférence, LERIA, Université d'Angers
	M. Thomas RAIMBAULT, Enseignant-chercheur, De Vinci Technology Lab, ESILV
Directeur de thèse :	**M. Stéphane LOISEAU**, Professeur, LERIA, Université d'Angers

Remerciements

Je remercie Francis Rousseaux d'avoir accepté de présider le jury de ma soutenance. Je remercie également Juliette Dibie et Mohand-Saïd Hacid d'avoir accepté d'être mes rapporteurs.

Merci à mon directeur de thèse Stéphane Loiseau pour ses remarques éclairées et son encadrement tout au long de cette thèse.

Merci à David Genest et Thomas Raimbault pour leur encadrement et leur soutien sans faille, l'un à 300 kilomètres, l'autre à 2 portes de bureau de distance.

Merci à Jean Rohmer sans qui ce doctorat n'aurait pas eu lieu.

Sincèrement merci à tout le personnel de l'ESILV pour son accueil et sa convivialité tout au long de ces 3 ans, avec une mention spéciale sans ordre particulier à Gaël Chareyron, Bérengère Branchet, Jérôme Da Rugna, Sonia Djebali, Carine Van Der Gucht, Corine Guillot, Nicole Warmberg, Mustapha Zakari, Najet Assaki et à ceux dont le nom m'échappe au moment ou j'écris ces lignes.

Merci également à mes amis, notamment Thomas Lechat, Valérian Merkling et Marc Legeay, sans qui le temps m'aurait paru bien trop long.

Merci évidemment à mes parents et ma famille, pour tout.

Merci à Kit Kat et à Nespresso.

RDF pour les exprimer,
OWL pour les structurer,
SPARQL pour les interroger
et le Linked Data pour tous, dans le Web des données, les lier

Bref, merci à tous

Table des matières

Introduction

Le World Wide Web, communément appelé Web, mis au point en 1991 par Tim Berners-Lee, est actuellement la plus grande source de données consultable par l'utilisateur. Il est constitué de documents reliés entre eux par des liens hypertextes et se base sur le réseau Internet pour fonctionner. Les différents langages et protocoles utilisés sur le Web sont encadrés par le World Wide Web Consortium (W3C), fondé en 1994 et dirigé par Tim Berners-Lee. Les documents Web sont hébergés sur des serveurs et consultables par des clients via le protocole HTTP à travers un navigateur Web. Chaque document possède une adresse unique, nommée URL, permettant de le localiser sur n'importe quel serveur d'Internet. Les documents Web sont composés de textes et de contenus multimédias. Dans sa forme actuelle – dite Web 2.0 – le contenu d'un document Web peut être généré dynamiquement grâce aux différents langages de mise en page (HTML, CSS, *etc.*) ou de programmation (Javascript, *etc.*). Ce mélange de contenus et de technologies permet une meilleure expérience utilisateur par rapport au Web « statique » des débuts. Malheureusement, ces technologies orientées vers l'humain rendent le traitement automatique du contenu du Web difficile. À la difficulté du traitement du langage naturel des textes et des différents contenus multimédias s'ajoute la difficulté de séparer contenu, langages de mise en page et langages de programmation.

Pour permettre la pleine utilisation des données du Web, Tim Berners-Lee, soutenu par le W3C, lance en 2001 le concept de Web Sémantique dans [Berners-Lee et al., 2001]. Le Web Sémantique est au départ un mouvement collaboratif pour le partage de données qui se présente comme une extension du Web, et qui contient uniquement des données bien formatées et facilement exploitables par un ordinateur. Le but ultime du Web Sémantique est de permettre le partage de toutes sortes de données et d'informations pour permettre un niveau d'interopérabilité sur le Web ; l'interopérabilité était faible à l'époque.

Le Web Sémantique est d'abord – et toujours – basé sur le framework RDF, [Wood et al., 2014], qui permet d'agencer des données sous forme de triplets (sujet, relation, objet). Cet triplets sont semblables à des courtes phrases, *e.g.* (Paul, est le frère de, Jacques), donnant un sens au données ainsi agencées, d'où le qualificatif sémantique. Ce framework permet la création de documents RDF, décrivant des ressources RDF par des ensembles de triplets. Chaque ressource RDF possède un identifiant unique, nommé URI. Le plus souvent, un URI est l'URL d'une page Web à propos de la ressource qu'elle identifie. Le Web Sémantique introduit ensuite *RDF Schema* (RDFS) [Guha and Brickley, 2004], puis Web Ontology Language (OWL), [Schreiber and Dean, 2004], qui permettent de définir des ontologies pour organiser le contenu des documents RDF. Ces ontologies permettent de définir des classes de ressources RDF, de les organiser en hiérarchies et de définir des relations entre elles. Le rôle des ontologies dans les documents RDF est similaire à celui du schéma des bases relationnelles. Les ontologies fixent un cadre structurel pour les données – plus souple que celui des bases de données classiques – et se basent sur les logiques de description pour apporter un vrai support *sémantique* au Web Sémantique en

permettant des interprétations et inférences logiques.

Avant 2008, le Web Sémantique est composée de plusieurs bases provenant principalement de chercheurs et de personnes enthousiasmées par le Web Sémantique. Le manque de système de gestion et d'interrogation de triplets empêche le Web Sémantique de passer d'un système réservé à une communauté de chercheurs à un système ouvert au plus grand nombre. En 2008, le W3C propose enfin un langage standard de requêtes SPARQL [Prud'hommeaux and Seaborne, 2008], qui permet d'interroger les documents RDF avec une syntaxe adaptée à leur structure en triplets. Les documents RDF deviennent alors des *bases RDF* hébergées sur des serveurs et interrogeables chacune via un point d'accès (endpoint) par des requêtes SPARQL. Après l'introduction de SPARQL, on commence à voir apparaître des bases de plusieurs millions de triplets, telles que DBPedia, [Bizer et al., 2009b] et Geonames[1], utilisables par tous. Tim Berners-Lee propose actuellement d'utiliser l'appellation *Web des données* au lieu de Web Sémantique afin d'éviter une confusion fréquente faite avec les systèmes d'interrogation en langage naturel de textes du Web qui utilisent parfois l'expression Web Sémantique.

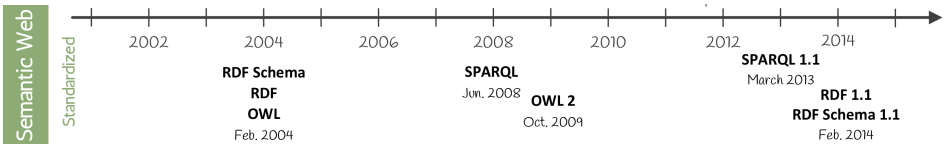


FIGURE 1 – Chronologie des technologies du Web des données

Avec l'apparition du langage SPARQL et des moteurs de gestion de données RDF qui l'accompagnent, le Web des données prend véritablement son essor. En 2009, un mouvement de publication de bases RDF nommé *Linked Data*, [Bizer et al., 2009a], apparaît. Le Linked Data est proche de l'image du Web des données imaginé par Tim Berners-Lee, il le qualifie d'ailleurs de « Web des données fait correctement ». Il s'agit d'un ensemble de bases RDF dans des domaines variés, qui respectent plusieurs règles de publication pour le partage de leurs données. Le Linked Data a suivi une forte croissance grâce à de nombreuses communautés impliquées dans l'alimentation et la mise à jour de bases dans différents domaines d'intérêts, tels que la biologie, les données gouvernementales, bibliographiques ou les connaissances encyclopédiques. La croissance du Linked Data a stimulé l'évolution des technologies à la base du Web des données, comme on peut le voir en figure 1 avec les dernières versions de OWL, SPARQL, RDF(S), apparus respectivement en 2009, 2013 et 2014. Deux règles essentielles du Linked Data doivent être citées : (i) « Tout URI de ressource RDF du Linked Data doit correspondre à une page Web qui décrit la ressource » ; (ii) « Chaque base du Linked Data doit établir des relations entre ses ressources et celle d'autres bases du Linked Data ». La première règle garantie que les données du Linked Data sont exploitables à la fois automatiquement grâce à RDF et par un utilisateur grâce à la lecture des pages correspondantes aux ressources. La deuxième règle permet de considérer le Linked Data comme un ensemble de bases interconnectés qui forme un immense graphe de données RDF.

À l'heure actuelle, quatorze ans après ses débuts, le Web des données devient mature. Le Linked Data est pour l'instant la partie la plus visible et connue du Web des données avec

[1] http://www.geonames.org/

plusieurs centaines de bases et plusieurs milliards de triplets. De grandes entreprises – *e.g.* Google, avec le Knowledge Graph[2] – intègrent les technologies du Web des données dans leurs services. De même, des moteurs de recherche spécialisés – *e.g.* Wolfram Alpha[3] – et des initiatives publiques – *e.g.* celles dans l'Open Data[4] – se basent sur le Web des données pour leur fonctionnement. Actuellement, le Web des données forme une sorte d'extension semi-indépendante du Web, qui se concentre sur la mise à disposition et la recherche de données, plutôt que sur l'affichage de contenu et de services. Les bases du Web des données sont créées le plus souvent en contenant un noyau de données et une ontologie faite par un expert. Ce noyau est alors progressivement augmenté par des mises à jour de la communauté de la base, constituées majoritairement d'ajouts et de modifications des données de la base. Ces mises à jour suivent parfois un processus d'évaluation qui filtre les modifications indésirables dans la base. De plus, certaines bases suivent également des phases d'examen au cours desquelles un expert adapte l'ontologie mais aussi corrige certaines données. Tout au long de leurs existences, l'interrogation de chaque base peut se faire par des requêtes SPARQL.

Trois constats sur le Web des données, en particulier sur le Linked Data, nous semblent importants et ont motivé les directions de recherche de cette thèse. Notre premier constat est que la qualité des données disponibles sur le Web des données n'est pas bien assurée. Le nombre de bases RDF dans le Web des données et leurs tailles sont en croissance constante au fur et à mesure que de nouvelles données sont ajoutées et que les données actuelles du Web sont converties en RDF. Cette accumulation de données se fait sans mécanisme standardisé ou reconnu d'évaluation de la qualité des données. L'évaluation de la qualité est une problématique cruciale pour pouvoir partager les données, tout particulièrement dans le Linked Data où les bases sont interconnectées, et donc dépendantes de la qualité des données des autres. La problématique de l'évaluation de la qualité de données est déjà étudiée dans le domaine des bases de données classiques, mais la structure particulière des bases RDF et leurs interconnexions entre elles et avec le Web nécessitent de nouvelles approches pour y répondre. Notre deuxième constat est que les ontologies doivent évoluer au cours de la vie d'une base, alors qu'elles sont souvent figées une fois pour toutes à la création de la base. Tout comme les schémas d'une base de données classique et ses données, l'ontologie d'une base RDF et ses données doivent être en adéquation. Il est donc nécessaire dans la vie d'une base d'effectuer un examen de l'adéquation entre ses données et son ontologie – plus ou moins périodiquement – pour accompagner les évolutions du domaine de la base. Bien que crucial, cet examen peut difficilement être effectué automatiquement, il doit être réalisé par un expert avec une bonne connaissance des données de la base RDF et du domaine qu'elle décrit. Sans examen et évolution réguliers de l'ontologie d'une base, on observe qu'un nombre croissant de conflits apparaissent entre les données ajoutées et leur structure attendue. Ces conflits sont particulièrement visibles dans les bases communautaires où les nombreux contributeurs sont généralement moins rigoureux et font évoluer la structure des données de manière informelle en fonction des tendances de leurs communautés. Les conflits que ces pratiques informelles engendrent rendent plus difficile l'exploitation automatique des données de la base et l'utilisation des moteurs d'inférence. Notre troisième constat porte sur la difficulté d'interrogation du Web des données ; en effet SPARQL est conçu pour interroger une base à la fois et non tout le Web des données. L'interrogation des bases du Web des données se fait base par base à l'aide du langage SPARQL. Le Web des données dont le but est le partage libre des

[2] http://googleblog.blogspot.gr/2012/05/introducing-knowledge-graph-things-not.html
[3] http://www.wolframalpha.com/
[4] http://www.opendatafrance.net/

données, n'a pas de structure centralisée permettant l'interrogation de façon distribuée de toutes les bases RDF. Il n'y a pas non plus, comme dans le Web classique, de moteur de recherche (re)connu.

Dans cette thèse, nous proposons quatre contributions pour améliorer l'évaluation de la qualité des données RDF d'une mise à jour, l'évolution de l'ontologie et des données par un expert et l'interrogation d'une base RDF (*cf.* tableau 1). Ces quatre contributions améliorent la gestion – automatique ou non – et l'exploitation des bases du Web des données tout au long de leur cycle de vie entre leur création et leur destruction. Nos contributions ont toutes été conçues de sorte à s'intégrer avec les moteurs et outils existants du Web des données.

Notre première contribution répond au problème de la qualité des données d'une base RDF par la proposition d'une méthode automatique d'évaluation de la qualité des données ajoutées ou modifiées dans une base RDF. Notre méthode évalue si les mises à jour peuvent être appliquées à une base en fonction de leur similarité avec les données existantes. Elle se présente comme un filtre qui détermine l'application ou non des mises à jour en fonction de leur similarité avec les données existantes dans la base. L'idée est de détecter les modifications qui risquent de créer des données aberrantes dans la base si elles sont appliquées et de les écarter pour empêcher la création de données à la structure anormale. Ce filtrage automatique est particulièrement adapté pour les bases communautaires qui subissent de nombreuses mises à jour de qualité variable en provenance de leurs communautés de contributeurs. Notre première contribution permet donc d'évaluer la qualité de mises à jour d'une base RDF, un peu à la manière des travaux [Kontokostas et al., 2014], [Bizer and Cyganiak, 2009] et [Bonatti et al., 2011], mais en se basant sur le critère original de la similarité des données.

Notre deuxième contribution répond à la problématique de l'évolution de l'ontologie et des données d'une base par un expert grâce à une méthode d'examen de l'adéquation de l'ontologie et des données. Notre méthode fournit une vue d'ensemble des données et des parties de l'ontologie en conflit qui nécessitent une correction d'un expert lors de l'examen périodique d'une base. L'idée est ici d'utiliser les définitions de l'ontologie pour identifier les données de la base qui sont en conflit avec elle. Parmi ces données en conflit, nous extrayons automatiquement les structures de données récurrentes qui apparaissent et qui donnent des indications utiles à l'expert pour ses corrections (*e.g.* une nouvelle pratique communautaire). Notre deuxième contribution permet donc d'aider un expert lors de l'évolution d'une bases RDF, comme pour les travaux [Köhler et al., 2011] et [Nikitina et al., 2012] mais en fournissant une vue d'ensemble originale des structures récurrentes autour des individus et de leurs relations en conflit avec l'ontologie.

Notre troisième contribution est une réponse à la problématique de l'interrogation du Web des données par une méthode performante d'interrogation d'un ensemble de bases RDF. Notre méthode interroge un ensemble de bases pour une même requête SPARQL en utilisant des sortes de résumés de bases pour déterminer quelles bases sont susceptibles de contenir un résultat à la requête SPARQL. Cette méthode se base sur l'ontologie et la structure des données d'une base RDF pour créer une sorte de résumé des données de la base, appelé aperçu de recherche. Cet aperçu de recherche est de taille réduite, ce qui rend son interrogation plus rapide que celle de sa base d'origine. De plus, par construction, son interrogation donne une indication sûre à propos de la présence ou l'absence de résultats à une requête SPARQL dans sa base d'origine. Les aperçus de recherche sont donc des documents RDF utilisés comme filtres rapides lors de l'interrogation d'un ensemble de bases. Ils évitent l'interrogation inutile des bases qui ne contiennent pas de résultat à une

requête. Notre troisième contribution permet donc d'interroger, pour une seule requête, un ensemble de bases en évitant les bases qui ne peuvent pas contenir de résultat. Pour cela, elle utilise des résumés des bases, proches de ceux décrit dans [Fokoue et al., 2012], aux propriétés originales.

Notre quatrième contribution est un ensemble d'implémentations et d'optimisations efficaces des méthodes de nos trois premières contributions.

	Nos contributions		
Thématique	Qualité	Évolution	Interrogation
Comment ?	Similarité	Détection de motifs	Aperçu de recherche
Qui ?	Contributeur	Expert	Utilisateur

TABLE 1 – Présentation de nos contributions.

Notre mémoire est organisé comme suit. Dans le chapitre 1, nous présentons en détail le Web des données et ses technologies. Nous faisons son état des lieux en nous concentrant particulièrement sur l'état des bases du Linked Data. Nous détaillons également les différents éléments nécessaires à la compréhension du mémoire et introduisons les définitions communes à toutes nos contributions. Dans le chapitre 2 nous détaillons notre méthode d'évaluation de la qualité des mises à jour de base RDF par similarité. Notre méthode d'évaluation compare les données d'une mise à jour avec les données de la base RDF pour déterminer si la mise à jour peut être ajoutée ou non. Dans le chapitre 3 nous détaillons notre méthode d'aide à l'examen de l'adéquation de l'ontologie et des données d'une base RDF. Notre méthode d'examen d'une base RDF indique à un expert les parties de l'ontologie et les structures récurrentes dans les données qui semblent nécessiter une évolution. Dans le chapitre 4 nous détaillons notre méthode d'interrogation d'un ensemble de bases à l'aide d'aperçus de recherche. Notre méthode d'interrogation utilise l'ontologie d'une base pour générer un aperçu de recherche utilisé ensuite pour empêcher l'interrogation de la base par des requêtes dont elle ne contient pas de résultat. Enfin dans le chapitre 5 nous présentons des implémentations efficaces de nos contributions. Les implémentations présentés dans ce chapitre sont plus performantes que les méthodes naïves décrites dans les chapitres précédents.

Chapitre 1

Web des données

Le Web des données est un mouvement collaboratif pour étendre le Web par des données structurées partagées. Son idée fondatrice, exprimée par Tim Berners-Lee en 2001 dans [Berners-Lee et al., 2001], s'inspire de la structure des pages Web – liées entre elles par des liens hypertextes – pour proposer une nouvelle représentation standardisée des données, exploitable aussi bien par l'humain que la machine. Le Web des données se base sur le framework RDF qui formate les données sous la forme de triplets. Un triplet est composé de trois éléments : un sujet, un prédicat et un objet. Ces triplets mettent en relation des ressources RDF qui désignent des ressources du Web, du monde réel ou des concepts généraux. Par exemple, le triplet (Tim Berners-Lee, dirige, W3C) met en relation le W3C avec son dirigeant. Chaque ressource RDF a un identifiant unique, qui est le plus souvent l'URL d'une page Web en rapport avec la ressource. Les langages RDFS et OWL sont utilisés pour organiser les ressources RDF en classes hiérarchisées – à la manière des classes de la programmation orienté objet – et pour définir les relations qui peuvent lier les ressources. Ces langages permettent également de faire des inférences à partir de données RDF en se basant sur les logiques de description. L'utilisation des langages RDFS et OWL pour structurer les données et leur organisation en triplets pour former des graphes de données rend les bases RDF plus souples que les bases de données relationnelles. Les bases RDF sont interrogeables par les requêtes du langage SPARQL.

L'incarnation la plus tangible du Web des données est le Linked Data [Bizer et al., 2009a], apparu en 2008, en même temps que SPARQL. Il s'agit d'un regroupement de bases qui concernent des domaines variés et qui suivent des règles communes de structuration et de publication des données. Ces bases sont reliées entre elles par des relations d'équivalence entre ressources RDF désignant les mêmes éléments dans des bases différentes. Les ressources RDF des bases du Linked Data sont également toutes associées à des pages Web lisibles par l'humain et fournissent toutes un moyen d'accès à leur contenu, *i.e.* par téléchargement de la base ou par requêtes SPARQL. Depuis 7 ans, le Linked Data a été un des moteurs de la croissance du Web des données, entraîné par de nombreux utilisateurs et contributeurs qui font vivre communautairement les centaines de bases du Linked Data.

Ce chapitre présente toutes les facettes du Web des données nécessaires pour la compréhension de cette thèse. Nous présentons les technologies du Web des données, ainsi que le Linked Data et nous en faisons un état des lieux. Enfin, nous définissons les concepts avancés de manipulation des documents RDF nécessaires à nos contributions.

En section 1 nous présentons le framework RDF ; ses différentes syntaxes sont présentées en section 2. Nous présentons les langages RDFS en section 3 et OWL en section 4.

Nous poursuivons ensuite avec le langage de requêtes SPARQL en section 5. Nous présentons le Linked Data et faisons son état des lieux en section 6. Enfin, nous donnons en section 7 les définitions de concepts avancés de gestion des ressources RDF et de leurs descriptions, qui sont utilisées dans les chapitres suivants.

Sommaire

1 RDF

RDF – *Resource Description Framework* – [Wood et al., 2014] est à la base des technologies du Web des données. Il s'agit d'un modèle de données simple pour la représentation de connaissances sur le Web dans des *documents RDF*.

Comme son nom l'indique, RDF sert à décrire des *ressources RDF*. Une ressource RDF est un concept ou un élément concret (toute sorte de chose, *e.g.* une personne, un pays, un événement, ...). Chaque ressource RDF est – théoriquement – unique dans tout le Web des données. Les descriptions des ressources RDF sont composées de *triplets RDF*, (sujet, relation, objet), qui peuvent être représentés sous forme de graphe tel que dans les figures 1.1 et 1.3. Un triplet RDF exprime une relation entre un sujet – la ressource décrite – et un objet. Autrement dit, un triplet décrit une propriété du sujet ayant pour valeur l'objet du triplet. La description d'une ressource correspond à l'ensemble des triplets qui contiennent la ressource en sujet.

FIGURE 1.1 – Représentation d'un triplet RDF décrivant la ressource « sujet » comme ayant pour propriété « relation » la valeur « objet ».

Considérons un triplet RDF (sujet, relation, objet) :

- le sujet est une ressource, soit identifiée explicitement par un URI, sinon correspondant à un nœud blanc.
  - Un URI[1] – *Uniform Resource Identifier*– est un identifiant unique sur le Web (*e.g.* une adresse page web à propos de la ressource). Cet identifiant est le plus souvent sous la forme d'une URL de page web. Un URI est dit *déréférençable* s'il désigne une page web existante à laquelle un utilisateur peut accéder via un navigateur Web. Par exemple `http://dbpedia.org/resource/Paris` représente la ville de Paris.
  - Un nœud blanc représente quelque chose qui n'a de sens qu'à l'intérieur du document, utile uniquement pour la description d'autres ressources. Par exemple une adresse qui regroupe un numéro de rue, une ville et un code postal, tel que le nœud blanc _:1 en figure 1.4 ; le nœud blanc permet un tel regroupement sans nécessiter un identifiant unique dans tout le Web. Il s'agit d'une version anonyme d'un URI dont l'identifiant a une portée limitée à son document. L'identifiant d'un nœud blanc est donc propre à un document RDF donné. Ainsi, lors de la fusion de deux documents RDF, on doit s'assurer que deux nœuds blancs de documents différents n'ont pas le même identifiant local.
- la relation est toujours une ressource identifiée (*i.e.* URI) ;
- l'objet est soit une ressource (identifiée ou non), soit une donnée brute, aussi appelée *littéral*.
  - un littéral représente une valeur, de type dit « primitif » (*i.e.* nombre, chaîne de caractère, date, *etc.*). Par exemple, "Marc" ou "1789-07-14".

[1] ou *IRI*, *Internationalized Resource Identifier* généralisation des URI gérant un plus grand nombre de caractères

Un *document RDF*, par exemple en figure 1.2, est également appelé *ensemble de triplets RDF* ou encore *graphe RDF*. Il peut en effet être représenté sous forme d'un multi-graphe orienté étiqueté, tel que par exemple en figure 1.3 et 1.4.

```
<Paul> <parentDe> <Jean> .
<Paul> <parentDe> <Marie> .
<Paul> <habite> <Paris> .
```

FIGURE 1.2 – Ensemble de triplets RDF (URIs simplifiés ici).

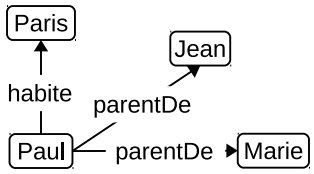


FIGURE 1.3 – Représentation sous forme de graphe de l'ensemble de triplets RDF en figure 1.2.

Pour plus de lisibilité il est possible d'abréger un URI en utilisant un préfixe qui remplace le début de l'adresse utilisée par l'URI. Par exemple, l'URI de l'élément `http://www.exemple.com/rdf#ressource` est remplacé par ex:ressource et tous les autres URIs qui commencent par `http://www.exemple.com/rdf#` utilisent le préfixe ex. On dit alors que ex est le préfixe du domaine `http://www.exemple.com/rdf#`.

Dans les chapitres suivants, nous utiliserons les préfixes :

- rdf: pour `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
- rdfs: pour `http://www.w3.org/2000/01/rdf-schema#`
- owl: pour `http://www.w3.org/2002/07/owl#`
- xsd: pour `http://www.w3.org/2001/XMLSchema#`
- dbpedia: pour `http://dbpedia.org/resource/`
- dbponto: pour `http://dbpedia.org/ontology/`
- dbpprop: pour `http://dbpedia.org/property/`

Les littéraux ne peuvent être sujet ou relation d'un triplet. Il est cependant possible de préciser 2 propriétés pour un littéral :

- son type : Les littéraux peuvent être typés selon la norme XML Schema Datatypes [Biron and Malhotra, 2004], ainsi que par d'autres types créés spécifiquement dans le document (*cf.* section 3). Par exemple, le littéral "1975"^^xsd:integer est une donnée brute de type entier.
- sa langue : On peut attacher à un littéral une étiquette pour indiquer la langue associée. Par exemple, "ceci est un littéral"@fr est un littéral de langue française.

La dernière version de la norme RDF (RDF 1.1) [Wood et al., 2014] précise la notion d'*ensemble de données RDF* – ou encore *base RDF* – comme étant un ensemble de graphes RDF. À l'intérieur d'une base RDF, ce que l'on appelle graphe RDF est un ensemble de triplets. Selon cette norme, toute base RDF contient un graphe par défaut anonyme et peut contenir des graphes nommés par des URIs, *i.e.* des sous-ensembles nommés de la base. Dans les chapitres suivants, nous supposerons que les triplets de la base sont contenus dans le graphe par défaut et par conséquent que les appellations *ensemble de données RDF* et *base RDF* sont équivalentes à *graphe RDF*.

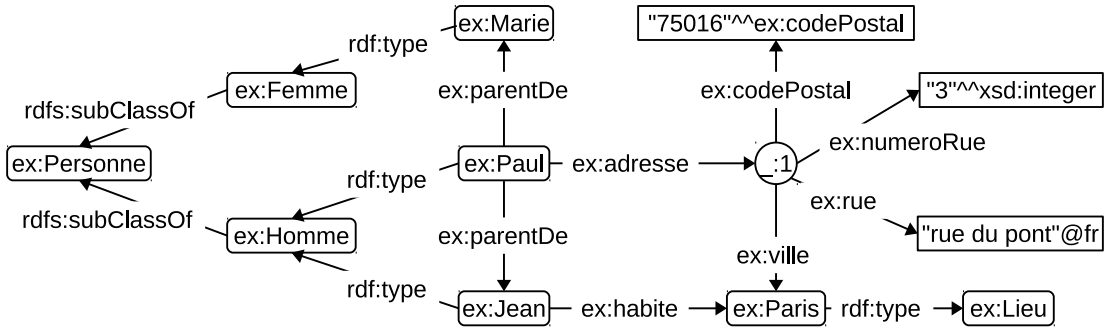


FIGURE 1.4 – Exemple de représentation graphique de document RDF avec URIs préfixés.

2 Syntaxes de RDF

Un document RDF peut être stocké dans des fichiers en suivant une des nombreuses syntaxes de RDF. Ces syntaxes permettent de représenter des documents RDF de différentes manières sans en changer le contenu de façons plus ou moins adaptées pour l'exploitation par la machine ou par l'humain.

La syntaxe RDF/XML [Beckett, 2004] est historiquement la première syntaxe à être apparue pour l'écriture de documents RDF. Cette syntaxe a été introduite en même temps que RDF, elle est basée sur XML et se place dans la continuité des technologies du Web. Son caractère très « verbeux », hérité de XML, la rend peu adaptée à l'exploitation par l'humain. Les fichiers RDF/XML, dont un exemple est en figure 1.5, utilisent les extensions .rdf.

```
<rdf:RDF>
    <rdf:Description rdf:about="http://www.exemple.com/rdf#Paul">
        <ex:parentDe rdf:resource="http://www.exemple.com/rdf#Jean"/>
        <ex:parentDe rdf:resource="http://www.exemple.com/rdf#Marie"/>
        <ex:habite rdf:resource="http://www.exemple.com/rdf#Paris"/>
    </rdf:Description>
</rdf:RDF>
```

FIGURE 1.5 – Document représenté en figure 1.3 au format RDF/XML.

La syntaxe Turtle [Carothers and Prud'hommeaux, 2014] est une syntaxe plus facilement lisible par l'humain qui permet d'écrire directement les triplets d'une base RDF. Elle permet une écriture intuitive des triplets avec des raccourcis qui permettent (i) l'utilisation de préfixes pour raccourcir les URIs, (ii) le regroupement des triplets ayant le même sujet et la même relation en les séparant d'une virgule et (iii) le regroupement des triplets ayant le même sujet en les séparant d'un point-virgule. Les fichiers Turtle, dont un exemple est en figure 1.6, utilisent l'extension .ttl.

```
@prefix ex: <http://www.exemple.com/rdf#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
ex:Paul ex:parentDe ex:Jean , ex:Marie ;
        ex:habite ex:Paris .
```

FIGURE 1.6 – Document représenté en figure 1.3 au format Turtle.

La syntaxe N-Triples [Seaborne and Carothers, 2014a] est une version simplifiée de Turtle ne permettant pas l'utilisation de préfixes ou le regroupement de triplets. Un

document N-Triples détaille chaque triplet ligne par ligne sans raccourcis d'écriture. Les fichiers N-Triples utilisent l'extension .nt.

L'introduction des graphes dans la norme RDF 1.1 a entraîné la création de deux nouvelles syntaxes inspirées de Turtle et N-Triples

La syntaxe TriG [Seaborne and Carothers, 2014b], inspirée de Turtle, permet de définir explicitement des graphes nommés dans un fichier RDF. Dans un fichier TriG, les triplets sont attribués à des graphes en étant encadrés par des accolades précédées par l'URI du graphe. Le graphe par défaut n'a pas d'URI, il est uniquement encadré par des accolades, tel que en figure 1.7. Les fichiers TriG utilisent l'extension .trig.

```
@prefix ex: <http://www.exemple.com/rdf#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
{
    ex:Paul ex:parentDe ex:Jean , ex:Marie ;
            ex:habite ex:Paris .
}
```

FIGURE 1.7 – Document représenté en figure 1.3 au format TriG.

La norme N-Quads [Carothers, 2014], inspirée de N-Triples, permet d'ajouter un quatrième élément aux triplets RDF, transformant certains triplets RDF en quadruplets N-Quads. Le quatrième élément d'un quadruplet N-Quads est réservé pour indiquer à quel graphe le triplet appartient (en indiquant l'URI du graphe). Les triplets RDF du graphe par défaut (qui n'a pas d'identifiant) sont écrits de la même manière que dans la norme N-Triples, le quatrième champ n'étant pas utilisé. Les fichiers N-Quads utilisent l'extension .nq.

3 Langage RDFS

Les ontologies sont des ensembles de concepts permettant de définir des regroupements de ressources – les *classes* –, de définir les relations utilisables pour les décrire – les *propriétés* – et de les hiérarchiser. Le W3C a fourni en 2004 un premier langage basique, RDFS – *RDF Schema* –, permettant de définir des ontologies simples. RDFS a ensuite été légèrement étendu en 2014 dans sa version 1.1, en même temps que RDF 1.1.

Traditionnellement on sépare dans un document RDF :

- les définitions des classes et des propriétés, qui forment l'ontologie,
- les définitions et mises en relations des instances des classes (aussi appelés individus), qui forment les faits du document.

Les bases RDF du Web des données peuvent utiliser plusieurs ontologies pour définir leurs données, cependant lorsque nous discutons des données d'une seule base, nous considérons l'union des ontologies utilisées dans la base comme formant « l'ontologie de la base ».

3.1 Classes

Une classe est un groupe de ressources ayant des propriétés similaires, *e.g.* Personne, Voiture, Pays, *etc.* On dit d'une ressource appartenant à une classe qu'elle est *instance*

de la classe. Une instance d'une classe est une ressource liée par la relation rdf:type à une classe. Les classes définies en RDFS sont elles-mêmes des instances de rdfs:Class, tel qu'en figure 1.8.

Les classes peuvent être organisées en hiérarchies pour indiquer des classes plus générales et des sous-classes plus spécialisées. Dans ces hiérarchies, on considère que les instances des sous-classes sont également instances des classes plus générales. Les hiérarchies de classes sont définies par la relation rdfs:subClassOf, tel qu'en figure 1.9.

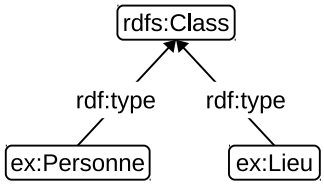


FIGURE 1.8 – Exemples de définition de classes.

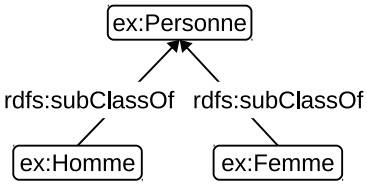


FIGURE 1.9 – Exemple de définition de sous-classes.

Le langage RDFS définit des ressources utiles à la définition de classes, notamment :

- rdfs:Resource est la classe par défaut de toutes les ressources définies selon RDFS.
- Tous les types de données des littéraux sont par défaut instances de la classe rdfs:-Datatype.
- Toutes les relations sont instances de la classe rdf:Property. Nous considérons de façon équivalente les termes « relation » et « propriété ».

3.2 Propriétés

Les propriétés sont les relations de triplet qui sont définies dans l'ontologie. Toute propriété d'une ontologie définie par le langage RDFS est instance de la classe rdf:Property. Tout comme les classes, les propriétés peuvent être organisées en hiérarchies par la relation rdfs:subPropertyOf, voir par exemple figure 1.10. Une hiérarchie de propriétés définit des niveaux de généralisation ou de spécialisation entre relations. On considère que deux ressources liées par une propriété sont également liées par ses propriétés plus générales.

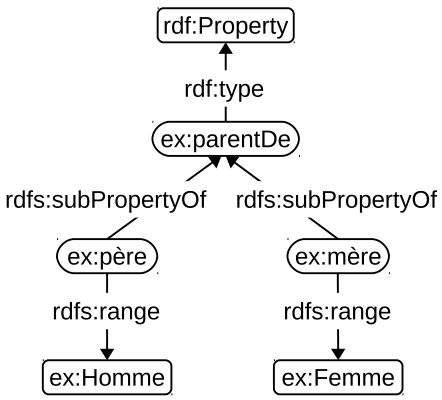


FIGURE 1.10 – Exemple de hiérarchie de propriétés.

3.3 Individus

Un *individu* est une ressource instance d'une classe définie dans l'ontologie – en dehors des classes définies par RDFS – dont des exemples sont donnés en figure 1.11. Pour un individu, la relation d'instanciation peut être traduite en langage naturel par « l'individu est un ... » ou « l'individu est une sorte ». Par définition, un individu est sujet d'au moins une relation d'instanciation, il est donc toujours représenté par un URI ou un nœud blanc, jamais par un littéral. Un individu instance d'une classe A est également appelé « individu de type A ».

Les individus sont les éléments de plus bas niveau d'un document RDF. En effet, les langages du W3C ne permettent pas de définir des relations de hiérarchisation entre individus. Selon le langage RDFS, toutes les ressources d'un document sont instances de rdfs:Resource. Ainsi, tous les individus d'un document ont toujours un type en commun.

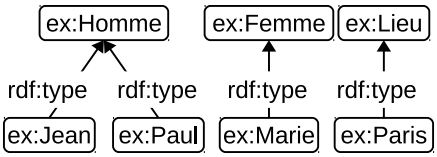


FIGURE 1.11 – Exemple d'instances des types Homme, Femme et Lieu définis en figures 1.8 et 1.9.

Le langage RDFS permet de définir le type attendu d'un individu en sujet (resp. en objet) d'une propriété par la relation rdfs:domain (resp. rdfs:range). Le type attendu du sujet d'un triplet est aussi appelé le domaine de la propriété. Le type attendu de l'objet d'un triplet est aussi appelé le co-domaine de la propriété (*range* en anglais). Lorsque plusieurs propriétés rdfs:domain (resp. rdfs:range) sont définies pour une propriété, son domaine (resp. co-domaine) est l'intersection des classes données en propriétés. Par exemple en figure 1.10, la relation ex:mère, liant une personne à sa mère, peut uniquement avoir des femmes en objet, son co-domaine est ex:Femme, défini par le triplet (ex:mère, rdfs:range, ex:Femme). Dans un autre exemple, représenté graphiquement en figure 1.12, la relation ex:meilleurActrice qui indique la meilleure actrice récompensée dans un film a uniquement des instances des classes ex:Femme et ex:Acteur en objet.

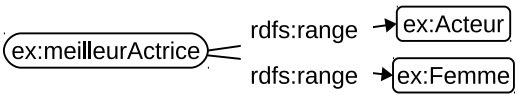


FIGURE 1.12 – Exemple de relation à domaine multiples.

Ensembles de ressources Le langage RDFS définit des *conteneurs* pour regrouper et lister des ressources. Ces conteneurs sont des individus de la classe rdfs:Container et de ses sous-classes rdf:Bag, rdf:Seq et rdf:Alt. Les éléments appartenant à ces conteneurs sont définis, voire ordonnés, par les propriétés de type rdfs:ContainerMembershipProperty : rdf:_1, rdf:_2, rdf:_3, ...

rdf:Bag définit un ensemble de ressources sans ordre défini. Par exemple, pour donner les ingrédients d'une recette de cuisine, tel que dans la figure 1.14.

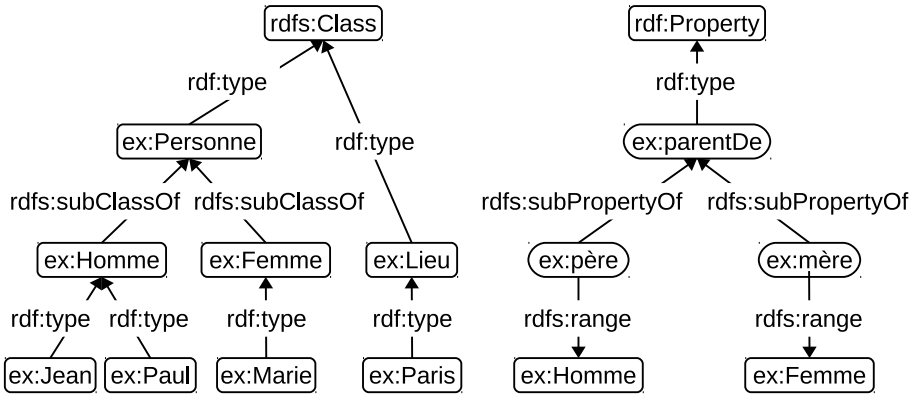


FIGURE 1.13 – Ontologie et instanciations résultantes des exemples 1.8, 1.9, 1.10 et 1.11 selon le langage RDFS.

rdf:Seq définit un ensemble de ressources accessible dans l'ordre indiqué par les propriétés d'appartenance.

rdf:Alt définit un ensemble de ressources parmi lesquelles on ne peut choisir qu'un à la fois (par défaut, le premier).

La norme RDFS permet également la définition de *collections*, c'est-à-dire d'ensembles fermés d'éléments. Contrairement aux conteneurs, les collections ne contiennent pas d'autres éléments que ceux définis. Ces collections sont instances de la classe rdf:List. On peut définir le premier élément d'une collection par la propriété rdf:first, et les autres éléments par rdf:rest. Une liste vide ou une partie de liste vide est représentée par l'élément rdf:nil.

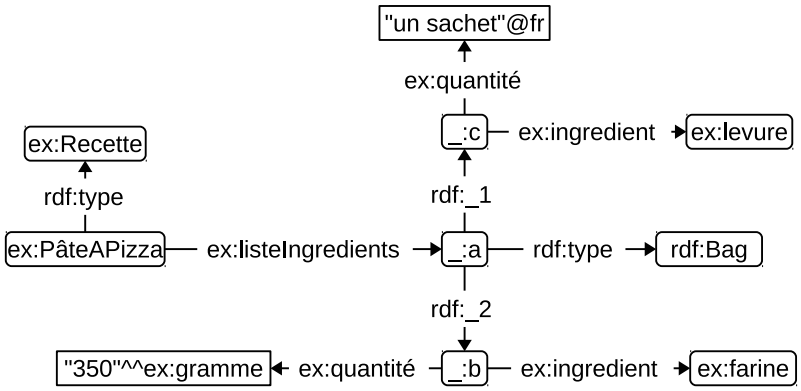


FIGURE 1.14 – Exemple d'utilisation d'un conteneur.

4 Langage OWL

Le langage OWL [Schreiber and Dean, 2004] est apparu en 2004, en même temps que RDF et RDFS, il a ensuite été étendu en 2009 dans la norme OWL 2 [OWL Working Group, 2009]. OWL est un langage qui étend RDFS, il introduit dans les documents RDF une sémantique logique tirée des logiques de description [Grau et al., 2012]. OWL apporte un langage plus riche pour la description de ressources en permettant, en plus d'une hiérarchie verticale, la définition de relations entre les classes et entre les propriétés , comme l'équivalence, la différence, l'union, le complément, *etc.*, ainsi que des raisonnements suivants les logiques de description.

Les logiques de description sont une famille de modèles formels de représentation des connaissances. Dans ces modèles on sépare d'un côté la terminologie dans une *T-Box* – l'ontologie – qui décrit les taxonomies et les relations. De l'autre côté sont définis les faits, ou les assertions, dans une *A-Box* contenant les connaissances factuelles. T-Box et A-Box forment une *base de connaissances* sur laquelle on peut effectuer des raisonnements grâce à une sémantique adéquate associée.

Les bases de connaissances en logiques de description reposent sur l'hypothèse d'un *monde ouvert* dans lequel l'absence d'une connaissance n'implique pas qu'elle est fausse (à l'opposé d'un *monde clos* où tout ce que l'on ignore est considéré comme faux). Cette hypothèse suppose qu'une base de connaissances ne contient pas toutes les connaissances disponibles. En suivant cette hypothèse, il est possible de définir des informations de manière dispersée, à la manière des bases RDF du Web des données.

Les raisonnements en logique de description sont basés sur la *classification* et *l'instanciation* des connaissances. La classification des connaissances correspond à l'organisation des classes et propriétés en hiérarchies. L'instanciation des connaissances correspond au typage des individus par les différentes classes du document (notamment les plus spécialisées). Ainsi, en pratique les raisonnements sur les documents OWL permettent de déduire de nouvelles connaissances sur les hiérarchies de classes et de propriétés et sur les types des individus du document.

Nous donnons ici quelques exemples des éléments définis par OWL :

- Toutes les classes définies en OWL sont instances de owl:Class, elle-même sous classe de rdfs:Class pour des raisons de compatibilité avec RDFS. De plus, toutes les classes sont par défaut sous-classes de owl:Thing.
- Les disjonctions de classes sont définies grâce à la relation owl:disjointWith, par exemple la classe ex:Homme est disjointe avec la classe ex:Femme par le triplet (ex:Homme, owl:-disjointWith, ex:Femme).
- Les classes OWL peuvent être définies selon des opérations ensemblistes, telle que les unions, par owl:unionOf, les intersections, par owl:intersectionOf et les compléments, par owl:complementOf.

4.1 OWL 1

Dans sa première spécification par le W3C en 2004, OWL a été séparé en trois sous-langages en fonction de leurs expressivités croissantes :

OWL Lite : OWL Lite permet uniquement des hiérarchies et contraintes simples, *e.g.* il supporte uniquement les contraintes de cardinalité 0 ou 1. Les limitations de l'expressivité de OWL Lite ne réduisent pas la complexité des raisonnements par rapport à OWL DL, leur but est de faciliter l'implémentation d'un raisonneur. C'est le moins complexe des 3 sous-langages OWL 1.

OWL DL : Ce sous-langage a été créé pour permettre une expressivité maximale tout en garantissant la décidabilité et un temps de calcul fini pour le raisonnement en logiques de description ; « DL » signifie *Description Logics*.

OWL Full : OWL Full permet l'utilisation de tous les éléments d'OWL sans contraintes, mais il n'offre aucune garantie de décidabilité ou de complexité pour les raisonnements.

Un document OWL Lite est un document OWL DL valide, et un document OWL DL est un document OWL Full valide. Les trois sous-langages d'OWL illustrent le dilemme des ontologies OWL pour lesquelles il faut choisir entre expressivité et décidabilité du raisonnement.

4.2 OWL 2

Le langage OWL 2 reprend les fonctionnalités de sa version précédente en ajoutant quelques éléments supplémentaires. Le principal apport d'OWL 2 est de définir des *profils*, des sous-langages d'OWL 2 qui limitent et contraignent les expressions utilisables selon l'usage voulu de l'ontologie. Contrairement aux sous-ensembles OWL Lite, DL et Full, les profils d'OWL 2 sont indépendants.

La norme OWL 2 définit trois profils :

OWL EL : OWL EL est adapté pour la définition d'ontologie contenant un très grand nombre de classes et de propriétés. Ce sous-langage limite l'expressivité des ontologies OWL 2 pour garantir des raisonnements en logiques de description de complexité au pire des cas polynomiale (PTIME-complet) selon la taille de l'ontologie. Il garantit également une interrogation de complexité au pire exponentielle (ou polynomiale sous certaines conditions). Le nom « EL » est tiré du sous-ensemble des logiques de description sur lequel ce profil se base : les logiques $\mathcal{EL}$ [Baader et al., 2005] qui permettent uniquement la quantification existentielle.

OWL RL : OWL RL est adapté pour permettre les raisonnements, issus de la logique classique, à grande échelle et sans sacrifier trop d'expressivité. Les raisonnements appliqués sur OWL RL peuvent être implémentés par des systèmes à base de règles (« RL » vient de *Rule Langage*). Ils sont au pire des cas co-NP-complets si ontologie et données sont mélangées, et PTIME-complets s'ils sont considérés séparément. Ce profil est parfois comparé aux différentes propositions de langages de règles pour le Web des données, telles que SWRL [Horrocks et al., 2004], pour la description de règles.

OWL QL : OWL QL est adapté pour les bases où on souhaite effectuer des raisonnements sur un grand nombre d'individus avec une ontologie de taille réduite. Ce sous-langage a été conçu de façon à ce que l'interrogation conjonctive des bases OWL QL puisse être effectuée par des systèmes performants de base de données relationnelle. OWL QL permet des raisonnements de complexité N-LOGSPACE et des interrogations de complexité NP-complète.

Il est possible de définir d'autres profils en définissant des sous-langages de OWL 2, *e.g.* OWL Lite et OWL DL peuvent êtres vus comme d'autres profils OWL 2. Les relations entre les sous-langages et profiles de OWL sont résumés en figure 1.15.

Après que OWL 1 ait proposé différents niveaux d'expressivité en fonction de leurs complexités de raisonnement, OWL 2 propose donc de nouveaux sous-langages dans lesquels l'expressivité est réglée en fonction de l'usage voulu, de la complexité et du niveau de raisonnements recherchés.

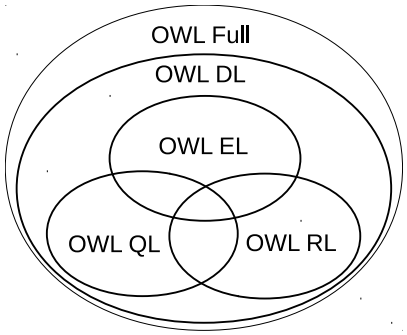


FIGURE 1.15 – Diagramme de Venn des sous-langages de OWL.

4.3 Raisonneurs

Les sous langages d'OWL ont été implémentés dans plusieurs raisonneurs prenant en charge différents niveaux d'expressivité. Nous donnons ici quelques exemples des raisonneurs récemment mis à jour. *ELK Reasoner* [Kazakov and Klinov, 2014] est un raisonneur sur les ontologies OWL EL. *Apache Jena*[2] est un framework Java répandu pour l'utilisation de données RDF qui intègre un raisonneur OWL prenant en compte un sous-langage d'OWL 2 proche de OWL EL. *Pellet* [Sirin et al., 2007] est un raisonneur OWL 2 prenant en charge les ontologies OWL EL et OWL DL. *Mastro* [Civili et al., 2013] est un système de gestion de bases RDF basé sur une base de données relationnelle classique et intégrant un raisonneur prenant en charge OWL Lite et OWL 2 QL. *Racer* [Haarslev et al., 2012] est un raisonneur prenant en charge les ontologies OWL DL et proposant son propre langage de requête d'interrogation nommé *nRQL*.

5 SPARQL

L'interrogation d'une base RDF se fait très généralement[3] par le langage SPARQL – SPARQL Protocol and RDF Query Language – défini par le W3C pour la première fois en 2008 (4 ans après RDF), puis étendu en 2013. Ce langage est un langage d'interrogation de bases de connaissances – à comparer avec le langage d'interrogation des bases de données relationnelles SQL – adapté pour la structure sous forme de triplets des bases RDF. La dernière norme SPARQL 1.1 permet l'interrogation, nommée SPARQL [Harris and Seaborne, 2013], l'ajout et la suppression, désignés par SPARQL Update [Gearon et al., 2013], des données d'une base RDF. Une base qui peut être interrogée par des requêtes SPARQL via Internet propose un point d'accès – appelé *endpoint SPARQL* – où envoyer les requêtes via HTTP.

Une requête SPARQL possède un entête et, le plus souvent, un corps de requête. L'entête contient le mot-clé de la requête – `SELECT`, `ASK`, `INSERT`, *etc.* – avec les éléments demandés en résultat de la requête. Le corps de requête contient un ensemble de « triplets de recherche » . Dans les triplets de recherche, un élément sujet, relation ou objet peut être remplacé par une variable, sous la forme d'un nom de variable préfixé du caractère `?`. Un ensemble de triplets de recherche forme un *motif de graphe*. On appelle solution d'un motif de graphe un ensemble de données RDF qui forme un morphisme du motif. Ce morphisme substitue les variables du motif par des ressources de façon à ce que le motif de graphe

[2] http://jena.apache.org/

[3] Exception notable : Freebase, une des plus grosse bases du Linked Data (*cf.* section 6), utilise son propre langage, MQL mais est inaccessible suite à son rachat par Google.

et l'ensemble de données RDF soient identiques. Le corps d'une requête commence par le mot clé WHERE suivi d'un motif de graphe entre accolades ou de plusieurs motifs organisés par des opérateurs.

Interrogation La syntaxe d'une requête d'interrogation SPARQL est comparable à celle du langage SQL pour les bases de données relationnelles.

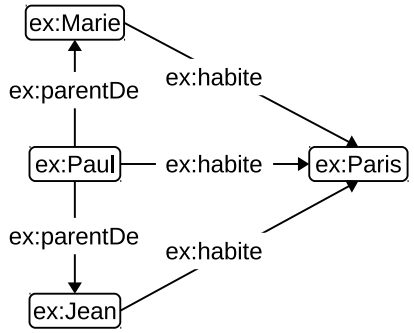


FIGURE 1.16 – Document RDF interrogé par les exemples de requêtes SPARQL suivantes.

Les requêtes d'interrogation SPARQL sont de 4 sortes différentes :

- Une requête SELECT extrait des résultats sous forme d'un tableau, comme avec SQL, contenant les ressources solutions à un motif de graphe donné et correspondantes aux variables présentes dans l'entête, tel que dans les exemples 1.17 et 1.18.

```
SELECT ?adresse
WHERE {
     ex:Jean ex:habite ?adresse .
}
```

FIGURE 1.17 – Exemple de requête SPARQL SELECT.

```
------------
| adresse  |
============
| ex:Paris |
------------
```

FIGURE 1.18 – Résultat de la requête SPARQL SELECT dans le document de la figure 1.16.

- Une requête ASK retourne une valeur booléenne s'il existe au moins une solution à un motif de graphe donné, tel qu'en exemple 1.19 (dont le résultat est true).

```
ASK
WHERE {
    ex:Paul ex:parentDe ?enfant .
}
```

FIGURE 1.19 – Exemple de requête SPARQL ASK.

- Une requête CONSTRUCT extrait les solutions à un motif de graphe donné sous la forme d'un nouveau graphe RDF, tel que dans les exemples 1.20 et 1.21. Dans cette requête, les variables sont utilisées dans un motif de graphe en entête pour générer de nouveaux triplets RDF à partir des résultats.

```
CONSTRUCT {
    ex:Julie ex:parentDe ?enfant .
} WHERE {
    ex:Paul ex:parentDe ?enfant .
}
```

FIGURE 1.20 – Exemple de requête SPARQL CONSTRUCT.

```
ex:Julie ex:parentDe ex:Jean .
ex:Julie ex:parentDe ex:Marie .
```

FIGURE 1.21 – Résultat de la requête SPARQL CONSTRUCT dans le document de la figure 1.16..

- Une requête DESCRIBE extrait tous les triplets de la base contenant les ressources données, tel que dans les exemples 1.22 et 1.23.

```
DESCRIBE ex:Paul
```

FIGURE 1.22 – Exemple de requête SPARQL DESCRIBE.

```
ex:Paul ex:habite ex:Paris .
ex:Paul ex:parentDe ex:Jean .
ex:Paul ex:parentDe ex:Marie .
```

FIGURE 1.23 – Résultat de la requête SPARQL DESCRIBE dans le document de la figure 1.16.

Les requêtes SPARQL disposent d'opérateurs et de filtres pour composer le corps des requêtes dont voici quelques exemples :

- Les 2 principaux opérateurs de disjonction :
  - L'opérateur UNION permet de définir deux motifs de graphes qui peuvent satisfaire séparément une requête, *c-à-d* il définit une disjonction entre deux motifs.
  - L'opérateur OPTIONAL permet de définir un sous-motif de graphe dont la résolution est optionnelle pour la satisfaction de la requête. L'utilisation de cet opérateur est comparable à la jointure gauche en SQL.
- Les filtres, définis par FILTER, vérifient que les variables satisfont certaines conditions exprimées par des fonctions ou des expressions telles que :
  - Les opérateurs d'égalité (=) et d'inégalité (<, >, ! =, <=, >=) définissent des conditions sur les valeurs des variables.
  - La fonction EXISTS retourne un booléen en fonction de l'existence d'un motif de graphe. Sa négation, introduite avec SPARQL 1.1, est NOT EXISTS.
  - Les fonctions isIRI, isBlank et isLiteral retournent un booléen selon la nature des ressources données, *i.e.* si la ressource est, respectivement, un URI, un nœud blanc ou un littéral.
  - La fonction datatype retourne l'URI tiré de la norme XML Schema Datatypes désignant le type d'un littéral donné.

Les filtres peuvent être organisés à l'aide des opérateurs d'égalité et d'inégalité ainsi que par les opérateurs logiques de conjonction (&&) et de disjonction (||).

D'un point de vue opérationnel, l'utilisation des opérateurs de disjonction rend la résolution des requêtes SPARQL co-NP-complète, d'après [Schmidt et al., 2010].

Modification La norme SPARQL 1.1 introduit, entre autres, les requêtes SPARQL Update d'ajout ou de suppression de triplets dans une base RDF. Cette norme définit deux types de requêtes centrées autour des mots-clés INSERT – exemples 1.25 et 1.26 – et DELETE – exemple 1.24 –. Ces requêtes permettent de créer de nouveaux triplets selon un patron de graphe. Elles peuvent utiliser une clause WHERE pour insérer des données générées à partir des données présentes.

```
DELETE {
    ex:Paul ex:habite ex:Paris .
    ?enfant ex:habite ex:Paris .
} WHERE {
    ex:Paul ex:parentDe ?enfant .
}
```

FIGURE 1.24 – Exemple de requête SPARQL DELETE.

```
INSERT {
ex:Paul ex:epoux ex:Julie .
ex:Julie ex:parentDe ?enfant .}
WHERE {
ex:Paul ex:parentDe ?enfant . } .
```

FIGURE 1.25 – Exemple de requête SPARQL INSERT.

```
ex:Paul ex:epoux ex:Julie .
ex:Julie ex:parentDe ex:Jean .
ex:Julie ex:parentDe ex:Marie .
```

FIGURE 1.26 – Triplets insérés dans le document de la figure 1.16 par la requête de la figure 1.25, représentés en figure 1.27.

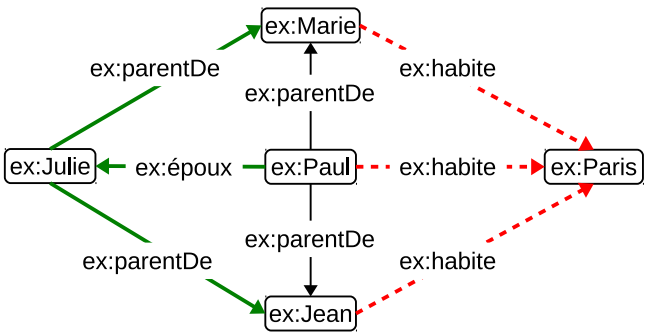


FIGURE 1.27 – Document RDF de la figure 1.16 avec les suppressions de la requête en figure 1.24 (en pointillés rouge) et les ajouts de la requête en figure 1.25 (en traits épais vert).

6 Linked Data

Le Web des données fournit le cadre pour la création de bases RDF et de services basés sur les données RDF. Le Linked Data est un ensemble de bases RDF libres d'accès et suivant plusieurs principes communs. Nous présentons ici les principes fondateurs du Linked Data et faisons son état des lieux.

Une base du Linked Data est accessible par un point d'accès HTTP interrogeable publiquement (*endpoint*) ou sous la forme d'une exportation du contenu de la base dans un fichier téléchargeable (un *dump* de la base). Toute base se réclamant du Linked Data doit suivre les règles suivantes :

- Toute ressource doit être identifiée par un URI HTTP pouvant être déréférencé.
- La page web correspondante à l'URI doit contenir des informations à propos de la ressource représentée par l'URI.
- La base doit contenir des liens vers d'autres bases du Linked Data. Le plus souvent ces liens se font par des relations d'équivalences, *i.e.* par owl:sameAs, entre ressources de bases différentes.

Bien que beaucoup de bases du Linked Data utilisent une ontologie exclusive, il existe également plusieurs ontologies communes à grand nombre de bases. Il est à noter que le langage RDFS est actuellement le plus utilisé, OWL l'est dans une moindre mesure. Les ontologies communes dans le Linked Data augmentent les possibilités d'utilisation et de fusion des données de bases différentes. Parmi ces ontologies communes, on trouve principalement les trois suivantes :

Dublin Core : *Dublin core*[4] est une ontologie définie en RDFS pour permettre la description de documents numériques, elle est maintenue par la *Dublin Core Metadata Initiative*.

FOAF : *Friend Of A Friend*[5] est une ontologie défini en OWL et conçue pour décrire sommairement et mettre en relation des personnes, des organisations et des documents.

SKOS : *Simple Knowledge Organization System*[6] est une ontologie définie en OWL qui permet de décrire des connaissances – *i.e.* des taxonomies – de façon complémentaire à OWL. Elle permet notamment la définition de relations de généralisation/spécialisation entre les individus.

Le W3C définit l'ontologie VoID [Cyganiak et al., 2011] pour renseigner les méta-données à propos des bases RDF dans les bases RDF. Cette ontologie[7] a été créée pour faciliter la découverte et le listage des bases du Web des données. Elle est conçue pour permettre la description des données internes à la base et ses liens avec d'autres bases. Elle utilise certaines relations de l'ontologie Dublin Core pour décrire les bases comme des documents numériques et de FOAF pour décrire notamment leurs sites. Elle permet également de

[4] identifiée par le domaine http://purl.org/dc/terms/
[5] identifiée par le domaine http://xmlns.com/foaf/0.1/
[6] identifiée par le domaine http://www.w3.org/2004/02/skos/core#
[7] utilisant le domaine http://rdfs.org/ns/void#

décrire les informations autour de la base telles que la licence de la base, l'ensemble des liens vers d'autres bases ou encore les détails techniques pour son accès. La description VoID d'une base RDF doit être placée dans un fichier suivant la syntaxe Turtle, nommé void.ttl, accessible à la racine du domaine de la base, tel que celui présenté en figure 1.28.

```
@prefix void: <http://rdfs.org/ns/void#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
ex:base a void:Dataset;
    dcterms:title "Triplets exemples";
    dcterms:description "Base formée des exemples précédents";
    foaf:homepage <http://www.exemple.com/rdf#>;
    void:triples 3;
    void:entities 4;
    .
```

FIGURE 1.28 – Exemple de document VoID pour un base contenant les triplets de la figure 1.16.

État du Linked Data Le Linked Data a entraîné beaucoup d'enthousiasme durant ses débuts et compte aujourd'hui plusieurs centaines de bases, disponibles librement sous forme de dumps – le plus souvent – ou accessibles par des endpoints. De part la nature ouverte du Linked Data, il n'existe pas d'autorité centrale chargée de répertorier et certifier les bases qui affirment appartenir au Linked Data. Une majorité des bases est actuellement répertoriée par l'Open Knowledge Fondation sur le site *datahub*[8].

L'ensemble des bases interconnectées du Linked Data forment le Linked Data Cloud, regroupant entre 500[9] et 1 000[10] ensembles de triplets[11] RDF, contenant au total environ 70 milliards[12] de triplets. Ces bases contiennent des données se rapportant à de nombreux domaines différents tels que la biologie, les données gouvernementales, géographiques, bibliographiques ou encore des connaissances encyclopédiques.

Aujourd'hui, plus de dix ans après l'apparition des premières normes du Web des données et sept ans après son lancement, il est possible de faire un état des lieux du Linked Data. La figure 1.29 montre graphiquement l'évolution du nombre et des interconnexions des bases du Linked Data. Cette représentation montre clairement la forte croissance du Linked Data Cloud en nombre de bases, de triplets et de liens inter-bases.

[8] http://datahub.io/group/lodcloud
[9] d'après http://lod-cloud.net/
[10] d'après http://stats.lod2.eu/
[11] bases accessibles via endpoint SPARQL ou dumps RDF
[12] d'après http://stats.lod2.eu/

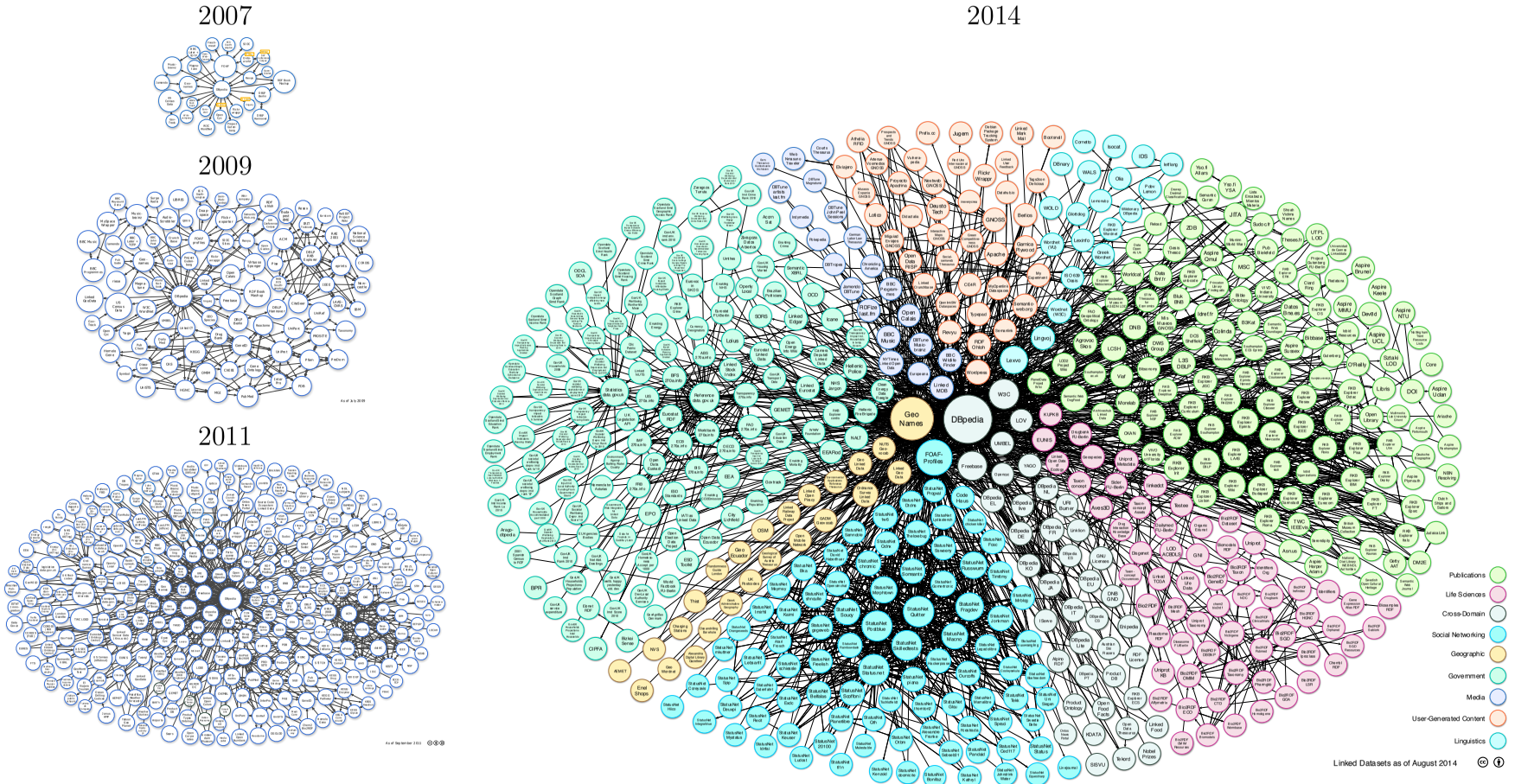


FIGURE 1.29 – Représentations du Linked Data Cloud au cours du temps : chaque cercle est une base et les couleurs désignent les différents domaines généraux (fournies par linkeddata.org).

Néanmoins, l'étude en détail des schémas de cette figure[13] met en évidence un biais dans la liste des bases représentées, probablement dû à un effet de « marketing » causé par l'enthousiasme de la communauté à montrer l'expansion du Linked Data. Un des exemples les plus frappants de ce biais est la catégorie *réseaux sociaux* apparue dans le schéma de 2014. La majorité de la centaine de bases de cette catégorie sont des instances de *StatusNet*, un système de microblog libre. Plusieurs éléments remettent en question la présence de ces bases dans le schéma du Linked Data : (i) aucune des adresses indiquées ne donne accès directement à un endpoint ou un fichier RDF, ces adresses sont celles des sites de microblog d'origine, (ii) il n'existe pas de moyen pour transformer le contenu d'un microblog en RDF, (iii) la taille des cercles des bases sur le schéma suggère qu'elles sont de tailles comparables ou supérieures à des bases comme *Geo Linked Data* comptant plusieurs millions de triplets alors que les sites concernés ne contiennent parfois que quelques dizaines de contenus de microblogging.

Ce genre d'incohérences entre la représentation et la réalité du Linked Data peut également être vu dans les données gouvernementales avec les quarante quatre bases *GovUk*, fractions des formatages en RDF des données gouvernementales du Royaume-Uni, qui sont toutes inaccessibles actuellement. La constatation de ces incohérences nous fait supposer que le Linked Data est actuellement plus proche de sa représentation de 2011 en termes de taille et de nombre de base. Malgré cela, l'effet marketing utilisé par certains membres de la communauté n'éclipse pas le fait que le Linked Data est en phase de croissance et que de plus en plus de bases sont publiées.

6.1 Bases majeures du Linked Data

Certaines de ces bases sont particulièrement connues et utilisées, elles sont liées à un très grand nombre d'autres bases, à un tel point qu'on peut les qualifier de « majeures » dans le Linked Data :

DBPedia : Le contenu de DBPedia [Lehmann et al., 2014] est issu du moissonnage des pages et des mises à jours de Wikipedia. Le système de moissonnage à l'origine de DBPedia crée des triplets principalement à partir des cadres de résumés formatés, les « infoboxes » visibles en figure 1.30, des pages de Wikipedia et en moindre proportion, du traitement du langage naturel des textes d'articles. La base DBPedia, accessible à travers son endpoint officiel[14] est mise à jour annuellement. La mise à jour de DBPedia est faite depuis la base DBPedia Live[15], miroir de DBPedia, qui est mise à jour directement à partir de Wikipedia. Les ajouts et suppressions des données des pages de Wikipedia sont extraites et appliquées à DBPedia Live quotidiennement et annuellement une mise à jour générale est extraite de DBPedia Live pour être appliquée à DBPedia. Un exemple de triplets extraits des pages de Wikipedia est donné en figure 1.30 et en figure 1.31.

DBPedia se base donc sur 3 éléments :

- Le système de moissonnage des pages de Wikipedia qui extrait les informations des éditions de page et des pages non-traitées sous forme de triplets.
- La base DBPedia Live qui subit en temps réel les modifications extraites par le système de moissonnage.

[13] en particulier du dernier en date, hébergé par http://lod-cloud.net/
[14] http://dbpedia.org/sparql
[15] http://live.dbpedia.org/sparql

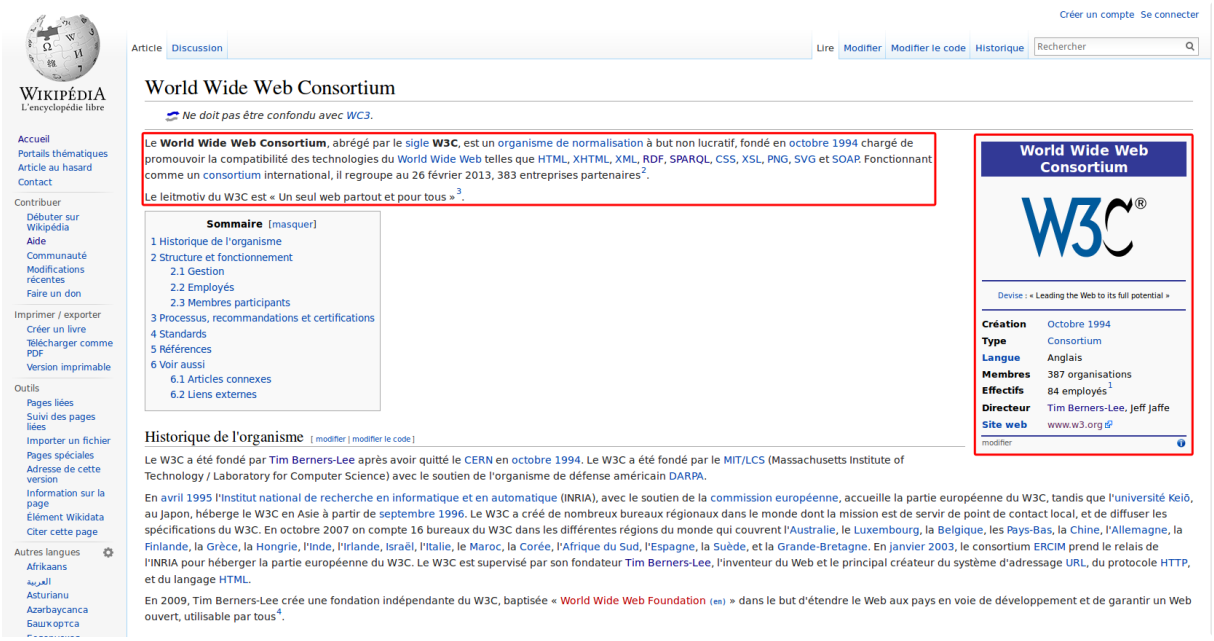


FIGURE 1.30 – Exemple de page Wikipedia avec les données extraites pour DBPedia encadrées en rouge.

```
dbpedia:World_Wide_Web_Consortium rdfs:label "World Wide Web Consortium"@fr ;
    dbponto:abstract "Le World Wide Web Consortium est un organisme..."@fr ;
    dbponto:abbreviation "W3C" ;
    dbpprop:formation "Octobre 1994"@fr ;
    dbponto:membership "387 organisations"@fr ;
    dbponto:motto "Leading the Web to Its Full Potential"@en ;
    dbpprop:numStaff 84 ;
    dbpprop:type Consortium ;
    dbpprop:leaderName dbpedia:Tim_Berners-Lee ;
    dbpprop:leaderTitle "Directeur"@fr ;
    dbpprop:website <http://www.w3.org> .
```

FIGURE 1.31 – Données extraites de la page Wikipedia en figure 1.30.

– La base DBPedia officielle, issue de DBPedia Live, renouvelée annuellement et augmentée par différents liens vers les autres bases du Linked Data.

La base DBPedia, en sa version internationale, contient plus de 583 millions de triplets pour 4,58 millions d'individus.

Freebase : Freebase[16] [Bollacker et al., 2008] est une base communautaire ouverte qui tire une partie de son contenu initial des autres bases du Linked Data (*e.g.* DBPedia). La base est en croissance constante par l'importation automatique d'autres bases ou d'autres sources. La qualité des données est assurée selon le même mode que pour Wikipedia : n'importe quel utilisateur peut modifier une donnée pour la corriger, une donnée restant longtemps sans correction peut être considérée comme correcte. Les contributeurs et correcteurs sont contrôlés par un groupe de « super-utilisateurs » choisis. Freebase ne propose pas d'endpoint SPARQL, elle utilise son propre langage

[16]Depuis son rachat par Google début 2015, la base Freebase est progressivement abandonnée, les personnes qui souhaite continuer à l'exploiter sont invitées à utiliser les bases qui héritent de son contenu : le Knowledge Graph de Google et le projet libre Wikidata [Erxleben et al., 2014].

de requête et met à disposition des dumps RDF. Freebase contient plus de 2 milliards de triplets concernant plus de 46 millions d'individus.

6.2 Analyses de l'état du Linked Data

Plusieurs travaux font une analyse de l'état du Linked Data selon différentes perspectives.

L'article [Demter et al., 2012] propose un framework d'analyse de l'état des bases et endpoints du Linked Data Cloud. D'après les résultats obtenus par ce framework[17] en février 2015, seuls 34% des bases répertoriées ont été accessibles ou téléchargeables sans erreurs. Le travail [Buil-Aranda et al., 2013] effectue une étude similaire en se concentrant sur les bases accessibles via un endpoint SPARQL. Il analyse différents attributs (accessibilité, interopérabilité, performance et mise à disposition) des 427 endpoints recensés sur le site *datahub*. Il donne une image mitigée de l'état du Linked Data dans lequel plus de 50% des endpoints ne supportent pas la dernière norme SPARQL 1.1 et 44% sont inaccessibles 95% du temps.

La conformité des bases qui se réclament du Linked Data Cloud avec les principes du Linked Data a été étudiée dans l'article [Hogan et al., 2012]. Ce travail a tenté d'évaluer la conformité avec les principes du Linked Data de 185 bases RDF en 2011, totalisant 947 millions de triplets. Les résultats de cette évaluation ont montré que les principes du Linked Data sont très inégalement suivis. Parmi les bases étudiées, 20% faisaient également une utilisation excessive des nœuds blancs, empêchant l'intégration de leurs données dans d'autres bases. 20% de ces bases utilisaient également en majorité des URIs ne pointant pas sur une page web existante.

La cohérence des données d'une base du Linked Data influe sur la qualité de toutes les bases auxquelles elle est liée par des relations d'équivalence. En effet, chaque relation d'équivalence entre ressources de bases différentes indique implicitement que les autres bases sont des sources d'informations acceptables et que les triplets qu'elles contiennent sont cohérents. L'article [Hogan et al., 2010] étudie la qualité et la cohérence générale de données moissonnées en 2009, totalisant 12 millions de triplets. Cet article a trouvé des incohérences principalement autour des types de données et des classes disjointes définies dans les ontologies. Pour contrer ces incohérences, différentes stratégies sont proposées pour les auteurs et les utilisateurs des bases. Parmi ces stratégies, les auteurs conseillent entre autres aux gestionnaires du Linked Data Cloud d'éviter l'utilisation des nœuds blancs et des conteneurs RDFS (*i.e.* rdf:Bag, rdf:Set, rdf:List, *etc.*)

Des travaux plus récents se sont concentrés sur la cohérence des données de la base DBPedia, qui est une des bases les plus connectées au reste du Linked Data Cloud[18]. Après avoir constaté 60 000 incohérences de différents types dans la base, l'article [Töpper et al., 2012] propose une méthode d'enrichissement de l'ontologie se concentrant particulièrement sur la résolution des incohérences dues aux définitions de domaines et de co-domaines. Le travail similaire présenté dans [Sheng et al., 2012] se concentre sur les définitions de classes, de types de données et de propriétés. Il étudie également les domaines et co-domaines, ainsi que les classes disjointes et propose quelques solutions pour l'amélioration de DBPedia. La plupart de ces incohérences ont les mêmes sources, elles proviennent à la

[17] affichés sur le site `http://stats.lod2.eu/`

[18] 27 millions de liens depuis DBPedia vers d'autres bases, 39 millions de liens depuis d'autres bases vers DBPedia, observables en figure 1.29

fois de l'extraction automatique des données de Wikipedia et les contributeurs (amateurs) de Wikipedia.

7 Concepts pour les bases RDF

Dans cette thèse, nous appelons *ressources nœuds* les ressources en sujets et en objets de triplets. Nous définissons en définition 1.1 deux ensembles pour tous documents RDF, *l'ensemble des ressources* et *l'ensemble des ressources nœuds*.

Définition 1.1 (Ensembles de ressources d'un ensemble de triplets RDF)**.** *Soit un ensemble de triplets RDF D.*

L'ensemble $\mathcal{R}_D$ contenant les sujets, relations et objets des triplets de D est appelé ensemble des ressources *de D.*

L'ensemble $\mathcal{N}_D$ contenant les sujets et objets des triplets de D est appelé ensemble des ressources nœuds *de D.*

L'ensemble $\mathcal{K}_D$ contenant les sujets et objets des triplets de D qui sont des nœuds blancs est appelé ensemble des ressources nœuds blancs *de D.*

La rédaction de cette thèse s'appuient également sur la représentation sous forme de graphe des documents RDF, de laquelle nous tirons deux appellations. Nous appelons *document RDF connexe* un document RDF dans lequel il y a un chemin connectant chaque ressource nœud du document à toutes les autres. En d'autres termes si le graphe qui représente le document RDF est un graphe connexe. Nous appelons *degré* d'une ressource nœud le nombre de triplets la contenant dans une base RDF. En d'autres termes, son degré dans le graphe qui représente la base RDF.

Le contenu d'un document RDF peut être vu comme la somme des descriptions des ressources qu'il contient. Dans la terminologie RDF, une description d'une ressource désigne en particulier l'ensemble des triplets autour d'une ressource nœud donnée, c'est-à-dire son *voisinage*. Ce voisinage ne se limite pas aux triplets contenant la ressource, il peut être étendu pour comprendre tous les triplets voisins jusqu'à un rang n. Ainsi, la définition 1.2 donne une fonction de voisinage au rang n.

Définition 1.2 (Fonction de voisinage)**.** *Soient un ensemble de triplets RDF B, $r \in \mathcal{N}_B$ et $n \in \mathbb{N}^*$.*

La fonction $\text{voisinage}_B^n(r) : \mathcal{N}_B \to B$ *telle que :*

- $\text{voisinage}_B^1(r) = \{t \mid t \in B, r \in \mathcal{N}_{\{t\}}\}$
- *Pour* $n > 1$, $\text{voisinage}_B^n(r) = voisinage_B^{n-1}(r) \cup \bigcup_{k \in \mathcal{N}_{\text{voisinage}_B^{n-1}(r)}} voisinage_B^1(k)$

est appelée fonction de voisinage *de r.*

Certains éléments de RDF et des langages RDFS et OWL font que l'extraction du voisinage d'une ressource doit être fait de façon non-naïve. Notre extraction du voisinage d'une ressource RDF, décrite dans [Maillot et al., 2014], traite les nœuds blancs d'une façon particulière. En RDF, les relations sont des relations binaires entre deux ressources représentant des concepts clairement identifiés. Néanmoins les nœuds blancs permettent de créer des ressources représentant des concepts non-identifiés, qui sont en pratique

uniquement utiles à la description d'autres ressources dans un document. Nous considérons que les nœuds blancs sont des éléments permettant d'utiliser des relations n-aires dans le langage RDF binaire. Ainsi, notre extraction du voisinage d'une ressource RDF traite les nœuds blancs comme des pseudo-relations.

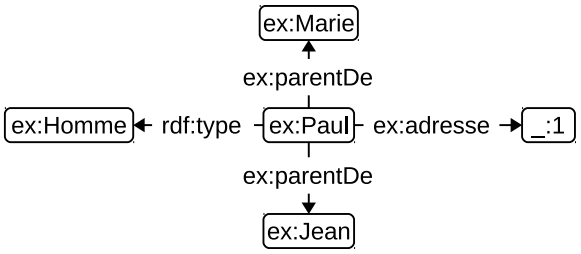


FIGURE 1.32 – Voisinage standard de ex:Paul au rang 1 dans le document de la figure 1.4.

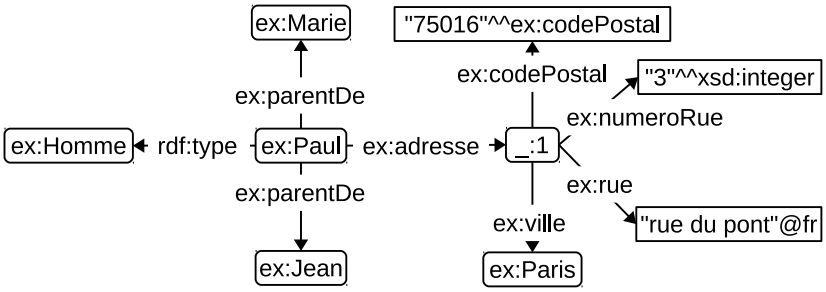


FIGURE 1.33 – Voisinage optimisé de ex:Paul au rang 1 dans le document de la figure 1.4.

Par exemple, le voisinage de ex:Paul de la figure 1.4 et représenté par une extraction du voisinage standard en figure 1.32 et notre extraction en figure 1.33. Le nœud blanc _:1 est utilisé pour regrouper les propriétés de l'adresse de ex:Paul autour d'une ressource qui est objet de la relation ex:adresse. Le voisinage optimisé en figure 1.33 permet d'avoir l'adresse complète de ex:Paul, contrairement au voisinage standard en figure 1.32.

8 Conclusion

Pour conclure ce chapitre, nous avons présenté le Web des données et fait l'état des lieux du Linked Data, son principal représentant. Il en ressort des problématiques de qualité des données, de validation et d'interrogation auxquelles nous proposons des solutions respectivement dans les chapitres 2, 3 et 4.

Chapitre 2

Évaluation de la qualité des mises à jour d'une base RDF par similarité

À leur création, les bases du Web des données contiennent un noyau de données initiales crées par un expert (ou un groupe d'experts) dans le domaine de connaissance de la base. Les mises à jour faites par les contributeurs font ensuite grandir le contenu de la base. Dans les bases du Linked Data ce sont les contributeurs issus des communautés formées autour des bases qui font ces mises à jour. Dans ces bases la qualité des données est particulièrement importante car elle influe sur la qualité des autres bases interconnectées du Linked Data. Pour maintenir la qualité des données, il faut alors évaluer la qualité des modifications des données et les filtrer selon le résultat de leur évaluation. De nombreuses propositions de méthodes d'évaluation ont été faites, mais aucune n'a réussi à s'imposer dans le Linked Data.

La méthode d'évaluation de la qualité classique consiste à se reposer sur l'ontologie. Cette méthode vérifie si le contenu des mises à jour respecte les définitions de l'ontologie pour décider si la mise à jour peut être appliquée. Lorsqu'une mise à jour ne respecte pas l'ontologie, on dit qu'elle est incohérente avec l'ontologie. Néanmoins, dans cette méthode il est possible que des mises à jour cohérentes avec l'ontologie créent des données qui vont à l'encontre du bon sens d'un observateur humain. Par exemple, (France, situé dans, Paris) est cohérente vis-à-vis de l'ontologie, la France et Paris étant tous les deux des lieux. L'ontologie n'empêche pas un pays d'être situé dans une ville ; néanmoins il est à l'encontre du bon sens de dire que la France est située dans Paris.

À l'opposé de cette situation, la méthode classique empêche parfois la création de données acceptées par les contributeurs malgré leurs incohérences avec l'ontologie. En effet, des données incohérentes sont régulièrement créées par la communauté lorsque l'ontologie de la base n'est plus adaptée aux données qu'elle contient. Par exemple, jusqu'à l'apparition des smartphones, on pouvait considérer les téléphones comme des appareils électroniques différents des ordinateurs (*e.g.* pas de système d'exploitation, entre autres). Avant que la distinction entre téléphones et smartphones ait été claire, il y a donc eu des téléphones ayant certains attributs réservés aux ordinateurs jusque-là. Lorsque les smartphones sont devenus suffisamment courants, ils ont été reconnus comme un genre à part entière dans les ontologies.

Ce chapitre présente notre méthode d'évaluation de la qualité de mises à jour d'une base RDF communautaire selon le critère de leur similarité avec les données existantes, schématisée en figure 2.1. Nous considérons que les données qui ont été ajoutées dans la base avant une mise à jour constituent un ensemble d'exemples de données de qualité

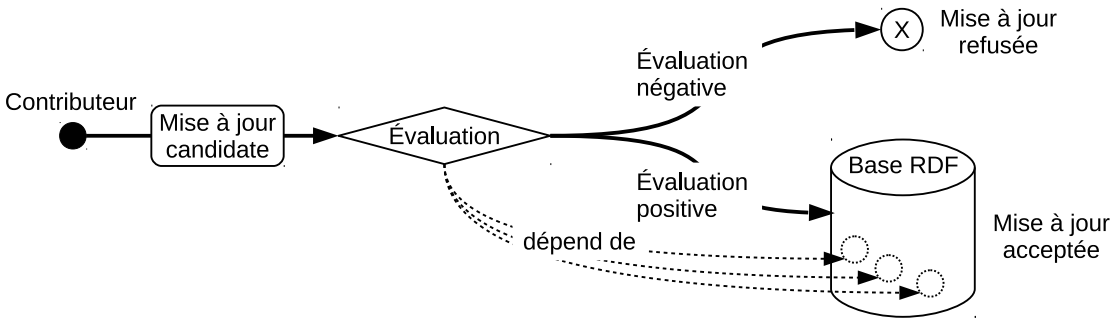


FIGURE 2.1 – Représentation schématique de notre méthode.

acceptable selon les critères de qualité de la base. Notre méthode considère les triplets de l'ontologie comme n'importe quels autres triplets de la base. Cette méthode a pour but d'évaluer les mises à jour de bases RDF afin de bloquer les mises à jour aberrantes (cohérentes ou non) dans leur contenu – ou simplement les signaler au contributeur – et autoriser les mises à jour similaires aux données déjà acceptées, mêmes si elles sont déclarées incohérentes. Pour cela notre méthode mesure la similarité des données d'une mise à jour avec le contenu de la base avant et après[1] application de la mise à jour. Nous présentons ensuite une méthode d'évaluation de la qualité d'une base seule. Cette méthode utilise notre évaluation de la qualité des mises à jour par similarité pour évaluer chaque partie de la base par rapport aux autres. Par exemple, pour une mise à jour ajoutant le triplet (France, situé dans, Paris) notre méthode va rechercher si les pays similaires à la France sont en général situé dans des villes similaires à Paris. Puisque des grands pays situés dans des villes n'existent pas, notre méthode rejettera la mise à jour.

En section 1 nous faisons un état de l'art des méthodes d'évaluation de la qualité des données RDF existantes. Nous définissons formellement les mises à jour et introduisons le concept de contexte de mise à jour en section 2. Nous présentons ensuite notre méthode d'évaluation de la qualité d'une mise à jour par similarité en section 3. Notre méthode d'évaluation est détaillée en 3 étapes : (3.1) la recherche des parties de la base à comparer avec la mise à jour, (3.2) la mesure de la similarité entre la mise à jour et ces parties de la base et (3.3) l'évaluation finale de la qualité de la base à partir de ces deux éléments. Enfin, nous présentons et discutons en section 4 les résultats de nos expérimentations.

Sommaire

[1] D'un point de vue théorique, car si la mise à jour est refusée elle ne peut être appliquée

1 État de l'art sur l'évaluation de la qualité des données RDF

La qualité d'une donnée est une notion subjective qui dépend de l'usage prévu de la donnée. Dans le Web des données, l'évaluation de la qualité de données RDF est généralement faite pour pouvoir filtrer les ajouts ou les modifications d'une base RDF. L'évaluation de la qualité est particulièrement importante dans le Linked Data où les bases interconnectées s'appuient mutuellement sur la qualité des autres pour étendre les informations qu'elles contiennent.

Les critères sur lesquels se basent les méthodes d'évaluation de la qualité des données RDF peuvent être résumés en deux questions :

« Qui ? » : En utilisant les informations sur la provenance des données, ces méthodes, schématisées en figure 2.2, évaluent la qualité d'une donnée RDF en fonction des méta-données qui l'entourent. Ces méta-données indiquent des détails supplémentaires sur les données tels que par exemple la date de soumission de la donnée, la fréquence de soumission de la source et la fréquence à laquelle sont modifiées ces données de la base. Elles sont souvent inspirées de méthodes utilisées dans les bases de données relationnelles accessibles sur le Web.

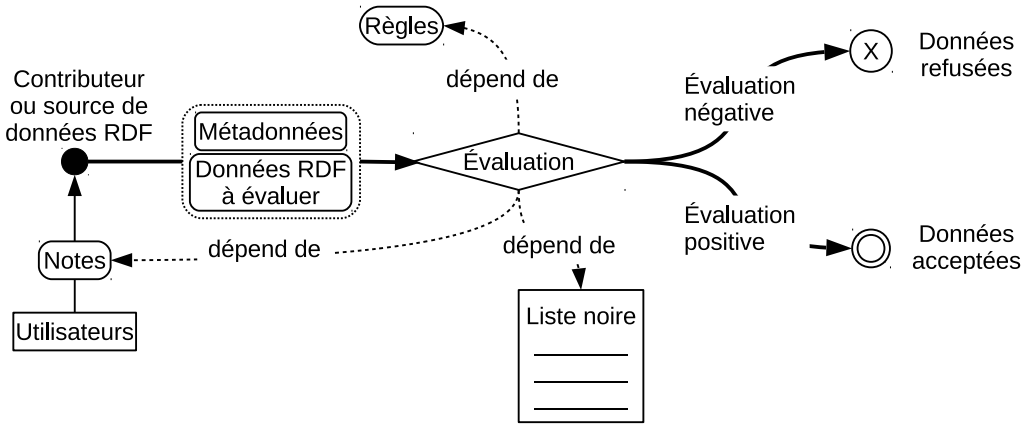


FIGURE 2.2 – Représentation schématique des méthodes basées sur les métadonnées.

« Quoi ? » : Ces méthodes, schématisées en figure 2.3, sont basées sur les données évaluées et disponibles pour leur évaluation. Elles utilisent généralement l'ontologie ou des ensembles de règles qui en dérivent.

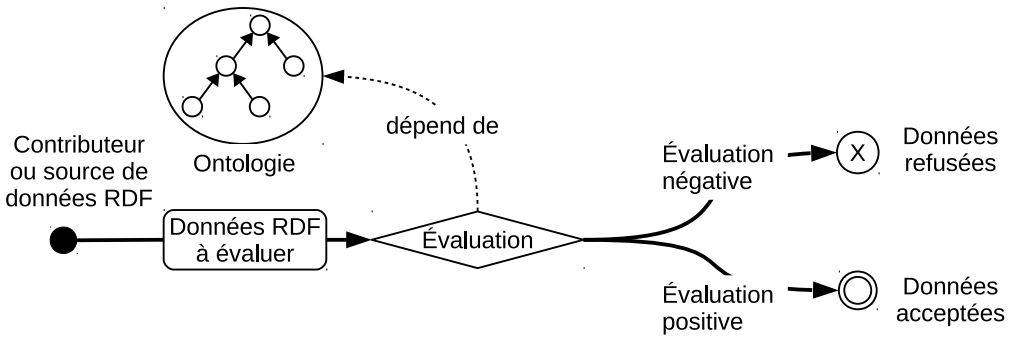


FIGURE 2.3 – Représentation schématique des méthodes basées sur les données.

Notre méthode, schématisée en figure 2.1, bien que ne s'appuyant pas sur l'ontologie ou autres règles, se situe dans la catégorie « Quoi ? ». Elle se concentre sur l'évaluation des mises à jour d'une base RDF en utilisant des sous-ensembles de la base.

Parmi les méthodes qui utilisent les métadonnées, nous trouvons les travaux suivants :

Le travail [Bizer and Cyganiak, 2009] propose le framework *WIQA* de filtrage des triplets affichés lors du parcours ou de la recherche de données RDF. Ce framework stocke des métadonnées associées à des graphes de triplets au sein d'une base. À partir de ces métadonnées et des données elles-mêmes, il propose un langage de règles pour qu'un utilisateur puisse définir la politique d'évaluation de la qualité des données. WIQA a été intégré dans un navigateur permettant de filtrer le contenu d'une base et d'obtenir une justification des choix de filtrage du navigateur.

Le travail [Bonatti et al., 2011] propose une étude des différentes façons d'évaluer la confiance en une source de données dans le Web des données. À partir de cette étude, il propose un framework qui utilise trois genres de mesures de confiance pour une information : (i) l'appartenance à une liste noire, (ii) l'autorité reconnue de la source et (iii) la centralité – *i.e.* indice PageRank, lié au nombre de liens en provenance ou en direction d'autres bases – de la source. La qualité des données est ici ramenée à la confiance accordée à sa source. Ce principe est couramment utilisé dans les évaluations des méthodes utilisant des métadonnées, elles sont dites basées sur la provenance des informations.

Remarquons que quelques travaux proposent des ontologies ou même des formats spécialisées pour représenter les métadonnées nécessaires à l'évaluation de la qualité dans les bases RDF. L'article [Schenk et al., 2011] propose une ontologie pour représenter la provenance de données RDF. Cette ontologie est faite pour faciliter la traçabilité des mises à jour et des triplets issus des raisonnements dans une base RDF. L'ontologie décrite dans l'article [Fürber and Hepp, 2011] est conçue pour décrire les propriétés de douze questions d'évaluation de la qualité d'une donnée RDF. Les auteurs fournissent une description de ces questions et de leurs variantes ainsi que des modèles de requête permettant d'insérer automatiquement les métadonnées dans une base. L'article [Jacobi et al., 2011] propose un framework pour l'évaluation de la confiance à accorder à des données RDF. Ce framework permet l'association à tous les éléments d'une base RDF une valeur de confiance à l'aide d'un format de donnée basé sur la syntaxe N3, [Berners-Lee and Connolly, 2011]. Cet article définit également le langage de règles *AIR* qui permet de définir des règles d'évaluation de la confiance à chaque triplet RDF d'une base.

Dans les méthodes qui utilisent les données et l'ontologie, nous trouvons les travaux suivants :

L'article [Guéret et al., 2012] étudie la qualité des données des bases interconnectées en fonction de mesures spécifiques aux graphes. La qualité des informations contenues dans un ou plusieurs documents RDF interconnectés est mesurée selon le degré et la centralité de ses ressources et selon l'apport des relations d'équivalences entre ressources.

Le framework *Sieve* est proposé dans [Mendes et al., 2012] pour le transfert des données d'une base vers une autre. Sieve permet d'encadrer la fusion des données grâce à des fonctions d'évaluations configurables en fonction des valeurs des propriétés, *e.g.* entre deux propriétés de valeurs similaires, prendre la plus précise.

L'article [Kontokostas et al., 2014] propose une méthode d'évaluation de la qualité des données d'une base par un ensemble de tests tirés de l'ontologie. Cette méthode utilise des cas-tests associés à différentes définitions ontologiques. Ces cas-tests vérifient des contraintes d'intégrité pour les bases RDF concernant les définitions de l'ontologie.

Les contraintes d'intégrité sont tirées des ontologies OWL et permettent de vérifier le respect des domaines, co-domaines, cardinalités, caractères symétriques ou réflexifs des relations et les disjonctions des classes. À ces contraintes d'intégrité tirées de l'ontologie, les auteurs ont ajouté des tests selon des critères de qualité tirés de la communauté de DBPedia. Ces tests communautaires testent des contraintes « de bon sens » qui ont été identifiées lors d'un sondage auprès de la communauté. Parmi ces contraintes de bon sens on trouve par exemple : une date de naissance doit être antérieure à la date de décès, ou encore les expressions régulières à respecter pour les littéraux de certains types.

Dans une catégorie d'approche différente, les auteurs de [Kontokostas et al., 2013] proposent d'utiliser la production participative pour évaluer la qualité des données d'une base. L'approche qu'ils proposent est utilisée sur DBPedia en proposant des micro-tâches rémunérées à des utilisateurs de Amazon Mechanical Turk pour qu'ils évaluent si une description de ressource contient des erreurs. Bien que dépendante du sérieux des utilisateurs, cette approche utilise la richesse des communautés du Web pour l'évaluation de la qualité de données RDF.

2 Mises à jour RDF et leurs contextes

Nous introduisons ici quelques définitions pour formaliser les *mises à jour RDF* et leurs *contextes* associés dans la base. Notons que nous désignons toujours les données d'une base RDF sans que les modifications d'une mise à jour ne lui soient appliquées, on parle de mise à jour « candidate ». Toute mise à jour d'une base RDF peut être vue comme étant composée de deux sections : une *section d'ajout* qui contient ce que la mise à jour ajoute à la base et une *section de suppression* qui contient ce que la mise à jour supprime dans la base ; l'une des deux sections pouvant être vide.

Les exemples de ce chapitre sont des mises à jour candidates à DBPedia, d'où est extrait le document RDF représenté en figure 2.4. Ce document parle de villes liées à des boissons et de quelques personnes qui leurs sont liées.

Définition 2.1 (Mise à jour RDF)**.** *Une mise à jour RDF candidate, pour être appliquée à une base RDF non vide B, est un couple d'ensembles de triplets RDF (A, R) tels que :*

- *A est appelé* section d'ajout*, avec $A \cap B = \emptyset$;*
- *$\mathcal{N}_A \cap \mathcal{N}_B \neq \emptyset$;*
- *R est appelé* section de suppression*, avec $R \subseteq B$;*
- *$A \cap R = \emptyset$;*
- *$A \cup R \neq \emptyset$;*
- *$A \cup R$ est un document RDF connexe.*

Une mise à jour RDF qui ajoute des informations à une base doit apporter de nouveaux éléments liés à des données déjà existantes. Une mise à jour qui supprime des informations peut uniquement supprimer des données déjà existantes dans la base. Les sections d'ajout et de suppression ne peuvent pas contenir de triplets en commun afin que l'ordre d'application de la suppression et de l'ajout dans la base n'ait pas de conséquence. L'ensemble

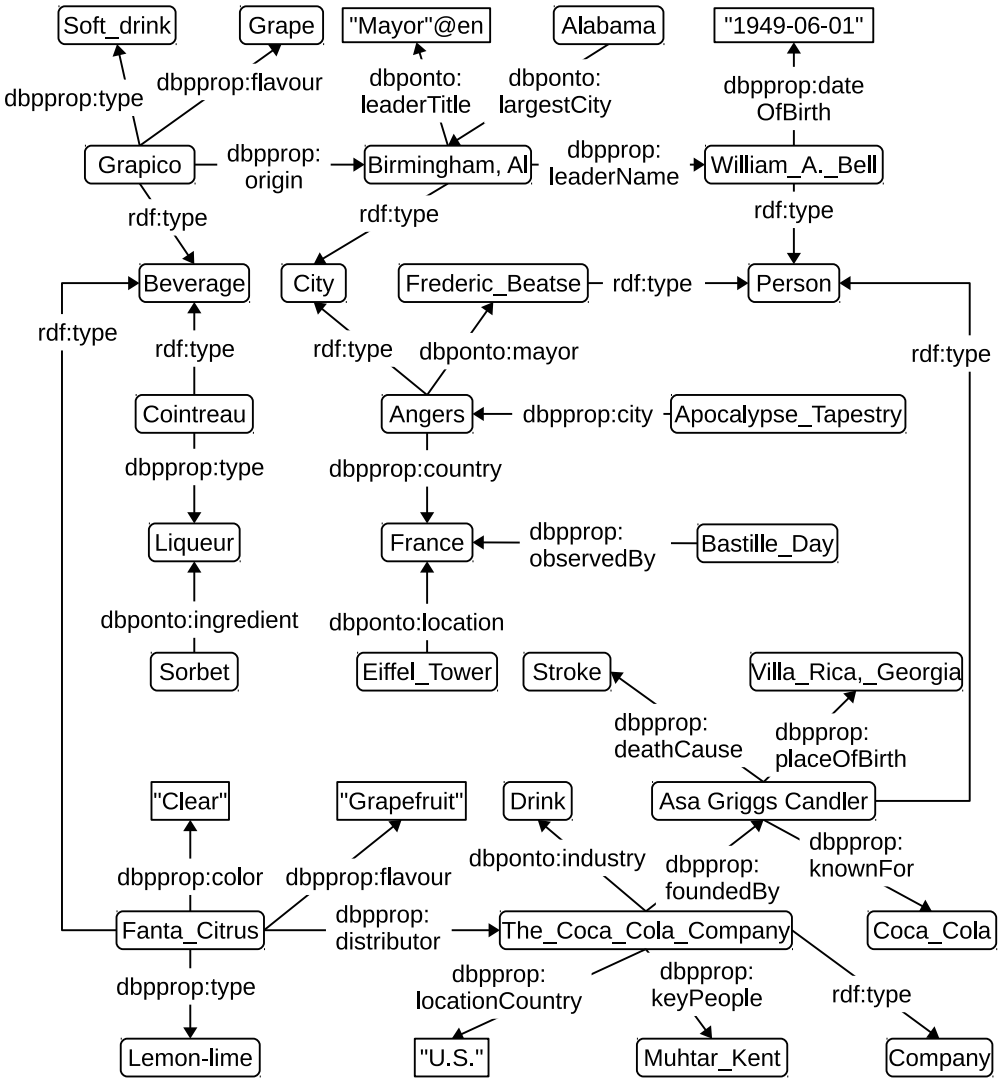


FIGURE 2.4 – Document RDF extrait de DBPedia.

de la mise à jour doit être non-vide (même si une des deux sections de la mise à jour peut être vide) et doit former un document connexe (autrement on la sépare en deux mises à jour distinctes).

D'après cette définition, une mise à jour peut apporter de nouvelles informations à propos des ressources existantes, les modifier ou les supprimer. Il est également possible de créer de nouvelles ressources, tant qu'elles sont liées à un élément de la base.

De la définition 2.1 nous pouvons classer les mises à jour RDF en trois catégories :

les *mises à jour d'ajout*, définies par $A \neq \emptyset$ et $R = \emptyset$,

les *mises à jour de modification*, définies par $A \neq \emptyset$ et $R \neq \emptyset$,

et les *mises à jour de suppression* définies par $A = \emptyset$ et $R \neq \emptyset$.

Notre but est d'évaluer une mise à jour par rapport à une base RDF en effectuant une comparaison de la mise à jour par rapport à la base. Pour pouvoir comparer les données d'une mise à jour à une base RDF, nous utilisons les voisinages des ressources de la mise à jour dans la base (définition 1.2) « avant » et « après » l'application de la

mise à jour. Ces voisinages forment les *contextes* des ressources que la mise à jour modifie. Plus précisément, les contextes d'une mise à jour sont obtenus grâce aux voisinages des sections d'ajout et de suppression (définition 2.2).

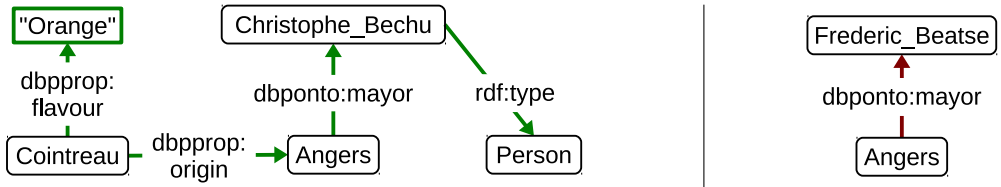
Définition 2.2 (Contextes de mise à jour RDF)**.** *Soit un base RDF non vide B, une mise à jour RDF $u = (A, R)$ candidate à B et $n \in \mathbb{N}$ un rang de voisinage.*

Les contextes d'une mise à jour *u candidate à B sont les deux ensembles de triplets RDF I_u et F_u avec :*

- *I_u appelé le* contexte initial *de u dans B tel que $I_u = \{t | t \in voisinage_B^n(r),\ r \in \mathcal{N}_{A \cup R}\}$;*
- *F_u appelé le* contexte final *de u dans B tel que $F_u = I_u \cup A \setminus R$.*

Le contexte initial représente l'état initial de la base autour des ressources concernées par la mise à jour candidate. Le contexte final représente l'état théorique du contexte de la mise à jour – et donc des triplets de la base – si celle-ci était appliquée.

Exemple 1 (Mise à jour u_1)**.** *La figure 2.5 est une représentation graphique de la section d'ajout et de la section de suppression de u_1. u_1 est une mise à jour fictive candidate à la base DBPedia, plus précisément au sous-ensemble représenté en figure 2.4, d'une part en ajoutant des informations à propos du goût et de l'origine de la liqueur « Cointreau » et d'autre part mettant à jour les informations à propos du nouveau maire de la ville d'Angers, d'où vient la liqueur.*



A_1 section d'ajout de u_1.

R_1 section de suppression de u_1.

FIGURE 2.5 – Représentation graphique de $u_1 = \{A_1, R_1\}$ candidate à DBPedia (éléments ajoutés en vert, éléments supprimés en rouge).

Exemple 2 (Contextes de u_1)**.** *Le contexte initial de u_1 avec un rang de voisinage de 1, noté I_{u_1}, est graphiquement représenté en figure 2.6 et le contexte final de u_1 avec un rang de voisinage de 1, noté F_{u_1}, est graphiquement représenté en figure 2.7. Ces contextes sont tous les deux tirés de DBPedia, et plus précisément de sa partie représentée en figure 2.4.*

3 Évaluer la qualité par similarité

Pour notre approche, nous considérons une mise à jour *candidate* à la modification d'une base. Ce n'est qu'après une évaluation positive – qui fait l'objet de ce chapitre – que la mise à jour est acceptée pour être appliquée.

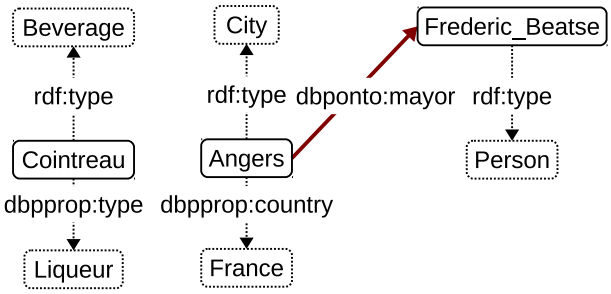


FIGURE 2.6 – Contexte initial I_{u_1} de u_1 (Éléments de la figure 2.4 en pointillés)

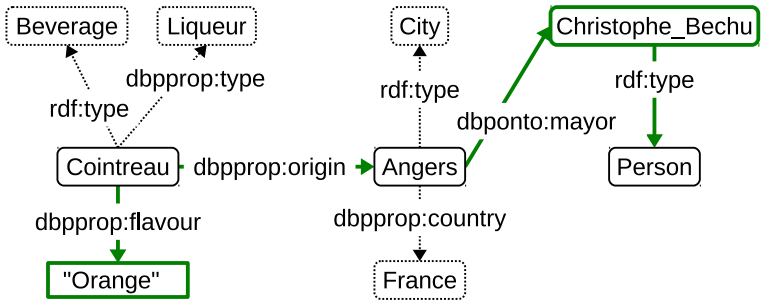


FIGURE 2.7 – Contexte final F_{u_1} de u_1 (Éléments de la figure 2.4 en pointillés)

Nous évaluons la qualité d'une mise à jour à partir de sa similarité avec les données de la base. Si les données d'une mise à jour ressemblent fortement aux données de la base, alors nous considérons qu'elle est en accord avec la base. Autrement dit, si on voit les données de la base comme des mises à jour acceptées précédemment, une mise à jour qui leur est similaire est acceptable. Plus précisément, nous considérons qu'une mise à jour est cohérente avec une base si on peut trouver suffisamment de parties de la base qui soient suffisamment similaires avec les contextes de la mise à jour. Nous procédons en 3 étapes :

(i) Nous recherchons dans la base des parties de la base comparables aux contextes de la mise à jour.

Ces parties, appelées *références*, sont des ensembles de triplets contenant des éléments en commun avec les contextes de la mise à jour.

(ii) Nous quantifions la similarité entre chaque partie de la base et les contextes de la mise à jour.

Cette quantification se base sur le nombre d'éléments en commun entre une partie de la base et les contextes de la mise à jour.

(iii) Nous concluons sur la qualité de la mise à jour.

À partir des mesures de similarité entre les contextes d'une mise à jour et différentes parties, nous pouvons conclure si la mise à jour est similaire au contenu de la base et est donc acceptable.

Nous faisons remarquer que notre méthode évalue les mises à jour d'ajout et de modification, les mises à jour de suppression pure ne peuvent pas être évaluées par notre approche. Nous évaluons une mise à jour candidate à une base RDF en considérant les données existantes comme un ensemble d'ajouts et de modifications ayant été acceptés

précédemment. La suppression de données ne laisse pas de traces dans la base, elle ne peut donc pas être utilisée pour notre évaluation.

3.1 Trouver des références dans la base

Deux ensembles de triplets RDF peuvent être comparés si leurs ressources nœud sont liées de façon *comparable* à – au moins – une ressource commune.

Définition 2.3 (Ensembles de triplets RDF comparables)**.** *Soient deux ensembles connexes et non-vides de triplets RDF G et H, $n \in \mathbb{N}^*$ tel que $n \geq 1$ et $p \in \mathbb{R}^{*+}$ tel que $p \geq 1$.*

L'ensemble G est comparable à H s'il existe une application $f : G \to H$ et si :
$$\begin{cases} |\mathcal{R}_G \cap \mathcal{R}_H| \geq n \\ |G| \leq p \times |H| \\ p \times |G| \leq |H| \end{cases}$$
On appelle n le nombre de ressources communes minimum *et p le* rapport de taille maximum*.*

En théorie des graphes on pourrait dire qu'un ensemble de triplets RDF est comparable à un autre s'il existe une transformation de l'un vers l'autre, qu'ils ont un nombre minimum de ressources communes et que le rapport entre leurs tailles est limité. La limitation du rapport de taille entre les ensembles comparables autorise des ensembles qui sont de tailles proches, voire égale, si p est proche de 1, ou qui ont des grands écarts de taille si p est grand.

Propriété 2.4. *Soient deux ensembles de triplets RDF G et H, tous deux connexes et non-vides.*

Si G est comparable à H, alors H est comparable à G.

Démonstration. Soit l'ensemble de triplets G comparable à l'ensemble de triplets H pour n nombre de ressources communes minimum et p rapport de taille maximum.

Il existe donc un triplet $t_G \in G$ tel que $\mathcal{R}_{\{t_G\}} \cap \mathcal{R}_H \neq \emptyset$. Prenons l'application $f : H \to \{t_G\}$ telle que $\forall t_H \in H$, $f(t_H) = t_G$. Donc H est comparable à G pour n nombre de ressources communes minimum et p rapport de taille maximum. □

Exemple 3. *En figure 2.8, l'ensemble de triplet G contient deux triplets qui décrivent la relation d'amitié entre* Pierre *et* Paul *et la relation professionnelle entre* Paul *et* Jacques*.*

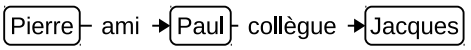
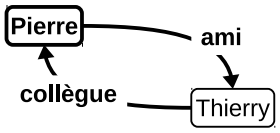


FIGURE 2.8 – Ensemble de triplets G.

*L'ensemble de triplets H_1, en figure 2.9, qui possède trois ressources communes avec G (*Pierre*,* collègue *et* ami*) et qui est de la même taille que G, est comparable à G selon l'application f_1.*

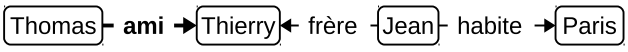
L'ensemble de triplets H_2, en figure 2.10, est comparable à G de part l'application f_2, la ressource commune ami *et un rapport de taille maximum de* 1.5*. On notera que l'application f_3 est également valable pour conclure que H_2 est comparable à G. La section suivante permet de connaître le niveau de similarité de cette comparaison.*

D'après la définition 2.3 les contextes d'une mise à jour peuvent être considérés comparables à la base toute entière si on n'exige qu'une ressource commune et qu'on ne limite



f_1 : (Pierre, ami, Paul) $\mapsto$ (Pierre, ami, Thierry)
(Paul, collègue, Jacques) $\mapsto$ (Thierry, collègue, Pierre)

FIGURE 2.9 – Ensemble de triplets H_1, comparable à G par l'application f_1 (ressources communes à G et H_1 en gras).



f_2 : (Pierre, ami, Paul) $\mapsto$ (Thomas, ami, Thierry)
(Paul, collègue, Jacques) $\mapsto$ (Jean, frère, Thierry)
f_3 : (Pierre, ami, Paul) $\mapsto$ (Thomas, ami, Thierry)
(Paul, collègue, Jacques) $\mapsto$ (Jean, habite, Paris)

FIGURE 2.10 – Ensemble de triplets H_2, comparable à G par l'application f_2 et par f_3 (ressources communes à G et H_2 en gras).

pas le rapport de taille entre ensembles comparables. Dans la pratique, les contextes d'une mise à jour sont généralement de taille nettement inférieure à celle de la base. Il est plus pertinent pour notre méthode de manipuler des ensembles comparables dans la base qui ont une taille de même ordre de grandeur.

Les sous-ensembles à comparer à la mise à jour de la base que nous recherchons particulièrement sont des ensembles de triplets dont la mise en relation des ressources est la plus similaire à celle des contextes et qui partagent avec la mise à jour au moins une ressource commune. Ces sous-ensembles sont les *références* pour notre évaluation.

En comparant la mise à jour aux références, notre méthode vérifie si les parties de la base qui partagent une partie de ses ressources, lui sont suffisamment similaires. Donc, en pratique, pour que notre comparaison soit la plus efficace, il est préférable d'utiliser les références de petite taille et avec beaucoup de ressources en commun avec les contextes.

Définition 2.5 (Référence d'une mise à jour)**.** *Soient une mise à jour u, candidate pour la base RDF non vide B et ses contextes I_u et F_u.*

Une référence *D de u dans B est telle que $D \subseteq B$ et D comparable à $I_u \cup F_u$.*

Comparer une mise à jour à une base entière signifie comparer les éléments des contextes initiaux et finaux de la mise à jour à chaque partie de la base comparable à la mise à jour. Nous appelons *références* ces parties de la base dépendantes de la mise à jour.

Exemple 4 (Références D_1 et D_2)**.** *De la base DBPedia (figure 2.4), nous extrayons deux références pour u_1, avec 4 ressources communes minimum et un rapport de taille maximum de 2, notées D_1 et D_2 et représentées en figure 2.11 et figure 2.12. D_1 et D_2 contiennent respectivement 7 et 5 ressources en communes et ont respectivement un rapport de taille maximum de 1 et 1.6 avec $I_{u_1} \cup F_{u_1}$. Plus généralement, D_1 suit le même modèle que u_1 avec une boisson liée à une ville qui est liée à une personne, alors que D_2 concerne une boisson liée à une entreprise qui est liée à une personne.*

Ces deux références font partie des sous-ensembles de DBPedia suffisamment similaires aux contextes de u_1 et qui contiennent le plus de ressources en commun. Elles ont été extraites de DBPedia à l'aide de la méthode décrite en chapitre 5 section 1.

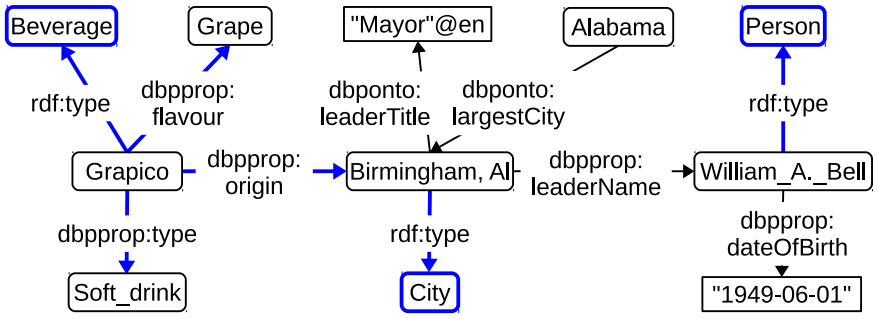


FIGURE 2.11 – Partie D_1 des données en figure 2.4, en tant que *référence* pour u_1 en figure 2.5 (Éléments en commun avec u_1 en bleu).

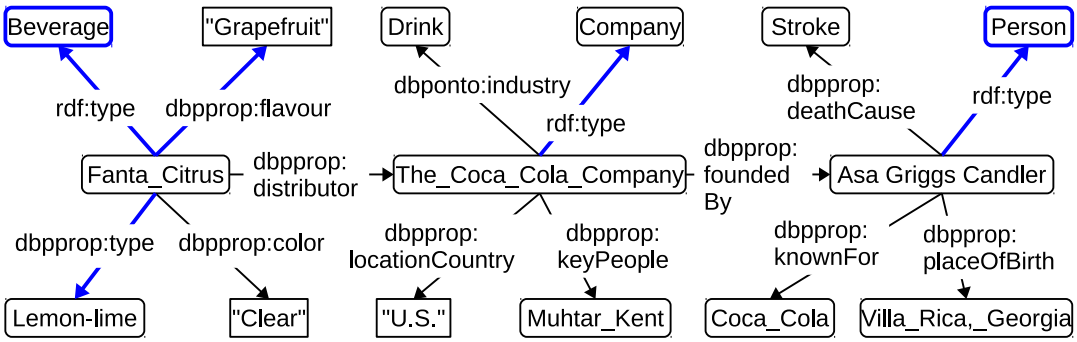


FIGURE 2.12 – Partie D_2 des données en figure 2.4, en tant que *référence* pour u_1 en figure 2.5 (Éléments en commun avec u_1 en bleu).

3.2 Mesurer la similarité

Pour évaluer la similarité entre deux ensembles de triplets RDF, il y a plusieurs mesures possibles.

Une des mesures les plus utilisées, qui n'est pas spécialement dédiée à RDF, est le *coefficient de similarité de Jaccard* [Jaccard, 1901]. Il mesure la similarité entre deux ensembles finis d'éléments en fonction du nombre d'éléments communs uniques par rapport au nombre total d'éléments uniques dans les deux ensembles. Le coefficient de similarité de Jaccard est une mesure naïve mais facilement implémentable ; sa complexité de calcul est polynomiale.

Une autre mesure de la similarité entre deux ensembles de triplets RDF est la *distance d'édition de graphe*, qui peut être utilisée grâce à la représentation sous forme de graphe des ensembles de triplets RDF. La distance d'édition de graphe est une mesure de la similarité basée sur le nombre d'opérations d'édition nécessaires pour transformer un graphe (*e.g.* un document RDF) en un autre ; son calcul est NP-difficile, [Gao et al., 2010].

Pour ce chapitre nous choisissons, pédagogiquement, de quantifier les similarités entre deux ensembles de triplets RDF avec une mesure de Jaccard adaptée au Web des données – de façon analogue à [Mottin et al., 2014] – en considérant non seulement les ressources RDF seules mais aussi les extrémités de triplets (*i.e.* les couples de ressources (sujet, prédicat) ou (prédicat, objet) apparaissant dans les triplets) comme éléments des ensembles à comparer.

Définition 2.6 (Mesure de similarité pondérée de Jaccard)**.** *Soient un document RDF non vide G, son ensemble de ressources $\mathcal{R}_G$ et $E^2_G = \{(e_1, e_2) \mid \exists r \in \mathcal{R}_G\ (e_1, e_2, r) \in G$ ou $(r, e_1, e_2) \in G\}$ l'ensemble des extrémités de triplets de G.*

Soit H un document RDF non-vide.

Soit une fonction poids : $\mathcal{R}_G \cup \mathcal{R}_H \cup E^2_G \cup E^2_H \to \mathbb{R}^*_+$.

La fonction $simJaccard(G, H) = \frac{\sum_{e \in (R_G \cap R_H) \cup (E^2_G \cap E^2_H)} poids(e)}{\sum_{e \in (R_G \cup R_H) \cup (E^2_G \cup E^2_H)} poids(e)}$ *est appelée* mesure de similarité pondérée de Jaccard.

Dans la définition 2.6 nous obtenons une mesure dépendante des ressources et des extrémités de triplets communes entre les deux ensembles de triplets RDF. En manipulant la fonction de pondération *poids*, il est possible de donner plus d'importance à certains éléments communs. Par exemple, nous pouvons donner plus d'importance aux extrémités de triplets qu'aux ressources seules, ou plus d'importance aux classes qu'aux individus en leurs accordant un poids plus grand.

Exemple 5. *Entre* I_{u_1}*, représentée en figure 2.6, et* D_1*, représentée en figure 2.11, on a*

- *5 ressources communes* $(R_{I_{u_1}} \cap R_{D_1})$ *:* {Beverage, rdf:type, dbpprop:type, City, Person }*, et*
- *3 extrémités communes* $(E^2_{I_{u_1}} \cap E^2_{D_1})$ *:* {(rdf:type, Beverage), (rdf:type, City), (rdf:type, Person)}*.*

On a

- *26 ressources au total* $(R_{I_{u_1}} \cup R_{D_1})$ *:* {Cointreau, dbponto:Beverage, rdf:type, City, Frederic_Beatse, dbponto:mayor, Cointreau, Angers, Person, dbpprop:type, dbpprop:country, France, Grape, "Mayor"@en, Alabama, dbpprop:flavour, dbponto:leaderTitle, dbponto:largestCity, Grapico, dbpprop:origin, Birmingham,_Al, dbpprop:leaderName, William_A._Bell, dbpprop:dateOfBirth, Soft_drink, "1949-06-01" }*, et*
- *29 extrémités de triplets au total* $(E^2_{I_{u_1}} \cup E^2_{D_1})$ *:* {(Cointreau, rdf:type), (rdf:type, Beverage), (Angers, rdf:type), (rdf:type, City), (Angers, dbponto:mayor), (dbponto:mayor, Frederic_Beatse), (Frederic_Beatse, rdf:type), (rdf:type, Person), (Cointreau, dbpprop:type), (dbpprop:type, Liqueur), (Angers, dbpprop:country), (dbpprop:country, France), (Grapico, rdf:type), (Grapico, dbpprop:origin), (Grapico, dbpprop:flavour), (dbpprop:flavour, Grape), (dbpprop:origin, Birmingham,_Al), (Birmingham,_Al, dbponto:leaderTitle), (dbponto:leaderTitle, "Mayor"@en), (Alabama, dbponto:largestCity), (dbponto:largestCity, Birmingham,_Al), (Birmingham,_Al, dbpprop:leaderName), (dbpprop:leaderName, William_A._Bell), (William_A._Bell, rdf:type), (Grapico, dbpprop:type), (dbpprop:type, Soft_drink), (Birmingham,_Al, rdf:type), (William_A._Bell, dbpprop:dateOfBirth), (dbpprop:dateOfBirth, "1949-06-01")}

Ainsi, en suivant la définition 2.6 pour calculer la mesure de similarité entre I_{u_1} *et* D_1 *avec une fonction de pondération donnant un poids de* 1 *à toutes les ressources et un poids de* 2 *à toutes les extrémités de triplets on obtient* $simJaccard(I_{u_1}, D_1) = \frac{5+3\times2}{26+29\times2} = 0.131$.

En utilisant la même fonction de pondération, on obtient :

- *la mesure de similarité entre* F_{u_1} *et* D_1*,* $simJaccard(F_{u_1}, D_1) = 0.139$*,*
- *la mesure de similarité entre* I_{u_1} *et* D_2*,* $simJaccard(I_{u_1}, D_2) = 0.247$*,*
- *la mesure de similarité entre* F_{u_1} *et* D_2*,* $simJaccard(F_{u_1}, D_2) = 0.236$*.*

Dans cet exemple, on choisit d'utiliser une fonction de pondération qui donne plus d'importance aux extrémités de triplets communes qu'aux ressources seules communes. Nous choisissons cette fonction pour mettre l'accent sur les similarités de mises en relation de ressources. D'après nous, les extrémités de triplet communes entre ensembles de triplets montrent que les ensembles contiennent des ressources différentes liées aux mêmes ressources par les mêmes relations, donc les deux ensembles de triplets concernent bien des ressources similaires.

La mesure de Jaccard évalue la similarité de deux ensembles en fonction de leurs éléments communs. Notre mesure de similarité dépend d'abord des éléments les plus atomiques des documents RDF, les ressources RDF. Néanmoins, la similarité basée sur les ressources RDF ne retranscrit pas la similarité des mises en relations des ressources au sein de deux documents RDF, *c-à-d* elle évalue si les mêmes ressources sont présentes mais pas si elles sont utilisées de la même façon. Pour permettre cette évaluation des mises en relation, notre mesure de similarité prend donc en compte les extrémités de triplets (associations deux à deux de ressources RDF).

Nous remarquons que les triplets complets ne sont pas parmi les éléments évalués dans notre mesure de similarité. Dans le cadre de notre évaluation, si une référence a des triplets entiers en commun avec le voisinage d'une mise à jour alors c'est que nous comparons le voisinage de la mise à jour avec lui-même (ou au moins une partie de lui-même). Ce cas particulier signifie que la mise à jour s'utilise elle-même comme référence durant son évaluation, ce qui risque de donner une évaluation biaisée. Notre méthode ignore donc toujours les triplets communs pour limiter ce genre de situation.

3.3 Évaluer la qualité d'une mise à jour

Pour évaluer la qualité d'une mise à jour, on mesure d'abord sa similarité par rapport à chacune des références obtenues.

Définition 2.7 (Mise à jour et référence positivement similaires)**.** *Soient une mise à jour u candidate à une base RDF non vide B, ses contextes I_u et F_u, et une référence D associée à u dans B.*

La mise à jour u et la référence D sont dites « positivement similaires » si simJaccard(F_u, D) − simJaccard$(I_u, D) > 0$.

On dit qu'une mise à jour et une référence sont positivement similaires si les données de la base autour de la mise à jour – ses contextes – sont plus similaires à la référence après l'application de la mise à jour qu'avant. La valeur de similarité – et donc la mesure choisie, Jaccard ou autre – entre contexte et référence seule n'a pas d'importance dans notre méthode, seul le signe de la différence entre l'état final et initial indique si la mise à jour apporte des informations similaires à ce qui est connu ou non.

Les références nous servent de points de comparaisons entre la mise à jour et la base. Notre évaluation de la qualité de la mise à jour se fait selon sa similarité avec un certain nombre de références.

Définition 2.8 (Évaluation de la qualité d'une mise à jour par similarité)**.** *Soient une mise à jour u, candidate pour une base RDF non vide B, ses contextes I_u et F_u, un ensemble des références D_u associées à u dans B, avec $D_u = \{D_1, ..., D_n\}$ et un nombre minimum de références positives $m \in \mathbb{N}^*$.*

L'évaluation de la qualité de la mise à jour u candidate pour B est positive si et seulement si $|\{D_i \mid D_i$ et u sont positivement similaires $\}| \geq m$.

L'évaluation de la qualité d'une mise à jour candidate pour une base nécessite de fixer le nombre minimum de références à trouver dans la base et le nombre minimum de ces références qui doivent être positivement similaires à la mise à jour.

Exemple 6. *Reprenons la mise à jour u_1 en figure 2.5, candidate pour DBPedia (i.e. la partie de DBPedia en figure 2.4) et ses contextes : I_{u_1} (figure 2.6) et F_{u_1} (figure 2.7). Fixons à 1 le nombre minimum de références positives (m). La différence de similarité entre la référence D_1 (figure 2.11) et les contextes de u_1 est positive ($simJaccard(F_{u_1}, D_1) - simJaccard(I_{u_1}, D_1) = 0.139 - 0.131$) et celle entre D_2 (figure 2.12) et les contextes de u_1 est négative ($simJaccard(F_{u_1}, D_2) - simJaccard(I_{u_1}, D_2) = 0.236 - 0.247$). Ainsi, u_1 est positivement similaire à D_1 ($0,004 > 0$) mais pas à D_2 ($-0,011 < 0$). L'évaluation de la mise à jour u_1 est donc positive : les modifications de la mise à jour créent des données similaires à au moins une ($\geq m$) des références. Donc, d'après les données actuellement dans la base, la mise à jour u_1 est de qualité et peut être appliquée à la base DBPedia.*

Dans l'exemple 6, nous avons comparé des ensembles de triplets RDF choisis de tailles comparables parmi des données réelles, issues de DBPedia. Il peut y avoir une grande différence de taille entre une mise à jour et une référence (*e.g.* comparer une mise à jour à propos d'une petite ville à une ville comme Paris) selon la façon dont elle est extraite. À cause de cette différence de taille, la mesure de similarité de Jaccard peut devenir très petite. Pour cela notre évaluation dépend uniquement du signe de la différence entre les mesures de similarité. Dans l'exemple 6 les mesures de similarité de D_1 sont presque de moitié inférieures à celles de D_2 mais permettent tout de même de valider u_1. Pour obtenir une évaluation positive, la taille des documents RDF n'importe pas, tant que les contextes de la mise à jour et les références ont des éléments communs et sont similaires.

Remarquons que dans notre méthode la structure (du graphe) des données est aussi importante que les données elles-mêmes. Par l'utilisation du voisinage des ressources et de mesures de similarité adaptée (telle que celle en définition 2.6), notre méthode identifie les données similaires ou différentes par la structure de leurs voisinages. Par exemple, on considère deux individus de la classe Personne tels que Wolfgang Amadeus Mozart et Michael Schumacher. Ces deux individus ont en commun des relations vers leurs noms, prénoms, *etc.* mais se différencient par leurs autre relations (l'un vers le monde de la musique et l'autre vers celui des sports automobiles).

3.4 Évaluer la qualité d'une base

Notre évaluation peut également être utilisée pour valider la qualité générale d'une base en considérant chaque partie de la base comme une mise à jour d'ajout. Cela nous permet d'évaluer l'homogénéité de la base et ainsi d'identifier des domaines de connaissance de la base qui contiennent trop peu de représentants.

Propriété 2.9 (Évaluation réflexive par similarité d'un ensemble de triplets RDF)**.** *Soit un ensemble de triplets RDF B et $n \in \mathbb{N}^*$ un rang de voisinage maximum.*

B est évaluée comme étant de qualité si $\forall r \in \mathcal{N}_B$ avec $H = voisinage^n_B(r)$, l'évaluation de chaque mise à jour $u = (H, \emptyset)$ pour $B \setminus H$ est positive.

Autrement dit, une base est validée si le voisinage de chacune de ses ressources est évalué comme étant de qualité par rapport au reste de la base. Ou encore, si chaque ressource de la base est décrite de manière similaire avec le reste de la base.

4 Expérimentation

Nous avons évalué avec notre méthode les mises à jour DBPedia Live de janvier 2014 telles qu'elles ont été appliquées sur la version 3.9 de DBPedia (470 millions de triplets). Ce sous-ensemble des mises à jour de DBPedia Live contenait 6 723 mises à jour d'ajout ou de modification, totalisant 211 119 triplets.

Théoriquement n'importe quel ensemble de parties de la base peut être utilisé comme ensemble de référence. Néanmoins il est évidemment plus efficace d'utiliser des parties de la base qui ont un minimum de points communs avec la mise à jour. L'extraction de références suffisamment ressemblantes à la mise à jour évaluée pour pouvoir être des points de comparaison utiles entre la base et la mise à jour n'est pas triviale. Nous proposons en chapitre 5 section 1 une approche performante d'extraction des références qui utilise des requêtes SPARQL relaxées. De façon schématique, cette méthode extrait de la base en priorité des références qui ont les structures les plus proches de celles des sections de la mise à jour. Plus cette méthode extrait de références, moins les références obtenues au cours de l'extraction ont de points communs avec la mise à jour. De ce fait, pour utiliser cette méthode lors de nos expérimentations nous avons fixé empiriquement à 50 le nombre de références à extraire par mise à jour. Cela nous permet à la fois d'obtenir des références qui sont similaire à la mise à jour et aussi d'obtenir un ensemble de références suffisamment grand pour qu'elles soient variées. D'une façon comparable, le nombre minimum de références positives doit nous permettre de pourvoir valider les mises à jour par similarité avec suffisamment de données structurellement proches d'un point de vue graphe présentes dans la base. Il doit également autoriser l'apparition de nouvelles structures de données soutenues par un petit nombre de données similaires dans la base. De façon empirique, nous avons fixé le nombre minimum de références positives de 5 pour nos expérimentations. Nous avons également fixé à 1 le rang du voisinage extrait pour les contextes et dans notre méthode d'extraction des références. Ce rang a été choisi en se basant sur les travaux, tels que [Guéret et al., 2011], qui ont étudié la densité des bases du Linked Data et montré qu'un voisinage de ressource au rang 4 dans DBPedia contient la quasi totalité du contenu de la base.

À la fin de nos tests, $0,1\%$ – 700 triplets – ont été évalués négativement par notre évaluation et n'auraient pas dû être appliqués. Leur non-validité provient essentiellement de l'utilisation de relations inexistantes dans l'ontologie, *e.g.* des relations nommées par une seule lettre.

Dans Wikipedia, les informations d'une page sont éditée et corrigées par une communauté d'utilisateurs, eux-mêmes encadrés par des modérateurs et des administrateurs. Les données des pages sont ensuite extraites par un framework créé par des experts pour être traduites en RDF et intégrées dans DBPedia. Notre méthode a été capable de trouver des mises à jour qui sont parvenues à passer à travers tous les processus de modération précédents. Des tests de notre méthode sur des données sans processus intermédiaire de modération entre la création de données et son intégration dans la base seront conduits dans de futurs travaux.

Remarquons que 30% des mises à jour ont été positivement évaluées par notre méthode alors qu'elles ne respectent pas les définitions de l'ontologie (essentiellement les domaines, co-domaines et disjonctions). Nous rappelons que notre méthode n'attache pas plus d'importance à l'ontologie qu'aux autre données. Un exemple intéressant des violations des définitions de l'ontologie faites dans les mises à jour évaluées se trouve dans la relation dbpprop:class. Il y a 10% des mises à jour acceptées qui ont utilisé la relation dbpprop:class

avec une instance de la classe dbponto:Species. Cependant, d'après l'ontologie, le co-domaine de dbpprop:class est dbponto:MeanOfTransportation. Donc notre méthode d'évaluation a trouvé suffisamment de parties de la base pour lesquelles l'ajout de la relation dbpprop:class avec un co-domaine incohérent ne fait pas diminuer la similarité des mises à jour. En étudiant l'ontologie de DBPedia, on observe que la relation correcte qui aurait dû être utilisée est probablement la relation dbponto:classis dont le co-domaine est dbponto:Species et qui possède une orthographe proche. Les pages Wikipedia correspondantes au ressources centrales des mises à jour utilisent le mot « class » dans les zones extraites pour DBPedia. Ce fait nous laisse supposer que la confusion entre les deux termes a eu lieu lors de la création de l'ontologie. Ainsi, dans notre méthode une erreur répandue est considérée comme une habitude des contributeurs pour la création de données qui a sert de règles *de facto* même si elle n'apparaît pas dans l'ontologie. On peut alors dire que notre méthode utilise les structures des données de la base qui sont suffisamment récurrentes pour être considérées comme la norme.

5 Conclusion

Pour conclure, nous avons proposé une méthode originale pour identifier les mises à jour contenant des mises en relation inhabituelles de ressources dans une base RDF. Notre méthode est adaptée pour les bases communautaires – telles que DBPedia et Freebase – recevant de nombreuses mises à jour simultanément. Nous pensons également que notre méthode peut être utilisée dans les bases communautaires comme une aide durant l'édition pour indiquer, avant soumission, la similarité de l'édition avec le contenu existant. Par exemple, notre méthode pourrait générer des avertissements additionnels sur les pages Wikipedia, en plus des états d'édition et avertissements existants, afin d'indiquer si les données d'une page Wikipedia suivent les habitudes de la communauté. Il est intéressant de noter que notre méthode d'évaluation de la qualité peut être utilisée en conjonction avec d'autres approches, telles que [Mendes et al., 2012], [Jacobi et al., 2011] et [Bonatti et al., 2011] pour évaluer la qualité d'une mise à jour avec des critères multiples : méta-données, règles, ...

Notre approche s'inscrit bien dans deux aspects des bases du Web des données qui nous paraissent importants : la base à mettre à jour contient au moins un exemple de données pour chacun des domaines de données qu'elle contient et la base mise à jour contient uniquement des données de qualité acceptable. Le premier aspect correspond à l'état actuel du Web des données dans lequel les bases apparaissent en contenant un noyau de données créées par des experts dans le domaine général de la base, ce noyau de données est ensuite étendu par toute la communauté d'utilisateurs. De ce fait, il n'y a pas de cas où une mise à jour est envoyée vers une base vide. Le second aspect suppose que les mises à jour antérieurs ont été validées (par exemple par des modérateurs ou à l'aide de la propriété 2.8) de sorte que la base contient uniquement des données validées. Pour vérifier cette seconde hypothèse, nous proposons d'utiliser notre proposition d'évaluation de la qualité pour valider toute une base (propriété 2.9).

Notre méthode repose en grande partie sur l'identification de parties de la base RDF, appelées références, qui sont similaires à la mise à jour. Nous proposons en chapitre 5 section 1 une approche performante d'extraction des références qui utilise des requêtes SPARQL relaxées.

Chapitre 3

Examen pour l'évolution d'une base RDF par détection de motifs

Dans le Web des données les ontologies servent à l'encadrement de la structure des données d'une base RDF. L'ontologie définit l'ensemble des classes et relations utilisables dans la base. Leurs définitions sont faites de façon à encadrer leurs utilisations par des contraintes pour préserver la cohérence des données de la base. Néanmoins, le caractère ouvert des bases communautaires du Web des données – qui appartiennent principalement au Linked Data – doit permettre une édition facile pour chaque contributeur sans nécessiter une connaissance approfondie de l'ontologie. Il en résulte un grand nombre de conflits dans les bases communautaires entre les données et les définitions de l'ontologie (*cf.* [Töpper et al., 2012, Sheng et al., 2012]). Ces conflits doivent être régulièrement résolus par un expert qui possède une connaissance suffisante des données de la base et du domaine qu'elle décrit. Au cours du temps dans la vie d'une base, certaines incohérences ou groupes d'incohérences sont révélateurs d'évolutions du langage ou de nouvelles pratiques communautaires qu'il est nécessaire de prendre en compte. L'expert doit alors examiner la base et décider des évolutions à apporter à l'ontologie ou aux données. Cet examen et la décision des évolutions résultantes sont difficilement automatisables. À chaque changement, il est nécessaire de vérifier s'ils n'entraînent pas l'apparition d'incohérences nouvelles. Il est également nécessaire de vérifier si les équivalences entre ressources de différentes bases se maintiennent à chaque évolution et de garder la trace des changements effectuées.

Ce chapitre présente notre méthode d'examen de l'adéquation de l'ontologie et des données pour l'évolution d'une base RDF. Le résultat d'un tel examen se veut être une aide précieuse pour un expert devant actualiser une base RDF. Notre méthode se base sur les données de la base qui sont incohérentes par rapport à l'ontologie pour identifier les motifs récurrents – les ensembles de ressources ou utilisations de relations semblable – qui y apparaissent. Ces motifs sont ensuite extraits pour fournir un diagnostic des incohérences à l'expert. Ce diagnostic permet d'identifier les incohérences récurrentes venant d'une inadéquation de l'ontologie aux données actuelles ou d'une mauvaise habitude des communautés de contributeurs. À partir de cela, le choix des modifications à apporter à l'ontologie ou aux données de la base est laissé à l'expert.

En section 1 nous faisons un état de l'art des méthodes d'évolution des bases RDF. Nous définissons en section 2 la notion de motif autour de ressources RDF dans une base. Notre méthode de diagnostic pour aider à l'examen de bases RDF par identification de motifs récurrents de description dans les données incohérentes avec l'ontologie est décrite en section 3. Enfin, en section 4 nous présentons nos expériences de détection de motifs

pour le diagnostic de DBPedia.

Sommaire

1 État de l'art sur l'évolution de l'ontologie et des données d'une base RDF

Dans le modèle actuel du Web des données, les ontologies sont fixées à un instant donné, généralement à la création des bases. Les bases RDF grandissent et subissent des changements de terminologies ou d'habitudes qui entourent les ressources qu'elles décrivent. Ces changements créent progressivement des conflits avec l'ontologie qui entraînent des incohérences dans la base. Ces modifications exigent qu'un expert se penche régulièrement sur ces conflits pour déterminer les opérations d'évolution de l'ontologie ou de correction des données qui sont nécessaires.

Dans la littérature, il existe deux grandes catégories pour ce problème d'évolution. Dans la première catégorie se trouve les méthodes qui se concentrent en priorité sur l'ontologie, avec aucun ou très peu de regards pour les données factuelles des bases RDF. Ces méthodes font souvent usage des raisonnements dans les logiques de description. Elles sont liées aux travaux tels que [Javed et al., 2013, Mahfoudh et al., 2015] qui tentent d'identifier et de formaliser les différents types de changements qu'une ontologie peut subir et leurs conséquences. La seconde catégorie rassemble les méthodes qui se basent plutôt sur les données factuelles que sur l'ontologie. L'augmentation de la quantité de données factuelles disponible apportée par l'expansion des bases communautaires du Linked Data a soutenu le développement de ces méthodes. Ces méthodes utilisent divers traitements des données factuelles – *e.g.* clustering, réseau bayésiens – pour identifier les évolutions potentielles de la base. Notre travail se concentre sur les données factuelles pour identifier des parties de la base nécessitant une évolution, il se rapproche donc dans cette dernière catégorie. Notre méthode effectue des regroupements parmi les ressources qui sont incohérentes avec l'ontologie pour identifier des motifs dans leurs voisinages.

Le travail présenté dans [Djedidi and Aufaure, 2010] propose une approche automatique d'application des changements de l'ontologie qui gère la résolution des incohérences que ces changements font apparaître. Cette approche se fonde sur un stockage des modèles des changements, incohérences et corrections associées. Elle détermine automatiquement les moyens de résolution des incohérences à appliquer en fonction d'une mesure de la qualité des données après application du changement. Cette approche a été implémentée dans le système Onto-Evo^{A}L.

L'article [Rieß et al., 2010] propose une approche similaire, appelée EvoPat, qui utilise des patrons d'application des modifications de l'ontologie. Cette approche adapte la notion de « mauvaise odeur » utilisée pour l'évaluation de patrons de conception en programmation pour identifier les incohérences créées par l'évolution de l'ontologie dans les bases RDF. Pour chaque « mauvaise odeur », une méthode de résolution de l'incohérence est associée. Ces méthodes et les « odeurs » sont représentées sous forme de requêtes SPARQL. Cette approche a été intégrée dans le système OntoWiki, [Auer et al., 2006].

Le travail présenté dans [Nikitina et al., 2012] propose une méthode interactive de présentation des évolutions d'ontologie. La méthode étudie les modifications de l'ontologie qui lui sont proposées par l'expert et les lui présente à l'expert de façon à ce qu'il ait à valider le nombre minimum de modifications. Plus précisément, les modifications sont présentées dans un ordre logique telle que si une modification validée en implique une autre, elle est également automatiquement validée.

L'article [Köhler et al., 2011] propose un logiciel utilisant le raisonnement en logiques

de description pour extraire les définitions ontologiques en conflit. Il identifie des définitions incohérentes, extrait leurs contextes pour présenter à un expert l'incohérence et les raisonnements qu'elle affecte. Étant basé sur le raisonnement, cet outil s'applique uniquement sur des ontologies OWL DL.

Enfin, l'article [Mortensen et al., 2013] propose d'utiliser la production participative pour la correction d'incohérences dans une base. Il propose une méthode basée sur la préparation de micro-taches de vérification de l'ontologie – transmises à des volontaires de Amazon Mechanical Turk – et un modèle d'inférence bayésien pour corriger l'ontologie. Dans cette méthode, toutes les étapes – identification de l'évolution nécessaire, correction des incohérences qu'elle génère – sont gérées par les volontaires et les experts qui les encadrent. Cette méthode est comparée à une correction manuelle effectuée par un expert, 86% des erreurs trouvées par l'expert sont corrigées par production participative sur Amazon Mechanical Turk.

L'article [Cherix et al., 2014] propose une approche de correction de l'ontologie par clustering des données. Cette approche, implémentée dans le framework CROCUS, utilise le clustering des individus d'une classe pour identifier les erreurs d'instanciation. Pour regrouper les individus d'une classe, les voisinages de chaque individu sont transformées en vecteurs numériques. Ces vecteurs sont ensuite utilisés par un algorithme de clustering. Les individus n'ayant pas été regroupés sont montrés à des juges humains qui peuvent alors décider de corriger la base. Cet article utilise les erreurs isolées pour identifier les données qui indiquent une évolution nécessaire de la base.

L'article [Fleischhacker et al., 2014] propose une méthode pour la détection des valeurs numériques anormales des littéraux des bases RDF. Il étudie les valeurs des littéraux associées à différents individus décrit dans plusieurs bases du Linked Data. Parmi ces valeurs, il recherche les valeurs anormales – trop éloignées de la moyenne – pour identifier des erreurs potentielles à corriger. Cette approche se limite à la recherche d'erreurs isolées dans les littéraux.

L'outil *LiQuate* présenté dans l'article [Ruckhaus et al., 2014] a pour but d'aider au maintien de la qualité des liens entre les bases du Linked Data. Pour cela, il utilise une approche d'évaluation basée sur les réseaux bayésiens pour identifier les incohérences, les ambiguïtés et les mises en relations incomplètes dans les liens entre ressources de bases différentes Le résultat final de cette évaluation suggère à un expert les ressources potentiellement mal liées entre elles. Cet outil a été utilisé pour tester la mise en relation de DBPedia et de deux bases en biologie.

L'approche présentée dans [Bühmann and Lehmann, 2013] propose d'améliorer semi-automatiquement les faits de la base par l'étude des structures courantes de l'ontologie. Plus précisément, un groupe d'ontologies est étudié pour déterminer les structures les plus répandues dans différentes bases. Ces structures sont ensuite converties en requêtes SPARQL pour extraire des individus dont l'instanciation et les propriétés peuvent être enrichis par un expert.

2 Motifs de description

Dans ce chapitre, nous nous plaçons dans la situation où un expert d'une base RDF souhaite l'examiner. Nous définissons d'abord le *graphe descriptif d'une ressource*, regroupement d'une ressource et de son voisinage.

Définition 3.1 (Graphe descriptif d'une ressource)**.** *Soit une base RDF B, une ressource $r \in \mathcal{R}_B$ et $n \in \mathbb{N}^*$.*

On appelle graphe descriptif au rang n de la ressource r dans B *le couple* $(r, voisinage^n_B(r))$, *et* r *le* centre *du graphe descriptif.*

Le graphe descriptif d'une ressource est composé de la ressource et de sa description dans la base RDF, c'est-à-dire le voisinage de la ressource jusqu'à un rang donné. Au rang 1, le graphe descriptif forme un graphe en étoile centré sur r.

Exemple 7. *Les deux exemples de graphes descriptifs donnés en figure 3.1*[1] *sont centrés respectivement autour des ressources « Renault 7 » et « Volkswagen Beetle » issues de DBPedia. Ces deux graphes contiennent le voisinage au rang 1 de chacune des deux ressources.*

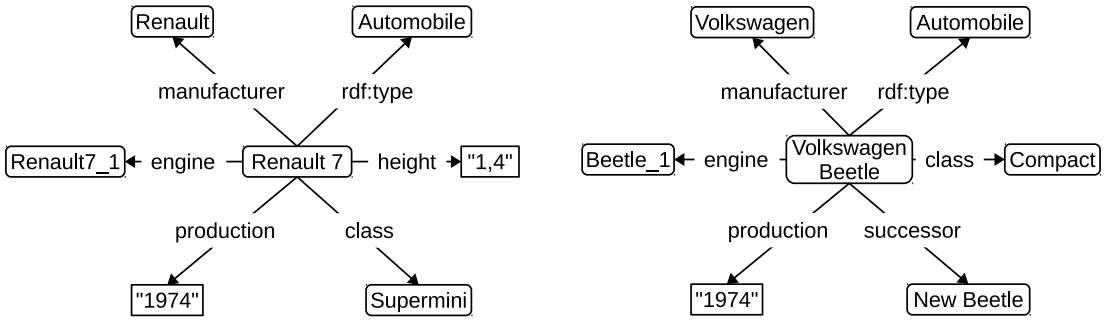


FIGURE 3.1 – Exemples de graphes descriptifs tirés de DBPedia, centrés autour de deux ressources : « Renault 7 » et « Volkswagen Beetle ».

Nous souhaitons obtenir les points communs de plusieurs graphes descriptifs, sous la forme d'un *motif de description* des graphes descriptifs. Un motif de description est composé de triplets qui contiennent des éléments communs aux graphes descriptifs de plusieurs ressources. Les ressources nœuds d'un motif de description sont soit des ressources des graphes descriptifs à son origine, soit des nœuds blancs.

[1]Dans cette figure et dans les figures suivantes, les noms des classes et individus ont été traduits pour plus de lisibilité.

Définition 3.2 (Motif de description)**.** *Soient une base RDF non vide B, un ensemble de ressources $R \subseteq \mathcal{R}_B$ appelé* ensemble des centres descriptifs*, avec $R = \{r_1, r_2, ..., r_n\}$ et $\mathcal{G}$ l'ensemble des graphes descriptifs des ressources de R de même rang dans B avec $\mathcal{G} = \{(r_1, D_{r_1}), (r_2, D_{r_2}), ..., (r_n, D_{r_n})\}$.*

L'ensemble de triplets non vide et connexe M, avec $r \in \mathcal{K}_M$, est un motif de description *si pour chaque élément (r_i, D_i) de $\mathcal{G}$ on peut obtenir M à partir d'un sous-ensemble de D_i en appliquant les règles suivantes :*

- *la ressource r_i est transformée en r ;*
- *une ressource nœud est transformée en un nœud blanc.*

Les ressources de R sont appelées des centres descriptifs de M *et le nœud r est appelé* nœud racine de M.

Un triplet d'un motif de description est composé soit d'éléments qui apparaissent de la même manière dans un triplet de chaque graphe descriptif (*i.e.* en étant sujet, relation ou objet), soit de nœuds blancs. Les centres descriptifs d'un motif sont les éléments à l'origine du motif. Ainsi un motif de description contient toujours un nœud blanc, appelé *nœud racine du motif*, dans les triplets qui contiennent les ressources communes des triplets qui contiennent les centre descriptifs dans leurs graphes descriptifs respectifs. D'un point de vue graphe, le voisinage de la racine du motif de description est structurellement similaire au (à une partie du) voisinage de chaque centre descriptif. Remarquons que dans un motif de description, toutes les ressources communes aux graphes descriptifs ne sont pas nécessairement présentes.

Définition 3.3 (Motif de description maximal)**.** *Soient une base RDF non vide B, une ensemble de ressources $R \subseteq \mathcal{R}_B$ avec $R = \{r_1, r_2, ..., r_n\}$ et $\mathcal{G}$ l'ensemble des graphes descriptifs des ressources de R de même rang dans B avec $\mathcal{G} = \{(r_1, D_{r_1}), (r_2, D_{r_2}), ..., (r_n, D_{r_n})\}$.*

Un motif de description M de R tel que $\mathcal{R}_M = \bigcap_{i=1}^{n} \mathcal{R}_{D_{r_i}}$ est appelé motif de description maximal *de R.*

Un motif de description qui contient toutes les ressources communes – en tenant compte si elles sont sujets ou objets d'un triplet – de tous les graphes descriptifs d'un ensemble de ressources est appelé *motif de description maximal*.

Exemple 8. *Les figures 3.2 et 3.3 contiennent des éléments communs aux graphes descriptifs représentés en figure 3.1. Ils contiennent tous les deux la relation qui lie les deux*

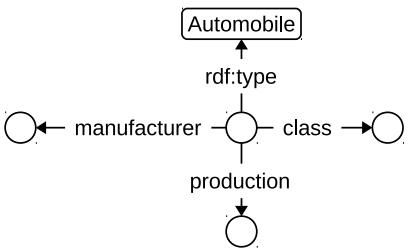


FIGURE 3.2 – Un motif de description des graphes descriptifs de l'exemple 7, figure 3.1.

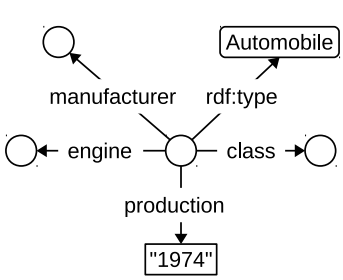


FIGURE 3.3 – Motif de description maximal des graphes descriptifs de l'exemple 7, figure 3.1.

voitures de la figure 3.1 à la classe Automobile. *Le motif en figure 3.2 contient les relations qui lient les deux voitures à leur fabricant (*manufacturer*), leur classe de véhicule (*class*) et leur année de production (*production*). Le motif en figure 3.3 est le motif maximal des deux voitures de la figure 3.1. Il contient l'année de production commune au deux voitures ("1974") et la relation qui les lie à leur moteur (*engine*), en plus des triplets du motif en figure 3.2.*

Définition 3.4 (Occurrence, taille et valeur de description d'un motif)**.** *Soient R un ensemble de ressources, M un motif de description de R pour un rang $n \in \mathbb{N}$ et $r \in \mathcal{K}_M$ la racine de M.*

La fonction $occ(M)$ telle que $occ(M) = |R|$ est appelée occurrence *de M.*

La fonction $taille(M)$ telle que $taille(M) = |M|$ est appelée taille *de M.*

La fonction $desc(M)$ telle que $desc(M) = \frac{\sum_{t \in V_1} |\mathcal{N}_{\{t\}} \setminus \mathcal{K}_{\{t\}}| + \sum_{t \in V_n \setminus V_1} |\mathcal{N}_{\{t\}} \setminus \mathcal{K}_{\{t\}}|}{|V_1| + |V_n \setminus V_1| \times 2}$ avec $V_1 = voisinage^1_M(r)$ et $V_n = voisinage^n_M(r)$

L'occurrence d'un motif de description est le nombre de centres descriptifs à l'origine du motif. La taille d'un motif de description est le nombre de triplets qu'il contient. La valeur de description d'un motif de description dépend du nombre de ressources nœuds qui ne sont pas des nœuds blancs dans chaque triplet du motif et du nombre total de ressources nœuds qu'il peut contenir. On peut considérer la valeur de description comme la proportion de ressources nœuds « nommées » dans le motif. Dans les triplets du voisinage de rang 1 de la racine du motif, chaque triplet contient au moins le nœud blanc racine. Le voisinage de rang 1 de la racine du motif contient donc, au plus, autant de ressources nœuds qui ne sont pas des nœuds blancs que de triplets. Au delà du rang 1, le voisinage de la racine contient des triplets qui peuvent contenir des nœuds blancs en sujet et/ou en objet ou aucun des deux. Les triplets du voisinage de rang supérieur à 1 contiennent donc, au plus, deux ressources nœuds qui ne sont pas des nœuds blancs.

Exemple 9. *Le motif en figure 3.2 est de taille 4 car il contient 4 triplets :* (_:1, rdf:type, Automobile), (_:1, class, _:2), (_:1, production, _:3) *et* (_:1, manufacturer, _:4)*. Il est d'occurrence 2 car ses centres descriptifs sont les 2 ressources* Renault 7 *et* Wolskwagen Beetle*. Il contient une ressources nœud qui n'est pas un nœud blanc au rang 1 :* Automobile*. Sa valeur de description est donc égale à $\frac{1}{4} = 0,25$.*

Le motif en figure 3.3 est de taille 5, a une occurrence de 2 et une valeur de description de $0,4$.

3 Diagnostic pour l'aide à la prise en compte de nouvelles habitudes de description de données

Nous avons défini les motifs de description comme les descriptions communes à un ensemble de ressources. Dans une base RDF, un motif de description suffisamment important pour un ensemble de ressources données est révélateur, selon nous, d'une habitude d'écriture suivie par les contributeurs de la base dans les descriptions des ressources.

Prenons le cas des instances d'une classe qui représentent des ressources similaires. Les instances de cette classe suivent naturellement le même motif, résultat d'une habitude de description dictée par l'ontologie où les propriétés sont définies. Prenons maintenant le cas d'un ensemble de ressources qui partagent le même motif de description sans qu'il soit défini par l'ontologie. Ce dernier cas indique une évolution possible de la base portée

par la communauté. L'intégration d'une nouvelle définition dans l'ontologie ou de son adaptation pour tenir compte de cette nouvelles habitude de description est certainement nécessaire.

Rappelons qu'au sein d'une base communautaire, on n'exige pas des contributeurs une connaissance approfondie de l'ontologie, ce qui les conduit régulièrement à ne pas en tenir compte en exprimant « librement » les données qu'il édite dans la base. Ces pratiques communautaires font qu'il n'est pas rare que des données factuelles entrent en conflit avec l'ontologie.

Pour détecter de nouvelles habitudes de description dans une base RDF, nous étudions en priorité les individus incohérents avec l'ontologie de la base. Les individus incohérents qui forment des ensembles de ressources partageant un motif sont susceptibles d'être les symptômes d'une habitude de description communautaire qui n'est pas prise en compte dans l'ontologie. Nous recoupons donc ensemble les habitudes de description avec les contraintes ontologiques non-respectées dans une base RDF, afin de les présenter à un expert pour l'aider à distinguer ce qui relève de « nouvelles pratiques communautaires de description » – où une évolution de l'ontologie semble nécessaire – de ce qui est un ensemble d'erreurs d'édition.

3.1 Détection d'habitudes

Les motifs de description permettent d'identifier facilement une pratique courante pour décrire des ressources. Pour un ensemble donné de ressources, nous proposons de détecter des habitudes de description selon les tailles, nombre d'occurrences et les informations que portent leurs motifs.

Définition 3.5 (Habitude de description)**.** *Soit un motif de description M, $o \in \mathbb{N}^*$ un seuil d'occurrence, $t \in \mathbb{N}^*$ un seuil de taille et $d \in \mathbb{R}^+$ un seuil de valeur de description.*

Le motif de description M représente une habitude de description *selon o, t et d si*

$$occ(M) \geq o$$

$$et\ taille(M) \geq t$$

$$et\ desc(M) \geq d.$$

Un motif de description indique une habitude communautaire s'il est suffisamment grand, apparaît suffisamment de fois dans la base et s'il contient suffisamment de ressources d'après les critères d'un expert de la base.

3.2 Contraintes d'intégrité

Les contraintes d'intégrité sont un concept issu des bases de données. Elles ne sont pas définies nativement dans le Web des données car elles reposent sur l'hypothèse d'une monde fermé pour les données alors que RDF repose sur un monde ouvert. Telles que décrites dans [Motik et al., 2009] pour le Web des données, les contraintes d'intégrité correspondent à des règles générées à partir de certaines définitions de l'ontologie dont le respect est requis pour maintenir la cohérence de la base RDF.

Certaines définitions ontologiques peuvent être considérées comme des contraintes d'intégrité pour la description de ressources. Elles sont en effet utilisées lors de raisonnements

en logiques de description de classification (organiser la hiérarchie des types dans l'ontologie) et d'instanciation (attribuer le type – le plus spécifique – à un individu ou un littéral). En pratique, ces contraintes définissent des règles de maintien de la cohérence à respecter lors de la création de données. Selon [Sirin and Tao, 2009], les *contraintes d'intégrité*[2] tirées d'une ontologie sont à classer de manière similaire à celles utilisées dans les bases de données relationnelles : (i) Les contraintes de typage, qui exigent que les ressources liées par une relation donnée aient un type précis, générées par les domaines et co-domaines des propriétés ainsi que par les disjonctions. (ii) Les contraintes d'unicité, qui exigent qu'une ressource ne puisse pas être présente de la même manière dans plus d'un triplet contenant la même relation, générées par les propriétés fonctionnelles. (iii) Les contraintes de définition, qui exigent qu'une ressource soit liée à une autre par un triplet contenant une relation ou des ressources nœuds précises, générées par les définitions de classes par restriction en OWL.

3.3 Nouvelles habitudes de description

Pour une contrainte d'intégrité donnée, nous générons d'un coté un[3] motif de description correspondant aux ressources qui respectent la contrainte d'intégrité, et d'un autre coté un motif de description correspondant aux ressources qui ne respectent pas la contrainte. Cette confrontation des deux motifs de description constitue pour nous un *diagnostic d'une contrainte d'intégrité*.

Une *nouvelle habitude potentielle de description* est une manière récurrent dans la communauté de décrire des ressources qui ne respectent pas une contrainte d'intégrité.

Exemple 10. *La figure 3.4 présente les graphes descriptifs de deux ressources issues de DBPedia et sujets de la relation « class »*[4].

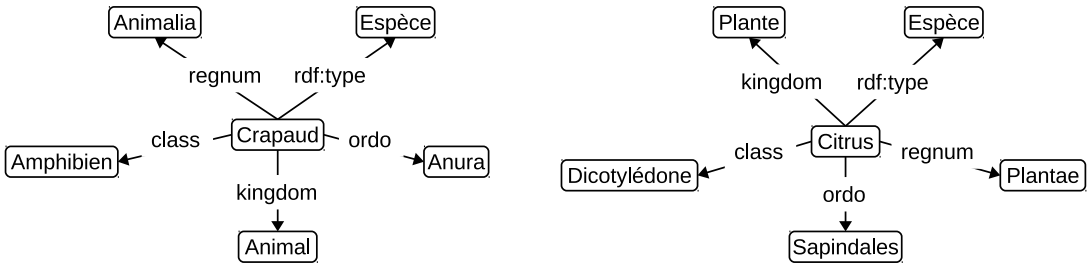


FIGURE 3.4 – Exemples de graphes descriptifs centrés autour de deux ressources : « Crapaud » et « Citrus ».

Selon l'ontologie de DBPedia, la relation class *a pour domaine la classe* MoyenDeTransport. *Les ressources utilisées en exemple 7 respectent le domaine de la relation, en effet leur typage par* MoyenDeTransport[5] *apparaît dans leur motif commun en figure 3.3. Par contre, les ressources en figure 3.4 ne respectent pas le domaine de la relation* class *(il n'y a aucun lien hiérarchique entre* Automobile *et* Espèce *dans DBPedia).*

Si l'on extrait le motif maximal des ressources qui ne respectent pas le domaine de la relation class, *on obtient le motif maximal représenté en figure 3.5.*

[2] Nous reprenons de la même manière cette désignation de *contrainte d'intégrité*

[3] *cf.* discussion c-après.

[4] Plus exactement la propriété `http://dbpedia.org/ontology/class`, utilisée pour indiquer la classification des moyens de transports selon leurs tailles, nombres de passagers, *etc.*

[5] Automobile est sous-classe de MoyenDeTransport dans DBPedia.

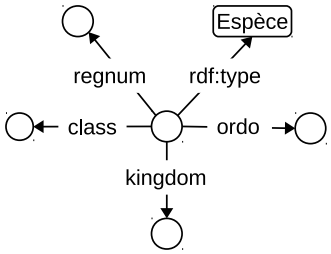


FIGURE 3.5 – Le motif de description maximum aux graphes descriptifs en figure 3.4.

Parmi les quatre ressources liées par la relation class *nous avons donc deux ressources qui ont en commun les motifs en figure 3.2 et figure 3.3, et deux ressources qui ont en commun le motif en figure 3.5. Les relations utilisées dans le motif en figure 3.3 et en figure 3.5 ne sont pas les mêmes. Les ressources* Crapaud *et* Citrus *de nos exemples ne respectent pas le domaine de* class *; elles suivent cependant le même motif. Dans cet échantillon de* 4 *ressources, avec un seuil d'occurrence à* 2*, un seuil de taille à* 4 *et un seuil de valeur de description à* 0.1*, nous pouvons identifier le motif en figure 3.5 comme une nouvelle habitude de description. En effet le motif en figure 3.5 a une occurrence de* 2*, une taille de* 5 *et une valeur de description de* 0.2 *qui sont supérieurs au seuils de détection des habitudes de description.*

Discussion La détection d'habitudes de description dépend de trois seuils pour l'occurrence, la taille et la valeur de description du motif de description.

- Il est préférable de choisir un seuil de taille suffisamment grand qui permette de faire ressortir les relations caractéristiques du motif de description, et pas uniquement les relations très récurrentes dans les bases.

  Par exemple, dans une base telle que DBPedia, où les ressources disposent toujours d'au moins un type, d'une propriété rdfs:label et d'une propriété owl:sameAs. Les motifs extraits des ressources de cette base contiennent toujours au moins trois triplets pour ces trois propriétés. Selon notre expérience, il est préférable de choisir un seuil de taille supérieur à 3.

- Le seuil d'occurrence est à choisir selon les caractéristiques de la base étudiée. Il semble préférable de le choisir à partir d'un pourcentage du nombre de ressources concernées par la contrainte étudiée.

- Le seuil de valeur de description reflète la proportion en ressources nœuds d'URIs et de littéraux par rapport aux nœuds blancs dans le motif. Quand le seuil de valeur de description tend vers zéro, seul l'aspect structurel, au sens graphe du terme, des graphes descriptifs est pris en compte dans le motif. Plus ce seuil est élevé et plus une caractérisation spécifique au motif apparaît.

D'un point de vue pratique, ces seuils permettent également de limiter le nombre de calculs effectués lors de l'extraction de motifs de description. Les motifs de taille trop faible ou trop peu descriptifs peuvent être ainsi évités (car peuvent être éliminés plus tôt durant l'extraction).

4 Expérimentation

Dans cette expérimentation, nous nous intéressons à la propriété class pour laquelle nombre d'utilisation dans la base DBPedia ne respectent pas la contrainte de typage du domaine[6].

Des travaux tels que [Sirin and Tao, 2009] et [Kontokostas et al., 2014] ont montré que les contraintes d'intégrité des bases RDF peuvent aisément être exprimées en requêtes SPARQL pour les vérifier. Nous avons utilisés de telles requêtes SPARQL pour extraire l'ensemble des ressources ne respectant pas la contrainte d'intégrité du domaine de la propriété class. Les motifs de description ont été extraits à partir de cet ensemble selon la méthode non-naïve décrite en chapitre 5 section 2. Nous avons utilisé des motifs de description basés sur un voisinage au rang 1 en nous basant sur les observations déjà faites au chapitre 2 section 4 à propos de la densité de la base DBPedia (au rang 4, un voisinage contient quasiment toute la base).

Nous avons opté empiriquement pour un seuil d'occurrence égal à 50% des ressources examinées, un seuil de taille de 4 et un seuil de valeur de description de 0.1. Nous avons extrait les motifs des graphes descriptifs de 200 ressources de DBPedia qui ne respectaient pas la contrainte. Parmi ces 200 ressources, 197 forment un ensemble qui suit le motif de description présenté en figure 3.5.

En transformant le motif en figure 3.5 en requête SPARQL – en remplaçant les nœuds blancs par des variables – nous avons recherché combien de ressources le suivent dans DBPedia. Cette recherche dans la base DBPedia montre que parmi les 267 000 ressources qui ne respectent pas le domaine de la propriété class, 227 503 (85%) suivent le motif en figure 3.5. Par conséquent le motif en figure 3.5 a une occurrence de 227 503 dans DBPedia, une taille de 6 et une valeur de description de 0.58. Par ses propriétés, ce motif semble représentatif d'une nouvelle habitude de description dans DBPedia pour les ressources liées par la propriété class et doit être présenté à un expert.

Cette « incohérence » récurrente a également été repérée dans des travaux qui ont étudié l'état de DBPedia par le parcours manuel des ressources en conflit avec l'ontologie, tel que dans [Sheng et al., 2012]. Après analyse de l'ontologie, des ressources concernées et des pages dont elles sont issues, cette erreur semble due à une confusion entre la relation class et la relation classis[7]. Cette hypothèse est renforcée par le fait que le domaine de classis est la classe Espèce. Face à notre diagnostic, un expert peut décider de considérer la nouvelle (et mauvaise) habitude de description comme une erreur généralisée et corriger les 227 503 occurrences concernées. Il peut également – maintenant que l'habitude est prise par la communauté – choisir de faire évoluer la description de class dans l'ontologie pour intégrer cette habitude de description.

5 Conclusion

Nous avons proposé une méthode originale pour l'examen des incohérences d'une base RDF qui met en évidence les motifs récurrents autour de ressources qui ne respectent pas les contraintes d'intégrité de l'ontologie. Ces motifs sont identifiés par la constitution du voisinage de ces ressources, appelé *graphe descriptif*. La structure commune et les ressources communes sont extraites sous la forme de graphes de synthèse, appelés *motifs*

[6] L'ontologie de DBPedia définit qu'une relation class lie une ressource de type « MoyenDeTransport » à une autre ressource.

[7] http://dbpedia.org/ontology/classis : « Troisième niveau de la classification classique (c'est-à-dire n'utilisant pas la notion de distance génétique) des espèces vivantes (voir systématique). » d'après l'ontologie de DBPedia.

de description. Ces motifs de description permettent d'identifier les attributs communs à des ensembles de ressources qui n'apparaissent pas dans l'ontologie. Rappelons que notre méthode est conçue pour les bases communautaires dans lesquelles les contributeurs sont libres de décrire un ensemble de ressources sans être limités par l'ontologie. Les motifs de description qui sont suffisamment remarquables dans la base caractérisent des *habitudes de description* des données. Lorsque ces habitudes font intervenir des incohérences dans les données selon l'ontologie de la base, elles peuvent être considérées par un expert comme les symptômes d'une évolution nécessaire de la base (de ses données ou de ses données).

L'extraction des motifs de description issus des graphes descriptifs d'un ensemble de ressources n'est pas triviale. Une méthode naïve consiste à comparer les graphes descriptifs de chaque ressource deux à deux pour progressivement trouver les points communs qui apparaissent dans plusieurs graphes descriptifs. Néanmoins une telle méthode exige un très grand nombre de comparaisons entre graphes descriptifs avant d'obtenir les motifs avec une occurrence suffisante pour nous intéresser. Dans le chapitre 5 section 2 nous proposons une méthode performante d'extraction des motifs qui minimise le nombre de comparaisons et qui peut donner un résultat partiel à tout moment avant d'avoir traité toutes les ressources.

Chapitre 4

Interrogation des bases du Linked Data à travers des aperçus de recherche

Le Linked Data forme une toile de bases RDF interconnectées, tout comme le Web forme une toile de pages interconnectées, et peut être considéré comme un seul immense graphe de données RDF. Bien que chaque base du Linked Data dispose d'un *endpoint* SPARQL pour accéder à ses données via des requêtes SPARQL, il est difficile en pratique aujourd'hui d'appréhender le Linked Data dans sa globalité. En effet, en l'absence de moteur de recherche dédié au Linked Data, celui-ci reste encore cloisonné base par base (*i.e.* endpoint par endpoint).

Ce chapitre présente notre méthode pour l'interrogation d'un ensemble de bases RDF et plus particulièrement de bases du Linked Data. Notre méthode utilise la génération de documents RDF compacts, sortes de résumés de bases RDF, appelés *aperçus de recherche* ou simplement *aperçus*. Leur utilisation permet de déterminer l'absence de résultats à une requête dans une base. Ainsi, de façon schématique, si par exemple une requête traite de « personnes qui conduisent des voitures », nous écartons immédiatement et catégoriquement – *i.e.* sans nécessité d'interroger les bases dans leur globalité – les bases qui ne traitent ni de personnes, ni de voitures. Notre méthode est adaptée à l'agencement actuel des données dans le Linked Data, où face à la multitude de bases RDF et la diversité des données, la réponse à une requête SPARQL est généralement présente dans seulement quelques bases. Son utilisation se fait en deux étapes :

(i) On génère un aperçu de recherche à partir du contenu et de l'ontologie de la base. Cette génération se base sur les types des individus de la base et regroupe les individus de chaque classe et les littéraux en conservant les relations qui les lient les uns aux autres. Il est possible de paramétrer la façon dont les individus et les littéraux sont fusionnés dans l'aperçu. L'aperçu généré est un document RDF de taille beaucoup plus petite que sa base d'origine. Il doit être actualisé uniquement lors des modifications de la base.

(ii) Les requêtes envoyées à la base sont d'abord envoyées à l'aperçu de recherche après une réécriture simple et immédiate de la requête pour l'adapter aux spécificités de l'aperçu de recherche. La structure de l'aperçu de recherche est générée de façon à ce que si une requête n'a pas de résultat dans l'aperçu, alors elle n'a pas de résultat dans la base. La présence de résultat(s) à une requête dans l'aperçu de recherche

indique la présence potentielle de résultat(s) dans sa base d'origine. La taille réduite de l'aperçu de recherche permet de l'interroger rapidement. On peut donc vérifier rapidement l'absence de résultat à une requête dans une base via l'interrogation de son aperçu. La figure 4.1 schématise ce principe.

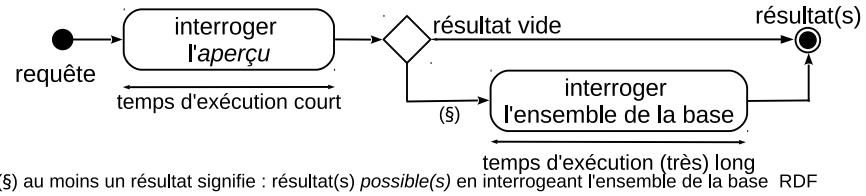


FIGURE 4.1 – Schéma de principe : interrogation de l'aperçu en premier, puis si la base semble pertinente interrogation de celle-ci.

En section 1 nous faisons un état de l'art des approches s'apparentant à la notre, soit parce qu'elles concernent les requêtes distribuées, soit parce qu'elles sont fondées sur les résumés de documents RDF. En section 2 nous définissons les aperçus de recherche, donnons les règles qui permettent leur génération et les règles de réécriture des requêtes qui les interrogent. Les règles de génération d'aperçus de recherche sont conçues pour effectuer une génération à partir du contenu d'une base à un instant donné et de recommencer cette génération lorsqu'il est nécessaire de mettre à jour l'aperçu de recherche. Nous donnons des exemples de génération et d'interrogation des aperçus de recherche en section 3. Enfin, en section 4 nous présentons les résultats de nos expérimentations sur les aperçus de recherche à partir d'un jeu de données tiré de DBPedia, d'un ensemble de requêtes inspirées de la littérature et de celles envoyées quotidiennement à DBPedia.

Sommaire

1 État de l'art sur les requêtes dans le Linked Data et les résumés de bases RDF

Notre méthode d'interrogation s'inscrit dans deux axes de recherche du Web des données. D'une part, elle répond à la problématique de l'interrogation d'un ensemble de bases RDF. Dans le Web des données, cette problématique est directement liées à l'interrogation des bases du Linked Data. Elles forment un ensemble de bases de données liées entre elles et dans des domaines complémentaires. Pour les exploiter au mieux, plusieurs travaux ont proposés des systèmes pour l'interrogation distribuée d'un ensemble de bases. D'autre part, notre méthode se base sur la génération de résumés de bases RDF. La structure des bases RDF et les relations de généralisation/spécialisation ou d'équivalence apportées par les ontologies amènent facilement la création de versions compactes des bases RDF en conservant des propriétés utiles. De nombreux travaux ont proposé des méthodes de génération de résumés de base RDF pour répondre à divers besoins.

Le travail présenté dans [Hartig et al., 2009] propose un système de requête pour le Linked Data qui utilise les relations d'équivalence entre ressources de différentes bases pour propager la recherche de résultats d'une requête. Cette approche interroge d'abord une base connue du Linked Data et utilise ensuite les relations owl:sameAs de ses ressources vers d'autres bases pour trouver d'autres résultats. Ce système utilise pleinement le caractère « Linked » du Linked Data. Contrairement à notre méthode, cette approche ne nécessite pas d'avoir une liste des bases du Linked Data à interroger tant que la base de départ est suffisamment liées dans le Linked Data. Elle se heurte néanmoins à des problèmes de temps de latence lors de l'interrogation des autres bases, alors que notre méthode est conçue pour limiter ces temps de latence.

Dans l'article [Görlitz and Staab, 2011] les requêtes sont distribuées sur un ensemble de base en fonction des informations données dans leurs fichiers de description VoID. Cette approche utilise les données statistiques données par les administrateurs des bases et contenues dans les descriptions VoID des bases pour optimiser l'exécution d'une requête. Cette méthode est gênée par le manque actuel de descriptions VoID dans les bases du Linked Data et par les limitations de l'ontologie VoID pour l'expression de statistiques.

Il existe un type de requête, dites requêtes fédérées, dont la résolution nécessite plusieurs bases différentes. Plus précisément, une requête fédérée permet de rechercher des données liées et réparties dans différentes bases. Elles recherchent les résultats dans l'union de toutes les bases disponibles. Elles diffèrent de notre approche qui est de rechercher la ou les sources de données qui contiennent séparément les résultats d'une requête. L'article [Schwarte et al., 2011] propose des techniques d'optimisation des requêtes SPARQL fédérées par l'optimisation des jointures de la requête. Cette optimisation est faite par une remise en ordre des triplets de la requête et une redistribution des triplets de la requête en fonction des endpoints à interroger. Une implémentation de ces techniques dans le framework FedX est comparée à diverses solutions commerciales et montre des performances encourageantes dans la réduction du nombre de requêtes intermédiaires à envoyer pour la résolution d'une requête fédérée.

La norme SPARQL 1.1 définit depuis 2013 le mot-clé SERVICE permettant de spécifier un endpoint particulier pour des parties du corps d'une requête SPARQL fédérée, [Prud'hommeaux and Aranda, 2013]. Similairement au travail précédent, l'article [Wang et al., 2013] propose un système pour l'optimisation de l'envoi de requêtes SPARQL fédérées par la parallélisation du traitement des requêtes vers chaque base. Ce système découpe une

requête en différentes sous-requêtes en fonction de leur sélectivité et les repartit vers les différentes bases selon une heuristique basée sur leurs temps de latence.

Dans le domaine des résumés pour le Web des données, les différentes approches proposées ont chacune des objectifs différents. L'approche de [Fokoue et al., 2006] permet de créer un résumé des assertions des faits d'une base en fusionnant tous les individus de même classe en un seul individu selon leur ontologie. Les propriétés conservées par le résumé et sa petite taille font qu'il est ensuite plus aisé de vérifier la consistance d'une base par l'application d'un raisonneur sur le résumé.

L'article [Zhang et al., 2007] propose une méthode de construction de résumés d'ontologies (sans individu) pour faciliter le parcours de l'ontologie par un utilisateur. Cette méthode s'inspire de la génération de résumés textuels en effectuant le regroupement de triplets RDF en « phrases RDF ». Ces résumés sont destinés à des utilisateurs pour faciliter la compréhension du contenu d'une base en lui présentant une synthèse des triplets les plus représentatifs de l'ontologie.

L'approche présentée dans [Campinas et al., 2012] a pour but de faciliter l'écriture de requêtes SPARQL pour plusieurs bases. Dans cette approche, le résumé est utilisé pour proposer les ressources utilisables dans une requête pendant son écriture. Le résumé est généré à partir de l'ontologie représenter les relations possibles entre les nœuds typés par différentes classes. Cette approche a été testée uniquement sur les bases Sindice, [Campinas et al., 2011].

L'article [Fokoue et al., 2012] propose une utilisation des résumés proche de notre méthode. Il propose une optimisation pour l'envoi de requêtes à un ensemble de bases par l'utilisation d'un seul résumé pour un ensemble de bases. Ce résumé ne prend pas en compte les littéraux, il est généré en fonction des types des individus de la base et de leur passage dans une fonction de hashage qui modifie leurs URIs pour indiquer leurs sources. Le rapport de taille entre les ensembles de bases et les résumés se situe entre 0.037% – lorsque les bases ont beaucoup de données en commun – et 9.2% – lorsque les bases sont toutes différentes. Cette approche et la nôtre ont en commun le fait d'utiliser des résumés générés en fonction de l'ontologie. Cette approche diffère de la nôtre par le fait qu'elle propose de sélectionner les sources utiles à la résolution d'une requête simple alors qu'un aperçu de recherche sert de filtre à l'interrogation d'une base.

2 Aperçus de recherche

L'idée pour générer des aperçus de recherche d'un document RDF est d'une part de fusionner des individus selon leur type et d'autre part de réduire le nombre de littéraux en les remplaçant par des littéraux de substitution. La fusion des individus de même type permet de réduire le nombre de triplets car on fusionne également les triplets qui sont communs aux individus d'un même type.

Dans ce chapitre, on désigne par *type* d'un individu l'ensemble des classes les plus spécialisées auxquelles il appartient.

Génération d'aperçus de recherche

Pour notre fusion des individus et les littéraux selon leurs types, il est nécessaire que chaque individu et littéral d'un document RDF soit typé. Nous nommons les documents RDF remplissant ces conditions des documents RDF explicitement et simplement typés, définis en définition 4.1.

Définition 4.1 (Document RDF explicitement et simplement typé (REST))**.** *Un* document REST *est un document RDF où*

(i) chaque individu est explicitement typé par une ou plusieurs classes RDFS ;

(ii) l'ensemble des individus et leurs types sont disjoints ;

(iii) chaque littéral est explicitement typé.

En d'autres termes, un document REST décrit des individus, des littéraux, leurs relations et leurs types ; de plus, un individu (*i.e.* instance d'une classe) ne peut pas être à son tour un type pour d'autres individus[1] (à l'exception de la définition des classes et propriétés).

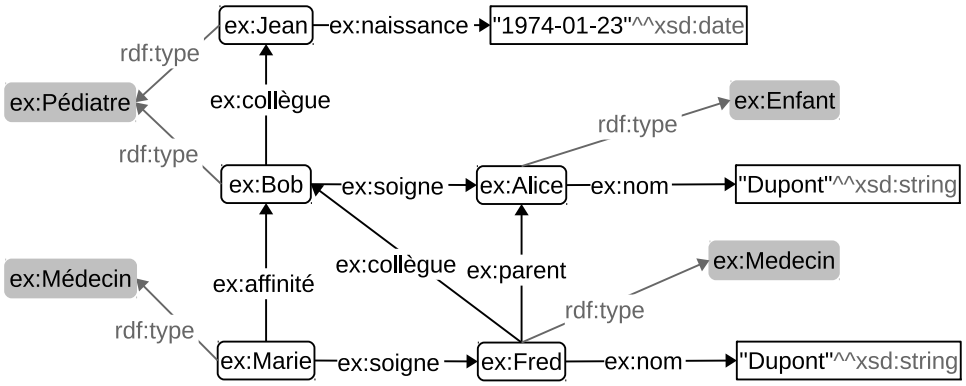


FIGURE 4.2 – Exemple de document REST.

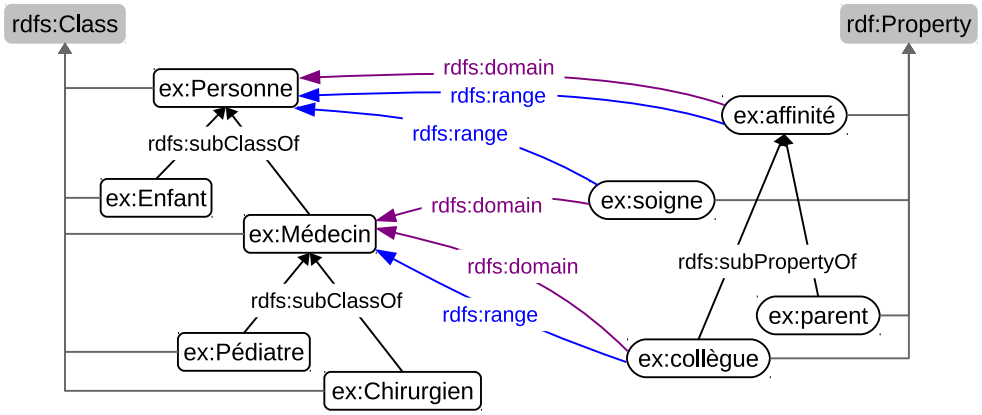


FIGURE 4.3 – Ontologie du document REST en figure 4.2.

La figure 4.2 et son ontologie en figure 4.3 est un exemple de document REST.

Rappelons que dans les ontologies RDFS (resp. OWL), les ressources d'une base sont par défaut instances de la classe rdfs:Resource (resp. owl:Thing) et les littéraux sont par défaut typés par le type de donnée xsd:anyType ou sont considérés comme étant de type xsd:string si aucun type n'est précisé. De plus pour des raisons pratiques, entre autres liées à la complexité des raisonnements, on sépare souvent en modélisation l'ontologie et les faits

[1] D'où le terme « simplement » de REST.

d'un document RDF, les individus et les classes sont donc en général disjoints [2]. Par conséquent, en pratique la plupart les documents RDF sont des documents REST.

Pour permettre le réglage fin de la compression des individus, nous définissons la *ligne de compression*. Cette ligne de compression est un ensemble de classes tirées de l'ontologie qui définit la manière selon laquelle les individus de la base vont être fusionnés.

Définition 4.2 (Ligne de compression d'individus)**.** *Soient C l'ensemble des classes d'une ontologie RDFS, et L un sous-ensemble non vide de C. L est appelée une* ligne de compression *de C si :*

- *pour toute classe $c \in C$ qui n'a pas de sous-classe, soit c est un élément de L, soit il existe une classe c' plus générale que c qui est un élément de L ;*
- *tous les éléments de L sont deux à deux hiérarchiquement incomparables (i.e. il n'y a pas dans L deux classes liées par la relation transitive* rdfs:subClassOf*).*

En d'autres termes, toutes les classes de l'ontologie sont représentées une et une seule fois, de manière directe (la classe est un élément de L) ou indirecte (il existe une classe plus générale ou plus spécialisée dans L) dans une ligne de compression.

Une ligne de compression constituée de toutes les classes les plus spécialisées d'une ontologie (*i.e.* les classes n'ayant pas de sous-classe) est appelée ligne de compression *maximale*. Une ligne de compression constituée uniquement des classes les plus générales (*e.g.*rdfs:Resource ou owl:Thing) est appelée ligne de compression *minimale*.

Par exemple sur l'ontologie en figure 4.2, les trois seules lignes de compression possibles sont : {ex:Personne }, {ex:Enfant, ex:Médecin } et {ex:Enfant, ex:Pédiatre, ex:Chirurgien } (*cf.*figure 4.4, la ligne de compression n°1 minimale, la ligne de compression n°2 et la ligne de compression n°3 maximale).

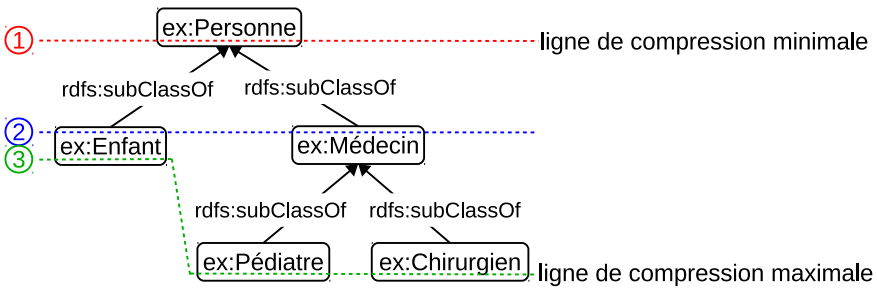


FIGURE 4.4 – Lignes de compression possibles de l'ontologie RDFS en figure 4.2.

La ligne de compression détermine un ensemble de classes dont les individus sont fusionnés lors de la génération de l'aperçu. Ces individus sont ceux dont les types font partie ou qui héritent d'une classe qui fait partie de la ligne de compression. Ces fusions réduisent fortement le nombre d'individus dans le document RDF tout en lui conservant certaines propriétés (*cf.* section 2). On peut dire que la ligne de compression détermine le niveau de détail de l'aperçu. Plus la ligne de compression va être proche de la ligne maximale, moins il y aura d'individus différents dans l'aperçu.

[2]Les classes sont liées par la relation d'instanciation à rdfs:Class ou owl:Class selon RDFS ou OWL, néanmoins cette notation est purement syntaxique, on ne considère pas les classes comme des individus de rdfs:Class ou owl:Class.

Dans un document RDF, les informations factuelles sont en grande partie portées par les littéraux (*e.g.* noms, dates, chiffres, etc) associés aux individus. Ces littéraux ne sont pas typés par l'ontologie et ne sont donc pas affectés par la ligne de compression dans les aperçus.

Pour pouvoir effectuer une compression des littéraux, nous définissons d'abord un *ensemble des cardinalités de types de littéraux* qui va nous permettre de définir le nombre de littéraux à obtenir dans un document après compression.

Définition 4.3 (Ensemble des cardinalités de types de littéraux)**.** *Soit T un ensemble de types de littéraux.*
Un ensemble R défini sur $T \times \mathbb{N}$ tel que $\forall (t,n) \in R$, $\nexists (t,n') \in R$ où $n \neq n'$ est appelé ensemble des cardinalités de T.

Un tel ensemble indique pour chaque type primitif le nombre de littéraux résultants dans l'aperçu.

À partir d'un ensemble des cardinalités de types de littéraux, nous définissons une *fonction de substitution de littéraux* pour déterminer quel est le résultat de la compression des littéraux de chaque type.

Définition 4.4 (Fonction de substitution de littéraux de type t)**.** *Soient t un type de littéraux, P_t l'ensemble ordonné de littéraux de type t et $\overline{P_t} \subseteq P_t$ un sous-ensemble de P_t, et $n \in \mathbb{N}$ le nombre de littéraux de substitution pour le type t.*
Une fonction $s_t^n(p) : P_t \to \overline{P_t}$ telle que :

- $\forall p, p' \in P_t$, *si* $p < p'$ *alors* $s_t^n(p) \leq s_t^n(p')$;
- *si* $0 < n < |P_t|$ *alors* $|\overline{P_t}| = n$ *et* $\forall \overline{p} \in \overline{P_t}$, $\exists p \in P_t$ *tel que* $s_t^n(p) = \overline{p}$;
- *sinon* $s_t^n(p) = p$.

est appelée fonction de substitution des littéraux de type t.

Une fonction de substitution des littéraux s_t^n remplace tous les littéraux de type t par n littéraux de substitution en conservant les inégalités partielles entre les littéraux de substitution.

Par exemple selon une fonction s^2_{string}, pour les littéraux "e", "m" et "w" de type string, l'image de "e" et "m" pourrait être "a" et l'image de "w" pourrait être "p" (où on a "a" < "p" de même que "e" < "w" et "m" < "w"). Dans un exemple plus concret, dans un document traitant de l'histoire de France qui détaillerait particulièrement les guerres napoléoniennes, la fonction de substitution pourrait regrouper toutes les dates des guerres napoléoniennes dans plusieurs groupes de façon à garder une certaine finesse dans les zones les plus riches en événements et répartir les autres dates dans seulement quelques groupes. Ainsi, la définition d'une fonction de substitution permet d'adapter la granularité de la réduction de littéraux pour un type particulier, voire sur une plage de valeurs données. D'autres exemples de fonctions de substitution pourraient être le remplacement des chaînes de caractères par leur première lettre, les dates par leur année ou les chiffres par le multiple de 100 inférieur le plus proche.

La ligne de compression et la fonction de substitution sont les paramètres qui nous permettent de régler la granularité de l'aperçu de recherche. La transformation d'un document RDF en aperçu est assurée par l'application des règles suivantes en définition 4.5.

Définition 4.5 (Aperçu de recherche)**.** *Soient D un document REST défini sur une ontologie RDFS O, L une ligne de compression d'individus de O, R un ensemble des cardinalités des types de littéraux de D, et un ensemble S de fonctions de substitution $s_{t_i}^{n_i}$ définies pour tout $(t_i, n_i) \in R$.*

Un aperçu de recherche *de D selon L, R et S est défini par dérivation de D en appliquant les règles ci-dessous, les* règles de réduction*, appliquées jusqu'à réduction maximale (i.e. jusqu'à ce qu'aucune règle ne puisse plus être appliquée) :*

r1 *spécialiser un individu typé plus généralement qu'un élément de L.*

Si un individu i est de type c_1, tel que i n'a pas de classe plus spécialisé que c_1 parmi ses types et c_1 est plus général qu'un élément c_2 de L, alors i devient aussi de type c_2 par ajout d'un triplet de prédicat rdf:type *entre i et c_2.*

r2 *fusionner deux individus distincts (*) ayant un même type.*

Si deux individus i_1 et i_2 partagent un même type c tel qu'il n'y a pas de classe plus spécialisée que c parmi les types de i_1 et i_2, alors i_1 et i_2 sont fusionnés.

r3 *fusionner deux individus distincts (*) dont les types appartiennent à L.*

Si c_1 et c_2 sont respectivement des types des individus i_1 et i_2 tels qu'il n'y a pas de classes plus spécialisées que c_1 et c_2 parmi les types de i_1 et i_2 et tel que $c_1 \in L$ et $c_2 \in L$, alors i_1 et i_2 sont fusionnés.

r4 *supprimer une relation redondante, i.e. supprimer les triplets contenant une relation entre deux nœuds*[3] *tel qu'il existe une autre relation plus spécifique entre ces deux mêmes nœuds.*

r5 *remplacer chaque littéral l de type t_{lit} avec $(t_{lit}, n) \in R$ par son littéral de remplacement $s_{t_{lit}}^{n}(l)$.*

(*) Deux individus distincts fusionnés signifie qu'ils sont remplacés par un même nœud blanc dans tous les triplets de A.

On appellera les règles **r1** à **r4** les *règles de réduction d'individu* et la règle **r5** la *règle de réduction de littéraux*. Les règles **r1** à **r3** fusionnent les individus selon leurs types et selon la ligne de compression. La règle **r4** supprime les triplets devenus redondants à cause des fusions. On notera que la règle **r5** peut être appliquée une seule fois en phase d'initialisation.

Exemple 11. *L'aperçu généré à partir du document en figure 4.2 et selon la ligne de compression n°1 de la figure 4.4 – {*ex:Personne *} – est représentée en 4.5. Cet aperçu contient 3 triplets de moins que son document d'origine à cause de la fusion de certains individus – le déroulement de la génération est détaillé en section 3 –.*

Dans cet exemple, si une requête qui interroge la base en nommant un individu précis – autre que ex:Alice *– ne peut pas avoir de résultats dans cet aperçu du fait de la fusion*

[3]Rappel : un nœud est soit un individu, soit un nœud blanc, soit un littéral

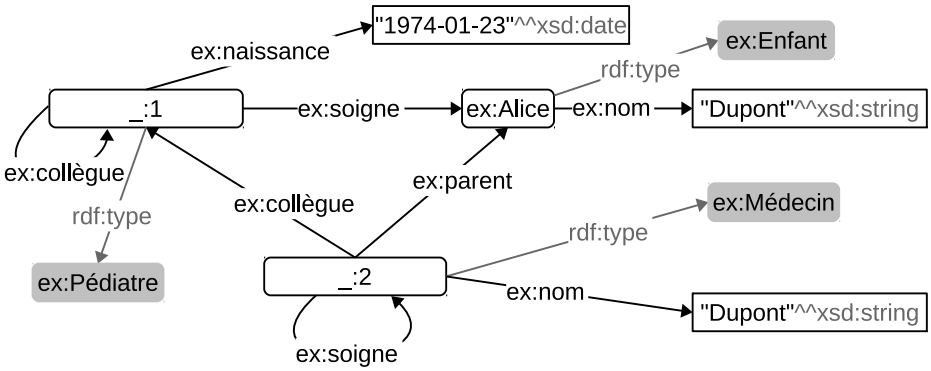


FIGURE 4.5 – Aperçu du document en figure 4.2 selon la ligne de compression n°1 de la figure 4.4.

des individus. Pour que les requêtes qui obtiennent des résultats dans la base obtiennent également des résultats dans l'aperçu, il est nécessaire de les réécrire.

Propriétés des aperçus de recherche

La construction selon la définition 4.5 d'un aperçu d'un document REST conduit immédiatement aux propriétés suivants.

Propriété 4.6 (Aperçu et document REST)**.** *Un aperçu (d'un document REST) est un document REST.*

Cette propriété, conservée par construction, présente un avantage pour l'exploitation d'un aperçu par les mêmes systèmes que ceux des bases.

Propriété 4.7 (Taille d'un aperçu)**.** *La taille d'un aperçu d'un document REST est plus petite ou égale que la taille du document.*

Nous désignons par taille d'un document RDF – et donc d'un document REST – le nombre de ses triplets.

La réécriture des requêtes transmises à l'aperçu est définie en définition 4.8. Cette réécriture a pour but d'éliminer les parties des requêtes destinées à un document REST et qui ne peuvent pas obtenir de résultats dans son aperçu de recherche.

Définition 4.8 (Réécriture en $Q^\star$ pour l'interrogation d'un aperçu)**.** *Soient D un document REST, V un aperçu de D, et Q une requête SPARQL.*

Soit $Q^\star$ la réécriture de Q selon les étapes suivantes :

(i) chaque URI d'individu (donc ni URI de prédicat, ni URI de type) est remplacée par une variable qui lui est propre ;

(ii) un filtre qui contient une variable qui n'apparaît pas dans le corps de requête, i.e. la clause `WHERE`*, est supprimé ;*

(iii) un filtre qui contient un test d'inégalité stricte, i.e. $>$ ou $<$, entre deux éléments est remplacé par un test d'inégalité large, respectivement $>=$ ou $<=$;

(iv) un filtre qui contient un test d'inégalité, i.e. $!=$, ou une négation, i.e. ! ou `NOT`*, est supprimé ;*

(v) les clauses `OPTIONAL` *et « Solution Modifiers » sont enlevées ;*

(vi) les littéraux sont remplacés selon la même fonction de substitution que celle utilisée lors de la génération de V.

Cette réécriture assure la propriété 4.9 car elle « généralise » une requête en remplaçant certains URIs par des variables et en supprimant ou modifiant certains filtres. Les étapes (i) et (vi) remplacent les URIs d'individus et les littéraux de la requête pour refléter les transformations effectuées dans l'aperçu. Les étapes (ii), (iii) et (iv) suppriment ou remplacent les filtres rendus obsolètes par la substitution des littéraux et la fusion des individus. Les filtres visés par ces étapes sont ceux qui utilisent des fonctions d'agrégation ou qui définissent des restrictions sur les valeurs des variables. L'étape (v) supprime les clauses `OPTIONAL`, inutiles pour vérifier s'il existe un résultat dans l'aperçu.

Propriété 4.9 (Résultat de $Q^\star$)**.** *Si l'interrogation par $Q^\star$ sur V retourne comme résultat l'ensemble vide, alors le résultat de l'interrogation par Q sur D est l'ensemble vide.*

Notons que l'ensemble vide[4] comme résultat à une interrogation signifie que le document RDF ne contient pas de ressource satisfaisant le corps de la requête.

Éléments de preuve :

- une variable peut correspondre à toute instance – *i.e.* individu identifié par un URI ou un nœud blanc –, hors considération du/des types ;
- une instance d'un type t_2 peut correspondre à une instance de type t_1, où t_1 est plus général que t_2 ;
- les littéraux sont substitués de façon à ce que les inégalités larges entre littéraux soient conservées pour leurs substituts. □

En d'autres termes, la requête $Q^\star$ est plus générale que Q.

Nous arrivons ici au cœur de notre approche, où face à la multitude de bases RDF présentes sur le Web des données nous interrogeons d'abord les aperçus de recherche des bases pour déterminer si une source est pertinente. Si une source est pertinente – *i.e.* si elle peut contenir un résultat à la requête – alors nous l'interrogeons concrètement et totalement (*cf.* schéma de principe en figure 4.1).

Les « *Solution Modifiers*[5] » (*e.g.*`ORDER BY`, `LIMIT` ou `OFFSET`) dans une requête SPARQL servent uniquement à la présentation des résultats. Ainsi, la réécriture de $Q^\star$ en épurant de Q ses « *Solution Modifiers* » présente une optimisation simple quant au temps d'interrogation de $Q^\star$ au vu de l'usage que nous souhaitons faire d'un aperçu : savoir si la source est non pertinente pour la résolution de la requête. Les résultats à une requête Q envoyée à un document RDF, s'il y en a, sont potentiellement beaucoup plus nombreux que les résultats de Q dans son aperçu, du fait de la réduction du nombre d'individus et de littéraux, donc les *Solution Modifiers* modifiant le nombre de résultats à afficher (*e.g.*`LIMIT` et `OFFSET`) sont inadaptés dans $Q^\star$.

[4] ensemble vide pour les requêtes SELECT, CONSTRUCT et DESCRIBE, false pour les requêtes ASK.
[5] http://www.w3.org/TR/sparql11-query/#solutionModifiers

Propriété 4.10 (Réécriture en $A^\star$)**.** *Soient D un document REST, V un aperçu de D, et Q une requête SPARQL.*

Soit $A^\star$ la réécriture de Q suivant les mêmes étapes de la définition 4.8 avec l'étape supplémentaire :

(vii) L'en-tête de la requête est transformée en en-tête de requête `ASK`*.*

Si l'interrogation par $A^\star$ sur V est négative (retourne `false`*), alors le résultat de l'interrogation par Q sur D est vide.*

En SPARQL, une interrogation de type `ASK` répond uniquement par `true` ou `false` s'il y a un résultat à l'interrogation ou non. Cette caractéristique coïncide exactement à l'utilisation que nous souhaitons faire d'un aperçu. De plus, le temps d'exécution pour obtenir une réponse en interrogation une base par une requête `ASK` est plus rapide que pour « la même » requête `SELECT`.

Le schéma global en figure 4.1 de fonctionnement de notre approche en recherche d'information peut donc être légèrement revu et optimisé en adoptant cette fois-ci le schéma global en figure 4.6.

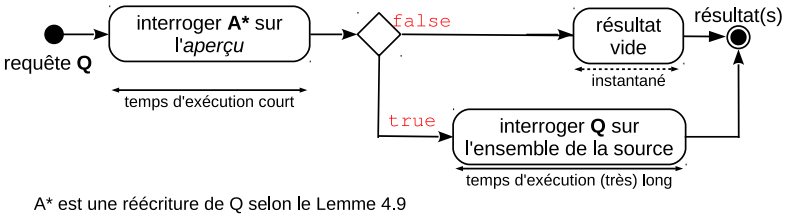


FIGURE 4.6 – Schéma global d'utilisation d'un aperçu, « version `ASK` ».

3 Exemples de génération et d'utilisation d'aperçus de recherche

Pour illustrer la génération et l'utilisation d'aperçus pour l'interrogation de documents RDF, nous présentons trois séries d'exemples pédagogiques. Nous présentons d'abord des exemples de génération d'aperçus sans aborder la réduction des littéraux, ensuite un exemple de réduction de littéraux et enfin des exemples d'interrogations de document RDF passant par les aperçus générés précédemment.

3.1 Générations d'aperçus par application des règles de réduction d'individus (définition 4.5)

Exemple de génération d'aperçu n°1 La partie du bas de la figure 4.7 présente l'aperçu de recherche dérivé du document REST en figure 4.2 – repris ici en partie haute. En effet, par application des règles de réduction des individus :

r2 ⇒ les individus ex:Bob et ex:Jean, tous deux de type ex:Pédiatre, sont fusionnés en un même nœud blanc, numéroté ici par 1 (*i.e.* _:1), ce qui entraîne la disparition d'une relation d'instanciation entre _:1 et ex:Pédiatre ; (a)

r2 ⇒ de même pour les individus ex:Marie et ex:Fred de type ex:Médecin qui sont fusionnés en _:2, ce qui entraîne la disparition de la relation d'instanciation entre _:2 et ex:Médecin ; (b)

r4 ⇒ la relation ex:affinité entre _:2 et _:1 est supprimée, car il existe une relation plus spécifique ex:collègue entre ces deux individus ; (c)

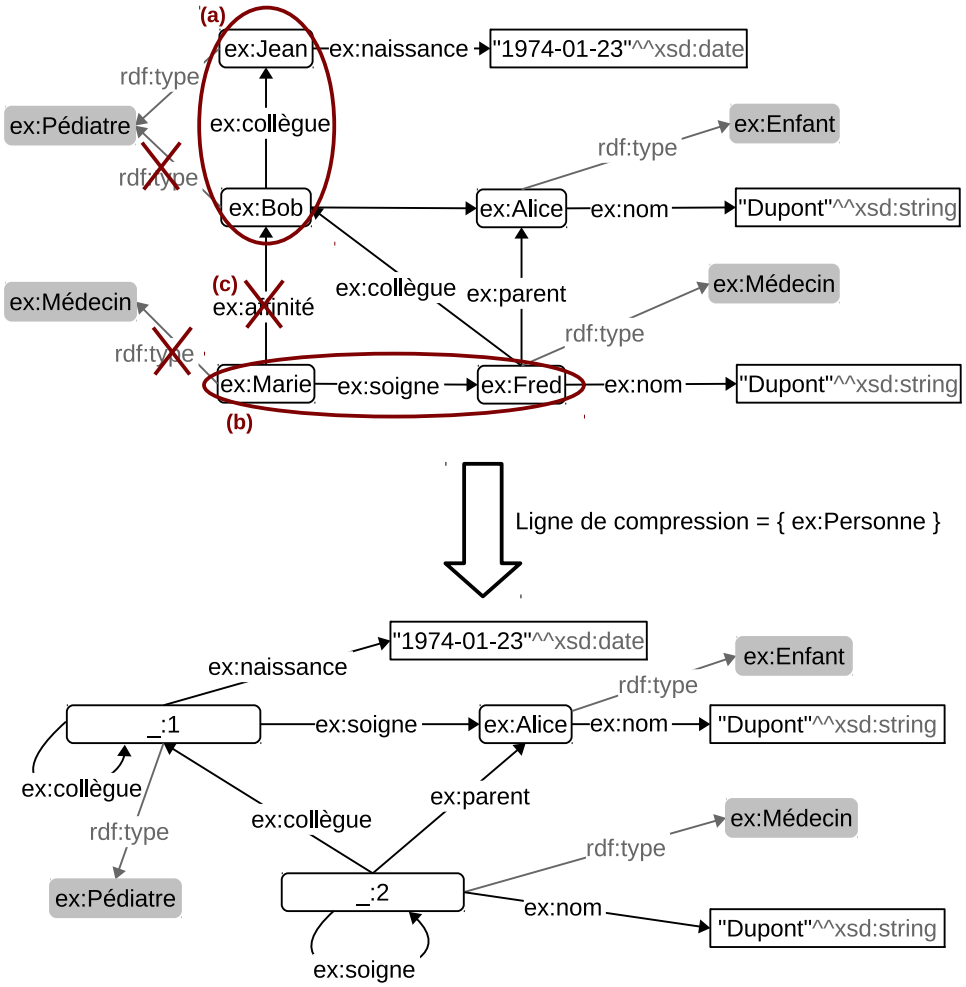


FIGURE 4.7 – Aperçu selon la ligne de compression minimale {ex:Personne } (n°1).

Exemple de génération d'aperçu n°2 La figure 4.8 présente l'aperçu de recherche dérivé du document REST en figure 4.2 selon la ligne de compression {ex:Enfant, ex:Médecin }. Cette fois-ci, par rapport à l'aperçu précédent, la règle **r3** peut en plus être appliquée :

r3 ⇒ les individus ex:Marie, ex:Fred et ex:Alice sont fusionnés en un même nœud blanc (_:2), car leurs types ex:Enfant ou ex:Médecin appartiennent à la ligne de compression ;

Il en résulte qu'une des deux relations ex:soigne entre _:2 et _:2 et une des deux relations ex:nom entre _:2 et le littéral "Dupont" sont supprimées.

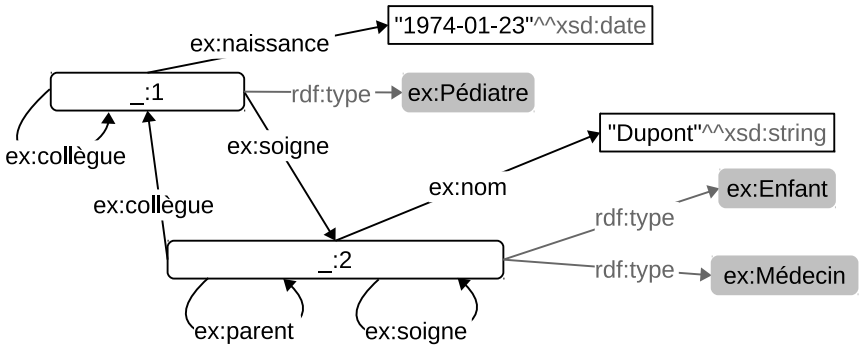


FIGURE 4.8 – Aperçu selon la ligne de compression {ex:Enfant, ex:Médecin } (n°2).

Exemple de génération d'aperçu n°3 La figure 4.9 présente le dernier aperçu de recherche possible, celui dérivé selon la ligne de compression maximale {ex:Enfant, ex:-Pédiatre, ex:Chirurgien }. Cette fois-ci par rapport aux aperçus 1 et 2, la règle **r1** peut en plus être appliquée :

r1 ⇒ les individus ex:Marie et ex:Fred, initialement de type ex:Médecin, sont tous les deux typés en ex:Pédiatre ;

r1 ⇒ idem, mais cette fois-ci avec le type ex:Chirurgien (ex:Marie et ex:Fred sont donc multi-typés en ex:Pédiatre et ex:Chirurgien à cette étape).

r2 ⇒ les individus ex:Jean, ex:Bob, ex:Marie et ex:Fred de type ex:Pédiatre sont fusionnés en le nœud blanc _:1 de types ex:Pédiatre et ex:Chirurgien ;

r3 ⇒ les individus _:1 et ex:Alice sont fusionnés en un même nœud blanc (_:2), car leurs types ex:Enfant ou ex:Pédiatre et ex:Chirurgien appartiennent à la ligne de compression ;

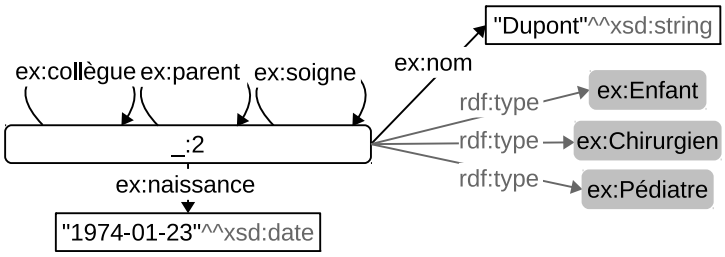


FIGURE 4.9 – Aperçu selon ligne de compression maximale {ex:Enfant, ex:Pédiatre, ex:Chirurgien } (n°3).

On observe dans les 3 aperçus que le nombre d'individu diminue en fonction de la ligne de compression. De 5 individus dans le document d'origine, on passe à 3 individus dans l'aperçu 1, 2 individus dans l'aperçu 2 et 1 seul individu dans l'aperçu 3. Cet exemple montre aussi que nombre de triplets dépend de la ligne de compression, on perd 28% de triplets dans les aperçus 1 et 2 et 42% dans l'aperçu 3.

D'un point de vue humain, on constate que les aperçus de recherche deviennent de plus en plus illisibles pour un observateur humain au fur et à mesure que la ligne de compression descend dans l'ontologie. L'aperçu de recherche 3 illustre bien cette difficulté de lire les aperçus de recherche puisque la fusion de tous les individus a créé un individu Enfant-Pédiatre-Chirurgien qui est collègue, soigneur et parent de lui-même.

3.2 Génération d'aperçu par application de la règle de réduction de littéraux (définition 4.5)

Supposons que la partie gauche de la figure 4.10 soit le résultat intermédiaire d'une génération d'aperçu (par application des règles de réduction d'individu) d'un document RDF décrivant 3 personnes (individus de type ex:Personne pour lesquels sont renseignés les noms et prénoms). La partie droite de la figure 4.10 est le résultat final de la génération de l'aperçu, en tenant compte en plus de la règle de réduction de littéraux selon une fonction de substitution décrite dans le paragraphe suivant.

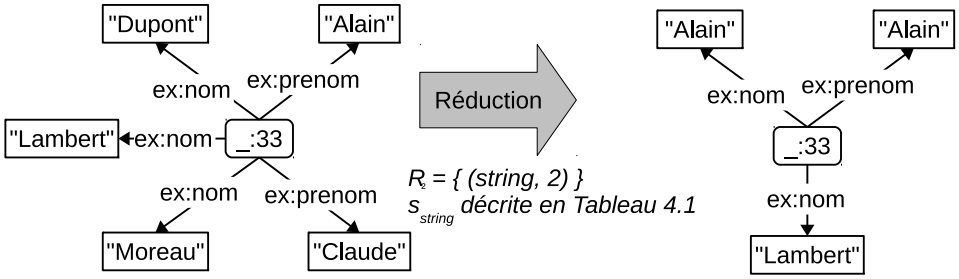


FIGURE 4.10 – À gauche, résultat de la fusion de 3 individus de type ex:Person avant réduction des littéraux, à droite, résultat de la fusion après réduction des littéraux en 2 groupes.

Ici la fonction de substitution utilisée pour la réduction des littéraux de ce document est décrite en tableau 4.1 : une fois tous les littéraux triés par ordre croissant et répartis en deux groupes, chaque littéral du document est remplacé (substitué) par le premier littéral de son groupe.

Groupe 1	Groupe 2
Alain	*Lambert*
Claude	Moreau
Dupont	

TABLE 4.1 – Description de la fonction de substitution de littéraux utilisée en figure 4.10

r5 ⇒ Les littéraux "Alain", "Claude" et "Dupont" sont remplacés par le littéral "Alain".

r5 ⇒ Les littéraux "Lambert" et "Moreau" sont remplacés par le littéral "Lambert".

Par ces substitutions sur ce simple exemple, le nombre de relations liant l'individu à des littéraux a diminué, passant de 5 à 3.

3.3 Interrogations de bases RDF passant par leurs aperçus associés (propriété 4.10)

Les exemples suivants montrent l'interrogation du document RDF D, représenté en figure 4.2. Ces exemples détaillent l'interrogation de D via ses aperçus (*cf.* section 3.1). Pour cela, deux requêtes sont utilisées, Q_1, figure 4.11, pour laquelle D ne contient pas de résultat et Q_2, figure 4.14, pour laquelle D contient des résultats.

L'utilisation normale d'aperçus de recherche se faire avec un seul aperçu par base, les 3 aperçus générés précédemment sont présentés ici ensembles à titre d'exemple.

Interrogation par une requête Q_1 dans D. La requête Q_1 recherche les médecins soignants des pédiatres. L'interrogation de D par la requête Q_1 passe par l'interrogation intermédiaire d'un aperçu (figure 4.6).

```
SELECT ?sujet ?objet
WHERE { ?sujet rdf:type ex:Médecin .
  ?objet rdf:type ex:Pédiatre .
  ?sujet ex:soigne ?objet .
}
```

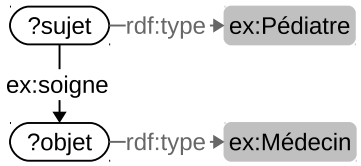


FIGURE 4.11 – Requête SPARQL Q_1 (à droite, sa représentation graphique) : Médecins soignants des pédiatres

Considérons maintenant l'aperçu de D généré selon la ligne de compression minimale, que l'on appellera aperçu n°1, représenté en figure 4.7. La requête Q_1 est au préalable réécrite en $Q_1^\star$, non représentée ici, selon la définition 4.8, puis en $A_1^\star$ (*cf.*figure 4.12) selon la propriété 4.10. L'interrogation de l'aperçu n°1 par $A_1^\star$ retourne une réponse négative

```
ASK { ?sujet rdf:type ex:Médecin .
  ?objet rdf:type ex:Pédiatre .
  ?sujet ex:soigne ?objet . }
```

FIGURE 4.12 – Requête Q_1 réécrite en $A_1^\star$.

(`false`). On en conclut (toujours par la propriété 4.10) qu'il n'y a pas de résultat à Q_1 dans D. On peut en effet constater qu'il n'y a pas de résultat à Q_1 dans D.

Considérons ensuite l'aperçu de D généré selon la ligne de compression {ex:Enfant, ex:Médecin }, que l'on appellera aperçu n°2, représenté en figure 4.8. De même que pour l'aperçu n°1, l'interrogation de l'aperçu n°2 par $A_1^\star$ retourne une réponse négative qui permet de conclure qu'il n'y a pas de résultats à Q_1 dans D. Il n'est donc pas nécessaire d'interroger D.

Considérons enfin l'aperçu généré selon la ligne de compression maximale, que l'on appellera aperçu n°3, représentée en figure 4.9. L'interrogation de l'aperçu n°3 par $A_1^\star$ retourne une réponse positive (`true`). On en conclut la présence *possible* de résultats à Q_1 dans D bien qu'il n'y en ait pas. Nous appelons ceci un *faux positif*.

Remarquons que ce faux positif est créé par le fait que la génération de cet aperçu a fusionné tous les individus en un seul, typé à la fois par ex:Médecin et ex:Chirurgien et lié à lui-même par une relation ex:soigne.

La figure 4.13 montre le résultat de l'interrogation de l'aperçu n°3 par la requête $Q_1^\star$.

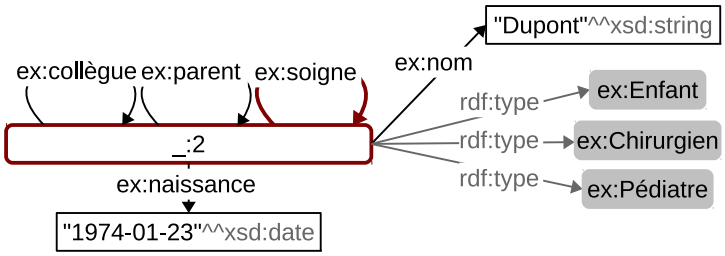


FIGURE 4.13 – Résultats (en traits épais) de la Requête Q_1 dans l'aperçu n°3 de D.

Interrogation par une requête Q_2 dans la base D. La requête Q_2, figure 4.11, recherche les médecins collègues de pédiatres. Quelle que soit la ligne de compression,

l'interrogation de chacun des trois aperçus de D par $A_2^\star$ (en figure 4.15), réécriture de Q_2, retourne une réponse positive qui signale la présence possible de résultats à la requête Q_2. La figure 4.16 et le tableau 4.2 présentent les résultats de l'envoi de la requête $Q_2^\star$ sur ces aperçus.

```
SELECT ?sujet ?objet
WHERE { ?sujet rdf:type ex:Médecin .
  ?objet rdf:type ex:Pédiatre .
  ?sujet ex:collègue ?objet .
}
```

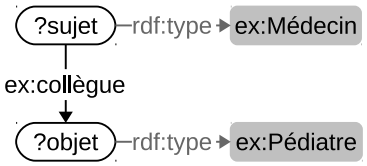


FIGURE 4.14 – Requête SPARQL Q_2 (à droite, sa représentation graphique) : Médecins collègues de pédiatres.

```
ASK { ?sujet rdf:type ex:Médecin .
  ?objet rdf:type ex:Pédiatre .
  ?sujet ex:collègue ?objet . }
```

FIGURE 4.15 – Requête Q_2 réécrite en $A_2^\star$.

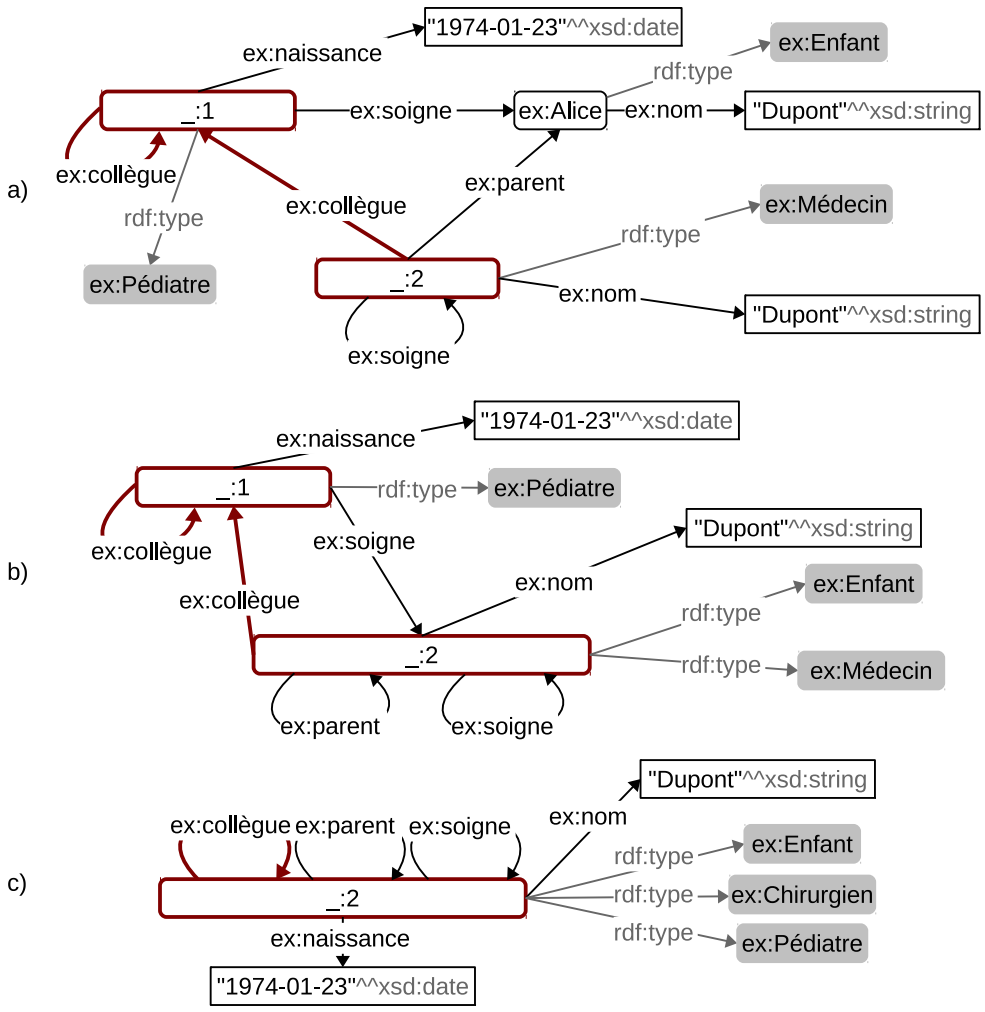


FIGURE 4.16 – Représentation des résultats de $Q_2^\star$ dans les différents aperçus (lignes n°1, 2 et 3).

L'interrogation du document D représenté en figure 4.2 par la requête Q_2 (figure 4.14) fournit en effet deux résultats (figure 4.17).

?sujet	?objet
_:1	_:1
_:1	_:2

Aperçu n°1

?sujet	?objet
_:1	_:1
_:1	_:2

Aperçu n°2

?sujet	?objet
_:1	_:1

Aperçu n°3

TABLE 4.2 – Résultats de $Q_2^\star$ dans les différents aperçus.

?sujet	?objet
ex:Bob	ex:Jean
ex:Fred	ex:Bob

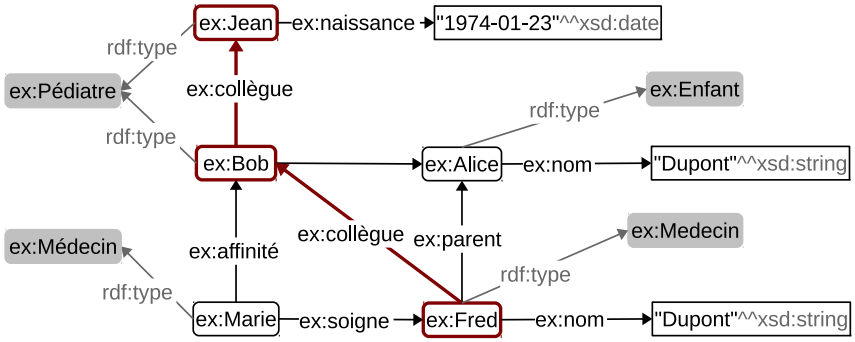


FIGURE 4.17 – Résultats de la requête Q_2 dans la base D.

4 Expérimentation multi-bases

Notons que le générateur proposé ici est mono-processus, une méthode de parallélisation de la génération d'aperçu est proposée en chapitre 5.

4.1 Jeu de données

Afin d'éviter tout biais dû aux aléas des temps d'accès réseau, mais ne pouvant pas charger en mémoire sur une machine standard aujourd'hui l'intégralité du Linked Data, nous avons extrait un sous-ensemble de la base généraliste DBPedia. Ce sous-ensemble a ensuite été réparti en 200 bases composées chacune d'environ 10 000 triplets, contenant des informations plus ou moins diversifiées autour des sept thématiques suivantes (*i.e.* centrées autour des classes) : dbponto:CelestialBody, dbponto:City, dbponto:Country, dbponto:Event, dbponto:Film, dbponto:Organisation et dbponto:Person. Notons que ces bases ont été extraites selon la méthode d'extraction détaillée dans [Maillot et al., 2014] afin de créer une sorte de « maquette du Linked Data Cloud », chargeable en mémoire. Cette maquette du Linked Data a été conçue pour reproduire un ensemble de bases RDF contenant des informations de domaines différents tout en partageant quelques URIs communes, d'une façon similaire au Linked Data Cloud. Les requêtes utilisées ont été choisies afin d'évaluer les performances des aperçus dans les situations les plus *courantes* [Gallego et al., 2011] et en situation de résolution de requêtes dites *difficiles* [Pérez et al., 2009, Schmidt et al., 2010].

Pour chaque base, nous avons généré trois aperçus différents selon trois lignes de compressions – représentées en figure 4.18 – passant à des profondeurs différentes dans la hiérarchie des classes de DBPedia, définies en OWL :

- La ligne de profondeur minimale contenant uniquement la classe owl:Thing la plus générale de l'ontologie, dont les aperçus seront appelés *aperçus A*.
- La ligne de profondeur 1 contenant les classes dbponto:CelestialBody, dbponto:City, dbponto:Country, dbponto:Event , dbponto:Organisation, dont les aperçus seront appelés *aperçus*

B.

- La ligne de profondeur 2 contenant la dernière thématique dbponto:Film et toutes les sous-classes des classes de la ligne de profondeur 1, dont les aperçus seront appelés *aperçus C*.

Dans tous les aperçus, les littéraux ont été compressés par une fonction réduisant tous les littéraux d'un même type à 10 valeurs.

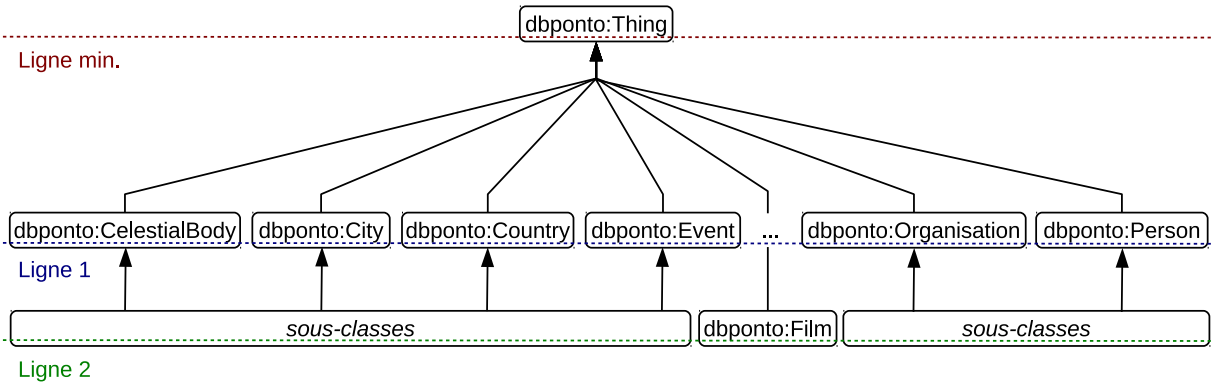


FIGURE 4.18 – Représentation (partielle) des lignes de compressions utilisées sur l'ontologie DBPedia

Le tableau 4.3 montre par thématique le nombre moyen de triplets dans les bases et dans les aperçus associés. Dans l'ensemble, les aperçus des bases ont une taille 10 fois inférieure à celle de leur source. Cette différence de taille va permettre d'interroger plus rapidement les aperçus que leurs sources. La faible taille des aperçus va également faciliter leur mise en place dans une structure centralisée, c'est-à-dire tous sur un même serveur de distribution de requêtes. Dans les expériences effectuées dans [Raimbault and Maillot, 2013], la génération d'aperçu sans compression des littéraux a permis une compression de seulement 55% environ. La réduction de taille des aperçu provient donc autant de la réduction du nombre d'individus dans le document que de la réduction du nombre de littéraux qui leurs sont liés.

On observe que les aperçus B et C peuvent parfois être de taille légèrement plus grande que celles des aperçus de profondeurs minimum. Cela est dû aux spécialisations apportées par la règle de réduction **r1**, qui ajoute de nouvelles relations de typage dans la base en fonction de la ligne de compression (la ligne *2* contient les 53 sous-classes des classes de la ligne 1).

4.2 Requêtes SPARQL

Les requêtes SPARQL utilisées sont listées dans le tableau 4.4. Ces requêtes ont été choisies pour tester l'effet de l'utilisation des aperçus dans les situations « habituelles » ou « extrêmes ». Elles peuvent être séparées en deux groupes :

- Les requêtes « courantes », comme les requêtes 1 à 3, selon l'étude faite par [Gallego et al., 2011] sur des requêtes soumises à DBPedia pendant un an. Ces requêtes utilisent des motifs dits *en étoile*, recherchant les informations autour d'un individu central ; le nombre de triplets qu'elles contiennent est compris entre 1 et 5.

Thématiques principales des bases	Nb. moyen de triplets dans les bases	Nb. moyen de triplets dans les aperçus A	Nb. moyen de triplets dans les aperçus B	Nb. moyen de triplets dans les aperçus C	Taux de compression moyen par thématique
CelestialBody	10 313	386	409	417	96,08 %
City	10 759	883	888	893	91,75 %
Country	11 373	1 041	1 053	1 062	90,75 %
Event	11 804	910	918	923	92,23 %
Film	13 607	795	793	797	94,16 %
Organisation	13 149	1 273	598	1 278	90,30 %
Person	12 052	1 319	1 323	1 332	89,01 %
Total	2 376 565	190 054	191 447	192 652	
Taux de compression moyen (%)		92,00 %	91,94 %	91,89 %	

TABLE 4.3 – Informations sur les bases de test et leurs aperçus

- Les requêtes « difficiles » d'après les articles [Pérez et al., 2009, Schmidt et al., 2010], comme
  - les requêtes 4 à 6 contenant le mot-clé `UNION`.
  - les requêtes 7 et 8 contenant le mot-clé `OPTIONAL`.

La résolution de ces requêtes SPARQL contenant des (conjonctions de) disjonctions portées par les opérateurs `UNION` et `OPTIONAL` a été prouvée dans [Schmidt et al., 2010] comme étant co-NP-complète.

4.3 Mise en place des tests

Chacune des 200 bases RDF a été montée avec ses aperçus sur un serveur Fuseki, basé sur Jena [Apache Software Foundation, 2012], où bases et aperçus sont accessibles par un *endpoint* SPARQL propre.

Pour chaque requête listée au tableau 4.4, les bases ont été interrogées une à une via son aperçu A (minimal) selon le schéma de principe en figure 4.6, puis de même avec l'aperçu B, et enfin avec l'aperçu C.

Afin d'écarter de nos mesures les temps d'accès réseau, tous ces serveurs ont été hébergés sur une même machine physique[6], donc exploitable en local. Notons qu'en plus de cela, puisque l'on utilise un protocole réseau pour interroger les bases, les temps d'accès – bien que minimes – aux aperçus et aux bases n'ont pas été pris en compte (*i.e.* temps soustraits des temps d'interrogation) afin de ne retenir que les temps calculatoires d'interrogation.

4.4 Résultats

Les temps totaux d'exécution, hors temps d'accès réseau, des différentes requêtes sur les aperçus et les bases sont présentés dans le tableau 4.5.

[6]Machine comportant un processeur Intel Core 2 cadencé à 2,40GHz, 4Go de mémoire RAM, et tournant sous Ubuntu 12.04.

Requête originale Q	Requête $A^\star$
q1	

`SELECT DISTINCT ?cible ?pred ?sujet`
`WHERE { ?cible ?pred ?sujet .`
`  FILTER( isLiteral(?sujet) ) }`
	`ASK { ?cible ?pred ?sujet . }`

q2	

`SELECT DISTINCT ?name`
`WHERE { ?cible dbponto:name ?name .`
`  ?cible dbponto:starring ?actor .`
`  FILTER(?actor = dbpedia:Robert_Downey,_Jr.)`
`}`
	`ASK { ?cible dbponto:name ?name .`
	`  ?cible dbponto:starring ?actor . }`

q3	

`SELECT DISTINCT`
`  ?abstract ?death ?birth ?quote`
`WHERE { ?person dbponto:name "Lucretius"@en .`
`  ?person dbponto:abstract ?abstract .`
`  ?person dbponto:deathYear ?death .`
`  ?person dbponto:birthYear ?birth .`
`  ?person dbponto:quote ?quote . }`
	`ASK { ?person dbponto:name "Lucretius"@en .`
	`  ?person dbponto:abstract ?abstract .`
	`  ?person dbponto:deathYear ?death .`
	`  ?person dbponto:birthYear ?birth .`
	`  ?person dbponto:quote ?quote . }`

q4	

`SELECT DISTINCT`
`  ?cible ?pred ?sujet`
`WHERE { ?cible ?pred ?sujet .`
`  { ?cible a dbponto:Person .`
`  ?sujet a dbponto:City . }`
`  UNION { ?sujet a dbponto:Person .`
`  ?cible a dbponto:City . } }`
	`ASK { ?cible ?pred ?sujet .`
	`  { ?cible a dbponto:Person .`
	`  ?sujet a dbponto:City . }`
	`  UNION { ?sujet a dbponto:Person .`
	`  ?cible a dbponto:City . } }`

q5	

`SELECT DISTINCT`
`  ?cible ?pred ?sujet ?syno`
`WHERE { ?cible ?pred ?sujet .`
`  { ?cible dbponto:placeOfDeath ?sujet . }`
`  UNION { ?cible dbponto:leaderName ?sujet .`
`  OPTIONAL {`
`  ?sujet owl:sameAs ?syno . } } }`
	`ASK { ?cible ?pred ?sujet .`
	`  { ?cible dbponto:placeOfDeath ?sujet .`
	`  } UNION {`
	`  ?cible dbponto:leaderName ?sujet .`
	`  } }`

q6	

`SELECT DISTINCT`
`  ?cible ?pred ?sujet`
`WHERE { ?cible ?pred ?sujet .`
`  { ?cible dbponto:placeOfDeath ?sujet . }`
`  UNION {`
`  ?cible dbponto:leaderName ?sujet . } }`
	`ASK { ?cible ?pred ?sujet .`
	`  { ?cible dbponto:placeOfDeath ?sujet . }`
	`  UNION {`
	`  ?cible dbponto:leaderName ?sujet . } }`

q7	

`SELECT DISTINCT`
`  ?cible ?pred ?sujet`
`WHERE { ?cible ?pred ?sujet .`
`  ?cible dbponto:placeOfDeath ?sujet .`
`  OPTIONAL {`
`  ?cible dbponto:leaderName ?sujet.  }  }`
	`ASK { ?cible ?pred ?sujet .`
	`  ?cible dbponto:placeOfDeath ?sujet . }`

q8	

`SELECT DISTINCT`
`  ?cible ?pred ?sujet`
`WHERE { ?cible ?pred ?sujet .`
`  ?cible a dbponto:Person .`
`  ?sujet a dbponto:City .`
`  OPTIONAL { ?sujet a dbponto:Person .`
`  ?cible a dbponto:City . } }`
	`ASK { ?cible ?pred ?sujet .`
	`  ?cible a dbponto:Person .`
	`  ?sujet a dbponto:City . }`

TABLE 4.4 – Requêtes utilisées pour l'évaluation de notre approche

Pour chaque sorte d'aperçu A, B ou C, le tableau 4.6 recense la proportion de bases dont on a pu éviter l'interrogation car sans réponse à la requête donnée (*cf.* schéma de principe en figure 4.6).

Si l'on observe le seul tableau 4.5, on constate globalement que l'utilisation – en local sur un même serveur – des aperçus (en première passe) lors de l'interrogation de bases RDF n'a pas d'influence sur les performances, c'est-à-dire n'apporte ni vraiment gain de temps (max. −9 %) ni vraiment perte de temps (max. +12 %) dans l'exécution d'une requête SPARQL.

N°	Requête	Temps sans utilisation d'aperçu	Temps via aperçus A minimaux	Temps via aperçus B	Temps via aperçus C	Nombre de résultats dans les bases
1	q1	2:24,85	2:31,18	2:30,88	2:30,92	1 122 670
2	q2	0:06,65	0:07,15	0:07,16	0:07,07	13
3	q3	0:06,32	0:06,83	0:06,97	0:06,88	0
4	**q4**	1:05,66	**1:05,26**	1:09,08	1:09,27	2 680
5	**q5**	1:27,42	**1:19,52**	**1:19,71**	**1:20,18**	9 558
6	**q6**	1:24,74	**1:18,42**	**1:18,20**	**1:18,54**	9 546
7	q7	4:20,66	4:26,90	4:26,76	4:26,75	1 477 402
8	q8	0:08,80	0:09,48	0:10,78	0:10,79	1 962

TABLE 4.5 – Temps d'exécution des requêtes, hors temps d'accès réseau

N°	Requête	Nombre de bases sans réponse	Efficacité des aperçus A	Efficacité des aperçus B	Efficacité des aperçu C
1	q1	0	—	—	—
2	q2	190	90,00 %	90,00 %	90,00 %
3	q3	200	88,00 %	88,00 %	88,00 %
4	**q4**	137	77,37 %	58,39 %	57,66 %
5	**q5**	89	100,00 %	100,00 %	100,00 %
6	**q6**	89	100,00 %	100,00 %	100,00 %
7	q7	0	—	—	—
8	q8	161	92,55 %	73,91 %	73,29 %

TABLE 4.6 – Précision de l'interrogation passant par les aperçus

Dans le tableau 4.6, on présente l'efficacité des aperçus lors de l'interrogation des bases, *c-à-d* le pourcentage de bases ne contenant pas de résultats et qui ont été correctement écartées. On constate qu'un certain nombre de bases n'ont pas eu besoin d'être interrogées car l'interrogation d'un aperçu associé a permis de l'éviter. Cela est d'autant plus vrai si la requête porte sur une thématique particulière où peu de bases sont susceptibles de contenir une réponse. Les transformations des individus et des littéraux dans les aperçus conservent donc suffisamment la structure des données de la base pour pouvoir filtrer les requêtes précises. Notons que dans le tableau 4.6, face à une requête très – trop – généraliste, lorsque toutes les bases contiennent une réponse, l'affichage de l'efficacité des aperçus est remplacé par « — » pour une meilleur lecture des résultats.

Deux remarques sont à faire sur les tableaux 4.5 et 4.6. D'une part, de manière prévisible, on observe une dégradation des performances, en temps et efficacité, lors de la descente de la ligne de compression dans la hiérarchie de l'ontologie. En effet, le nombre de faux positifs est plus grand via les aperçus C, où le nombre d'individus dans un tel aperçu est de seulement 4 contre 7 individus en moyenne – compris entre 5 et 10 – pour un aperçu A. D'autre part, on remarque que le système d'indexation utilisé par le moteur Jena hébergeant les bases est plutôt performant dans la mesure où le temps d'exécution d'une requête dépend peu de la taille de la base, à l'exception de requêtes avec `UNION` dont la résolution reste co-NP-difficile. Dans le cadre réel d'interrogation du Linked Data où les

bases sont distribuées, et où les temps d'accès réseau ne sont pas négligeables, l'utilisation d'aperçus de recherche avant d'envoyer les requêtes à leur base d'origine apporte un véritable avantage. Seul un stockage en local ou centralisé[7] des aperçus – de très petite taille – est nécessaire. Pour permettre ces stockages des aperçus, nous proposons en chapitre 5 section 3 des optimisations et des méthodes pour la génération et le maintien des aperçus de recherche.

5 Conclusion

Nous avons proposé une méthode originale d'interrogation d'un ensemble de bases RDF en écartant du spectre des recherches les bases dont les données ne sont pas pertinentes pour une requête donnée. Pour cela, l'interrogation s'effectue en premier lieu sur document RDF de taille réduite, généré pour chaque base, appelé aperçu de recherche. Les résultats de nos expérimentations ont montré que l'utilisation des aperçus de recherche apportent un avantage non-négligeable pour la résolution de requêtes contenant des disjonctions. Ils ont également montrés que les aperçus de recherche n'entraîne pas de perte de temps lorsqu'ils sont hébergés sur la même machine que leurs bases d'origine, indépendamment de tout temps d'accès réseau. Cette observation nous laisse affirmer que les aperçus sont efficaces dans un système d'interrogation distribué – comme à travers le Linked Data– où les temps d'accès aux bases ne sont pas négligeables. Ainsi, une des perspective de notre approche serait son utilisation depuis un serveur local, voire central, qui regrouperait les aperçus de recherche d'un ensemble de bases et gérerait l'aiguillage des requêtes. Dans cette perspective, il serait intéressant de proposer une extension du protocole SPARQL pour (i) prendre en compte l'interrogation via les aperçus de recherche et (ii) veiller à la synchronisation des aperçus de recherche avec leurs bases d'origine.

Tels que présentés dans ce chapitre, les aperçus de recherche sont initialement générés par l'application des règles de réduction sur une base et doivent être re-générés ou actualisés lors des mises à jour de la base. Nous proposons au chapitre 5 section 3 deux optimisations pour faciliter la génération des aperçus en donnant (i) une méthode de génération parallèle des aperçus et (ii) une méthode de mise à jour des aperçus de recherche pour qu'ils accompagnent les mises à jour de la base.

[7] à la manière du système DNS sur le Web avec des serveurs « locaux » et un mécanisme de synchronisation entre serveurs

Chapitre 5

Système EE-I

Dans ce chapitre, nous présentons les implémentations des méthodes présentées dans les chapitres précédents. Ces implémentations sont regroupées dans le système EE-I – Système d'Évaluation, d'Examen et d'Interrogation – que nous avons conçu.

En section 1, nous présentons l'algorithme d'extraction des références utiles à l'évaluation d'une mise à jour dans notre méthode d'évaluation de la qualité de données RDF. Nous présentons l'algorithme d'extraction des motifs de descriptions qui apparaissent dans le voisinage d'un ensemble de ressources RDF, utilisé dans notre méthode d'examen de base RDF, en section 2. Enfin, nous présentons les améliorations de notre méthode d'interrogation basée sur les aperçus de recherche en section 3. La première amélioration est une méthode parallèle de génération d'aperçus de recherche, la seconde amélioration est un ensemble de règles qui permettent de modifier un aperçu en même temps que sa base d'origine.

Sommaire

1 Extraction des références pour l'évaluation de la qualité de données RDF

Nous proposons ici un algorithme pour l'extraction des références utilisées dans notre méthode d'évaluation de la qualité de données RDF (chapitre 2). Pour rappel, une référence est une partie d'une base suffisamment proche structurellement des contextes d'une mise à jour (*cf.* définition 2.2) pour être considérée comme comparable à la mise à jour (*cf.* définition 2.3).

Une référence est utilisée pour évaluer la qualité d'une mise à jour candidate à la base par la mesure de sa similarité avec les contextes de la mise à jour. Théoriquement, on peut considérer la base toute entière comme une référence pour l'évaluation d'une mise à jour, néanmoins en pratique il est préférable de choisir une partie de la base dont la taille est du même ordre de grandeur que les contextes de la mise à jour. Pour cela, la recherche des références d'une mise à jour présentée ici consiste à trouver les ressources de la base dont le voisinage ressemble à celui des ressources de la mise à jour. Ces ressources doivent être reliées entre elles de la même façon que celles de la mise à jour. De façon schématique, une référence est un point de comparaison pour l'évaluation les données d'une mise à jour par rapport à celles de la base. Nous cherchons donc à obtenir des références qui représentent la base tout en ayant des points communs avec la mise à jour.

Une approche semblable est présentée dans [Mottin et al., 2014] dans un système de « requête par l'exemple » utilisant le pré-calcul du voisinage de chaque ressource de la base pour retourner un résultat similaire à un exemple donné. Ce système est analogue à notre recherche de références si on considère la mise à jour comme un exemple du genre de références recherchées. Ce qui différencie notre méthode est que nous préférons éviter d'avoir recours à tout pré-calcul ad-hoc pour nous baser sur SPARQL et utiliser la relaxation de requête SPARQL pour la recherche de données structurellement ressemblantes.

Pour obtenir les références nécessaires pour l'évaluation de la qualité d'une mise à jour, nous utilisons une requête SPARQL construite à partir de la mise à jour et nous la *relaxons* autant que nécessaire jusqu'à ce que les résultats nous donnent un certain nombre de références. Moins il y a de relaxation, plus les références seront « satisfaisantes », *i.e.* avec une structure proche de la mise à jour. La requête est générée à partir des parties la mise à jour et de son contexte initial, elle est ensuite relaxée par des techniques de relaxation de requête SPARQL – telles que celles présentées dans [Hurtado et al., 2008, Elbassuoni et al., 2011] – qui la rendent de plus en plus générale jusqu'à ce qu'on obtienne suffisamment de résultats pour l'évaluation.

Cet algorithme a été implémenté en Java pour les expérimentations présentées dans le chapitre 2.

Méthode de relaxation

La première étape de notre méthode d'extraction des références est de générer une *requête initiale* à partir de la mises à jour et de ses contextes. Pour générer cette requête, nous identifions d'abord deux sortes de ressources dans la mise à jour, les *ressources cibles*, les ressources les plus concernées par la mise à jour – *i.e.* les ressources dont la description est la plus modifiée par la mise à jour – et les *ressources de description*.

Définition 5.1 (Ressource cible et ressource de description)**.** *Soient une mise à jour* $u = (A, R)$ *et* $k \in \mathbb{N}$ *un* seuil d'importance.

On appelle ressource cible *de* u *une ressource nœud* $c \in \mathcal{N}_A \cup \mathcal{N}_R$ *telle que* c *a un degré dans* $A \cup R$ *supérieur ou égal à* k.

On appelle ressource de description *de u une ressource de $\mathcal{R}_{A \cup R}$ qui n'est pas une ressource cible.*

Une ressource cible est une des ressources qui peut être considéré comme étant au centre de la mise à jour, c'est un élément central – *e.g.* la ressource au centre d'un graphe en étoile – apparaissant dans plus de k triplets de la mise à jour. Une ressource de description est une des ressources utilisées pour décrire les ressources cibles.

La requête initiale pour l'extraction de références pour une mise à jour $u = (A, R)$ est générée à partir de l'union du contexte initial de u et des triplets ajoutés – $I_u \cup A$ – en remplaçant principalement les ressources cibles par des variables (voir définition 5.2).

Définition 5.2 (Génération de requête initiale d'extraction des références pour une mise à jour)**.** *Soit $u = (A, R)$ une mise à jour candidate à la base RDF B et C_u l'ensemble des ressources cibles de u.*

La requête initiale pour l'extraction de références Q_u^{init} pour une mise à jour u est une requête SELECT *contenant un en-tête Q_e et un corps de requête Q_c. Le contenu de Q_e et les triplets de Q_c sont le résultat de la transformation des triplets de $A \cup I_u$. Q_c contient les triplets transformés de $A \cup I_u$ selon les règles suivantes :*

r1 *Remplacer une ressource appartenant à C_u par une variable dans chaque triplet où la ressource apparaît et l'ajouter à Q_e ;*

r2 *Dans un triplet dont le sujet et l'objet ont été remplacés par des variables, remplacer la relation par une variable et l'ajouter à Q_e ;*

r3 *Remplacer un littéral par une variable ;*

Lorsque la requête initiale est utilisée pour interroger la base, elle recherche les ressources dont le voisinage contient des ressources communes avec celui des ressources cibles de la mise à jour.

Les transformations effectuées pour créer la requête initiale ne sont pas faites pour garantir des résultats à la requête dans la base. La requête initiale peut contenir des triplets de A intacts, qui l'empêchent d'obtenir un résultat dans la base. Dans les meilleures conditions – où la requête initiale ne contient plus de triplet empêchant l'obtention de résultats – les résultats de la requête initiale sont au minimum les ressources cibles de la mise à jour qui sont déjà présentes dans la base et les relations qui les lient.

Exemple 12 (Génération de la requête initiale depuis u_1)**.** *Dans u_1, la mise à jour de l'exemple 1 du chapitre 2 qui est représentée en figure 2.5, les ressources cibles sont* dbpedia:cointreau *et* dbpedia:angers *avec un seuil d'importance de* 3*.*

La requête initiale pour l'extraction de références pour l'évaluation de u_1 est :

```
SELECT ?cointreau ?origin ?angers
WHERE {
    ?cointreau dbpprop:flavour ?lit1 .
    ?cointreau a dbponto:Beverage .
    ?cointreau dbpprop:type dbpedia:Liqueur .
    ?cointreau ?origin ?angers .
    ?angers a dbponto:City .
    ?angers dbponto:country dbpedia:France .
    ?angers dbponto:mayor dbpedia:christophe_bechu .
```

```
?angers dbponto:mayor dbpedia:frederic_beatse .
dbpedia:christophe_bechu a dbponto:Person .
dbpedia:frederic_beatse a dbponto:Person . }
```

La requête initiale recherche les ressources de la base dont la description est exactement la même que celle des ressources cibles de la mise à jour. Nous recherchons les autres ressources de la base, dont la description est proche mais pas nécessairement identique à des ressources cibles de la mise à jour.

Pour obtenir ces ressources, nous relaxons la requête initiale pour obtenir de plus en plus de ressources aux descriptions similaires. Autrement dit, nous allons petit à petit rendre la requête plus générale en changeant en variable les ressources qui sont les plus caractéristiques de la mise à jour.

Dans notre exemple centré autour de Cointreau et de la ville d'Angers, le fait que Christophe Béchu soit maire d'Angers ou que le Cointreau soit une liqueur, sont particulièrement caractéristiques de notre mise à jour. Nous recherchons des références concernant des boissons, qui ne sont pas nécessairement des liqueurs, liées à des villes ayant un maire.

La relaxation de la requête est faite progressivement jusqu'à ce que son nombre de résultats dépasse un nombre minimum de références d_{min} fixé pour l'évaluation. Plus précisément, la relaxation de la requête est faite par remplacement des ressources nœuds ou par la transformation des triplets selon leurs nombres d'occurrences dans la base. Le but des relaxations est d'éliminer les éléments qui limite le nombre de résultats à la requête dans la base.

Définition 5.3 (Fonction d'occurrence des ressources RDF)**.** *Soit B un ensemble de triplets RDF et $r \in \mathcal{R}_B$ une ressource de B.*

La fonction occurrence *: $E \to \mathbb{N}$ avec $\forall e \in E$,* occurrence$(r)= |\{t \,|t \in B$ *et r est sujet, relation ou objet de t $\}|$ est appelée* fonction d'occurrence des ressources RDF*.*

La fonction d'occurrence de ressource dans une base RDF retourne le nombre de triplets dans lequel apparaît une ressource.

Définition 5.4 (Règles de relaxation de requête pour l'extraction de références)**.** *Soit une base RDF B, une mise à jour u candidate à B, $d_{min} \in \mathbb{N}$ un nombre minimum de références, $n \in \mathbb{N}$ un rang de voisinage et la requête initiale pour l'extraction de références $Q_u^{init} = (Q_e, Q_c)$*

Soit la requête $Q = Q_u^{init}$ et B_Q son ensemble de résultats dans B.

L'ensemble des références peut être obtenu à partir de B_Q après application des règles suivantes sur Q_c. Les règles suivantes sont appelées règles de relaxation*, elles sont à appliquer tant que $|B_Q| < d_{min}$ et qu'elles peuvent s'appliquer :*

r1 *S'il existe un triplet (s, p, o) $\in Q_c$, hors d'une clause* OPTIONAL*, dans lequel* s *et* o *ne sont pas des variables,*

- **r1.1** *si* occurrence(s) $<$ occurrence(o) *alors remplacer* s *par une variable,*
- **r1.2** *sinon remplacer* o *par une variable ;*

r2 *Sinon pour le triplet (s, p, o) dont* occurrence(p) *est minimale parmi les occurrences de relations des triplets de Q_c, hors d'une clause* OPTIONAL*,*

r2.1 *si* s *(ou resp.* o*) n'est pas une variable alors remplacer* s *(ou resp.* o*) par une variable,*

r2.2 *sinon remplacer (*s, p, o*) par une clause* OPTIONAL *contenant (*s, p, o*).*

Après application des règles de relaxation, une référence D_i *est obtenue à partir d'un résultat* $B_Q^i \in B_Q$ *telle que* $D_i = \bigcup_{r_Q \in B_Q^i} voisinage_B^n(r_Q)$

Les règles de relaxation transforment d'abord les triplets qui ne contiennent pas de variables, puis elles transforment en variables les sujets et objets des triplets du corps de la requête qui apparaissent le moins souvent dans la base. Elles transforment ensuite les triplets – dans lesquels seule la relation n'est pas une variable – en clauses optionnelles. Par l'application de ces règles, les références sont extraites par la recherche progressive de ressources qui ont de moins en moins de points communs avec la mise à jour.

On obtient une référence de la mise à jour en extrayant le voisinage dans la base des ressources contenues dans un résultat de la requête.

Exemple 13 (Extraction des références de u_1)**.** *Dans cet exemple, nous cherchons* 100 *références pour évaluer* u_1*. En appliquant les règles de relaxation de la définition 5.4 sur* $Q_{u_1}^{init}$*, avec* $d_{min} = 100$*, nous suivons les étapes suivantes :*

1. *Les triplets qui ne contiennent pas de variables sont traités par la règle* ***r1****. Dans tous les triplets de la requête,* dbpedia:christophe_bechu *et* dbpedia:frederic_beatse *sont remplacés respectivement par les variables* ?christophe_bechu *et* ?frederic_beatse*.*

   Cette requête n'obtient pas de résultats dans DBPedia (première itération).

2. *Dans le triplet (* ?cointreau, dbpprop:flavour, ?lit1*), la relation* dbpprop:flavour *a le plus faible nombre d'occurrences dans DBPedia parmi les relations des triplets de la requête. Par la règle* ***r2.2****, le triplet est placé à l'intérieur d'une clause* OPTIONAL*.*

   Cette requête n'obtient pas de résultat dans DBPedia (deuxième itération).

3. *Dans le triplet (* ?angers, dbponto:country, dbpedia:France*), la relation* dbponto:country *est devenue la relation avec le plus petit nombre d'occurrences dans la requête (en dehors d'une clause* OPTIONAL*). Par la règle* ***r2.1****, l'objet* dbpedia:France *est remplacée par une variable* ?France*.*

   Cette requête obtient 38 *résultats (*$< d_{min}$*) dans DBPedia (troisième itération).*

4. *(* ?angers, dbpedia:country, ?France*) est toujours le triplet dont la relation a la plus petite occurrence dans DBPedia. Il est placé dans une clause* OPTIONAL*.*

   La requête ainsi formée obtient 916 *résultats dans DBPedia, plus que le nombre minimum de références recherché (quatrième et dernière itération).*

Les références de u_1 *sont obtenues en recherchant dans DBPedia le voisinage des 100 premiers résultats donnés par le moteur de requête SPARQL. On observe que les références obtenues sont des liqueurs ou des boissons associées à des villes et à leurs différents dirigeants.*

Cette méthode se concentre sur les caractéristiques structurelles des mises à jour et des références, sans considération pour l'ontologie. En incluant le voisinage des ressources extraites par notre méthode, nous espérons ajouter des éléments à la référence qui pourraient potentiellement être utiles à l'évaluation sans avoir pu être extraits directement par notre méthode.

Algorithme d'extraction des références

Les algorithmes 5.1 et 5.2 sont les traductions algorithmiques des définitions 5.2 et 5.4. L'algorithme 5.1 sert à la génération de l'en-tête et du corps de la requête initiale pour l'extraction de références. L'algorithme 5.2 relaxe le corps de la requête jusqu'à ce que son nombre de résultats accumulés soit supérieur au nombre minimum de références.

Algorithme 5.1 Génération de la requête initiale.

Entrée: MiseAJour m, Entier k ▷ k seuil d'importance
1: Ensemble<Ressource> cibles = m.cibles(k) ▷ Retourne les ressources cibles au rang k
2: Ensemble<Variable> $q_{\text{en-tete}}$ = {}
3: Ensemble<Triplet> q_{corps} = m.A + m.I_u
4: Map<Ressource, Variable> swaps = {}
5: **pour tout** c **dans** cibles **faire** ▷ **r1** : Création et ajout des variables à Q_e pour chaque ressource cible
6: Variable v_c
7: $q_{\text{en-tete}}$ += v_c
8: swap[c] = v_c
9: **fin pour**
10: **pour tout** t **dans** q_{corps} **faire**
11: **si** t.objet ∈ cibles **et** t.sujet ∈ cibles **alors** ▷ **r2**
12: Variable $v_{\text{t.relation}}$
13: t.relation = $v_{\text{t.relation}}$
14: **fin si**
15: **si** t.sujet ∈ cibles **alors** ▷ **r1** : Remplacement des sujets
16: t.sujet = swaps[t.sujet]
17: **fin si**
18: **si** t.objet ∈ cibles **alors** ▷ **r1** : Remplacement des objets
19: t.objet = swaps[t.objet]
20: **fin si**
21: **si** estLitteral(t.objet) **alors** ▷ **r3**
22: Variable $v_{\text{t.objet}}$
23: t.objet = $v_{\text{t.objet}}$
24: **fin si**
25: **fin pour**
Sortie: ($q_{\text{en-tete}}$, q_{corps})

Algorithme 5.2 Relaxation de requête pour l'extraction de références.

Entrée: ($q_{\text{en-tete}}$, q_{corps}), int nbRefMin ▷ Requete initiale et nombre minimum de références
1: Ensemble<ResultatDeRequete> results = b.execute(Requete($q_{\text{en-tete}}$, q_{corps}))
2: boolean modif = true
3: Liste relationsParOcc =
4: **pour tout** t **de** q_{corps} **faire** relationsParOcc += t.relation
5: **fin pour**

```
 6: triParOccurrence(b,relationsParOcc)     ▷ Tri croissant des relations par occurrence dans b
 7: tant que (|results| < nbRefMin) et modif faire
 8:     modif = false
 9:     pour tout t de q_corps faire                                    ▷ Application de r1
10:         si non estVariable(t.sujet) et non estVariable(t.objet) alors
11:             Variable v_t
12:             modif = true
13:             si occurrence(b, t.sujet) < occurrence(b, t.objet) alors
14:                 t.sujet = v_t
15:             sinon
16:                 t.objet = v_t
17:             fin si
18:             aller à la ligne 40
19:         fin si
20:     fin pour
21:     si non modif alors                                              ▷ Application de r2
22:         pour tout t de q_corps faire
23:             si t.relation == relationsParOcc.premierElement() alors      ▷ La relation du
    triplet est la relation d'occurrence minimum dans q_corps
24:                 Variable v_t
25:                 modif = true
26:                 si estVariable(t.objet) alors
27:                     t.sujet = v_t
28:                 sinon si estVariable(t.sujet) alors
29:                     t.objet = v_t
30:                 sinon
31:                     q_corps -= t
32:                     q_corps += OPTIONAL{ t }
33:                     si occurrence(q_corps, t.relation) == 0 alors
34:                         relationsParOcc.supprimerPremierElement()
35:                     fin si
36:                 fin si
37:             fin si
38:         fin pour
39:     fin si
40:     results += (b.execute(Requete(q_en-tete, q_corps)))
41: fin tant que
Sortie: results
```

2 Extraction des motifs de description pour l'examen de base RDF

Pour rappel, un motif de description est une sorte de graphe descriptif de synthèse défini pour notre méthode d'examen pour l'évolution d'une base RDF en chapitre 3. Un motif décrit les points communs dans les graphes descriptifs de plusieurs ressources RDF d'une base. Lorsque l'ontologie et les données sont cohérentes, les groupes de ressources qui partagent un motif commun coïncident avec les classes définies dans l'ontologie. Dans le chapitre 3 nous nous sommes intéressés au cas où ces groupes ne correspondent pas aux classes définies dans l'ontologie. Ce cas est important à identifier lors de l'examen

d'une base RDF par un expert pour qu'il puisse déterminer si ce conflit entre ontologie et données indique une évolution nécessaire de la base. Notre méthode est une aide à cet examen en extrayant les motifs suivis par les groupes de ressources en conflit avec l'ontologie.

Nous proposons une méthode d'extraction des motifs de description d'un groupe de ressources, détaillée dans l'algorithme 5.3, et schématisée en figure 5.1. Elle est inspirée du problème d'extraction de la plus longue sous-séquence commune de deux chaînes de caractère (voir [Bergroth et al., 2000]) en considérant les motifs possibles comme des caractères. Elle consiste à parcourir un ensemble de graphes descriptifs et d'ajouter progressivement tous les motifs maximaux qui peuvent être identifiés en eux à une liste qui sera retournée à l'expert.

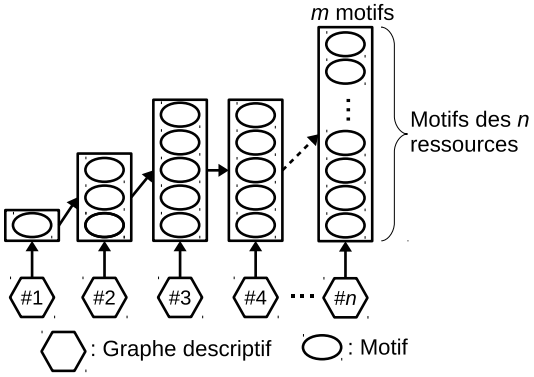


FIGURE 5.1 – Schéma de fonctionnement de l'algorithme 5.3.

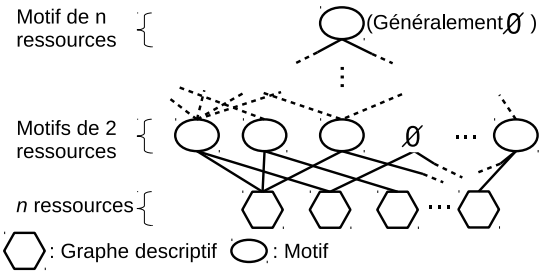


FIGURE 5.2 – Illustration personnelle de l'extraction de motifs telle que proposée dans [Colucci et al., 2014].

Notre méthode a un résultat similaire à la méthode de clustering utilisant des sortes de motifs de ressources RDF décrite dans [Colucci et al., 2014] et représentée selon notre interprétation en figure 5.2. Cette autre méthode revient à comparer les graphes descriptifs des ressources deux à deux pour en extraire les motifs, puis à comparer les motifs entre eux jusqu'à obtenir les motifs communs à toutes les ressources. Elle a le défaut de nécessiter le calcul de tous les motifs de ressources deux à deux et de tous les motifs de motifs deux à deux jusqu'à obtention du motif final, sans pouvoir éviter le calcul de motifs vides.

La méthode de recherche des motifs d'un ensemble de ressources que nous proposons (i) peut donner le même motif final, (ii) peut être arrêtée à tout moment pour obtenir un résultat partiel et (iii) ne calcule jamais de motif vide.

Pour un ensemble de ressources RDF et pour chacun de leurs graphes descriptifs, notre méthode d'extraction de motifs extrait les motifs maximaux qui peuvent être identifiés dans les triplets du graphe descriptif et par comparaison avec les motifs précédemment extraits. Ces motifs sont ajoutés progressivement à la liste des motifs qui sera retournée par l'algorithme. Bien que cela n'apparaisse pas dans l'algorithme 5.3 pour des raisons de lisibilité, le nombre de motifs extraits des graphes descriptifs peut être limité par un choix de valeurs minimum de taille ou de valeur de description (*cf.* définition 3.4). Pour une liste de ressources l'algorithme 5.3 ajoute la liste des motifs de chaque graphe descriptif ressource à la liste de motifs connus. Cet algorithme peut être interrompu à n'importe quelle itération de la boucle en ligne 2 pour obtenir un résultat partiel. Dans l'algorithme 5.3 on utilise la relation d'inclusion d'un motif dans un autre. Nous définissons l'inclusion de motif de la façon suivante : Un motif m_1 est inclus dans un motif m_2 si m_1 peut être considéré comme un motif de m_2. Nous utilisons également les opérateurs d'intersection et de soustraction, que nous définissons tels que :

- L'intersection $m_1 \cap m_2$ de deux motifs m_1 et m_2 est leur motif maximal commun.
- La soustraction d'un motif m_2 à un motif m_1 est le motif maximal extrait de m_1 tel que $(m_1 \setminus m_2) \cap m_2 = \emptyset$.

Algorithme 5.3 Procédure d'ajout des motifs d'un graphe descriptif g à la liste des motifs *connus*.

```
Entrée: Liste<Ressource> liste
 1: Liste<Motif> connus = {}
 2: pour tout Ressource res de liste faire
 3:     GrapheDescriptif g = graphDesc(res)
 4:     Pile<Motif> cibles = {g.voisinage}
 5:     tant que cibles ≠ ∅ faire
 6:         Motif courant = cibles.pop()
 7:         pour tout elem de connus faire
 8:             si courant ∩ elem ≠ ∅ alors
 9:                 si courant = elem ou elem ⊂ courant alors
10:                     elem.sources += courant.sources
11:                 fin si
12:                 si elem ⊂ courant alors
13:                     Motif nmotif = courant \ elem
14:                     nmotif.sources = courant.sources
15:                     cibles += nmotif
16:                 sinon si courant ⊂ elem alors
17:                     courant.sources += elem.sources
18:                 sinon
19:                     Motif nmotif = courant ∩ elem
20:                     nmotif.sources = elem.sources + courant.sources
21:                     cibles += nmotif
22:                 fin si
23:             fin si
24:         fin pour
25:         si courant ∉ connus alors
26:             connus += courant
27:         fin si
28:     fin tant que
29: fin pour
Sortie: connus
```

Dans l'algorithme 5.3, chaque motif contient ses triplets et garde trace de ses « sources » – *i.e.* ses centres descriptifs – afin de pouvoir calculer son occurrence. L'algorithme compare chaque graphe descriptif courant, aux motifs connus et tant qu'ils ont un élément en commun :

- Si le nouveau motif est déjà parmi les motifs connus ou contient un motif connu, alors on ajoute ses sources à la liste des sources du motif déjà présent dans la liste (lignes 7 et 8).
- Si le nouveau motif contient un motif connu, on crée un nouveau motif avec les chemins spécifiques au nouveau motif et on l'ajoute à la liste des motifs à ajouter (lignes 9 à 12).

- Si le nouveau motif est contenu dans un motif connu, on ajoute les sources du motif connus à celle du nouveau motif (lignes 13 et 14).
- Sinon un nouveau motif est créé, contenant les chemins communs avec un motif connu et les sources adéquates, il est ajouté à la liste des motifs à ajouter (lignes 15 à 18)

Enfin, si le nouveau motif n'apparaît pas dans la liste des éléments connus, il y est ajouté (lignes 22 et 23).

Exemple d'extraction de motif

Dans cet exemple, nous extrayons les motifs de 3 ressources : Renault 7, Volkswagen Beetle et Porte-avion Charles de Gaulle en utilisant leurs graphes descriptifs au rang 2, avec un seuil d'occurrence de 1, un seuil de taille de 3 et un seuil de valeur de description de 0.1. Pour chacune de ces ressources, on extrait les graphes descriptifs contenant leurs voisinages au rang 2, représentés respectivement en figures 5.3, 5.4 et 5.6.

L'extraction des motifs de ces trois ressource est décomposée en trois itérations de la boucle en ligne 2 de l'algorithme 5.3.

Première itération : À la première itération, il n'y a pas de motif connus.

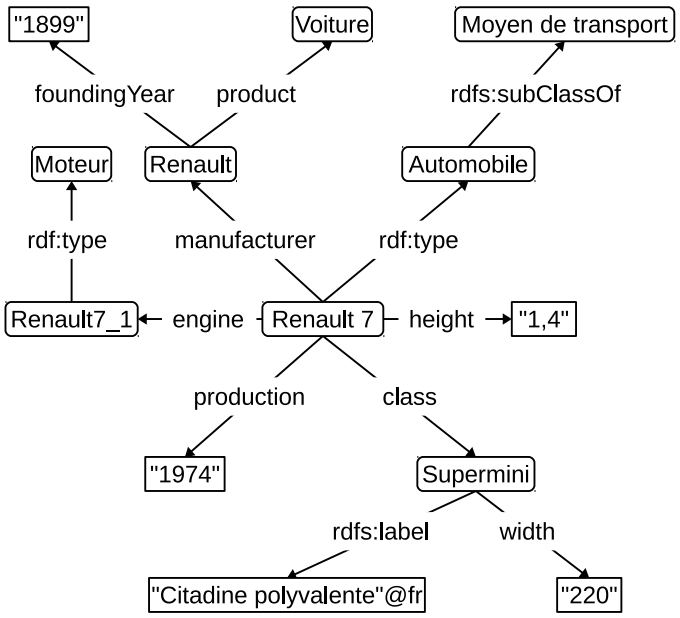


FIGURE 5.3 – Voisinage au rang 2, $D_{Renault7}$, de Renault 7 dans DBPedia.

Il n'y a aucun motif connu auquel comparer de graphe descriptif $D_{Renault7}$, la fonction retourne donc la liste l_1 en tableau 5.1.

Motif	Sources
$D_{Renault7}$	Renault 7

TABLE 5.1 – Liste de résultats t_1 de la première itération.

Deuxième itération : À la deuxième itération nous reprenons les résultats obtenus lors de l'appel précédent pour y ajouter les motifs de Volkswagen Beetle. En comparant $D_{Renault7}$ et D_{Beetle} on trouve plusieurs éléments en commun, on ajoute donc le motif maximal commun au deux m_1, représenté en figure 5.5, à la liste des motifs connus. Ce motif est lui-même comparé de nouveau à $D_{Renault7}$ et D_{Beetle} sans trouver de nouveau motif à extraire, l'itération donne donc comme résultat l_2 tel qu'en tableau 5.2.

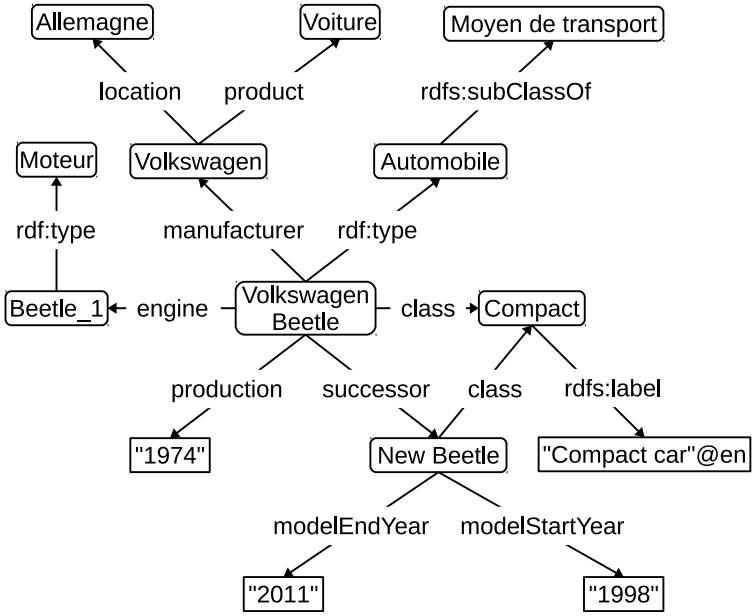


FIGURE 5.4 – Voisinage au rang 2, D_{Beetle}, de Volkswagen Beetle dans DBPedia.

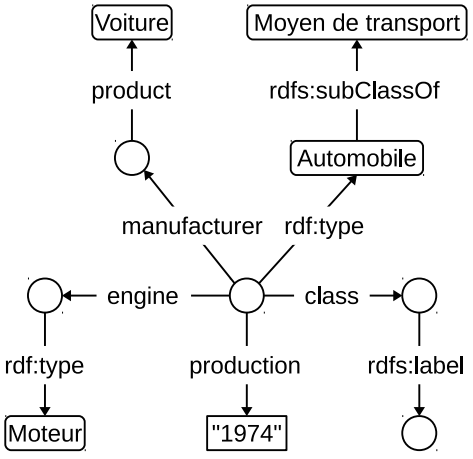


FIGURE 5.5 – Motif de description m_1 de Renault 7 et Volkswagen Beetle.

Motif	Sources
$D_{Renault7}$	Renault 7
D_{Beetle}	Volkswagen Beetle
m_1	Renault 7, Volkswagen Beetle

TABLE 5.2 – Liste de résultats l_2 de la deuxième itération.

Troisième itération : À la troisième itération nous reprenons les résultats obtenus lors de l'itération précédent pour y ajouter les motifs de Porte-avion Charles de Gaulle. En comparant d'abord $D_{Charles}$ à $D_{Renault7}$, on trouve plusieurs éléments communs qui forment le motif m_2, représenté en figure 5.7, qui est ajouté à la liste des motifs cibles avec pour sources Porte-avion Charles de Gaulle et Renault 7 (nous nommons m_2^a la version de m_2 associée à ces sources). En comparant ensuite $D_{Charles}$ à D_{Beetle}, on retrouve m_2 qui est cette fois ajouté aux motifs cibles avec pour sources Porte-avion Charles de Gaulle et Volkswagen Beetle (nous nommons m_2^b la version de m_2 associée à ces sources). En comparant $D_{Charles}$ à m_1, on retrouve encore m_2, cette fois avec pour sources Porte-avion Charles de Gaulle, Renault 7 et Volkswagen Beetle, et on l'ajoute aux motifs cibles. $D_{Charles}$ a été comparé à tous les motifs et il n'est pas connu, il est donc ajouté à la liste des motifs connus.

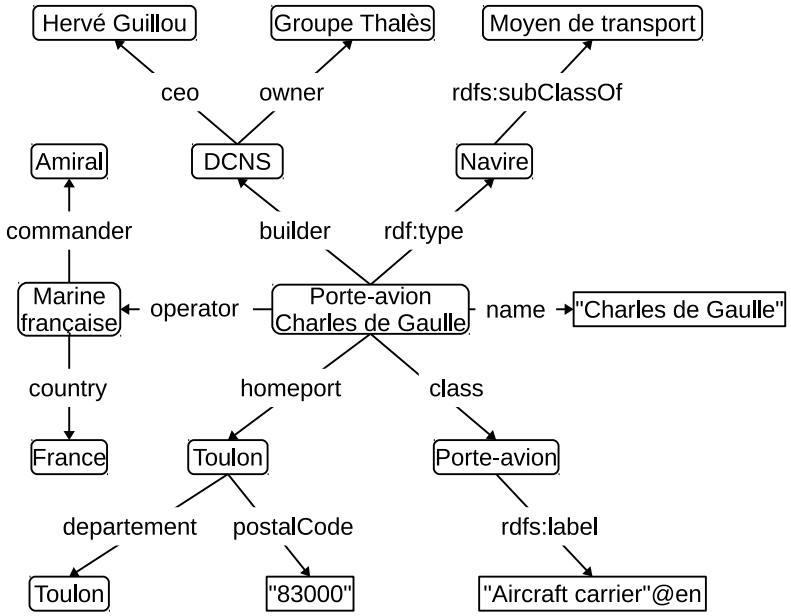


FIGURE 5.6 – Voisinage au rang 2, $D_{Charles}$, de Porte-avion Charles de Gaulle dans DBPedia.

Le dernier motif ajouté à la pile des motifs cibles, m_2 est comparé aux motifs connus $D_{Charles}$, $D_{Renault7}$, D_{Beetle} et m_1. À l'issu de ces comparaisons, il n'y a pas de nouveau motif à extraire et m_2 n'est pas un motif connu, il est donc ajouté aux motifs connus. Les deux autres versions de m_2, m_2^a et m_2^b, sont ensuite comparées aux motifs connus jusqu'à ce qu'elles soient comparées à m_2 que l'ont vient d'ajouter. Ces deux versions sont égales à m_2, elles ne sont donc pas ajoutées aux motifs connus.

La troisième itération donne donc pour liste des motifs connus la liste l_3 en tableau 5.3

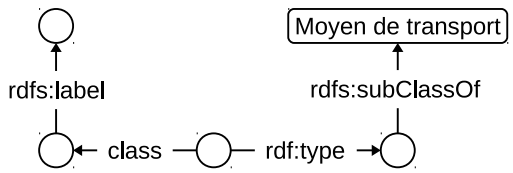


FIGURE 5.7 – Motif de description m_2 de Renault 7, Volkswagen Beetle et Porte-avion Charles de Gaulle.

Le seuil d'occurrence de notre extraction nous fait retirer du résultat final les motifs qui n'ont qu'une seule source. Les motifs restants, d'occurrence supérieure à 1, sont les motifs m_1 et m_2. Le motif m_1 est de taille 9 et de valeur de description de 0.46 et le motif

Motif	Sources
$D_{Renault7}$	Renault 7
D_{Beetle}	Volkswagen Beetle
m_1	Renault 7, Volkswagen Beetle
$D_{Charles}$	Porte-avion Charles de Gaulle
m_2	Renault 7, Volkswagen Beetle, Porte-avion Charles de Gaulle

TABLE 5.3 – Liste de résultats l_3 de la troisième itération.

m_2 est de taille 4 et de valeur de description 0.16. Les tailles et valeurs de description des deux motifs sont toutes deux aux dessus des seuils de notre extraction. Au final, l'algorithme d'extraction des motifs de Renault 7, Volkswagen Beetle et Porte-avion Charles de Gaulle a pour résultats les motifs en tableau 5.4.

Motif	Sources
m_1	Renault 7, Volkswagen Beetle
m_2	Renault 7, Volkswagen Beetle, Porte-avion Charles de Gaulle

TABLE 5.4 – Liste des motifs de Renault 7, Volkswagen Beetle et Porte-avion Charles de Gaulle au rang 2.

3 Génération parallèle et incrémentale d'aperçus de recherche pour l'interrogation

Pour rappel, les aperçus de recherche (chapitre 4) sont des documents RDF qui servent de résumés d'une base et qui permettent de vérifier si une base ne contient aucun résultat à une requête donnée. Les aperçus regroupent les individus qui ont un type en commun en conservant les relations qui les relient.

La génération d'aperçus de recherche, définition 4.5, crée un aperçu à partir d'un document RDF dans lequel on connaît le type de chaque individu. Dans notre modèle d'utilisation, un aperçu est généré une fois pour une base à un moment donné et doit être re-généré régulièrement pour prendre en compte les modifications de la base. Il nous manque donc une méthode qui permettrait d'effectuer cette re-génération de façon efficace ou une méthode faisant en sorte que le contenu de l'aperçu de recherche soit à jour par rapport à celui de la base.

Nous proposons ici 2 améliorations indépendantes de la génération d'aperçus de recherche qui permettent :

- d'une part, la génération d'un aperçu en utilisant la parallélisation par la séparation des données traitées,
- d'autre part, de faire en sorte que le contenu d'un aperçu soit mis à jour en même temps que le contenu de la base.

Optimisations basiques L'implémentation des règles de réduction – et des méthodes de génération avancées suivantes – données en définition 4.5 nécessite de disposer de plusieurs structures d'indexation :

- un index liant chaque individu a ses types les plus spécialisés et chaque type à ses individus ;
- une structure d'accès rapide aux classes plus générales ou plus spécialisées qu'une classe donnée.

Ces index sont utilisés massivement lors de la génération et sont très utiles pour les optimisations suivantes.

Les règles de réduction **r2** et **r3** sont formulées en considérant les individus deux à deux : « fusionner deux individus distincts » en fonction de leurs types. À l'aide des index, ces fusions sont très aisément faites en fusionnant en bloc tous les individus de chaque classe concernée.

D'un point de vue pratique, certaines bases, *e.g.* DBPedia, stockent le typage de chaque individu par chaque type de sa hiérarchie. Dans notre méthode de génération, nous ne considérons que les classes les plus spécialisées et nous supprimons les types plus généraux en considérant que le moteur SPARQL effectue l'inférence à l'interrogation. Néanmoins, dans le Linked Data actuel, l'inférence lors de requête SPARQL est très rare, c'est pourquoi il est nécessaire :

- soit de stocker ces relations de typage avant la génération de l'aperçu – ou parcourir l'ontologie – pour les ajouter de nouveau à la fin de la génération, ce qui ajoute une opération supplémentaire à la génération ;
- soit d'utiliser un moteur d'inférence lors de l'interrogation des aperçus, ce qui crée une perte de performance lors de l'interrogation.

Dans nos expérimentations, nous avons choisi la première option, en l'absence d'un moteur d'inférence jugé suffisamment performant. Ce choix a été renforcé par la faible profondeur de l'ontologie utilisée dans nos expérimentations – et en général dans le Web des données –, ce qui a limité le nombre de triplets à ajouter à chaque individu dans nos aperçus.

3.1 Génération parallèle d'aperçus de recherche

La génération d'aperçus de recherche se fait en appliquant autant que possible les règles de réduction des individus et des littéraux sur un document REST (définition 4.1). La réduction des individus regroupe les individus ayant un type en commun et les fusionne en un seul nœud blanc. La réduction des littéraux utilise une fonction de substitution pour remplacer un groupe de littéraux par un seul littéral.

Une fois la fonction de substitution des littéraux choisie, la réduction des littéraux ne dépend pas des littéraux réduits. Elle donne le même résultat sur un ensemble de triplets, qu'ils soient dans le même document ou non, *c-à-d* les littéraux de substitution sont constants pour une fonction de substitution donnée. On peut donc appliquer en parallèle les règles de réduction des littéraux sur des parties d'un document RDF et obtenir le même résultat que si on les avait appliquées sur le document complet.

En revanche, le résultat de la réduction des individus dépend des individus réduits et de leurs types. Les individus de parties différentes d'un document forment des nœuds de réduction différents d'une partie à l'autre et peuvent être différents des nœuds de réduction de l'aperçu du document entier. Pour pouvoir paralléliser la réduction des individus, un traitement particulier des individus est donc nécessaire.

Nous proposons de paralléliser la génération d'aperçus en utilisant le fait qu'un sous-document d'un document REST est un document REST si le type de chaque individu est conservé dans le sous-document (définition 4.1 (i)). Ainsi, en séparant un document REST en sous-documents REST contenant une partie des triplets dans lesquels tous les types des individus apparaissent, nous pouvons générer indépendamment les aperçus de parties de ce document. L'aperçu final est obtenu par la fusion des aperçus des sous-documents.

La fusion finale des aperçus des sous-documents est faite par une nouvelle étape de réduction, effectuée sur le document résultant de la combinaison des sous-aperçus. Cette dernière étape de génération fusionne les différents individus de même type de chaque sous-aperçu. Les règles de réduction des littéraux peuvent être appliquées avant la séparation ou durant la fusion, ou même lors des générations des sous-aperçus. Nous conseillons de les appliquer avant la séparation des sous-documents afin de réduire au plus tôt le nombre de triplets de la base.

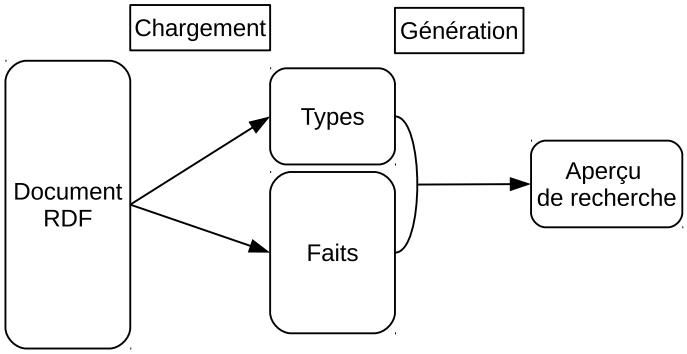


FIGURE 5.8 – Schéma de la génération normale d'un aperçu de recherche.

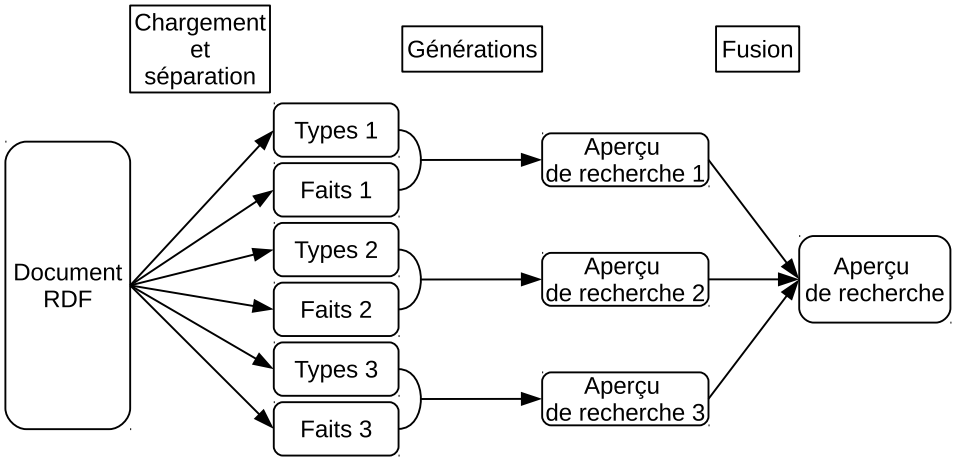


FIGURE 5.9 – Schéma de la génération parallèle d'un aperçu de recherche.

La figure 5.8 représente une génération normale d'aperçus de recherche, avec application itérative des règles de réduction à partir des types des individus sur les faits d'une base RDF. La figure 5.9 représente une génération parallèle d'aperçus de recherche, elle est possible par l'ajout de 2 nouvelles étapes par rapport à la génération normale d'aperçus :

- La séparation des sous-documents : On sépare les triplets de la base en plusieurs sous-documents. Dans un sous-document, pour chaque individu apparaissant en sujet ou en objet d'un triplet, les triplets de typages sont ajoutés dans le sous-document.

En pratique, la séparation peut être faite naïvement en séparant les triplets de la base en plusieurs parties de tailles équivalentes, puis en extrayant les relations de typage pour chaque individu apparaissant dans le sous-document. De façon moins naïve, nous conseillons une séparation se basant sur l'extraction intelligente du voisinage des individus de la même classe (*cf.* chapitre 1, section 7). Cette séparation non naïve crée des sous-documents où les individus ont des descriptions similaires, parce qu'ils sont de la même classe, mais où le contenu de chaque sous-document est différent, puisqu'on trouve des individus de classes différentes d'un sous-document à l'autre. Ainsi les réductions effectuées dans chaque sous-aperçu sont également différentes, et la fusion finale a moins de nœuds de mêmes types à fusionner.

- La fusion des sous-aperçus : Cette étape fusionne les nœuds blancs eux-mêmes résultats de la fusion d'individus dans chaque sous-aperçu. Lors de cette fusion, les nœuds blancs de chaque sous-aperçu doivent avoir des identifiants différents afin de ne pas être confondus d'un sous-aperçu à l'autre. En effet, les identifiants des nœuds blancs ont une signification uniquement à l'intérieur de leur document d'origine, il est nécessaire d'utiliser des identifiants uniques lors des générations ou de les renommer lors de la fusion.

Exemple 14. *Pour illustrer notre méthode de parallélisation de la génération d'aperçus, nous reprenons la génération faite en chapitre 4 section 3.1 à partir du document représenté en figure 4.2, particulièrement celle dont le résultat est l'aperçu représenté en figure 4.7 avec la ligne de compression {ex:Personne }.*

On sépare le document d'origine en deux documents, représentés en figures 5.10 et 5.11. La séparation est faite de façon à séparer les classes qui contiennent le plus d'individus. Le sous-document 1 a été créé pour contenir en majorité les triplets des descriptions des individus de la classe ex:Pédiatre (ex:Jean et ex:Bob). Le sous-document 2 a été créé pour contenir principalement ceux de la classe ex:Médecin (ex:Marie et ex:Fred). Les triplets restant, provenant des descriptions des individus d'autres classes (ex:Alice), sont répartis dans les deux sous-documents.

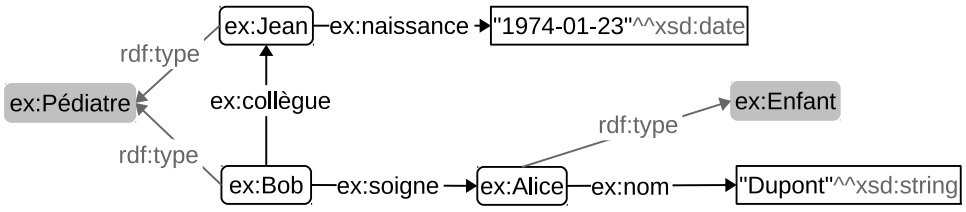


FIGURE 5.10 – Sous-document 1 tiré du document en figure 4.2.

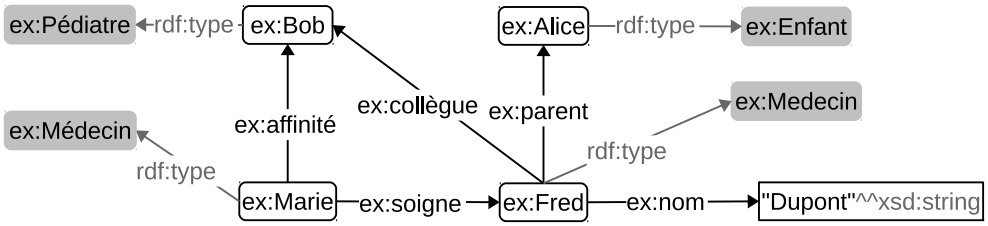


FIGURE 5.11 – Sous-document 2 tiré du document en figure 4.2.

L'application des règles de réduction sur ces deux documents génère deux sous-aperçus, représentés en figures 5.12 et 5.13.

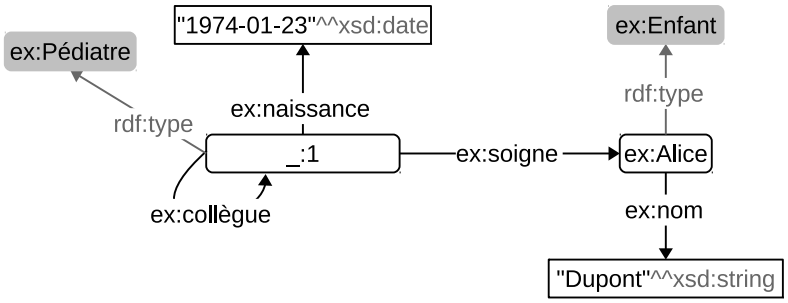


FIGURE 5.12 – Sous-aperçu 1 généré à partir du sous-document en figure 5.10 avec pour ligne de compression {ex:Personne }.

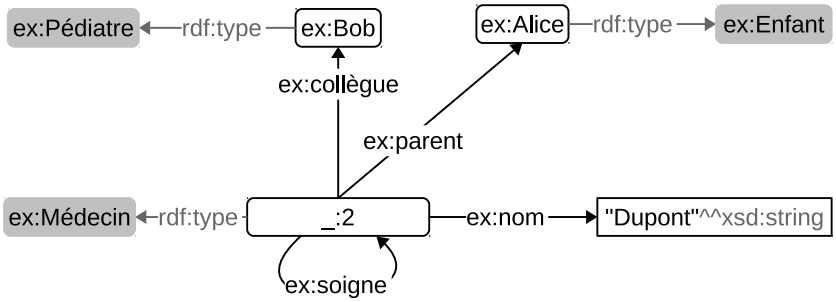


FIGURE 5.13 – Sous-aperçu 2 généré à partir du sous-document en figure 5.11 avec pour ligne de compression {ex:Personne }.

Ces sous-aperçus sont ensuite fusionnés et réduit une dernière fois par la fusion de ex:Bob avec _:1 pour obtenir l'aperçu représenté en figure 4.7.

La séparation des données en sous-documents indépendant permet d'appliquer les règles de réduction en parallèle. Le résultat final est obtenu en effectuant une ultime application sur la réunion des aperçus intermédiaires (avec leurs nœuds blancs étiquetés par des identifiants différents).

3.2 Génération incrémentale d'aperçus de recherche

Dans le modèle d'utilisation proposé en chapitre 4, les aperçus de recherche sont à re-générer régulièrement pour que leur contenu accompagne les changements de leur base d'origine. Dans le Linked Data où les bases sont en constante évolution, ce modèle d'utilisation est contraignant. Il est nécessaire de faire en sorte que les aperçus puissent subir les mêmes mises à jour que leurs sources.

Pour permettre aux aperçus d'être tenus à jour en même temps que leurs bases d'origine, nous proposons des règles de modification de l'aperçu à appliquer lors des mises à jour de la base.

Ces règles de modification gèrent l'ajout et la suppression d'une relation entre deux individus et la modification – nouvelle instanciation ou suppression – du type d'un individu. Ces règles assurent que l'aperçu est toujours cohérent avec la base et qu'aucun triplet résiduel ne reste dans l'aperçu après une modification.

Définition 5.5 (Règle d'ajout de relation entre individus dans l'aperçu)**.** *Soit le triplet (a, r, b) avec a typé par l'ensemble de type T_a et b par l'ensemble de type T_b, ajouté à un document REST dont A est l'aperçu.*

r1 *S'il existe deux nœuds blancs* $na \in \mathcal{N}_A$ *typé par* T_{na} *avec* $T_a \subseteq T_{na}$ *et* $nb \in \mathcal{N}_A$ *typé par* T_b *avec* $T_b \subseteq T_{nb}$*, alors :*

- **r1.1** *si* $\nexists(na, r, nb) \in A$ *alors on ajoute dans* A *un triplet* (na, r, nb) *;*
  Si $\exists(na, r_{bis}, nb) \in A$ *avec* r_{bis} *généralisation de* r*, alors on supprime le triplet* (na, r_{bis}, nb)*.*
- **r1.2** *sinon* A *n'est pas modifié ;*

r2 *sinon, on ajoute* (a, r, b) *à l'aperçu* A *et on applique sur* A *les règles de réduction de la définition 4.5.*

L'ajout d'un triplet dans un aperçu est trivial dans le cas où l'aperçu contient des nœuds blancs instance des types du sujet et de l'objet du triplet (règles **r1**). Il suffit alors d'ajouter la relation du triplet entre ces nœuds blancs, si elle ou une de ses spécialisations n'est pas déjà présente. S'il n'existe pas de nœuds blancs pour les types du sujet et/ou de l'objet, on applique les règles de réduction sur l'aperçu plus le nouveau triplet. De cette façon, les individus qui le doivent sont fusionnés/spécialisés si nécessaire (règle **r2**) et liés aux nœuds blancs déjà existants.

Exemple 15. *On dispose du document représenté en figure 4.2 avec son aperçu utilisant la ligne de compression* {ex:Personne } *représentée en figure 4.5. On lui ajoute le triplet* (ex:Marie, ex:soigne, ex:Bob)*, représenté en figure 5.14.*

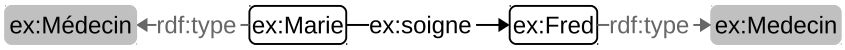


FIGURE 5.14 – Triplet ajouté au document en figure 4.2 et à l'aperçu en figure 4.5 (avec rappel des types des individus).

En suivant la définition 5.5 on ajoute le triplet (ex:Marie, ex:soigne, ex:Bob) *à l'aperçu :*

- *L'individu* ex:Marie *est de type* ex:Médecin *et le nœud blanc* _:2 *est de même type dans l'aperçu, idem pour l'individu* ex:Bob *et le nœud blanc* _:1 *de types* ex:Pédiatre*, la règle* ***r1*** *s'applique.*
- *Il n'existe pas de triplet* (_:2, ex:soigne, _:1) *dans l'aperçu, on l'ajoute donc selon la règle* ***r1.1****. La relation* ex:soigne *n'a pas relation plus générale, il ne peut donc y avoir de triplet à supprimer selon cette règle.*

L'aperçu de recherche obtenu après cet ajout est représenté en figure 5.15.

Définition 5.6 (Règle de suppression d'une relation entre individus dans l'aperçu)**.** *Soit un triplet* (a, r, b)*, à supprimer dans un document REST* D *avec* A *son aperçu de recherche, avec* $a \in \mathcal{N}_D$ *typé par les éléments de* $T_a = \{ta_1, ta_2, \ldots, ta_k\}$*,* $b \in \mathcal{N}_D$ *typé par les éléments de* $T_b = \{tb_1, tb_2, \ldots, tb_l\}$ *et* $r \neq$ rdf:type*.*

Soient les nœuds $na \in \mathcal{N}_A$ *typés par les éléments de* $T_{na} = T_a \cup \{tna_1, tna_2, \ldots, tna_m\}$*, avec* $\forall i \in [1, m]$*,* $tna_i \notin T_a$ *et* $nb \in \mathcal{N}_A$ *typé par les éléments de* $T_{nb} = T_b \cup \{tnb_1, tnb_2, \ldots, tnb_p\}$*, avec* $\forall i \in [1, p]$*,* $tnb_i \notin T_b$*.*

r1 *S'il existe au moins un triplet* (c, r, d) *dans* D *avec,* $\forall i \in [1, m]$ *et* $\forall j \in [1, p]$*,* c *de type* tna_i *et* d *de type* tnb_j *alors on ne modifie pas* A*,*

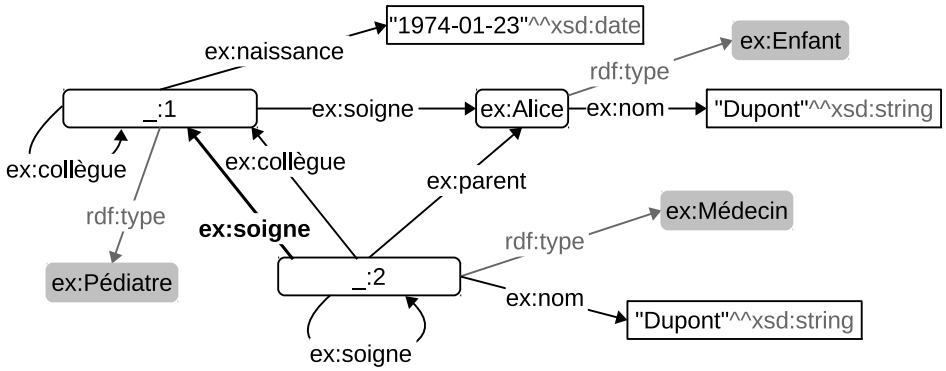


FIGURE 5.15 – Aperçu du document en figure 4.2 après l'ajout du triplet (ex:Marie, ex:soigne, ex:Bob) (modification en traits épais).

r2 *sinon on supprime (na, r, nb) de A.*

Lors de la suppression d'un triplet, on vérifie s'il est toujours nécessaire de faire apparaître la relation du triplet entre les nœuds représentants les sujets et objets du triplet. Pour cela, on vérifie pour chaque combinaison des types de ces nœuds représentants s'il existe encore un triplet entre deux individus de ces types avec la relation du triplet à supprimer. En pratique, la présence d'un triplet ayant la relation `r` pour un couple d'individus de types $\texttt{tna}_i$ et $\texttt{tnb}_j$ est vérifiée par la requête SPARQL simple : `ASK {?s <r>?o .?s a <tna`$_i$`> .?o a <tnb`$_j$`> . }`.

Dans le cas d'un aperçu de recherche contenant des nœuds blancs avec de nombreux types différents – *e.g.* avec une ligne de compression très basse dans la hiérarchie des classes de l'ontologie –, ces règles de suppression peuvent entraîner un très grand nombre de requêtes (jusqu'à $\frac{n!}{2(n-2)!}$ pour n le nombre de types à vérifier).

Exemple 16. *Nous reprenons le document et l'aperçu obtenus à la fin de l'exemple 15 pour supprimer le triplet (ex:Bob, ex:soigne, ex:Alice), représenté en figure 5.16.*

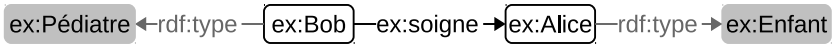


FIGURE 5.16 – Triplet supprimé du document en figure 4.2 et de l'aperçu en figure 4.5 (avec rappel des types des individus).

Il n'existe qu'un triplet contenant la relation ex:soigne entre un individu de la classe ex:Pédiatre, représenté par le nœud blanc _:1 dans l'aperçu, et ex:Alice qui est le seul individu de la classe ex:Enfant. Donc, d'après la règle ***r1*** *de la définition 5.6, le triplet (_:1, ex:soigne, ex:Alice) est supprimé de l'aperçu. L'aperçu de recherche obtenue après cette suppression est représentée en figure 5.17.*

Les relations d'instanciations sont à la base des fusions effectuées dans les aperçus, il est donc nécessaire de les traiter de façon séparée.

Définition 5.7 (Règle de modification d'une relation d'instanciation dans l'aperçu)**.** *Soit un individu a typé originellement par l'ensemble de types T_a et dont l'ensemble des types est changé en l'ensemble T'_a dans un document D dont l'aperçu de recherche est A, avec na $\in \mathcal{N}_A$ typé par T_a :*

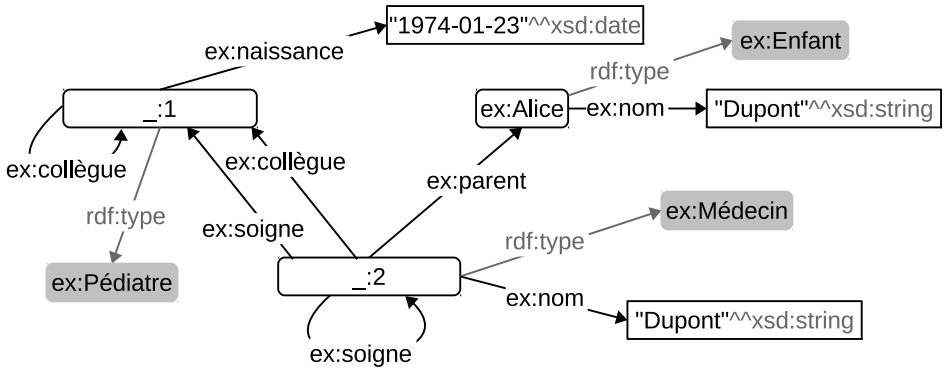


FIGURE 5.17 – Aperçu du document en figure 4.2 après l'ajout du triplet (ex:Marie, ex:soigne, ex:Bob) et la suppression du triplet (ex:Bob, ex:soigne, ex:Alice).

r1 *S'il n'existe plus aucune instance typée par T_a dans D, alors pour tous les triplets (a, r, b) avec b typé par l'ensemble de types T_b et nb son représentant dans A de type T_b, on supprime de A les triplets (na, r, nb) (définition 5.6) ;*

r2 *Pour tout triplet t de D contenant a en sujet ou en objet, on ajoute t à A avec a typé par T'_a (définition 5.5).*

Lors de la modification du type d'un individu, on vérifie si l'individu était l'unique porteur de relations pour les individus de ce type (règle **r1**). On ré-injecte ensuite le voisinage de l'individu avec son nouveau typage pour qu'il soit spécialisé si nécessaire et intégré à sa nouvelle place dans l'aperçu (règle **r2**).

Exemple 17. *En reprenant le document et l'aperçu de recherche de l'exemple 16, nous changeons le type de ex:Alice de ex:Enfant en ex:Médecin.*

D'après la définition 5.7, on effectue les opérations suivantes :

- *L'individu ex:Alice est le seul individu de la classe ex:Enfant dans l'ensemble de triplet, on supprime donc les triplets (ex:Alice , ex:nom, "Dupont") et (_:2, ex:parent, ex:Alice) de l'aperçu, par la règle **r1**.*
- *On ajoute ensuite par la règle **r2** les deux triplets précédemment supprimés en considérant ex:Alice comme individu de la classe ex:Médecin.*
  - *On ajoute le triplet (ex:Alice , ex:nom, "Dupont") à l'aperçu. ex:Alice est de type ex:Médecin comme le nœud blanc _:2 de l'aperçu et il existe déjà un triplet (_:2, ex:nom, "Dupont") dans l'aperçu, par conséquent par la règle **r1.2** de la définition 5.5 l'aperçu n'est pas modifié.*
  - *On ajoute le triplet (_:2, ex:parent, ex:Alice) à l'aperçu. Il n'existe pas de triplet (_:2, ex:parent, _:2) dans l'aperçu, il est donc ajouté.*

L'aperçu obtenu après la modification du type de ex:Alice est représenté en figure 5.18. Le document final après les modifications des trois exemples précédents est représenté en figure 5.19.

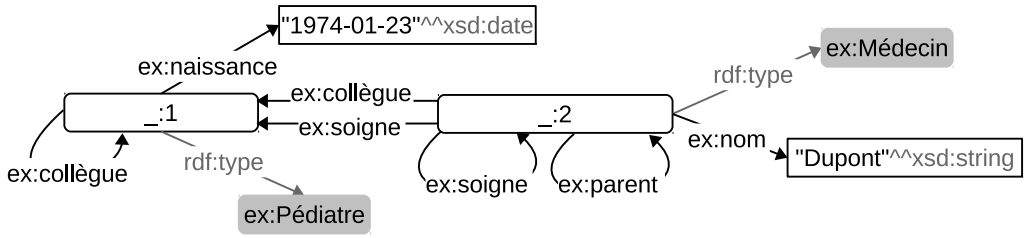


FIGURE 5.18 – Aperçu du document en figure 4.2 après l'ajout du triplet (ex:Marie, ex:soigne, ex:Bob), la suppression du triplet (ex:Bob, ex:soigne, ex:Alice) et la modification du type de ex:Alice en ex:Médecin.

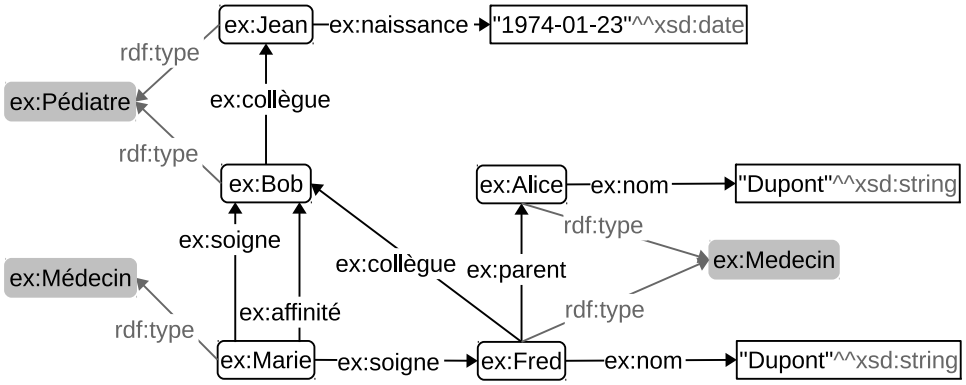


FIGURE 5.19 – Document en figure 4.2 après les modifications des exemples 15, 16 et 17.

4 Conclusion

Pour conclure, nous avons présenté plusieurs implémentations qui complètent les aspects pratiques des chapitres précédents. Ces implémentations ont été utilisées lors de nos différentes expérimentations. Les algorithmes et règles présentées reposent sur des requêtes SPARQL pour interagir avec les bases RDF, leurs performances sont donc en partie dépendantes des performances des moteurs SPARQL.

Conclusion

Le Web des données forme un nouveau domaine d'innovations et d'exploitations en pleine expansion. Aujourd'hui, presque 15 ans après sa première spécification par Tim Berners-Lee, le Web des données arrive à une certaine maturité. Cette maturité entraîne un nouveau regain des recherches dans l'optimisation et la maintenance des bases.

Les contributions de cette thèse sont des réponses originales à plusieurs problèmes récurrents qui apparaissent tout au long de la vie des bases du Web des données. Elles répondent ensembles aux trois problématiques du maintien de la qualité des données, de l'évolution des données et de l'ontologie d'une base, et de l'interrogation des données dans un milieu distribué, tel que le Linked Data. Nos contributions se basent toutes sur des technologies existantes et couramment utilisées dans le Web des données afin de faciliter leurs mises en place dans des bases existantes et profiter des optimisations qui sont progressivement faites sur ces technologies.

En pratique, nos contributions visent les trois catégories générales d'utilisateurs qu'on peut trouver sur le Web des données : l'évaluation de la qualité aide les contributeurs, c'est-à-dire ceux qui ajoutent et mettent à jour les données d'une base RDF ; l'examen de la base aide les experts responsables de son évolution, c'est-à-dire les seuls habilités à modifier à la fois l'ontologie et les données d'une base RDF ; l'interrogation aide tous les utilisateurs, c'est-à-dire toutes les personnes qui consultent le Web des données.

Notre méthode d'évaluation des mises à jour se démarque d'une partie des travaux existants en ne donnant pas de rôle particulier à l'ontologie, c'est-à-dire en considérant l'ontologie comme une partie quelconque de la base. Dans cette situation, notre contribution utilise l'abondance actuelle de données – apportée par la croissance du Linked Data – pour évaluer les mises à jour en fonction des structures de données récurrentes dans la base plutôt qu'en fonction des structures définies officiellement dans l'ontologie de la base. Notre méthode d'évolution s'intéresse à l'adéquation entre données et ontologie ; elle se base également sur les structures de données récurrentes dans la base, mais elle les sélectionne en fonction de l'ontologie et les utilise pour faire évoluer ontologie et données. Notre méthode d'interrogation est totalement guidée par l'ontologie de la base pour la génération des aperçus de recherche. Nos trois contributions adoptent donc chacune un point de vue différent vis-à-vis de l'ontologie.

Nous avons adopté ces différents points de vue après avoir constaté un manque de respect de l'ontologie croissant dans les bases du Linked Data. Ce manque de respect est en partie expliqué par l'absence de raisonneur efficace et répandu dans le Web des données. L'existence d'un tel raisonneur donnerait une importance plus évidente à l'ontologie pour les utilisateurs du fait de son utilisation possible lors de l'interrogation des bases RDF. Au début des recherches sur le Web des données, de nombreuses personnes ont présenté la possibilité de raisonnement sur les données RDF comme l'atout majeur du Web des

données. Aujourd'hui, les raisonneurs ne sont utilisés que marginalement lors de l'interrogation, du fait probable de la complexité des algorithmes associés. Le constat est qu'à l'heure actuelle les bases connues, comme DBPedia ou Geonames par exemple, ajoutent toutes les relations de typage lors de la création de données. Face à la faiblesse actuelle des raisonneurs dans le Web des données, nous avons orienté nos travaux d'évaluation et d'évolution de sorte qu'ils soient peu contraints par l'ontologie, sans pour autant l'écarter.

Les expérimentations effectuées pour nos trois contributions ont donné des résultats réels encourageants et prometteurs. Des perspectives peuvent être proposées.

Nous avons testé notre méthode d'évaluation de la qualité sur DBPedia qui est une des bases les plus importantes du Linked Data en termes de contenu et de communauté. Nous souhaitons par la suite tester notre méthode avec une autre base communautaire qui aborderait par exemple un domaine de connaissance différent ou qui aurait une méthode de contribution différente. À noter qu'il est actuellement difficile d'obtenir les données des mises à jour des bases du Linked Data autres que DBPedia. Notre méthode d'évaluation de la qualité a été conçue de façon à pouvoir s'intégrer avec d'autres méthodes d'évaluation des données. Une étude des méthodes d'évaluation qui pourraient le mieux s'intégrer avec la nôtre est à faire.

Notre méthode d'aide à l'évolution de l'ontologie et des données laisse la décision finale complexe et délicate des modifications à apporter à un expert ; une évaluation de son efficacité est donc difficile. Nous avons pu observer que notre méthode met bien en évidence les problèmes de DBPedia déjà repérés manuellement dans d'autres travaux. Une étude des motifs détectés par notre méthode dans d'autres bases étudiées de la même manière que DBPedia serait intéressante à mener.

Notre méthode d'interrogation distribuée a été testée sur un ensemble de bases virtuelles mais contenant des données réelles. Une étude de notre approche d'interrogation sur les bases du Linked Data dans leur intégralité reste à faire. Pour cela, il nous est encore nécessaire de tester la parallélisation de la génération d'aperçus présentée en chapitre 5, section 3 en effectuant la génération des aperçus de recherche de bases réelles du Linked Data, qui comptent plusieurs millions de triplets. De plus, il serait intéressant de tester notre approche en hébergeant les aperçus de recherche soit localement, soit sur un serveur central servant de « hub » d'interrogation du Linked Data. Enfin, une extension du protocole SPARQL pour ce mode d'interrogation serait des plus intéressante.

Index des définitions

Table des figures

Bibliographie

[Apache Software Foundation, 2012] Apache Software Foundation (2012). Jena Framework. http://jena.apache.org. (Cité en page 83.)

[Auer et al., 2006] Auer, S., Dietzold, S., and Riechert, T. (2006). Ontowiki–a tool for social, semantic collaboration. In *The Semantic Web-ISWC 2006*, pages 736–749. Springer. (Cité en page 55.)

[Baader et al., 2005] Baader, F., Brandt, S., and Lutz, C. (2005). Pushing the el envelope. In *IJCAI*, volume 5, pages 364–369. (Cité en page 23.)

[Beckett, 2004] Beckett, D. (2004). RDF/xml syntax specification (revised). W3C recommendation, W3C. http ://www.w3.org/TR/REC-rdf-syntax/. (Cité en page 17.)

[Bergroth et al., 2000] Bergroth, L., Hakonen, H., and Raita, T. (2000). A survey of longest common subsequence algorithms. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, pages 39–48. IEEE. (Cité en page 94.)

[Berners-Lee and Connolly, 2011] Berners-Lee, T. and Connolly, D. (2011). Notation3 (N3) : A readable RDF syntax. Team submission, W3C. http://www.w3.org/TeamSubmission/n3/. (Cité en page 40.)

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., Lassila, O., et al. (2001). The semantic web. *Scientific american*, 284(5) :28–37. (Cité en pages 7 et 13.)

[Biron and Malhotra, 2004] Biron, P. V. and Malhotra, A. (2004). XML Schema Part 2 : Datatypes Second Edition. W3C Recommendation, W3C. http://www.w3.org/TR/xmlschema-2/. (Cité en page 16.)

[Bizer and Cyganiak, 2009] Bizer, C. and Cyganiak, R. (2009). Quality-driven information filtering using the wiqa policy framework. *Web Semantics : Science, Services and Agents on the World Wide Web*, 7(1) :1–10. (Cité en pages 10 et 40.)

[Bizer et al., 2009a] Bizer, C., Heath, T., and Berners-Lee, T. (2009a). Linked data-the story so far. *Semantic Services, Interoperability and Web Applications : Emerging Concepts*, pages 205–227. (Cité en pages 8 et 13.)

[Bizer et al., 2009b] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009b). Dbpedia - A crystallization point for the web of data. *Web Semantics : science, services and agents on the world wide web*, 7(3) :154–165. (Cité en page 8.)

[Bollacker et al., 2008] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase : a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM. (Cité en page 32.)

[Bonatti et al., 2011] Bonatti, P. A., Hogan, A., Polleres, A., and Sauro, L. (2011). Robust and scalable linked data reasoning incorporating provenance and trust annotations. *Web Semantics : Science, Services and Agents on the World Wide Web*, 9(2) :165–201. (Cité en pages 10, 40 et 52.)

[Bühmann and Lehmann, 2013] Bühmann, L. and Lehmann, J. (2013). Pattern based knowledge base enrichment. In *The Semantic Web–ISWC 2013*, pages 33–48. Springer. (Cité en page 56.)

[Buil-Aranda et al., 2013] Buil-Aranda, C., Hogan, A., Umbrich, J., and Vandenbussche, P.-Y. (2013). Sparql web-querying infrastructure : Ready for action ? In *The Semantic Web–ISWC 2013*, pages 277–293. Springer. (Cité en page 33.)

[Campinas et al., 2011] Campinas, S., Ceccarelli, D., Perry, T. E., Delbru, R., Balog, K., and Tummarello, G. (2011). The sindice-2011 dataset for entity-oriented search in the web of data. In *Proceedings of the 1st International Workshop on Entity-Oriented Search (EOS)*, pages 26–32. (Cité en page 68.)

[Campinas et al., 2012] Campinas, S., Perry, T. E., Ceccarelli, D., Delbru, R., and Tummarello, G. (2012). Introducing rdf graph summary with application to assisted sparql formulation. In *Database and Expert Systems Applications (DEXA), 2012 23rd International Workshop on*, pages 261–266. IEEE. (Cité en page 68.)

[Carothers, 2014] Carothers, G. (2014). RDF 1.1 n-quads. W3C recommendation, W3C. http ://www.w3.org/TR/2014/REC-n-quads-20140225/. (Cité en page 18.)

[Carothers and Prud'hommeaux, 2014] Carothers, G. and Prud'hommeaux, E. (2014). RDF 1.1 turtle. W3C recommendation, W3C. http://www.w3.org/TR/turtle/. (Cité en page 17.)

[Cherix et al., 2014] Cherix, D., Usbeck, R., Both, A., and Lehmann, J. (2014). Lessons learned—the case of crocus : Cluster-based ontology data cleansing. In *The Semantic Web : ESWC 2014 Satellite Events*, pages 14–24. Springer. (Cité en page 56.)

[Civili et al., 2013] Civili, C., Console, M., De Giacomo, G., Lembo, D., Lenzerini, M., Lepore, L., Mancini, R., Poggi, A., Rosati, R., Ruzzi, M., et al. (2013). Mastro studio : Managing ontology-based data access applications. *Proceedings of the VLDB Endowment*, 6(12) :1314–1317. (Cité en page 24.)

[Colucci et al., 2014] Colucci, S., Giannini, S., Donini, F. M., and Di Sciascio, E. (2014). A deductive approach to the identification and description of clusters in linked open data. In *ECAI 14*. (Cité en pages 94 et 115.)

[Cyganiak et al., 2011] Cyganiak, R., Zhao, J., Hausenblas, M., and Alexander, K. (2011). Describing linked datasets with the VoID vocabulary. W3C note, W3C. http://www.w3.org/TR/void/. (Cité en page 28.)

[Demter et al., 2012] Demter, J., Auer, S., Martin, M., and Lehmann, J. (2012). Lodstats—an extensible framework for high-performance dataset analytics. In *Proceedings of the EKAW 2012*, Lecture Notes in Computer Science (LNCS) 7603. Springer. (Cité en page 33.)

[Djedidi and Aufaure, 2010] Djedidi, R. and Aufaure, M.-A. (2010). Onto-evo a l an ontology evolution approach guided by pattern modeling and quality evaluation. In *Foundations of information and knowledge systems*, pages 286–305. Springer. (Cité en page 55.)

[Elbassuoni et al., 2011] Elbassuoni, S., Ramanath, M., and Weikum, G. (2011). Query relaxation for entity-relationship search. In *The Semanic Web : Research and Applications*, pages 62–76. Springer. (Cité en page 88.)

[Erxleben et al., 2014] Erxleben, F., Günther, M., Krötzsch, M., Mendez, J., and Vrandečić, D. (2014). Introducing wikidata to the linked data web. In *The Semantic Web–ISWC 2014*, pages 50–65. Springer. (Cité en page 32.)

[Fleischhacker et al., 2014] Fleischhacker, D., Paulheim, H., Bryl, V., Völker, J., and Bizer, C. (2014). Detecting errors in numerical linked data using cross-checked outlier detection. In *The Semantic Web–ISWC 2014*, pages 357–372. Springer. (Cité en page 56.)

[Fokoue et al., 2006] Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., and Srinivas, K. (2006). The summary abox : Cutting ontologies down to size. In *Proc. of the Internat. Semantic Web Conference (ISWC'06)*, volume 4273, pages 343–356. Springer. (Cité en page 68.)

[Fokoue et al., 2012] Fokoue, A., Meneguzzi, F., Sensoy, M., and Pan, J. Z. (2012). Querying linked ontological data through distributed summarization. In *AAAI*. (Cité en pages 11 et 68.)

[Fürber and Hepp, 2011] Fürber, C. and Hepp, M. (2011). Towards a vocabulary for data quality management in semantic web architectures. In *Proceedings of the 1st International Workshop on Linked Web Data Management*, pages 1–8. ACM. (Cité en page 40.)

[Gallego et al., 2011] Gallego, M. A., Fernández, J. D., Martínez-Prieto, M. A., and de la Fuente, P. (2011). An empirical study of real-world sparql queries. In *Internat. Workshop on Usage Analysis and the Web of Data (USEWOD'2011)*. ACM. (Cité en pages 81 et 82.)

[Gao et al., 2010] Gao, X., Xiao, B., Tao, D., and Li, X. (2010). A survey of graph edit distance. *Pattern Analysis and applications*, 13(1) :113–129. (Cité en page 47.)

[Gearon et al., 2013] Gearon, P., Passant, A., and Polleres, A. (2013). SPARQL 1.1 Update. W3C Recommendation, W3C. http://www.w3.org/TR/sparql11-update/. (Cité en page 24.)

[Görlitz and Staab, 2011] Görlitz, O. and Staab, S. (2011). Splendid : Sparql endpoint federation exploiting void descriptions. *COLD*, 782. (Cité en page 67.)

[Grau et al., 2012] Grau, B. C., Patel-Schneider, P., and Motik, B. (2012). OWL 2 web ontology language direct semantics (second edition). W3C recommendation, W3C. http://www.w3.org/TR/owl2-direct-semantics/. (Cité en page 21.)

[Guéret et al., 2012] Guéret, C., Groth, P., Stadler, C., and Lehmann, J. (2012). Assessing linked data mappings using network measures. In *The Semantic Web : Research and Applications*, pages 87–102. Springer. (Cité en page 40.)

[Guéret et al., 2011] Guéret, C., Wang, S., Groth, P., and Schlobach, S. (2011). Multi-scale analysis of the web of data : a challenge to the complex system's community. *Advances in Complex Systems*, 14(04) :587–609. (Cité en page 51.)

[Guha and Brickley, 2004] Guha, R. and Brickley, D. (2004). RDF vocabulary description language 1.0 : RDF schema. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. (Cité en page 7.)

[Haarslev et al., 2012] Haarslev, V., Hidde, K., Möller, R., and Wessel, M. (2012). The racerpro knowledge representation and reasoning system. *Semantic Web Journal*, 3(3) :267–277. (Cité en page 24.)

[Harris and Seaborne, 2013] Harris, S. and Seaborne, A. (2013). SPARQL 1.1 Query Language. W3C Recommendation, W3C. http://www.w3.org/TR/sparql11-query/. (Cité en page 24.)

[Hartig et al., 2009] Hartig, O., Bizer, C., and Freytag, J.-C. (2009). Executing sparql queries over the web of linked data. In *Proceedings of the 8th International Semantic Web Conference*, pages 293–309. Springer-Verlag. (Cité en page 67.)

[Hogan et al., 2010] Hogan, A., Harth, A., Passant, A., Decker, S., and Polleres, A. (2010). Weaving the pedantic web. In *Proceedings of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010*. (Cité en page 33.)

[Hogan et al., 2012] Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., and Decker, S. (2012). An empirical survey of linked data conformance. *Web Semantics : Science, Services and Agents on the World Wide Web*, 14 :14–44. (Cité en page 33.)

[Horrocks et al., 2004] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL : A Semantic Web Rule Language Combining OWL and RuleML. Team submission, W3C. http://www.w3.org/Submission/SWRL/. (Cité en page 23.)

[Hurtado et al., 2008] Hurtado, C. A., Poulovassilis, A., and Wood, P. T. (2008). Query relaxation in rdf. In *Journal on data semantics X*, pages 31–61. Springer. (Cité en page 88.)

[Jaccard, 1901] Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, (37) :547–579. (Cité en page 47.)

[Jacobi et al., 2011] Jacobi, I., Kagal, L., and Khandelwal, A. (2011). Rule-based trust assessment on the semantic web. In *Rule-Based Reasoning, Programming, and Applications*, pages 227–241. Springer. (Cité en pages 40 et 52.)

[Javed et al., 2013] Javed, M., Abgaz, Y. M., and Pahl, C. (2013). Ontology change management and identification of change patterns. *Journal on Data Semantics*, 2(2-3) :119–143. (Cité en page 55.)

[Kazakov and Klinov, 2014] Kazakov, Y. and Klinov, P. (2014). Goal-directed tracing of inferences in el ontologies. In *The Semantic Web–ISWC 2014*, pages 196–211. Springer. (Cité en page 24.)

[Köhler et al., 2011] Köhler, S., Bauer, S., Mungall, C. J., Carletti, G., Smith, C. L., Schofield, P., Gkoutos, G. V., and Robinson, P. N. (2011). Improving ontologies by automatic reasoning and evaluation of logical definitions. *BMC bioinformatics*, 12(1) :418. (Cité en pages 10 et 55.)

[Kontokostas et al., 2014] Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., and Zaveri, A. (2014). Test-driven evaluation of linked data quality. In *Proceedings of the 23rd international conference on World Wide Web*, pages 747–758. International World Wide Web Conferences Steering Committee. (Cité en pages 10, 40 et 63.)

[Kontokostas et al., 2013] Kontokostas, D., Zaveri, A., Auer, S., and Lehmann, J. (2013). Triplecheckmate : A tool for crowdsourcing the quality assessment of linked data. In *Knowledge Engineering and the Semantic Web*, pages 265–272. Springer. (Cité en page 41.)

[Lehmann et al., 2014] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2014). DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*. (Cité en page 31.)

[Mahfoudh et al., 2015] Mahfoudh, M., Thiry, L., Forestier, G., and Hassenforder, M. (2015). Une nouvelle formalisation des changements ontologiques composés et complexes. In *15èmes Journées Francophones Extraction et Gestion des Connaissances, EGC 2015, 27-30 Janvier 2015, Luxembourg*, pages 263–274. (Cité en page 55.)

[Maillot et al., 2014] Maillot, P., Raimbault, T., Genest, D., and Loiseau, S. (2014). Targeted linked data extractor. In *ICAART 2014 - Proceedings of the 6th International Conference on Agents and Artificial Intelligence*. (Cité en pages 34 et 81.)

[Mendes et al., 2012] Mendes, P. N., Mühleisen, H., and Bizer, C. (2012). Sieve : linked data quality assessment and fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 116–123. ACM. (Cité en pages 40 et 52.)

[Mortensen et al., 2013] Mortensen, J. M., Musen, M. A., and Noy, N. F. (2013). Crowdsourcing the verification of relationships in biomedical ontologies. In *AMIA Annual Symposium Proceedings*, volume 2013, page 1020. American Medical Informatics Association. (Cité en page 56.)

[Motik et al., 2009] Motik, B., Horrocks, I., and Sattler, U. (2009). Bridging the gap between owl and relational databases. *Web Semantics : Science, Services and Agents on the World Wide Web*, 7(2) :74–89. (Cité en page 60.)

[Mottin et al., 2014] Mottin, D., Lissandrini, M., Velegrakis, Y., and Palpanas, T. (2014). Exemplar queries : Give me an example of what you need. *Proceedings of the VLDB Endowment*, 7(5). (Cité en pages 47 et 88.)

[Nikitina et al., 2012] Nikitina, N., Rudolph, S., and Glimm, B. (2012). Interactive ontology revision. *Web Semantics : Science, Services and Agents on the World Wide Web*, 12 :118–130. (Cité en pages 10 et 55.)

[OWL Working Group, 2009] OWL Working Group, W. (27 October 2009). OWL 2 Web Ontology Language : Document Overview. Technical report. http://www.w3.org/TR/owl2-overview/. (Cité en page 21.)

[Pérez et al., 2009] Pérez, J., Arenas, M., and Gutierrez, C. (2009). Semantics and complexity of sparql. *Transactions on Database Systems (TODS)*, 34(3) :16. (Cité en pages 81 et 83.)

[Prud'hommeaux and Aranda, 2013] Prud'hommeaux, E. and Aranda, C. B. (2013). SPARQL 1.1 federated query. W3C recommendation, W3C. http://www.w3.org/TR/sparql11-federated-query/. (Cité en page 67.)

[Prud'hommeaux and Seaborne, 2008] Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C recommendation, W3C. http://www.w3.org/TR/rdf-sparql-query/. (Cité en page 8.)

[Raimbault and Maillot, 2013] Raimbault, T. and Maillot, P. (2013). Vues d'ensembles de documents rdf. In *Actes du 31ème congrès INFORSID*, pages 387–402. (Cité en page 82.)

[Rieß et al., 2010] Rieß, C., Heino, N., Tramp, S., and Auer, S. (2010). Evopat–pattern-based evolution and refactoring of rdf knowledge bases. In *The Semantic Web–ISWC 2010*, pages 647–662. Springer. (Cité en page 55.)

[Ruckhaus et al., 2014] Ruckhaus, E., Vidal, M.-E., Castillo, S., Burguillos, O., and Baldizan, O. (2014). Analyzing linked data quality with liquate. In *The Semantic Web : ESWC 2014 Satellite Events*, pages 488–493. Springer. (Cité en page 56.)

[Schenk et al., 2011] Schenk, S., Dividino, R., and Staab, S. (2011). Using provenance to debug changing ontologies. *Web Semantics : Science, Services and Agents on the World Wide Web*, 9(3) :284–298. (Cité en page 40.)

[Schmidt et al., 2010] Schmidt, M., Meier, M., and Lausen, G. (2010). Foundations of sparql query optimization. In *Proc. of the Internat. Conference on Database Theory (ICDT'10)*, pages 4–33. ACM. (Cité en pages 27, 81 et 83.)

[Schreiber and Dean, 2004] Schreiber, G. and Dean, M. (2004). OWL web ontology language reference. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-owl-ref-20040210/. (Cité en pages 7 et 21.)

[Schwarte et al., 2011] Schwarte, A., Haase, P., Hose, K., Schenkel, R., and Schmidt, M. (2011). Fedx : Optimization techniques for federated query processing on linked data. In *The Semantic Web–ISWC 2011*, pages 601–616. Springer. (Cité en page 67.)

[Seaborne and Carothers, 2014a] Seaborne, A. and Carothers, G. (2014a). RDF 1.1 n-triples. W3C recommendation, W3C. http ://www.w3.org/TR/2014/REC-n-triples-20140225/. (Cité en page 17.)

[Seaborne and Carothers, 2014b] Seaborne, A. and Carothers, G. (2014b). RDF 1.1 trig. W3C recommendation, W3C. http://www.w3.org/TR/trig/. (Cité en page 18.)

[Sheng et al., 2012] Sheng, Z., Wang, X., Shi, H., and Feng, Z. (2012). Checking and handling inconsistency of dbpedia. In *Web Information Systems and Mining*, pages 480–488. Springer. (Cité en pages 33, 53 et 63.)

[Sirin et al., 2007] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007). Pellet : A practical owl-dl reasoner. *Web Semantics : science, services and agents on the World Wide Web*, 5(2) :51–53. (Cité en page 24.)

[Sirin and Tao, 2009] Sirin, E. and Tao, J. (2009). Towards integrity constraints in owl. In *OWLED*, volume 529. (Cité en pages 61 et 63.)

[Töpper et al., 2012] Töpper, G., Knuth, M., and Sack, H. (2012). Dbpedia ontology enrichment for inconsistency detection. In *Proceedings of the 8th International Conference on Semantic Systems*, pages 33–40. ACM. (Cité en pages 33 et 53.)

[Wang et al., 2013] Wang, X., Tiropanis, T., and Davis, H. C. (2013). LHD : optimising linked data query processing using parallelisation. In *Proceedings of the WWW2013 Workshop on Linked Data on the Web*. (Cité en page 67.)

[Wood et al., 2014] Wood, D., Lanthaler, M., and Cyganiak, R. (2014). RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, W3C. http://www.w3.org/TR/rdf11-concepts/. (Cité en pages 7, 15 et 16.)

[Zhang et al., 2007] Zhang, X., Cheng, G., and Qu, Y. (2007). Ontology summarization based on rdf sentence graph. In *Proc. of the Internat. World Wide Web Conference (WWW'07)*, pages 707–716. ACM. (Cité en page 68.)

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

université angers

Thèse de Doctorat

Pierre MAILLOT

Nouvelles méthodes pour l'évaluation, l'évolution et l'interrogation des bases du Web des données

New methods to evaluate, check and query the Web of data

Résumé

Le Web des données offre un environnement de partage et de diffusion des données, selon un cadre particulier qui permet une exploitation des données tant par l'humain que par la machine. Pour cela, le framework RDF propose de formater les données en phrases élémentaires de la forme (sujet, relation, objet), appelées triplets. Les bases du Web des données, dites bases RDF, sont des ensembles de triplets. Dans une base RDF, l'ontologie organise la description des données factuelles. Le nombre et la taille des bases du Web des données n'a pas cessé de croître depuis sa création en 2001. Cette croissance s'est même accélérée depuis l'apparition du mouvement du Linked Data en 2008 qui encourage le partage et l'interconnexion de bases publiquement accessibles sur Internet. L'utilisation et l'ajout de données sont faits par des communautés d'utilisateurs avec le soutien d'outils insuffisamment matures pour diagnostiquer le contenu d'une base ou pour interroger ensemble les bases du Web des données. Notre thèse propose trois méthodes pour encadrer le développement, tant factuel qu'ontologique, et pour améliorer l'interrogation des bases du Web des données. Nous proposons d'abord une méthode pour évaluer la qualité des modifications des données factuelles lors d'une mise à jour par un contributeur. Nous proposons ensuite une méthode pour faciliter l'examen de la base par la mise en évidence de groupes de données factuelles en conflit avec l'ontologie. Nous proposons enfin une méthode d'interrogation dans un environnement distribué qui interroge uniquement les bases susceptibles de fournir une réponse.

Mots clés

Web Sémantique, Linked Data, RDF, SPARQL, Interrogation distribuée, Qualité des données, Extraction de motifs, Base de connaissances communautaires.

Abstract

The Web of data is a mean to share and broadcast data user-readable data as well as machine-readable data. This is possible thanks to RDF which propose the formatting of data into short sentences (subject, relation, object) called triples. Bases from the Web of data, called RDF bases, are sets of triples. In a RDF base, the ontology – structural data – organize the description of factual data. Since the Web of Data creation in 2001, the number and sizes of RDF bases have been constantly rising. This increase has accelerated since the apparition of Linked Data, which promote the sharing and interlinking of publicly available bases by user communities. The exploitation – interrogation and edition – by theses communities is made without adequate solution to evaluate the quality of new data, check the current state of the bases or query together a set of bases. This thesis proposes three methods to help the expansion at factual and ontological level and the querying of bases from the Web of Data. We propose a method designed to help an expert to check factual data in conflict with the ontology. Finally we propose a method for distributed querying limiting the sending of queries to bases that may contain answers.

Key Words

Semantic Web, Linked Data, RDF, SPARQL, Distributed interrogation, Data quality, Pattern extraction, Crowd-sourced knowledge base.