THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *12/12/2013* **par :**

CAMILLE PRADEL

D'un langage de haut niveau à des requêtes graphes permettant d'interroger le web sémantique

---

Directeurs de Thèse :

Ollivier HAEMMERLÉ
Professeur des Universités
Université Toulouse II - Le Mirail

Nathalie HERNANDEZ
Maître de Conférences
Université Toulouse II - Le Mirail

Rapporteurs :

Isabelle MIRBEL
Maître de Conférences HDR
Université Nice Sophia Antipolis

Marie-Christine ROUSSET
Professeur des Universités
Université de Grenoble

Éxaminateurs :

Nathalie AUSSENAC-GILLES
Directrice de Recherches
IRIT - CNRS

Patrice BUCHE
Ingénieur de Recherche HDR
Montpellier SupAgro

Claudette CAYROL
Professeur des Universités
Université Toulouse III - Paul Sabatier

Olivier CURÉ
Maître de Conférences HDR
Université de Marne-la-Vallée

École doctorale et spécialité :
MITT ; Domaine STIC : Intelligence Artificielle

Unité de Recherche :
Institut de Recherche en Informatique de Toulouse

Résumé

Les modèles graphiques sont de bons candidats pour la représentation de connaissances sur le Web, où tout est graphes : du graphe de machines connectées via Internet au "Giant Global Graph" de Tim Berners-Lee, en passant par les triplets RDF et les ontologies.

Dans ce contexte, le problème crucial de l'interrogation ontologique est le suivant : est-ce qu'une base de connaissances composée d'une partie terminologique et d'une partie assertionnelle implique la requête, autrement dit, existe-t-il une réponse à la question ? Ces dernières années, des logiques de description ont été proposées dans lesquelles l'expressivité de l'ontologie est réduite de façon à rendre l'interrogation calculable (familles DL-Lite et EL). OWL 2 restreint OWL-DL dans ce sens en se fondant sur ces familles.

Nous nous inscrivons dans le contexte d'utilisation de formalismes graphiques pour la représentation (RDF, RDFS et OWL) et l'interrogation (SPARQL) de connaissances. Alors que les langages d'interrogation fondés sur des graphes sont présentés par leurs promoteurs comme étant naturels et intuitifs, les utilisateurs ne pensent pas leurs requêtes en termes de graphes. Les utilisateurs souhaitent des langages simples, proches de la langue naturelle, voire limités à des mots-clés. Nous proposons de définir un moyen générique permettant de transformer une requête exprimée en langue naturelle vers une requête exprimée dans le langage de graphe SPARQL, à l'aide de patrons de requêtes. Le début de ce travail coïncide avec les actions actuelles du W3C visant à préparer une nouvelle version de RDF, ainsi qu'avec le processus de standardisation de SPARQL 1.1 gérant l'implication dans les requêtes.

Abstract

Graph models are suitable candidates for KR on the Web, where everything is a graph, from the graph of machines connected to the Internet, the "Giant Global Graph" as described by Tim Berners-Lee, to RDF graphs and ontologies.

In that context, the ontological query answering problem is the following: given a knowledge base composed of a terminological component and an assertional component and a query, does the knowledge base implies the query, i.e. is there an answer to the query in the knowledge base ? Recently, new description logic languages have been proposed where the ontological expressivity is restricted so that query answering becomes tractable. The most prominent members are the DL-Lite and the EL families. In the same way, the OWL-DL language has been restricted and this has led to OWL2, based on the DL-Lite and EL families.

We work in the framework of using graph formalisms for knowledge representation (RDF, RDF-S and OWL) and interrogation (SPARQL). Even if interrogation languages based on graphs have long been presented as a natural and intuitive way of expressing information needs, end-users do not think their queries in terms of graphs. They need simple languages that are as close as possible to natural language, or at least mainly limited to keywords. We propose to define a generic way of translating a query expressed in a high-level language into the SPARQL query language, by means of query patterns. The beginning of this work coincides with the current activity of the W3C that launches an initiative to prepare a possible new version of RDF and is in the process of standardizing SPARQL 1.1 with entailments.

Remerciements

Je tiens à remercier tout d'abord Ollivier Haemmerlé et Nathalie Hernandez pour avoir dirigé ma thèse. Ils ont su m'insuffler leur dynamisme et leur ambition, et me faire découvrir tous les aspects ~~du métier~~ de la passion d'enseignant-chercheur.

Je souhaite exprimer toute ma reconnaissance à Marie-Christine Rousset et à Isabelle Mirbel pour l'honneur qu'elles me font en acceptant d'être les rapportrices pour mon mémoire ; leur lecture attentive et leurs remarques ont permis d'en améliorer la rédaction.

Je veux également remercier vivement Claudette Cayrol, Nathalie Aussenac-Gilles, Patrice Buche et Olivier Curé d'avoir accepté de faire partie de mon jury de thèse.

Je remercie tous les membres de l'équipe Melodi pour leur réponses à mes sollicitations, les conseils qu'ils m'ont prodigués et les nombreuses discussions qui ont permis d'enrichir mon travail.

Je remercie Anne Hernandez pour ses relectures rigoureuses de nos publications anglaises. Dans la même famille, je remercie Raphaël et Elsa pour m'avoir appris à travailler en autonomie.

Je tiens également à remercier Rudi Studer ainsi que tous les membres de l'équipe *Knowledge Management* de m'avoir accueilli à Karlsruhe pendant trois mois. J'y ai vécu le printemps le plus froid de ma vie, mais j'y ai également connu l'un des accueils les plus chaleureux.

Je remercie l'ensemble des stagiaires avec qui j'ai eu la chance de travailler, Anna, Samuel, Fabien, Guillaume et Pascal, qui ont su me tenir compagnie et produire des travaux de qualité tout en subissant les affres de mes premières expériences d'encadrement.

Je remercie mes étudiants pour tout ce qu'ils m'ont apporté. Dans les périodes de doute et de confusion, récurrentes au cours d'une thèse, chaque cours est une bouffée d'air frais.

Je veux aussi exprimer toute ma sympathie, mes encouragements et mes vœux de réussite envers les doctorants de l'Irit. Merci pour cette bonne humeur, ces échanges (scientifiques et autres) passionnants, ces soirées et ces Mr. Freeze !

J'ai également une pensée affectueuse pour mes amis, ceux qui me permettent

de sortir du labo. Malgré des choix différents et des vies d'adultes occupés, on maintient des projets communs et c'est un véritable bonheur !

Enfin, je tiens à remercier sincèrement ma famille, et plus particulièrement mes parents. Leur soutien inconditionnel, leur patience, les encouragements qu'ils m'ont prodigués et la confiance qu'ils m'ont témoignée sont ici en partie récompensés.

Table des matières

Introduction

Le world wide web (ou web), lancé en 1989, est devenu aujourd'hui la première source de connaissances au monde. Dans sa forme originelle, il consiste en un système d'information décentralisé, constitué d'un ensemble de documents (ou pages web) reliés les uns aux autres par des liens hypertexte. Un site web est un ensemble de pages web liées accessible à une adresse donnée. Pour accéder au contenu d'un site web, il faut "visiter" son adresse à l'aide d'un navigateur web. Dans la mesure où il est nécessaire de connaître au préalable l'adresse d'un site pour le consulter, ce fonctionnement ne favorise pas la découverte de nouvelles connaissances par un utilisateur.

Le web était à ses débuts constitué d'un nombre relativement faible de sites auxquels on accédait par l'intermédiaire d'annuaires web qui représentaient des portes d'entrée sur le web. Un annuaire web est un site web proposant des liens vers d'autres sites, classés par catégories et sous-catégories et visant à orienter le visiteur vers les sites web relatifs à ses centres d'intérêt. L'explosion de la taille du web et de la diversité de ses contenus a rendu cette méthode d'accès à l'information inadaptée.

Les moteurs de recherche ont constitué une réponse efficace à ces évolutions. Ces systèmes, le plus souvent commerciaux, sont encore aujourd'hui le moyen privilégié d'accès aux informations contenues sur le web. Le web et les usages qu'en font les utilisateurs étant en constante mutation, les moteurs de recherche doivent s'adapter sans cesse pour rester concurrentiels.

La principale limite de ce fonctionnement réside dans le fait que les machines n'appréhendent pas le sens des documents qu'elles traitent. En effet, les pages web ne sont à l'origine que de simples données textuelles accompagnées d'informations de mise en page. Ceci restreint les capacités des moteurs de recherche actuels, notamment en ce qui concerne les requêtes complexes ou nécessitant la fusion d'informations contenues dans des documents distincts [Hallett et al., 2007].

Le web sémantique [Berners-Lee et al., 2001], une évolution du web documentaire qui interconnecte non plus seulement des documents mais des ressources en tous genres, veut résoudre ces problèmes en ajoutant une couche de sémantique au web : les données contenues sur le web sémantique sont formellement définies

et liées de façon à faciliter la découverte, l'intégration, l'exploitation et le partage d'informations. Le web sémantique se fonde sur l'héritage des travaux de recherche en représentation des connaissances et utilise le formalisme des ontologies [Gruber, 1993] pour rendre son contenu compréhensible par les machines. Là encore, les moteurs de recherche s'adaptent et apprennent peu à peu à exploiter le potentiel de ces nouveautés.

L'avènement des ordiphones, des tablettes tactiles et de l'internet mobile a, lui aussi, bouleversé les usages du web. On estime qu'aujourd'hui les périphériques mobiles sont utilisés à hauteur de 40% du temps pour consommer des services d'internet.

Les moteurs de recherche dans leur forme traditionnelle montrent leurs limites face à ces transformations, et le besoin de mettre en place de nouveaux paradigmes d'accès à l'information se fait sentir. Les interfaces en langue naturelle, systèmes dans lesquels l'utilisateur interagit en s'exprimant dans la langue utilisée quotidiennement, ne sont pas une idée nouvelle et ont déjà fait l'objet de nombreux travaux, que ce soit pour interfacer des systèmes de gestion de base de données ou de corpus de texte. Ces travaux ont permis le développement de systèmes élaborés. Cependant ces systèmes doivent faire un compromis entre la portabilité et de bonnes performances de recherche [Kaufmann and Bernstein, 2010]. La représentation de connaissances fondée sur des ontologies, pour la première fois à très grande échelle sur le web sémantique, a relancé l'intérêt pour les interfaces en langue naturelle. On pense en effet qu'une ontologie de domaine, vecteur des connaissances liées à ce domaine, peut être exploitée par une interface en langue naturelle générique pour s'adapter naturellement au domaine.

On constate aujourd'hui que les acteurs majeurs du web sont en train de prendre ce virage. Facebook, par exemple, a mis en place en janvier 2013 *Graph Search*[1], un nouveau moteur de recherche interne répondant à des requêtes en langue naturelle sur les utilisateurs et les informations qui les caractérisent (amis, centre d'intérêt...). De son côté, Google enrichit depuis quelques années son *Knowledge Graph*[2] et a très récemment déployé une mise à jour majeure de son algorithme de recherche, nommée *Hummingbird* et visant à retourner directement, en plus d'une liste de documents pertinents, la réponse à la requête posée par l'utilisateur [Monde, 2013].

Le travail présenté dans ce mémoire se situe dans ce contexte. L'approche que nous proposons veut fournir aux utilisateurs finals un moyen d'interroger des bases de connaissances sous forme de graphes à l'aide de requêtes exprimées en langue naturelle, et ce dans le but d'éviter à ces utilisateurs de se confronter à la complexité de la formulation d'une requête graphe dans un langage tel que SPARQL. Cette

1. https://www.facebook.com/about/graphsearch
2. http://www.google.com/insidesearch/features/search/knowledge.html

approche se caractérise par l'utilisation de *patrons de requêtes* préétablis pour guider le processus d'interprétation. Nos travaux se fondent en effet sur le postulat selon lequel, dans les applications réelles, les requêtes formulées par les utilisateurs sont pour l'essentiel des variations autour de quelques familles typiques de requêtes.

Dans notre approche, chaque patron de requêtes représente une de ces familles de requêtes. Les patrons de requêtes sont constitués d'un graphe représentant le besoin en informations couvert par le patron et faisant référence à des ressources de la base de connaissances cible, et d'un modèle de phrase descriptive permettant de générer des phrases en langue naturelle présentées à l'utilisateur.

Le processus d'interprétation proposé est décomposé en deux étapes principales, avec un résultat intermédiaire qui est la *requête pivot*. La requête pivot consiste en une première interprétation de la requête utilisateur dans laquelle le besoin en information est exprimé sous une forme proche d'une requête par mots-clés, mais dans laquelle il est possible d'exprimer des relations entre les mots-clés. Cette organisation présente deux avantages principaux : elle permet de représenter les informations importantes issues de l'analyse syntaxique de la requête utilisateur, et elle facilite la mise en œuvre du multilinguisme dans notre approche. En effet, la requête pivot est un format intermédiaire indépendant de la langue, et peut donc être traitée de la même façon quel que soit le langage employé par l'utilisateur. Ainsi, pour adapter notre approche à un nouveau langage, il suffit de modifier la première étape.

Cette première grande étape consiste en l'interprétation de la requête utilisateur et sa traduction en requête pivot. Pour cela, des opérations classiques de traitement automatique des langues (identification des entités nommées, détermination des catégories grammaticales, analyse de dépendances) sont appliquées à la requête en langue naturelle, puis des règles de transformation préétablies sont utilisées pour générer la requête pivot à partir de l'arbre de dépendances de la phrase. Dans la seconde étape, les patrons de requêtes sont associés à la requête pivot pour obtenir une liste d'interprétations possibles de cette requête (et donc de la requête utilisateur d'origine) qui sont ensuite ordonnées en fonction de leurs pertinences supposées. Les interprétations peuvent ainsi être soumises à l'utilisateur sous forme de phrases descriptives générées à partir des modèles de phrases descriptives de chaque patron.

Notre approche a été implémentée dans un système appelé *Swip*, pour *Semantic Web Interface using Patterns*. Ce prototype fait un usage poussé des capacités d'un moteur de requêtes SPARQL pour effectuer les opérations d'association de patrons, et a fait l'objet de plusieurs évaluations, notamment dans le cadre de la compétition *Question Answering over Linked Data* [Lopez et al., 2013] (*QALD*).

Ce mémoire présente le contexte, la démarche et les résultats de notre travail visant à exploiter des patrons de requêtes pour interpréter des requêtes utilisateur

exprimées en langue naturelle et les traduire en requêtes graphes formelles destinées à interroger une base de connaissances sur un domaine donné. Il est structuré en trois parties.

La première partie expose le contexte scientifique et technologique. Le chapitre 1 présente le web sémantique, ses enjeux, certains de ses verrous, ses applications et les technologies qui le caractérisent. Dans le chapitre 2, après un aperçu des différentes méthodes et outils issus du Traitement Automatique des Langues et utiles au domaine des interfaces en langue naturelle, nous établissons un état de l'art des principales approches et systèmes proposés jusque-là.

La deuxième partie détaille l'approche que nous proposons. Une des particularités de cette approche est qu'elle est fondée sur l'utilisation de patrons de requêtes, qui sont présentés dans le chapitre 3, de façon intuitive puis formelle. Le chapitre 4 décrit l'étape de traduction de la requête utilisateur en langue naturelle vers la requête pivot, et le chapitre 5 la seconde étape, celle de formalisation de la requête pivot.

La troisième et dernière partie présente les contributions du point de vue pratique de notre travail. Le chapitre 6 présente l'implémentation en insistant sur l'exploitation poussée de SPARQL lors de l'étape de formalisation de la requête pivot. Enfin, le chapitre 7 expose et analyse les résultats obtenus aux différentes expérimentations menées pour évaluer notre approche et son implémentation.

Première partie

État de l'art

Chapitre 1

Le web sémantique : notions et mise en œuvre

Les travaux menés depuis un demi-siècle sur la représentation de connaissances à l'aide de graphes étiquetés trouvent aujourd'hui une concrétisation à très grande échelle sur le web sémantique, qui constitue le champ d'application privilégié des travaux présentés dans cette thèse et est à ce titre présenté dans ce chapitre. Le web sémantique se veut une évolution du web actuel et se définit d'un point de vue technique par une pile de recommandations définies par le W3C [1] (*World Wide Web Consortium*), qui est l'organisme de normalisation des technologies du web. Nous définissons ici les motivations et concepts généraux du web sémantique, ainsi que quelques-unes des recommandations nécessaires à la compréhension de nos travaux.

La section 1.1 inscrit le web sémantique dans l'histoire de la représentation des connaissances en présentant les formalismes qui ont particulièrement influencé les choix faits pour sa mise en place. Les principes généraux sont introduits dans la section 1.2, puis le modèle de graphe RDF, la définition d'ontologies en RDFS et OWL, et l'interrogation de données avec SPARQL sont détaillés dans les sections 1.3, 1.4 et 1.5 respectivement. La section 1.6 fait l'état des lieux du déploiement du web sémantique via le web de données et présente quelques verrous limitant ce déploiement. Enfin, la section 1.7 conclut ce premier chapitre.

1.1 La représentation des connaissances

Depuis des siècles, les philosophes ont tenté de classifier les choses et d'analyser leurs propriétés afin de mieux comprendre le monde qui les entoure. Le philosophe grec Platon (429-347 avant J.C.) établissait déjà des catégories à partir de questions

1. http://www.w3.org/

fondamentales sur la réalité, l'existence et la nature réelle des choses. Ce champ de la philosophie est aujourd'hui connu sous le terme *Ontologie* ; cette discipline consiste en "l'étude de l'être en tant qu'être", d'après la définition d'Aristote (384-322 avant J.C.), étudiant de Platon.

Parmi les autres contributions majeures à la représentation des connaissances, on compte les *taxonomies* (ou *taxinomies*) introduites par Carolus Linneaus (1707-1778) comme un moyen de classifier les formes de vie. Le terme *taxonomie* fait maintenant référence à la science de la classification, mais aussi aux schémas de classifications hiérarchiques, produits de cette science. Dans ce deuxième sens, une taxonomie est une structure hiérarchique organisant les différents concepts d'un domaine selon la relation de subsomption (ou relation "sorte de"). Aux $XVIII^{ème}$ et $XIX^{ème}$ siècles, les connaissances en biologie, en médecine ou encore en astronomie ont permis la construction de très grandes taxonomies, certaines faisant encore référence aujourd'hui.

La communauté de représentation des connaissances s'est approprié le terme *ontologie* au cours des années quatre-vingt dix pour désigner l'objet issu d'un processus de modélisation de connaissances. La définition la plus consensuelle d'une ontologie est celle introduite par Gruber [Gruber, 1993] et étendue par Borst [Borst, 1997] : une ontologie est une spécification formelle explicite d'une conceptualisation partagée d'un domaine donné. [Studer et al., 1998] détaille cette définition : *formelle* se réfère au fait que la spécification doit être lisible par une machine, *explicite* signifie que les types des concepts et les contraintes sur leur utilisation sont explicitement définis, *conceptualisation* se réfère à un modèle abstrait d'un certain phénomène du monde reposant sur l'identification des concepts pertinents de ce phénomène et *partagée* se rapporte à la notion selon laquelle une ontologie capture la connaissance consensuelle, qui n'est pas propre à un individu mais validée par un groupe.

Il est possible de faire un parallèle avec le monde des bases de données relationnelles, où le schéma d'une base de données peut être vu comme une ontologie, et les données des assertions utilisant le vocabulaire de cette ontologie. Il reste cependant une différence fondamentale : la base de données relationnelle fait l'hypothèse d'un monde clos, c'est-à-dire que toutes les informations sont présentes dans la base et que ce qui n'y est pas asserté est considéré comme faux.

Plusieurs formalismes ont été proposés et exploités par différentes communautés de l'Intelligence Artificielle (IA) pour représenter et inférer des connaissances. Deux familles de langages ont particulièrement influencé les choix faits pour la représentation des connaissances au sein du web sémantique : les *logiques de description* et les *graphes conceptuels*. Ces formalismes ont un ancêtre commun, les *réseaux sémantiques*. Dans la suite, nous définissons la notion de connaissance, puis présentons brièvement chacun de ces trois formalismes, avant de décrire les choix

effectués pour le web sémantique. Les technologies destinées à mettre en œuvre ces choix seront présentées dans les sections suivantes.

1.1.1 Donnée, information, connaissance

Il convient de marquer les différences entre les notions de donnée, d'information et de connaissance, qui sont parfois assimilées. Les frontières entre ces notions ne sont cependant pas claires et peuvent varier en fonction du contexte.

Une donnée est le résultat d'une mesure. Elle peut être collectée par un capteur, une personne, ou extraite d'une base de données. Une donnée n'est pas porteuse de sens en elle-même ; elle n'est ni vraie, ni fausse. Sa finalité est d'être transmise, stockée et surtout traitée. La mesure de température issue d'une station météo est un exemple de donnée.

Des données peuvent être structurées pour produire de l'information. Cette information porte alors un sens, soit pour le système, soit pour l'humain. Les relevés de la température sur la place du Capitole, à intervalle précis et sur une période donnée sont un exemple d'information.

L'information devient connaissance lorsqu'elle est reçue dans un contexte donné et interprétée par un humain sur la base des connaissances qu'il possède déjà suite à son expérience personnelle. L'énoncé "la température à Toulouse est en moyenne plus élevée durant les mois d'été que durant les mois d'hiver" est un exemple de connaissance. Dans le domaine de l'Intelligence Artificielle (IA), l'information devient connaissance lorsqu'elle sert de fondement à une inférence menant à de nouvelles connaissances. [Kayser, 1997] définit une inférence comme une façon générique de désigner l'ensemble des mécanismes par lesquels des entrées (perceptives ou non) sont combinées à des connaissances préalables afin d'obtenir des comportements élaborés.

On peut distinguer les *connaissances assertionnelles* des *connaissances ontologiques*. Les connaissances assertionnelles (ou factuelles) décrivent des faits précis et établis (par exemple, *Elsa est née le 5 avril 2013*, ou *Nathalie est la mère de Raphaël*), alors que les connaissances ontologiques (ou terminologiques, ou intentionnelles) décrivent le monde à un plus haut niveau d'abstraction (par exemple, *un humain a exactement une date de naissance, une mère biologique et un père biologique*). On peut également voir les connaissances ontologiques comme les "règles" qui régissent le monde décrit et les connaissances assertionnelles comme la description du "contenu" de ce monde.

1.1.2 Les réseaux sémantiques

Un réseau sémantique est une représentation graphique de connaissances humaines [Quillian, 1966; Collins and Quillian, 1969]. Il prend la forme d'un graphe

étiqueté et orienté et permet ainsi d'exprimer des relations entre des nœuds. La figure 1.1 illustre un réseau sémantique.

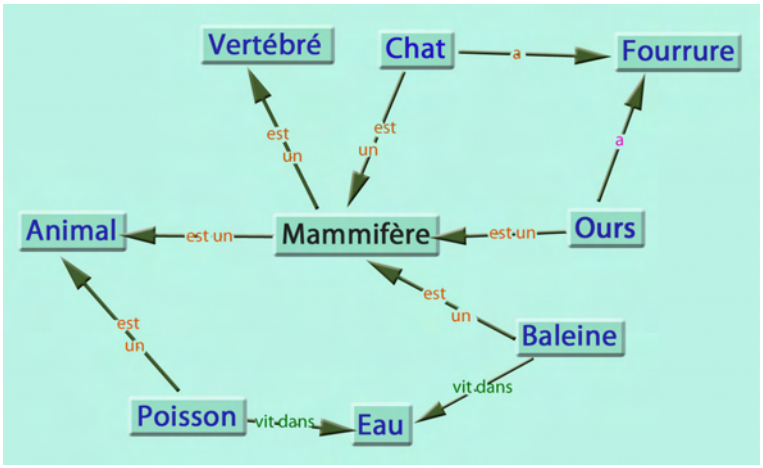


FIGURE 1.1 – Exemple de réseau sémantique (tiré de *Wikipedia*)

Les réseaux sémantiques ont été principalement critiqués pour l'absence de distinction entre connaissances ontologiques et factuelles, et le manque d'une sémantique formelle (ils peuvent être interprétés intuitivement de différentes manières). Les graphes conceptuels et les logiques de description ont par la suite été développés pour répondre à ces critiques.

1.1.3 Les graphes conceptuels

Les graphes conceptuels (GC) sont nés d'une volonté de corriger les défauts des réseaux sémantiques tout en conservant l'aspect graphique rendant leur utilisation plus intuitive que des formalismes fondés sur la logique. Dans le modèle des graphes conceptuels [Sowa, 1976, 1984; Chein and Mugnier, 1992], il y a une séparation claire entre partie ontologique et partie assertionnelle : un support représente la connaissance ontologique et les graphes conceptuels construits sur ce support représentent la connaissance assertionnelle relative à la connaissance ontologique.

Un support est une structure $S = (T_C, T_R, I)$, où :

- T_C est un ensemble fini de types de concept muni d'un ordre partiel $\leq$ et possédant un supremum $\top$ et un infimum $\bot$;
- T_R est un ensemble fini de relations d'arité quelconque, muni d'un ordre partiel $\leq$, tel que seules les relations de même arité sont comparables ;
- I est un ensemble non nécessairement fini dont les éléments sont appelés marqueurs individuels ; le symbole $*$ désigne le marqueur générique, avec $* \notin I$. L'ensemble des marqueurs $I \cup \{*\}$ est muni de l'ordre partiel $\leq$, tel

que $*$ est plus grand que n'importe quel marqueur individuel ($\forall m \in I, m \leq *$) et deux marqueurs individuels ne sont pas comparables.

La figure 1.2 montre un sous-ensemble d'une hiérarchie de concepts sur le domaine du cinéma.

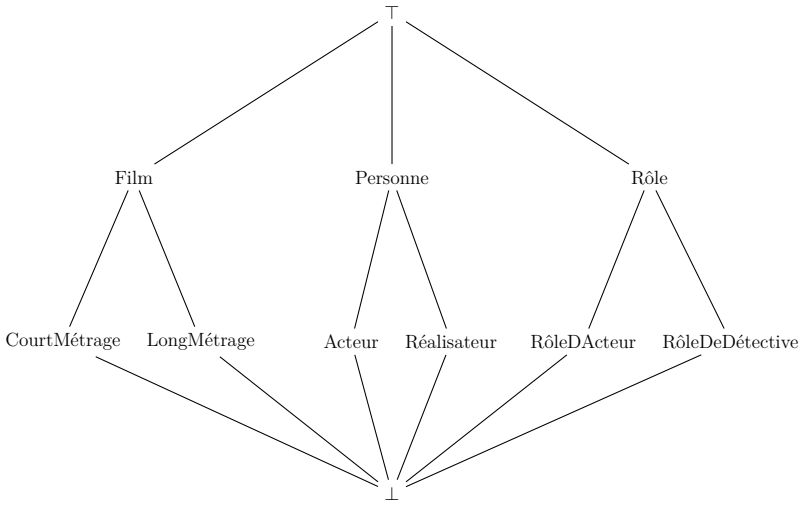


FIGURE 1.2 – Sous-ensemble d'une hiérarchie de concepts sur le domaine du cinéma

Un graphe conceptuel $G = (C_G, R_G, E_G, l_G)$ sur un support $S = (T_C, T_R, I)$ est un multigraphe, non orienté et biparti, où C_G est l'ensemble des sommets concepts, R_G l'ensemble des sommets relations, E_G le multi-ensemble des arêtes et l_G la fonction d'étiquetage des sommets et des arêtes, telle que :

- un sommet concept est étiqueté par un type de concept t et un marqueur m tels que $(t, m) \in T_C \times (I \cup \{*\})$;
- un sommet relation est étiqueté par une relation $r \in T_R$, et le nombre d'arêtes incidentes à ce sommet est égal à l'arité de r ;
- les k arêtes incidentes à un sommet relation sont totalement ordonnées. On note $(c_1 \dots c_k)$ la liste des arguments du sommet relation selon cet ordre.

La figure 1.3 exprime que le film *The Artist* a pour acteur Jean Dujardin et pour réalisateur Michel Hazanavicius.

FIGURE 1.3 – Un graphe conceptuel utilisant les concepts de la figure 1.2

Les GC sont munis d'une sémantique en logique du premier ordre (LPO), par l'intermédiaire d'une fonction appelée Φ. L'interprétation $\Phi(S)$ d'un support S en LPO est telle que :

– $\forall(t,\, t') \in T_C^2,\, \Phi(t \leq t') = \forall x,\, t(x) \rightarrow t'(x)$
– $\forall(t,\, t') \in T_R^2,\, \Phi(t \leq t') = \forall x_1, \ldots,\, x_n,\, t(x_1, \ldots,\, x_n) \rightarrow t'(x_1, \ldots,\, x_n)$
– $\forall i \in I,\, \Phi(i) = i$

L'interprétation $\Phi(G)$ d'un graphe conceptuel G en LPO est telle que:
– $\forall c \in C_G$ avec $l_G(c) = (t,\, *)$,$\Phi(c) = t(x)$, où x est une nouvelle variable,
– $\forall c \in C_G$ avec $l_G(c) = (t,\, i)$ et $i \in I$, $\Phi(c) = t(m)$,
– $\forall r \in R_G$ avec $l_G(r) = t$, r d'arité k, et $(c_1 \ldots c_k)$ les k voisins de t, $\Phi(r) = r(x_1 \ldots x_k)$, avec $\forall i = 1, \ldots, k$, x_i est une constante ou la variable introduite dans $\Phi(c_i)$.

La communauté des graphes conceptuels a développé un courant de recherche exploitant les opérations de la *théorie des graphes*, dont l'homomorphisme de graphes, pour effectuer des raisonnements. En particulier, la vérification de l'existence d'un lien de subsomption entre deux graphes peut être effectuée à l'aide de la recherche d'un homomorphisme de graphes particulier, en ce qu'il autorise des restrictions d'étiquettes. L'étiquette de l'image d'un sommet peut être une spécialisation de l'étiquette du sommet antécédent. Cette opération, appelée *projection*, présente de bonnes propriétés algorithmiques : la recherche d'une projection d'un graphe dans un autre est un problème NP-complet, mais de nombreux cas polynomiaux imposant des contraintes raisonnables sur la structure des graphes ont été mis en évidence.

1.1.4 Les logiques de description

Les logiques de description (LD), issues des travaux de Brachman [Brachman and Schmolze, 1985], adoptent un point de vue plus logique et se focalisent sur l'axiomatisation de l'ontologie.

Les LD sont des fragments (le plus souvent) décidables de la logique du premier ordre (LPO), et forment donc une famille de langages offrant différents niveaux de compromis entre expressivité et complexité algorithmique. L'idée générale consiste à décrire des concepts de façon à pouvoir ensuite classifier automatiquement des instances dans ces concepts. La palette d'opérateurs autorisés pour la définition de ces concepts dépend du langage de LD choisi. Une base de connaissances en LD est composée d'une *Tbox* (partie ontologique) et d'une *Abox* (partie assertionnelle). La Tbox peut être vue comme une ontologie ; elle définit un modèle pour la Abox qui contient les données assertionnelles liées à cette ontologie.

L'univers décrit en LD est constitué d'individus (ou instances), appartenant à des concepts (ou classes), et reliés entre-eux par des rôles (ou propriétés). Un individu représente un élément et correspond à un terme en LPO ; $jeanDujardin$, $theArtist$ sont des exemples d'individus. Un concept représente un ensemble d'éléments et correspond en LPO à un prédicat unaire ; $Acteur$, $Film$ sont des exemples de concepts et $Acteur(jeanDujardin)$ exprime le fait que l'élément $jeanDujardin$

appartient à l'ensemble *Acteur*. Un rôle représente une relation binaire entre éléments et correspond en LPO à un prédicat binaire ; *joueDans*, *aPourPère* sont des exemples de rôles.

Des concepts complexes peuvent être définis via deux types d'opérateurs :

- les contructeurs ensemblistes :
  - le concept universel (*top*) $\top$ correspond à l'ensemble de l'univers Δ,
  - le concept absurde (*bottom*) $\bot$ correspond à l'ensemble vide $\emptyset$,
  - le complément $\neg C$ correspond à l'ensemble $\{x|\neg C(x)\}$,
  - l'union $C \sqcup D$ correspond à l'ensemble $\{x|C(x)\} \cup \{x|D(x)\}$,
  - l'intersection $C \sqcap D$ correspond à l'ensemble $\{x|C(x)\} \cap \{x|D(x)\}$,
  - l'extension $\{a_1, a_2, \ldots, a_n\}$ correspond à l'ensemble $\{a_1, a_2, \ldots, a_n\}$,
- les restrictions :
  - la restriction existentielle $\exists rC$ correspond à l'ensemble $\{x|\exists y,\, r(x,y) \wedge C(y)\}$,
  - la restriction universelle $\forall rC$ correspond à l'ensemble $\{x|\forall y,\, r(x,y) \rightarrow C(y)\}$,
  - la restriction de cardinalité $= nrC$ correspond à l'ensemble $\{x|\,|\{y|r(x,y) \wedge C(y)\}| = n\}$,
  - la restriction de cardinalité minimale $\geq nrC$ correspond à l'ensemble $\{x|\,|\{y|r(x,y) \wedge C(y)\}| \geq n\}$,
  - la restriction de cardinalité maximale $\leq nrC$ correspond à l'ensemble $\{x|\,|\{y|r(x,y) \wedge C(y)\}| \leq n\}$.

Ainsi, l'expression $\exists\ ajouéDans\ Film$ désigne l'ensemble des élements qui ont joué dans un film. De même, $\forall\ aPourEnfant\ Fille$ désigne l'ensemble des éléments qui n'ont que des filles comme enfant(s) ou pas d'enfant du tout.

Dans l'expression d'une restriction, lorsque le concept C n'est pas spécifié, il est implicitement remplacé par le concept universel $\top$. Ainsi, $\geq 2\ aPourEnfant$ désigne l'ensemble des éléments qui ont au moins deux enfants.

Des rôles complexes peuvent également être définis :

- le rôle inverse r^- correspond à la relation $\{(x,y)|r(y,x)\}$,
- le rôle composé $r \circ s$ correspond à la relation $\{(x,y)|\exists z,\, r(x,z) \wedge s(z,y)\}$,
- le complément $\neg r$ correspond à la relation $\{(x,y)|\neg r(x,y)\}$,

Ainsi, les expressions équivalentes $joueDans(jeanDujardin,\ theArtist)$ et $joueDans^-(theArtist,\ jeanDujardin)$ expriment que (l'acteur) Jean Dujardin joue dans (le film) *The Artist*.

Enfin, les axiomes d'inclusion permettent de déclarer des relations de subsomption entre concepts et propriétés :

- l'inclusion de concept $C \sqsubseteq D$ exprime que le concept D subsume le concept C : $\forall x,\, C(x) \rightarrow D(x)$
- l'inclusion de rôle $r \sqsubseteq s$ exprime que le rôle s subsume le rôle r :

$\forall x,\ y,\ r(x,y) \rightarrow s(x,y)$

Par exemple, $LongMétrage \sqsubseteq Film$ exprime que tout élément du concept $LongMétrage$ est également élément de $Film$, et l'expression $aPourContributeur \sqsubseteq aPourActeur$ exprime qu'une relation $aPourActeur$ entre deux éléments implique une relation $aPourContributeur$ entre ces deux mêmes éléments.

1.1.5 Des formalismes historiques vers ceux du web sémantique

Les LD et les GC semblent adaptés pour représenter des connaissances à l'échelle du web. Dotés d'une sémantique formelle, ils présentent de bonnes propriétés de complexité moyennant quelques limitations maîtrisées. Enfin, tous les deux font l'hypothèse du monde ouvert (c'est-à-dire qu'un système qui raisonne avec de telles connaissances doit tenir compte du fait qu'il n'a pas accès à l'intégralité des connaissances), ce qui est indispensable dans un système d'information décentralisé tel que le web.

Les LD se sont particulièrement illustrées dans l'expression d'axiomes riches et le raisonnement au niveau ontologique. Les GC exploitent un formalisme de graphe intuitif et consistant avec la structure du web qui prend lui-même la forme d'un grand graphe de documents reliés par des liens hypertexte ; ils ont de plus montré de très bons résultats dans l'interrogation de données au niveau assertionnel. Poussée ces dernières années par les problématiques du web sémantique, la communauté des LD a développé de nouvelles familles de LD, dites légères, moins expressives mais plus adaptées au problème de l'interrogation de données, et celle des GC s'est attachée à étudier comment exprimer les principaux axiomes ontologiques à l'aide de règles de graphes.

En définissant les formalismes et technologies destinés à représenter les connaissances sur le web (RDF(S) et OWL, présentés plus loin dans ce chapitre), le W3C a voulu tirer parti du meilleur de ces deux univers.

1.2 Principes généraux

Dans son ouvrage *Weaving the Web* [Berners-Lee et al., 2000], Tim Berners-Lee donne sa vision du futur du web : *I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A "Semantic Web", which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The "intelligent agents" people have touted for ages will finally materialize.*

Ces quelques lignes résument très bien les motivations du web sémantique. Tim Berners-Lee en donnera ensuite une définiton plus aboutie dans [Berners-Lee et al., 2001]. Le web sémantique désigne une évolution du web syntaxique. Ce dernier est entièrement destiné à la consultation de documents par des utilisateurs humains. Dans le web sémantique, les contenus sont également compréhensibles et exploitables par des agents logiciels. Sa mise en place permettrait l'automatisation d'un grand nombre de tâches et un accès plus facile à la connaissance, en s'appuyant sur l'utilisation d'un langage de description commun, permettant d'exprimer à la fois des ontologies et des assertions (éventuellement sous formes d'annotations dans des documents web) utilisant le vocabulaire de ces ontologies.

D'un point de vue technologique, le web sémantique peut se définir par la pile de standards qui le composent et fixent le cadre de sa mise en place. Un schéma largement diffusé, appelé "semantic web layer cake" et présenté dans la figure 1.4, décrit cette pile de standards et constitue à lui seul un résumé de la feuille de route dressée par Tim Berners-Lee (le schéma de la figure est une évolution de la version originale proposée en 2000 [Berners-Lee, 2000]). L'agencement en pile illustre l'organisation et les dépendances entre les standards, chaque couche exploitant les couches inférieures.

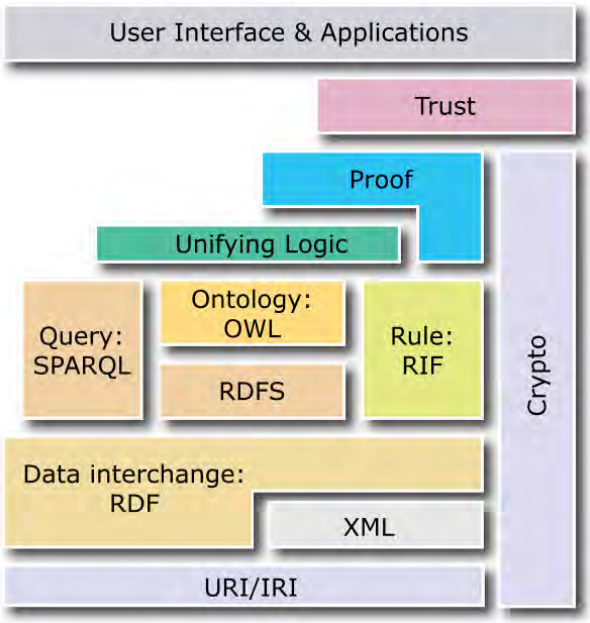


FIGURE 1.4 – La pile des standards du web sémantique

Le web sémantique repose sur quelques concepts fondamentaux que nous présentons dans la suite. L'*URL* (*Universal Resource Locator*) est une chaîne de caractères encodée en ASCII qui permet d'identifier de manière unique un document sur

la toile. L'*URI* (*Universal Resource Identifier*) a une fonction analogue mais plus large puisqu'il permet d'identifier n'importe quelle ressource, même abstraite, et l'*IRI* (*International Resource Identifier*) est une version internationalisée (utilisant Unicode) de l'URI. Par exemple, `http://www.irit.fr/` est un URL qui identifie la page d'accueil du site web de l'Irit ; puisque cette chaîne de caractères est un URL, elle est également un URI et un IRI. `http://www.irit.fr/ontologies/cinema#JeanDujardin` est un URI (et donc un IRI) qui identifie l'acteur Jean Dujardin, alors que `http://www.irit.fr/ontologies/cinema#BéréniceBejo` est uniquement un IRI car elle contient des caractères non compris dans le jeu de caractères ASCII.

RDF (*Resource Description Framework*) constitue la couche de plus bas niveau parmi les standards spécifiques au web sémantique. Il s'agit d'un modèle de graphe étiqueté permettant de décrire des ressources de façon formelle. Il exploite le formalisme des URI et impose une forme de triplets *sujet* - *prédicat* - *objet* pour l'expression de connaissances. Une description plus détaillée de ses caractéristiques est réalisée dans la section 1.3.

Les langages de schémas que sont RDFS et OWL, en définissant des ontologies intégrées au cadre du web sémantique, permettent de mener des inférences sur des données RDF. En RDFS, il est possible d'exprimer les axiomes les plus simples (dont la subsomption entre classes et entre propriétés) et de définir ainsi des ontologies dites légères. OWL offre une plus grande expressivité en définissant des constructeurs inspirés de ceux des LD permettant d'exprimer des classes. Ces langages sont présentés dans la section 1.4.

SPARQL (*SPARQL Protocol and RDF Query Language*) [Group, 2013] est le standard préconisé par le W3C pour interroger des bases de triplets RDF. La recommandation SPARQL 1.0 définit trois outils liés à ce standard assurant l'interopérabilité dans l'échange de données : un langage de requête (*SPARQL Query Language for RDF* [Prud'hommeaux and Seaborne, 2008]), un protocole permettant l'échange de requêtes et de résultats (*SPARQL Protocol for RDF* [Clark, 2008]), et une spécification XML des résultats (*SPARQL Query Results XML Format* [Beckett and Broekstra, 2013]). La recommandation SPARQL 1.1 définit de nouvelles fonctionnalités, les plus marquantes pour notre travail étant une méthode de mise à jour de données RDF (*SPARQL Update* [Gearon et al., 2013]) et de nouvelles fonctions pour le langage de requêtes, dont notamment les fonctions d'agrégats. Dans nos travaux, nous exploitons particulièrement les spécificités du langage d'interrogation et les capacités de mise à jour. Ces éléments sont présentés plus en détail dans la section 1.5.

Enfin, les autres couches de la pile des recommandations sont moins essentielles à la présentation de notre travail. *RIF* [Kifer and Boley, 2013] (*Rule Interchange Format*) est un format d'échange de règles créant une interface entre les différents

langages de règles existant. La couche *cryptography* (cryptographie) permet d'assurer grâce à la signature numérique de données RDF que les informations échangées proviennent d'une source légitime. Ces briques sont encore en cours de définiton et n'ont pas fait l'objet d'une recommandation officielle.

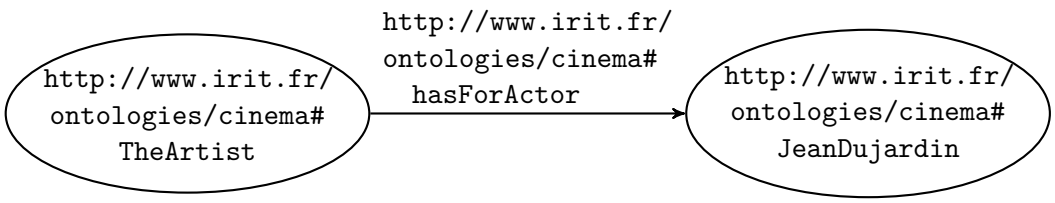
1.3 Le modèle de graphe RDF

Triplet et URI RDF est un modèle de graphe dont l'unité atomique de connaissance est le triplet. Un triplet RDF décrit une ressource (le sujet) qui est liée par une propriété (le prédicat) à une valeur de propriété (l'objet). Un ensemble de triplets RDF peut être vu comme un graphe orienté étiqueté dont les nœuds correspondent aux ressources (sujets et objets des triplets) et les arcs aux prédicats (dirigés du sujet vers l'objet de chaque triplet). Le triplet <`http://www.irit.fr/ontologies/cinema#TheArtist`, `http://www.irit.fr/ontologies/cinema#hasForActor`, `http://www.irit.fr/ontologies/cinema#JeanDujardin`> est représenté sous forme de graphe dans la figure 1.5(a). L'utilisation d'URI pour identifier les ressources décrites garantit l'homogénéité de RDF avec l'architecture du web.

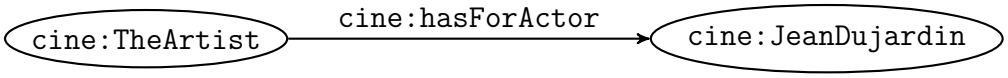
Préfixes Pour des raisons de lisibilité, nous adoptons dans la suite de ce travail le mécanisme de préfixe permettant d'alléger la notation de ces URI. Cette méthode, inspirée des espaces de nommage XML, permet de substituer le début d'un URI (qui est le plus souvent dans un contexte local peu pertinent et commun à plusieurs URI) par un identifiant plus court. Par exemple, en définissant un préfixe qui a pour nom `cine` et pour valeur `http://www.irit.fr/ontologies/cinema#`, le *nom qualifié* (*qualified name* ou *Qname*) `cine:TheArtist` est une version courte de l'URI `http://www.irit.fr/ontologies/cinema#TheArtist` ; la représentation graphique du triplet précédent devient bien plus lisible, comme montré en figure 1.5(b). L'annexe A donne la liste des préfixes utilisés dans ce document et leurs valeurs respectives.

Littéral et nœud vide Un nœud de graphe RDF peut également contenir un *littéral*. Un littéral peut être typé ou non. Un littéral non typé est appelé *littéral simple* (*plain literal*) ; il est éventuellement accompagné d'une étiquette précisant, selon la norme RFC-3066 [2], la langue utilisée pour l'expression de sa valeur. Les types par défaut de RDF sont ceux définis par *XML schema*. Enfin, un nœud de graphe RDF peut être un *nœud vide* (*blank node*), c'est-à-dire un nœud qui fait référence à une ressource anonyme. Un nœud vide n'est associé à aucun URI et ne peut, par conséquent, pas être identifié en dehors du document RDF où il est

2. `http://www.isi.edu/in-notes/rfc3066.txt`



(a)



(b)

FIGURE 1.5 – Un même triplet RDF représenté sous forme graphique, 1.5(a) avec des URI complets et 1.5(b) avec des noms qualifiés

utilisé. Il permet également de mettre en œuvre l'un des mécanismes recommandés par le W3C pour exprimer des relations n-aires, qui ne sont pas naturellement exprimables en RDF [Noy and Rector, 2006]. La figure 1.6 illustre un exemple d'utilisation de nœud vide et de littéral. Un nœud vide est employé pour représenter l'événement que constitue la naissance de l'acteur Jean Dujardin, et les valeurs de lieu et de date reliées à cette ressource anonyme permettent de spécifier les lieu et date de naissance de ce même acteur. La valeur de la propriété `bio:date` est d'ailleurs un littéral, de valeur *"1972-06-19"* et de type `xsd:date`. La valeur de la propriété `foaf:name` de la ressource `cine:JeanDujardin` est un littéral simple de valeur *"Jean Dujardin"* et sans information de langue.

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

- le sac (`rdf:Bag`) est un groupe non ordonné qui peut contenir des doublons,
- la séquence (`rdf:Seq`) est un groupe ordonné qui peut contenir des doublons,
- l'alternative (`rdf:Alt`) est un groupe dont les éléments représentent chacun un choix possible d'une alternative.

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

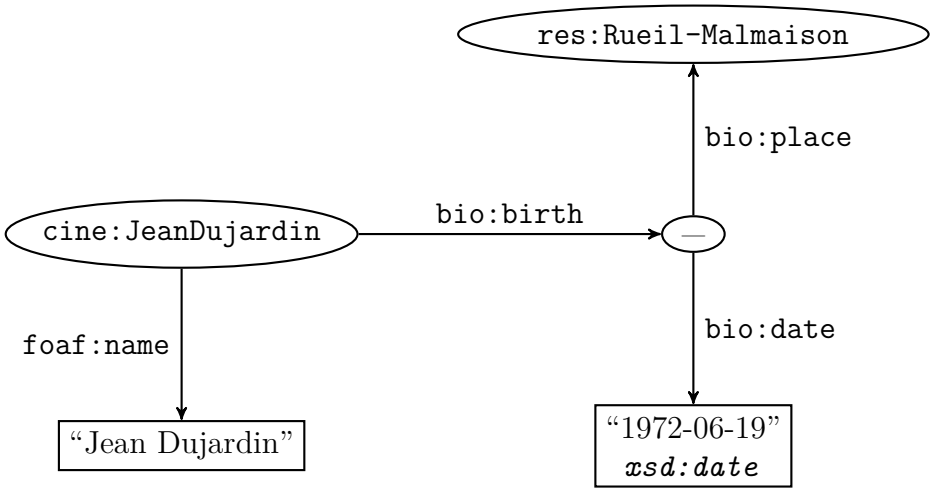


FIGURE 1.6 – Graphe RDF impliquant un nœud vide et des littéraux

par la propriété `rdf:first` à l'élément même contenu dans la collection et par la propriété `rdf:rest` au reste de la liste, c'est à dire une nouvelle ressource de type `rdf:List` ; ce schéma est répété jusqu'au dernier élément de la liste, qui est lui relié par la propriété `rdf:rest` à la ressource `rdf:nil` représentant la liste vide (définie dans le vocabulaire RDF et de type `rdf:List`). La figure 1.7 illustre une façon de représenter la liste des doctorants d'une équipe de recherche.

Syntaxes concrètes Nous avons jusque-là décrit les principales caractéristiques de RDF. Ce modèle de graphe constitue la syntaxe abstraite de RDF, qui est formellement décrite dans [Klyne and Carroll, 2004]. Toutefois, cette syntaxe abstraite ne permet ni d'exprimer ni d'échanger des données RDF. Pour cela, le modèle doit être doté d'une *syntaxe concrète*, ou *format de sérialisation*. Il en existe plusieurs, dont les plus populaires sont RDF/XML, Turtle, N-Triples et N3. RDF/XML est le seul format standardisé dans la recommandation RDF 1.0 ; il est fondé sur XML, ce qui le rend verbeux et lourd pour une utilisation manuelle. N-Triples est la notation la plus simple : un triplet par ligne et pas de mécanisme de préfixe. Ces caractéristiques la rendent idéale pour l'analyse rapide de grands documents par un programme, mais elle est également très volumineuse. Le format N3 (ou Notation 3), développé notamment par Tim Berners-Lee, se veut une alternative à RDF/XML facilement lisible par un humain ; il est doté d'une syntaxe assez complexe qui le rend peu intuitif. Enfin, le format Turtle est un sous-ensemble de N3, épuré de ses fonctionnalités complexes et peu utilisées ; il est simple, compact et intuitif, ce qui explique son succès grandissant.

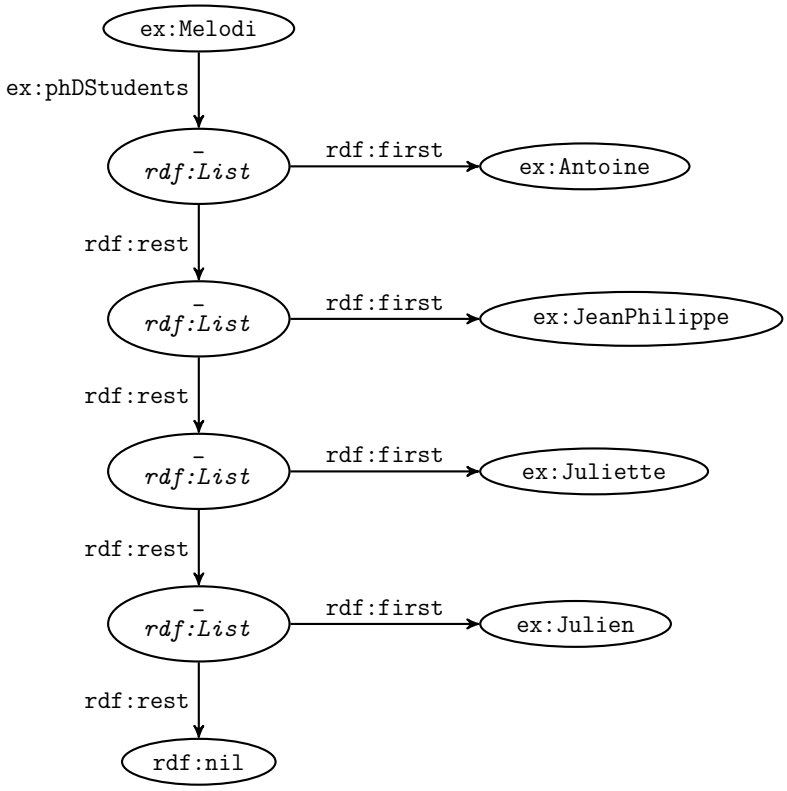


FIGURE 1.7 – Une liste exprimée en RDF

Turtle Dans la suite de ce travail, les données RDF seront le plus souvent représentées sous forme graphique. Nous choisissons cependant de présenter rapidement la syntaxe Turtle, celle-ci étant partie prenante du langage d'interrogation SPARQL présenté en 1.5. En Turtle, les URI complets s'écrivent entre chevrons (`<` et `>`) ; par exemple `<http://www.irit.fr/ontologies/cinema#JeanDujardin>`. Le triplet RDF de la figure 1.5 est exprimé en Turtle dans le listing 1.

Listing 1

```
<http://www.irit.fr/ontologies/cinema#TheArtist>
    <http://www.irit.fr/ontologies/cinema#hasForActor>
    <http://www.irit.fr/ontologies/cinema#JeanDujardin>.
```

Les noms qualifiés se notent sous la forme <nom_de_préfixe>:<partie_locale>, comme par exemple `cine:JeanDujardin`. Un préfixe et sa valeur se déclarent de la façon suivante: `@prefix` <nom_de_préfixe>: `<`<espace_de_nommage>`>`.. Le triplet RDF de la figure 1.5 peut alors s'exprimer dans une forme plus compacte, comme montré dans le listing 2.

Listing 2

```
@prefix cine: <http://www.irit.fr/ontologies/cinema#>.
cine:TheArtist cine:hasForActor cine:JeanDujardin.
```

Les littéraux simples se notent entre guillemets doubles, par exemple `"Jean Dujardin"`, et les littéraux typés selon cette syntaxe : `"`<valeur>`"^^`<type>. Les nœuds vides s'écrivent de la façon suivante : `_:`<identifiant>, par exemple `_:1` ; dans un même document, un nœud vide avec un même identifiant représente une même ressource. Le listing 3 présente une possible sérialisation Turtle du graphe RDF de la figure 1.6.

Listing 3

```
# triplets exprimant la date et le lieu de naissance de Jean Dujardin
@prefix cine: <http://www.irit.fr/ontologies/cinema#>.
@prefix foaf: <http://purl.org/vocab/bio/0.1/>.
@prefix bio: <http://purl.org/vocab/bio/0.1/>.
@prefix res: <hthttp://dbpedia.org/resource/>.
cine:JeanDujardin   foaf:name   "Jean␣Dujardin".
cine:JeanDujardin   bio:birth   _:birthEvent.
_:birthEvent        bio:date    "1972-06-19"^^xsd:date.
_:birthEvent        bio:place   res:Rueil-Malmaison.
```

Turtle permet également de "factoriser" des triplets partageant un même sujet (respectivement un même sujet et un même prédicat) ; pour cela, le premier triplet est exprimé intégralement, un point-virgule (respectivement une virgule) sépare chaque triplet, et les triplets suivants sont exprimés en précisant uniquement la partie changeante. Le listing 4 exprime à nouveau le graphe de la figure 1.6 en utilisant cette abréviation.

Listing 4

```
# triplets exprimant la date et le lieu de naissance de Jean Dujardin
@prefix cine: <http://www.irit.fr/ontologies/cinema#>.
@prefix foaf: <http://purl.org/vocab/bio/0.1/>.
@prefix bio: <http://purl.org/vocab/bio/0.1/>.
@prefix res: <hthttp://dbpedia.org/resource/>.
cine:JeanDujardin  foaf:name  "Jean␣Dujardin";
                   bio:birth  _:birthEvent.
_:birthEvent  bio:date   "1972-06-19"^^xsd:date;
              bio:place  res:Rueil-Malmaison.
```

Enfin, il n'est pas nécessaire d'identifier un nœud vide ; celui-ci est alors écrit `[]` et ne peut plus être référencé autrepart dans le document. Il reste cependant possible de préciser entre les crochets tous les triplets dont le nœud vide concerné est sujet. Cette méthode est utilisée dans le listing 5.

Listing 5

```
# triplets exprimant la date et le lieu de naissance de Jean Dujardin
@prefix cine: <http://www.irit.fr/ontologies/cinema#>.
@prefix foaf: <http://purl.org/vocab/bio/0.1/>.
```

```
@prefix bio: <http://purl.org/vocab/bio/0.1/>.
@prefix res: <hthttp://dbpedia.org/resource/>.
cine:JeanDujardin foaf:name "Jean Dujardin";
                  bio:birth [bio:date "1972-06-19"^^xsd:date;
                             bio:place res:Rueil-Malmaison.]
```

1.4 Les ontologies pour le web sémantique

RDFS (*RDF Schema*) et *OWL* (*Web Ontology Language*) sont les formalismes proposés par le W3C permettant de définir des ontologies intégrées au cadre du web sémantique, aussi appelées schémas ou ontologies légères lorsqu'elles sont définies en RDFS et par conséquent faiblement expressives.

Nous présentons ici les principales notions de ces formalismes, en nous focalisant sur les axiomes que nous exploitons dans la suite de ce travail, qui sont également les axiomes les plus répandus dans les bases de connaissances actuelles. Une description complète de RDFS et OWL et de leurs sémantiques est donnée dans [Hitzler et al., 2011].

1.4.1 Classes et instances

RDF fournit déjà un mécanisme permettant de typer des ressources, c'est-à-dire de les marquer comme appartenant à un groupe de ressources partageant des propriétés communes. L'ensemble des ressources appartenant à un même groupe constitue ce que l'on appelle une *classe*, et chaque resource est une *instance* de cette classe. La propriété `rdf:type` permet d'exprimer l'appartenance d'une ressource à une classe. Par exemple, le triplet de la figure 1.8 exprime que la ressource `cine:TheArtist` est une instance de la classe `cine:FullLength` (long métrage).

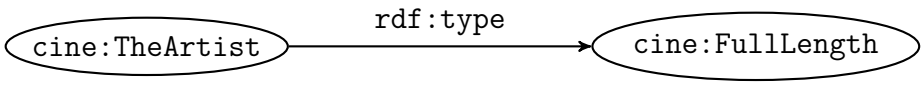


FIGURE 1.8 – Déclaration d'appartenance à une classe pour une ressource

RDFS permet de déclarer des noms de classes personnalisés. Pour cela, il suffit de typer l'URI correspondant à une classe que l'on veut introduire comme appartenant à la classe de toutes les classes : la *méta-classe* `rdfs:Class` prédéfinie dans le vocabulaire RDFS. Dans la figure 1.9, `cine:FullLength` est déclaré comme étant une classe.

Il est important de noter que le simple fait de déclarer qu'une ressource $r1$ est reliée à une ressource $r2$ par la propriété `rdf:type` implique que $r2$ est une classe. Ainsi, le triplet 1.8 implique le triplet 1.9. De même, le triplet 1.9 implique celui de la figure 1.10 :

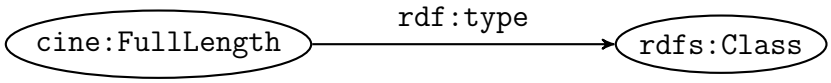


FIGURE 1.9 – Déclaration d'une classe

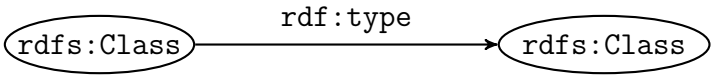


FIGURE 1.10 – La resource `rdfs:Class` est une classe

Dans la suite de ce travail, une grande partie des graphes RDF présentés utiliseront, par souci de simplification des figures, une convention permettant d'exprimer de façon plus compacte l'appartenance d'une ressource à une classe ; pour cela, la classe sera indiquée en italique, dans le même nœud que la ressource et en dessous de l'indentifiant de celle-ci. Par exemple, le graphe de la figure 1.11 est équivalent à celui de la figure 1.8. Il est important de garder en tête que, bien que n'apparaissant pas dans la figure, l'arc entre les deux nœuds (la ressource et son type) appartient réellement au graphe, ce qui n'est pas le cas des nœuds de littéraux typés qui, dans la syntaxe abstraite RDF, comportent en eux-mêmes l'information de leur type.

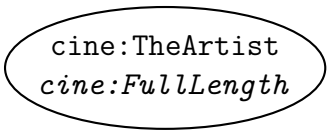


FIGURE 1.11 – Convention permettant d'exprimer de façon plus compacte l'appartenance d'une ressource à une classe

1.4.2 Hiérarchie de classes

RDFS permet également d'exprimer le fait qu'une classe $c1$ est incluse dans une classe $c2$. Pour cela, le vocabulaire RDFS définit la relation `rdfs:subClassOf`, dite *relation de subsomption* ou *relation taxonomique*. On dit alors que $c1$ est une *sous-classe* de $c2$ et que $c2$ est une *superclasse* de $c1$. Le triplet 1.12 exprime que `cine:FullLength` est une sous-classe de `cine:Movie`.

Exprimer le fait qu'une classe $c1$ est une sous-classe de $c2$ est équivalent à dire que chaque instance de $c1$ est également instance de $c2$. Les triplets 1.8 et 1.12 impliquent donc le triplet de la figure 1.13. Le raisonnement inverse (constater que toutes les instances de c_1 sont également instances de c_2 et en déduire que $c1$ est une

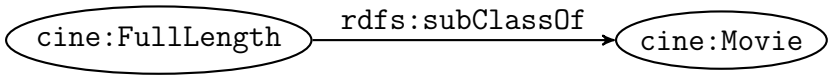


FIGURE 1.12 – Déclaration d'une relation de subsomption entre classes

sous-classe de $c2$) est cependant impossible à cause de l'hypothèse du monde ouvert présentée plus haut ; il est en effet impossible de connaître la liste exhaustive des instances d'une classe (à l'exception des classes énumérées exprimables en OWL que nous ne présenterons pas ici).

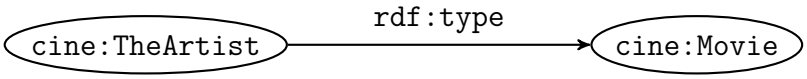


FIGURE 1.13 – Triplet induit des triplets 1.8 et 1.12

Ce mécanisme d'héritage permet d'établir une hiérarchie des concepts d'un domaine et de structurer ainsi les connaissances liées à ce domaine de façon intuitive. L'héritage multiple est autorisé, c'est-à-dire qu'une classe peut avoir plusieurs superclasses distinctes. La figure 1.14 montre un sous-ensemble d'une hiérarchie de classes qui pourrait être construite pour représenter le domaine du cinema. Ce graphe utilise la propriété `rdfs:subClassOf` pour exprimer que les classes `cine:ShortFilm` (court métrage) et `cine:FullLength` sont des sous-classes de `cine:Movie` (courts métrages et longs métrages sont des types de film), la classe `cine:Movie` a pour superclasse `cine:Work` (un film est un type d'œuvre), la classe `cine:AwardWinningDirector` (représentant l'ensemble des réalisateurs primés) a pour superclasses `cine:Director` et `cine:AwardWinningPerson` (un réalisateur primé est à la fois un réalisateur et une personne primée), ces classes étant elles-mêmes des sous-classe de `foaf:Person` (une personne primée est un type de personne, tout comme un réalisateur).

Signalons enfin que la propriété `rdfs:subClassOf` est transitive, c'est à dire que si une classe $c1$ est une sous-classe de $c2$ qui est elle-même une sous-classe de $c3$, alors $c1$ est une sous-classe de $c3$. Les conséquences de cette transitivité sont illustrées en pointillés dans la figure 1.14 : des déclarations précédentes on peut déduire que les classes `cine:ShortFilm` et `cine:FullLength` sont elles aussi des sous-classes de `cine:Work`, et que la classe `cine:AwardWinningDirector` est une sous-classe de `foaf:Person`.

1.4.3 Propriétés

RDFS permet également de définir des propriétés, c'est-à-dire des entités qui sont utilisées comme deuxième membre d'un triplet RDF. La classe `rdfs:property`

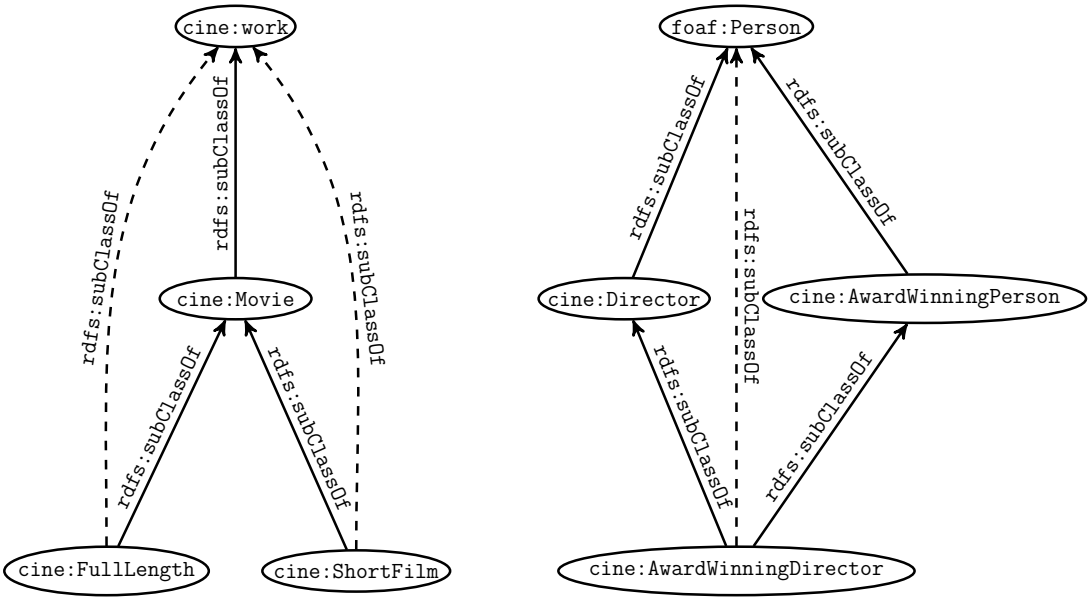


FIGURE 1.14 – Illustration de la transitivité de la propriété `rdfs:subClassOf`

est prévue à cet effet : une fois déclarée comme appartenant à cette classe, une ressource est alors considérée comme une propriété. Le triplet de la figure 1.15 exprime que la ressource `cine:hasForActor` est une propriété. On peut également noter que le simple usage d'une ressource comme deuxième membre d'un triplet suffit à la déclarer comme propriété. Ainsi, le triplet 1.5 implique le triplet 1.15.

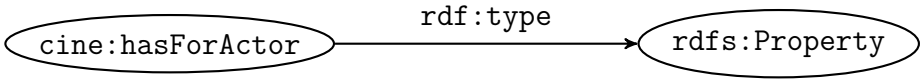


FIGURE 1.15 – Déclaration d'une propriété

De la même façon que `rdfs:subClassOf` permet de construire des taxonomies de classes, `rdfs:subPropertyOf` exprime la propriété de subsomption entre propriétés. Une propriété $p1$ subsume une propriété $p2$ si et seulement si, pour toutes ressources $r1$ et $r2$, le triplet $< r1, p2, r2 >$ implique le triplet $< r1, p1, r2 >$. La figure 1.16 exprime que la propriété `cine:hasForActor` est subsumée par `cine:hasContributor`.

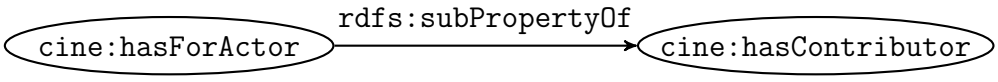


FIGURE 1.16 – Expression de la subsomption entre propriétés

De plus, RDFS offre les propriétés `rdfs:domain` et `rdfs:range` qui permettent de définir respectivement le *domaine* (*domain*) et le *co-domaine* (*range*) d'une propriété. Le domaine (resp. co-domaine) d'une propriété désigne la classe à laquelle appartient toute ressource utilisée comme sujet (resp. objet) dans un triplet dans lequel cette propriété a la place du prédicat. Le graphe de la figure 1.17(a) exprime que le domaine de la propriété `cine:hasForActor` est `cine:Movie` et que son co-domaine est `cine:Actor`. Avec ces connaissances supplémentaires, on peut inférer du triplet 1.5 les types des ressources `cine:TheArtist` et `cine:JeanDujardin`, comme illustré dans la figure 1.17(b).

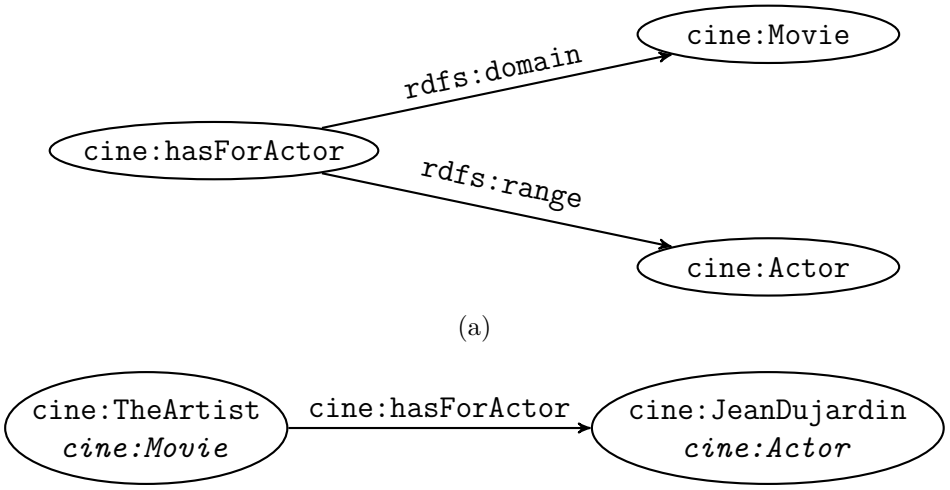


FIGURE 1.17 – Spécification des domaine et co-domaine d'une propriété

Les restrictions de domaine et co-domaine constituent un lien sémantique entre classes et propriétés ; elles sont (en RDFS) le seul moyen de décrire les interdépendances entre ces entités de natures différentes et d'orienter la façon de les utiliser.

1.4.4 Étiquettes

Les éléments présentés jusqu'ici permettent de définir de façon formelle des connaissances afin que celles-ci puissent être exploitées par un programme informatique. Ils n'offrent cependant aucun moyen de faire le lien avec les "connaissances humaines" de l'utilisateur censé bénéficier de ce programme. Or la modélisation, l'exploitation et l'inférence de nouvelles connaissances n'a de sens que si l'utilisateur final en profite. Les URI utilisés dans les exemples précédents sont assez explicites et peuvent permettre de deviner le concept ou l'entité du monde réel qui leur est attaché. Cependant, un URI n'a pas d'obligation d'intelligibilité, et peut donc être constitué de n'importe quelle chaîne de caractères.

Les propriétés d'annotations pallient ce problème en comblant le fossé entre la modélisation ontologique d'un domaine et le lexique de ce domaine. `rdfs:label`, la plus répandue de ces propriétés, permet d'associer à une ressource un terme du langage utilisé pour dénoter cette ressource. Ces *étiquettes* (*labels*) peuvent être définies sur des instances, mais aussi sur des classes et des propriétés. La figure 1.18 exprime que le film *Pulp Fiction* est appelé "Pulp Fiction" par les anglophones et par les français, et "Fiction Pulpeuse" par les Canadiens francophones.

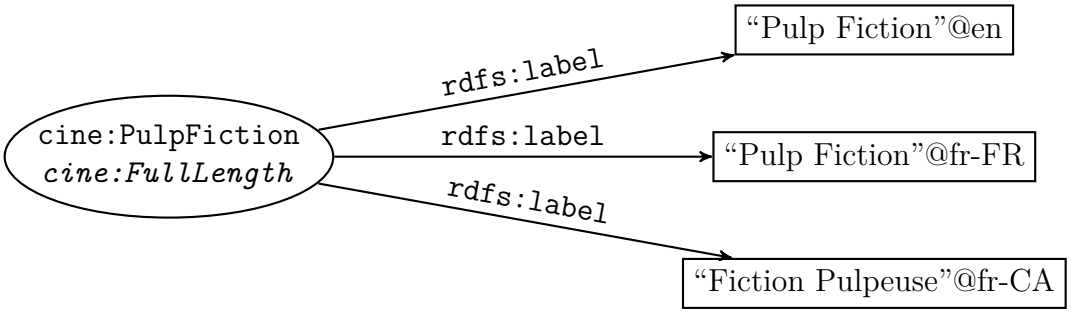


FIGURE 1.18 – Association d'étiquettes à une ressource

D'autres propriétés d'annotations sont populaires, comme `dc:title`, issu du vocabulaire du *Dublin Core*[3], qui permet de définir le titre d'une œuvre, ou `foaf:name`, provenant du vocabulaire *FOAF*[4], qui permet d'exprimer le nom d'une personne. Bien qu'ayant une signification plus précise que `rdfs:label`, ces propriétés ont approximativement le même rôle : elles permettent d'exprimer une représentation lexicale d'une ressource. C'est pourquoi dans l'approche présentée dans la suite de ce travail, nous ne faisons pas la distinction et appelons *étiquettes* les valeurs de toutes ces propriétés. Dans la pratique (c'est-à-dire dans l'implémentation présentée dans le chapitre 6), les propriétés d'annotations considérées dépendent du jeu de données ciblé.

1.4.5 Ontologies lourdes en OWL

OWL [McGuiness and van Harmelen, 2004] définit des opérateurs inspirés de ceux des LD permettant d'exprimer des classes. OWL se divise en plusieurs déclinaisons à l'expressivité décroissante : OWL Full, OWL DL et OWL Lite. Chaque sous-langage est restreint à un ensemble d'opérateurs donné. OWL Full est le plus expressif mais n'assure pas la décidabilité des algorithmes d'inférence. OWL DL est une restriction de OWL Full visant à obtenir l'expressivité la plus grande tout

3. http://dublincore.org/
4. http://xmlns.com/foaf/spec/

en garantissant la décidabilité et la complétude des algorithmes d'inférence. Enfin, OWL Lite est un sous-ensemble de OWL DL se limitant aux opérateurs les plus simples. En pratique, OWL Lite est très peu utilisé et n'est plus mentionné dans la spécification OWL 2 [Group, 2012]. Cette dernière ajoute de nouvelles fonctions [Golbreich and Wallace, 2012] et définit trois nouveaux sous-langages, appelés *profils* [Motik et al., 2012], tous moins expressifs que OWL DL et présentant chacun une palette d'opérateurs adaptée à des besoins précis : OWL 2 EL assure une complexité polynomiale des raisonnements, OWL QL facilite l'interrogation de données, et OWL 2 RL est un langage de règles.

Un exemple d'axiome exprimable en OWL (et non en RDFS) est la déclaration de propriétés inverses. Une propriété $p1$ est l'inverse d'une propriété $p2$ (et inversement, la propriété inverse étant symétrique) si et seulement si, pour toutes ressources $r1$ et $r2$, le triplet $< r1, p1, r2 >$ implique le triplet $< r2, p2, r1 >$. La figure 1.19 exprime que la propriété `cine:isActorIn` est l'inverse de la propriété `cine:hasForActor` ; les étiquettes de chacune des propriétés et la relation `owl:inverseOf` induite sont également représentées.

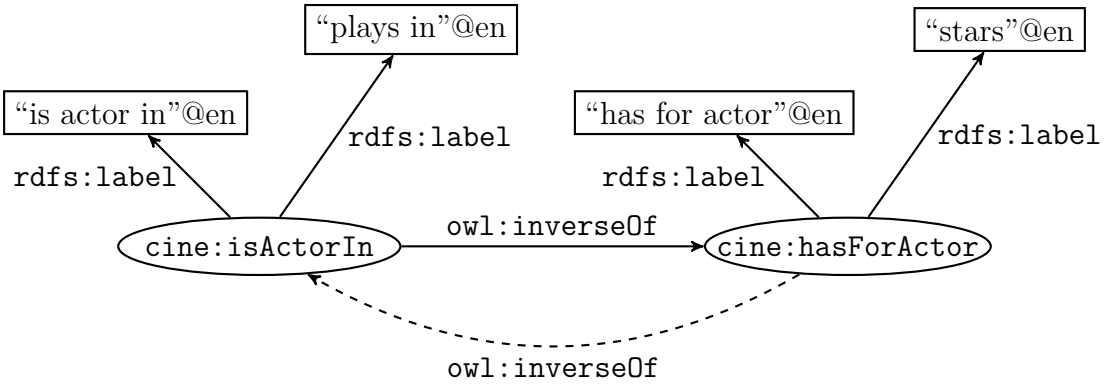


FIGURE 1.19 – Déclaration de deux propriétés inverses

Comme nous l'expliquons dans la section 1.6.1 décrivant le *web de données*, la version actuellement déployée du web sémantique, très peu de bases de connaissances font à ce jour usage des possibilités d'expressivité proposées par OWL. Cependant, les ontologies que nous avons développées pour les besoins applicatifs exploitent des axiomes de OWL DL, et l'approche que nous proposons prend en compte certains axiomes exprimables en OWL, comme la déclaration de propriétés inverses (à l'aide de la propriété symétrique `owl:inverseOf`).

1.4.6 Base de connaissances

La spécification d'un domaine au travers d'une ontologie et d'un ensemble de données relatives à cette ontologie constitue ce que l'on appelle une *base de connaissances dirigée par une ontologie*, souvent abrégée dans ce travail en base de connaissances. Une base de connaissances cible généralement un domaine donné. Elle est formée d'une ontologie qui représente les connaissances axiomatiques du domaine, et d'un ensemble d'assertions établissant les faits liés à ce domaine.

La figure 1.20 montre un sous-ensemble d'une base de connaissances sur le domaine du cinéma. La partie supérieure du schéma montre les connaissances ontologiques. Elle exprime notamment qu'un long métrage est un type de film, qu'un acteur est un type de personne et qu'un rôle d'acteur (lorsque le personnage incarné dans le film est lui-même un acteur) est un type de rôle. Ce fragment d'ontologie déclare également trois propriétés exprimables entre des instances de ces classes. Ces propriétés sont d'ailleurs exploitées dans la partie inférieure du schéma, relative aux assertions, qui exprime que l'acteur Jean Dujardin joue le rôle

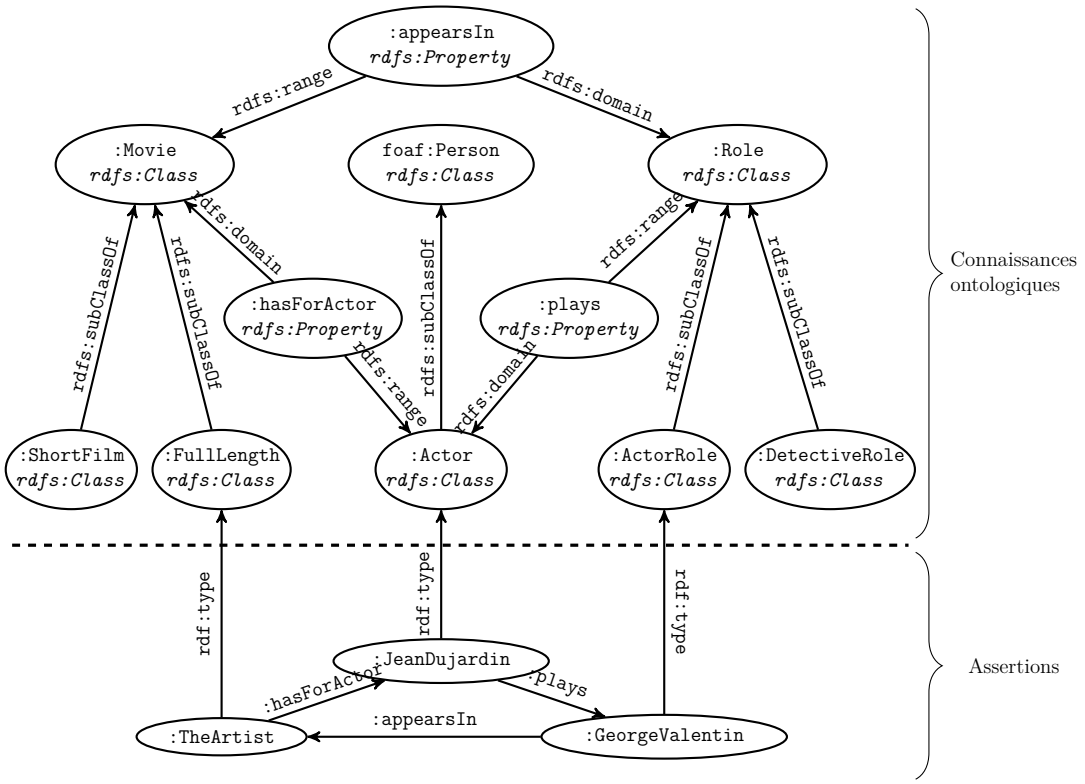


FIGURE 1.20 – Sous-ensemble d'une base de connaissances sur le domaine du cinéma

de George Valentin dans le film *The Artist*. Par souci de clarté, les étiquettes des ressources ne sont pas représentées sur le schéma. On remarque que les propriétés apparaissent comme des nœuds dans la partie ontologique alors qu'elles sont utilisés sur les arcs de la partie assertionnelle.

La base de connaissances que nous considérons dans notre travail consiste en une taxonomie de classes et de propriétés, des instances de classes, des littéraux et des instances exprimées entre ces entités sous la forme de triplets ; il est également possible d'assigner des étiquettes aux ressources et d'exprimer qu'une propriété est l'inverse d'une autre. La définition suivante décrit formellement cette base de connaissances et introduit les notations utilisées dans la suite de ce travail.

Définition 1 (Base de connaissances) *Une* base de connaissances KB *est un n-uplet* (V,P,E) *où :*

- V *est un ensemble fini de sommets.* V *est l'union disjointe* $\mathcal{C} \uplus \mathcal{I} \uplus \mathcal{L}$*, où :*
  - $\mathcal{C}$ *est l'ensemble de toutes les classes,*
  - $\mathcal{I}$ *est l'ensemble de toutes les instances,*
  - *et* $\mathcal{L}$ *est l'ensemble des valeurs de littéraux.*
- P *est un ensemble fini de propriétés, partitionné de la façon suivante :* $P = \mathcal{R}_o \uplus \mathcal{R}_d \uplus \{instanceOf, subclassOf, subpropertyOf, inversePropertyOf\}$*, où :*
  - $\mathcal{R}_o$ *est l'ensemble des propriétés d'objet,*
  - $\mathcal{R}_d$ *est l'ensemble des propriétés de type de données,*
  - $instanceOf$ *exprime l'appartenance d'une entité à une classe,*
  - $subclassOf$ *est la relation taxonomique entre classes,*
  - $subpropertyOf$ *est la relation taxonomique entre propriétés,*
  - *et* $inverseOf$ *permet d'exprimer qu'une propriété d'objet est l'inverse d'une autre.*
- E *est un ensemble fini d'arcs de la forme* $p(v_1,v_2)$ *remplissant l'une des conditions suivantes :*
  - $p \in \mathcal{R}_o$ *et* $v_1,v_2 \in \mathcal{I}$*,*
  - $p \in \mathcal{R}_d$*,* $v_1 \in \mathcal{I}$ *et* $v_2 \in \mathcal{L}$*,*
  - $p = instanceOf$*,* $v_1 \in \mathcal{I}$ *et* $v_2 \in \mathcal{C}$*,*
  - $p = subclassOf$ *et* $v_1,v_2 \in \mathcal{C}$*,*
  - $p = subpropertyOf$ *et* $v_1,v_2 \in P$*,*

Une valeur de littéral peut être spécifiée comme étant d'un type donné ; ℓ *est l'ensemble des types de littéraux ; une fonction type :* $\mathcal{L} \rightarrow \ell \cup \emptyset$ *associe son type à chaque littéral.*
$\mathcal{R}_d$ *contient, parmi d'autre propriétés, "label" qui capture l'expression lexicale d'une entité.*
$\mathcal{E} = \mathcal{C} \cup \mathcal{I} \cup \ell \cup \mathcal{L} \cup \mathcal{R}_o \cup \mathcal{R}_d$ *est l'ensemble de tous les éléments de la base de connaissances.*

On peut noter que les bases de connaissances du web de données, principalement définies par des schémas RDF et parfois par des ontologies OWL, peuvent facilement être réduites au type de base de connaissances décrit ci-dessus. Notre approche n'exploite pas directement les autres types d'axiomes (classes distinctes, chaînes de propriétés...), mais ceux-ci, s'ils existent, sont néanmoins pris en compte d'une certaine façon dans la mesure où les connaissances qu'ils apportent ont servi à inférer de nouvelles assertions dans le jeu de données interrogé.

1.5 L'interrogation en SPARQL

Bien que pouvant s'adapter à n'importe quel langage de requête fondé sur l'appariement de graphes, l'approche présentée dans la suite a été conçue pour s'intégrer au cadre du web sémantique, et SPARQL [Group, 2013] s'est donc naturellement imposé comme le langage cible. Nous décrivons ici les grands principes de ce langage.

Structure d'une requête Le langage SPARQL a été conçu pour interroger des bases de triplets RDF. Il est à RDF ce que SQL est aux bases de données relationnelles. Il utilise d'ailleurs une syntaxe proche de SQL, et notamment la structure générale `SELECT WHERE` d'une requête. Le contenu de la clause `WHERE` est un motif de graphe RDF dont chaque occurrence dans la base de triplets interrogée constitue une réponse à la requête.

Motif de graphe Dans ce motif de graphe, les triplets s'expriment dans une syntaxe très proche de Turtle, qui autorise en plus l'utilisation de variables ; les variables commencent par `?` ou `$` directement suivi de l'identifiant de la variable, comme par exemple `?maVariable`. Un motif de graphe M est formé de l'un des éléments suivants :

- un motif de triplet (s p o`.`) constitué de termes RDF et de variables,
- une jointure de deux motifs (M_1 M_2),
- une union de deux motifs (`{`M_1`} UNION {`M_2`}`),
- un motif optionnel (`OPTIONAL {`M_1`}`),
- un motif de filtre (`FILTER` F) où F est soit une expression booléenne fondée sur des opérateurs prédéfinis, soit une négation de motif (`NOT EXIST {`G_1`}`),
- un motif de graphe nommé (`GRAPH` g `{`G_1`}`) où g est l'URI, le nom qualifié ou une variable faisant référence à un graphe nommé,
- une sous-requête (`{ SELECT ... WHERE {`G_1`} }`).

Les préfixes se déclarent comme en Turtle, mais en amont de la clause `WHERE`, avant le mot-clé `SELECT`, et sans le caractère `@`.

Résultat Le traitement d'une requête (c'est-à-dire la recherche des résultats de cette requête) se fait par appariement de graphe (*graph pattern matching*). Pour répondre à une requête, un moteur SPARQL détermine l'ensemble des valeurs que peuvent prendre les variables du motif de graphe de sorte que le graphe obtenu après instanciation de ces variables soit un sous-graphe partiel du graphe cible ; chaque appariement variables-valeurs menant à un sous-graphe partiel du graphe cible est une réponse à la requête. Le contenu de la clause `SELECT` liste l'ensemble des variables dont la valeur doit être retournée pour chaque réponse.

La requête SPARQL présentée ci-dessous demande qui joue George Valentin dans *The Artist*. Elle définit le graphe requête illustré dans la figure 1.21. Ce graphe contenant une variable peut être apparié à celui de la base de connaissances de la figure 1.20. Lors de cet appariement, la variable `?actor` prend la valeur `:JeanDujardin` ; cette ressource est donc considérée comme une réponse à la requête.

Listing 6

```
PREFIX : <http://www.irit.fr/ontologies/cinema#>
SELECT ?actor
WHERE
{
  ?actor      :plays         :GeorgeValentin.
  :TheArtist  :hasForActor   ?actor.
}
```

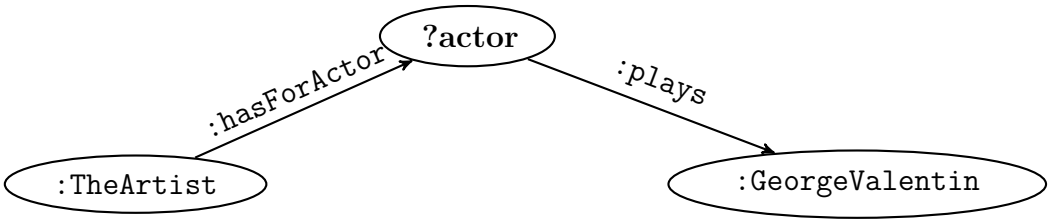


FIGURE 1.21 – Graphe requête demandant qui joue George Valentin dans *The Artist*

1.6 Déploiement en cours

Les solutions technologiques principales destinées à soutenir le web sémantique sont maintenant matures. L'évolution vers un web sémantique est donc techniquement possible. Cependant, quelques verrous scientifiques subsistent et freinent encore le déploiement à grande échelle du web sémantique. Le *web de données liées* est une version dégradée du web sémantique, qui contourne ces verrous.

1.6.1 Le web de données liées

Le *web de données liées* (ou simplement *web de données*) constitue une approche dégradée, simple et pragmatique des idées du web sémantique ; il consiste en la publication sur le web de jeux de données RDF liés les uns aux autres par l'utilisation de schémas ou de ressources communes. Il peut également être vu comme une étape intermédiaire vers la mise en place du web sémantique.

Le web de données est une initiative du W3C visant à encourager la publication de données structurées reliées entre-elles pour augmenter leur utilité. Il est naturellement construit sur les standards du web sémantique. La méthode de publication de données en accord avec les principes du web de données peut se résumer en quatre principes introduits par Tim Berners-Lee :

1. utiliser des URI pour identifier les choses ;
2. les URI doivent utiliser le protocole *http* afin de rendre les ressources accessibles via ce protocole ; on parle d'URI *déréférençables* ;
3. adapter le format des informations servies en cas de déréférencement d'un URI en fonction de la nature de l'agent à l'origine de la requête ; par exemple, si un humain a effectué cette requête à l'aide d'un navigateur web, les informations sont envoyées sous la forme d'un document HTML, ou s'il s'agit d'un agent logiciel, les informations sont envoyées en RDF ;
4. ajouter des liens à des éléments de jeux de données externes en utilisant leurs URI.

Un grand nombre d'entrepôts de données ont déjà été publiés selon ces principes. La figure 1.22 montre la version la plus récente du nuage du web de données, schéma emblématique du mouvement du web de données, mis à jour en 2011. On y voit les principaux jeux de données qui forment le web de données, ainsi que les relations qui les unissent.

La base de connaissances DBpedia a pris beaucoup d'importance avec le développement du web de données et est au cœur de la toile tissée par les liens entre chaque jeu de données [Auer et al., 2007; Bizer et al., 2009b]. Cette popularité s'explique par la largeur du spectre qu'elle couvre. En effet, DBpedia est le résultat d'efforts universitaires et communautaires d'extraction de données sur la base de Wikipedia [5]. D'autres jeux de données jouent un rôle dans la cohésion du web de données de par l'omniprésence des domaines qu'ils modélisent dans de multiples autres domaines ; par exemple, *FOAF* permet de décrire des personnes, et *GeoNames* d'exprimer des informations géospatiales. Parmi les autres jeux de données notables, on en dénombre quelques-uns provenant des communautés de recherche en sciences de la vie qui, comme nous l'avons dit plus haut, entretiennent une

5. http://www.wikipedia.org/

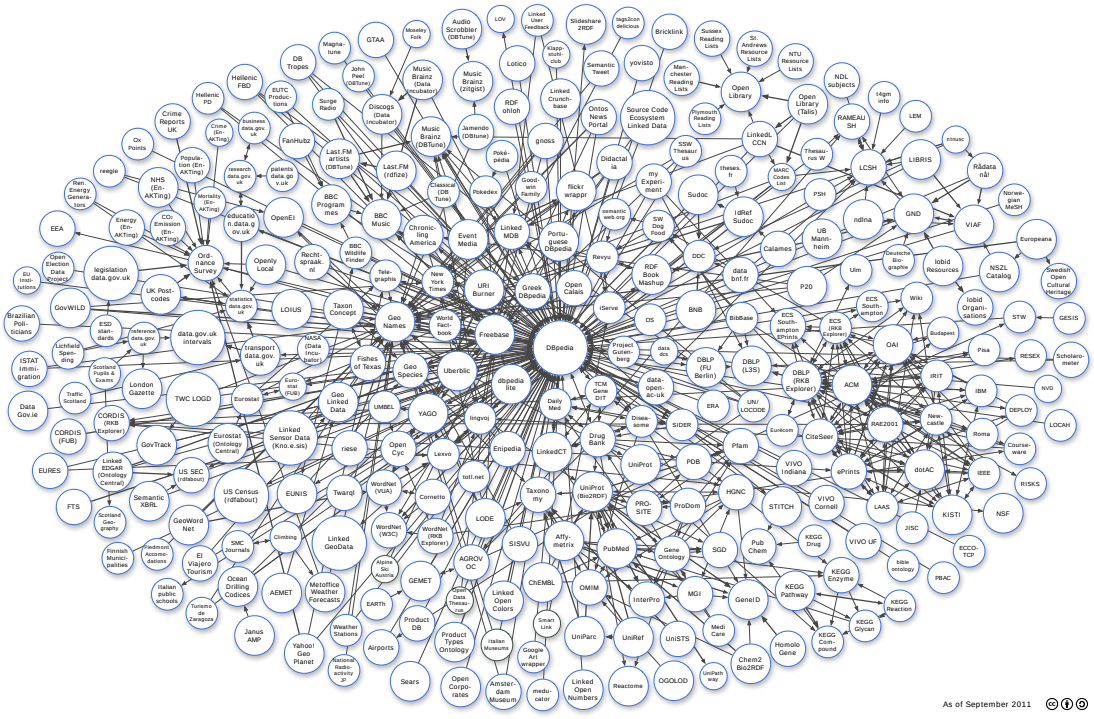


FIGURE 1.22 – Nuage du web de données liées en septembre 2011

longue tradition d'exploitation des méthodes de représentation des connaissances. D'autres, construits par la communnauté ou par des acteurs privés, ciblent des domaines d'intérêt du grand public, comme la musique pour *Musicbrainz*. Enfin, d'autres encore, plus récents, ont été mis en ligne par des collectivités publiques (états, régions, communes...) ; ils sont le fruit de courants politiques (principalement dans les pays anglo-saxons) visant à promouvoir la transparence dans le fonctionnement des organisations publiques et à encourager toute nouvelle idée d'applications exploitant ces données.

L'initiative du web de données a rencontré un franc succès. D'importants projets de recherche nationaux et internationaux, tels que *Linked Open Data 2*[6] *(LOD2)* [Bizer et al., 2009a], *PlanetData*[7], *Linked open data Around The Clock*[8] *(LATC)* ou encore *Datalift*[9], ainsi que des dynamiques politiques à différentes échelles, ont participé à cet engouement.

Ainsi, le web de données a su atteindre une masse critique de données liées les

6. http://lod2.eu/Welcome.html
7. http://planet-data.eu/
8. http://latc-project.eu/
9. http://datalift.org/

unes aux autres, rendues accessibles de façon uniforme et dans un format commun. Le spectre des application possibles est immense et commence à peine à être exploré. Cependant, la faible qualité de certaines données accessibles constitue un frein important au développement de telles applications. La plupart des jeux de données sont en effet fondés sur des ontologies très simples qui consistent le plus souvent en une taxonomie de classes ; certains de ces jeux de données sont de simples exports au format RDF de bases de données relationnelles préalablement existantes. Les propriétés de domaine, de co-domaine et de subsomption de relations sont rarement utilisées ; les autres types d'axiomes sont quasiment inexistants. Cette pauvreté en connaissances ontologiques limite les possibilités de raisonnements. De plus, les étiquettes sur les éléments de l'ontologie sont régulièrement omises [Ell et al., 2011], ce qui complique l'interfaçage avec les utilisateurs. Enfin, certaines données assertionnelles sont incohérentes ou ne respectent pas les schémas sur lesquels elles sont fondées.

Pour faire face à ce problème, certains travaux visent à maximiser la robustesse des applications exploitant ces données. Cet axe de recherche est exploré dans les nombreux ateliers orientés web de données, dans les conférences sur le web sémantique et dans des numéros spéciaux de revues [Schlobach and Knoblock, 2012]. Une approche complémentaire du problème consiste à chercher des méthodes pour identifier les erreurs, oublis et incohérences afin de "nettoyer" ces données [Poveda-Villalón et al., 2012; Ferré and Rudolph, 2012; Atencia et al., 2012].

1.6.2 Les verrous

Le web sémantique, tel que l'a imaginé Tim Berners-Lee [Berners-Lee et al., 2001], n'est pas encore une réalité. Certains problèmes n'ont en effet toujours pas été résolus et demeurent ouverts. Nous citons ici ceux qui nous semblent les plus importants.

La traduction manuelle des connaissances circulant sur le web en respectant les formalismes du web sémantique serait bien trop coûteuse. Des méthodes sont nécessaires pour automatiser cette traduction. La construction d'ontologies à partir de textes [Cimiano, 2006], l'annotation sémantique de textes à l'aide d'ontologies [Uren et al., 2006], et la génération de triplets RDF à partir de sources plus structurées telles que des bases de données relationelles ou des tableaux sont des domaines de recherche très actifs et les progrès récents sont encourageants. Les triplets générés par les systèmes actuels ne sont cependant pas toujours pertinents et n'exploitent pas encore toutes les possibilités offertes par les formalismes de représentation des connaissances.

Pour révéler leur véritable potentiel, les bases de connaissances ne doivent pas rester isolées sur le web. La mise en relation de ces bases de connaissances permet l'émergence de nouvelles connaissances. Les recherches autour de cette

problématique charnière, appelée *intégration de connaissances*, sont également en pleine effervescence. Elles s'articulent autour de deux sous-problèmes :

- l'*alignement d'ontologies et d'instances* vise à construire des liens sémantiques entre les différents éléments des bases de connaissances accessibles sur le web [Rahm and Bernstein, 2001; Kalfoglou and Schorlemmer, 2003; Shvaiko and Euzenat, 2005; Euzenat and Shvaiko, 2007] ;
- la mise en place d'agents *médiateurs* qui assurent la correspondance des schémas de chaque base de connaissances par le biais d'un *schéma médiateur*.

Il est également critique de pouvoir exploiter les très grandes quantités de données structurées de façon complexe et exprimées dans des formats hétérogènes. L'*évaluation de requêtes en présence d'ontologie* (*Ontology-Based Data Access*, *OBDA*) est un paradigme proposé pour répondre à ce besoin. L'objectif est de pouvoir trouver dans une couche de données (formée d'une ou plusieurs bases de données, éventuellement réparties et exploitant des formats potentiellement différents) les réponses à des requêtes exprimées grâce à un modèle conceptuel (une ontologie exprimée en logiques de decription, RDFS ou OWL). Pour cela, il est nécessaire de définir une association entre la couche de données et le modèle conceptuel.

Enfin, l'exploitation et l'inférence de connaissances ne sont d'aucune utilité si elles ne profitent pas à l'Homme. Au-delà du scénario classique de l'assistant personnel qui sélectionne une liste de restaurants à proximité de l'utilisateur, en adéquation avec ses goûts et son budget, un réel besoin d'interfaçage avec l'humain reste à combler. L'utilisateur final doit pouvoir accéder aux connaissances, et ce sans être confronté aux formalismes employés par les bases de connaissances avec lesquelles il interagit. Le travail présenté dans la suite est d'ailleurs essentiellement motivé par ce besoin.

1.7 Conclusion

Dans ce chapitre, nous avons présenté le web sémantique, qui constitue le cadre technologique de notre travail. Le web sémantique s'inscrit dans l'architecture du web classique et repose sur une pile de technologies qui ont désormais atteint un stade de maturité suffisant pour permettre son déploiement à grande échelle. Certains verrous empêchent cependant le web sémantique d'atteindre son plein potentiel. L'accès de l'utilisateur final aux connaissances du web sémantique figure parmi ces problèmes non résolus. Le travail décrit dans la suite est notre contribution à la résolution de ce problème. Dans le chapitre suivant, nous réalisons un état de l'art des approches partageant nos objectifs que nous avons recensées à ce jour.

Chapitre 2

Interfaces en langue naturelle

Les interfaces en langue naturelle (*natural language interfaces*), ou systèmes de question-réponse (*question answering systems*), sont définies dans [Hirschman and Gaizauskas, 2001]. Elles ont pour but de donner une réponse pertinente et concise à des questions énoncées en langue naturelle par un utilisateur. Elles se veulent un premier pas vers la concrétisation d'un des plus vieux et plus populaires fantasmes de l'intelligence artificielle, à savoir la possibilité pour un humain de communiquer avec une machine comme s'il s'agissait d'un autre humain. D'un point de vue plus réaliste, les interfaces en langue naturelle offrent une alternative très intéressante aux requêtes par mots-clés à l'expressivité limitée et aux langages de requête formels à la complexité inadaptée pour les utilisateurs finals. Elles font l'objet de recherches actives depuis la fin des années soixante. Les produits de ces recherches sont matérialisés par des logiciels, appelés *interfaces en langue naturelle* ou *systèmes de question-réponse*, dans lesquels l'utilisateur peut directement exprimer son besoin en information en langue naturelle et obtenir une réponse adaptée. Ce chapitre présente les approches proposées à ce jour.

La section 2.1 constitue un aperçu des notions, méthodes et outils utilisés en traitement automatique des langues, ce domaine étant indispensable au développement d'interfaces en langue naturelle. Les notions de question et d'interface en langue naturelle sont définies dans les sections 2.2 et 2.3 respectivement, où des typologies sont proposées. Les sections 2.4, 2.5 et 2.6 présentent chacune une des trois grandes familles d'interfaces en langue naturelle : les interfaces entre la langue naturelle et les bases de données, les interfaces entre la langue naturelle et des textes, et les interfaces entre la langue naturelle et les bases de connaissances. La section 2.7 se focalise sur certaines spécificités de systèmes de question-réponse que nous avons jugées intéressantes. Enfin, la section 2.8 conclut le chapitre.

2.1 Le Traitement automatique des langues pour les interfaces en langue naturelle

Le *Traitement Automatique des Langues* (TAL) ou *Traitement Automatique du Langage Naturel* (TALN) vise à appliquer des méthodes informatiques pour exploiter des ressources exprimées en langue naturelle. C'est une discipline à l'interface entre la linguistique, l'informatique et l'intelligence artificielle.

Le langage naturel s'oppose au langage artificiel (comme un langage de programmation ou un langage formel de représentation de l'information). Un langage artificiel est constitué d'un vocabulaire intégralement déterminé et fermé et obéit à des règles syntaxiques et sémantiques strictes. À l'inverse, le langage naturel utilise un vocabulaire non exhaustivement défini et possède une sémantique ambiguë et des règles syntaxiques moins formelles, plus nombreuses, parfois contradictoires, et qui ne sont pas toujours respectées à l'usage.

L'histoire du TAL remonte aux années cinquante, avec la définition de ce que l'on appelle aujourd'hui le *test de Turing* [Turing, 1950]. La complexité des problèmes liés au TAL a toujours été sous-estimée. En 1954, IBM organisait une démonstration d'un système de traduction automatique du russe vers l'anglais ; le système présenté fonctionnait avec six règles grammaticales et un vocabulaire de 250 termes. Plus de soixante phrases furent traduites avec succès. Ces résultats impressionnants soulevèrent une vague d'optimisme et l'on pensait alors que le problème de traduction automatique serait résolu en trois à cinq ans. Dix années de recherches infructueuses eurent raison de cet optimisme.

Deux principales raisons rendent le TAL difficile. La première est l'ambiguïté du langage naturel. Cette ambiguïté se manifeste à plusieurs niveaux. Au niveau phonétique, les homophones sont des mots distincts qui se prononcent de la même façon (par exemple "laid" et "lait"). Au niveau lexical, on trouve des ambiguïtés lexicales polysémiques ("Antoine sent la rose" peut être paraphrasé en "Antoine hume la rose" ou "Antoine a l'odeur d'une rose") et des ambiguïtés lexicales homonymiques (dans la phrase "Cet ours a mangé un avocat," la forme avocat correspond à deux termes distincts, l'un désigant un fruit, l'autre une personne). Au niveau grammatical, certaines phrases admettent plusieurs structures syntaxiques (la phrase "Julien a vu un homme avec un télescope" peut être paraphrasée en "Julien a vu un homme grace à un télescope" ou "Julien a vu un homme qui avait un télescope"). La figure 2.1 donne un autre exemple d'ambiguïté syntaxique tiré d'une réplique célèbre [Institute, 2005] du comédien américain Groucho Marx, "One morning I shot an elephant in my pajamas. How he got in my pajamas, I don't know."

La seconde raison est l'implicite qui est présent dans presque tout énoncé naturel. La compréhension d'un énoncé en langue naturelle est effectuée chez l'humain

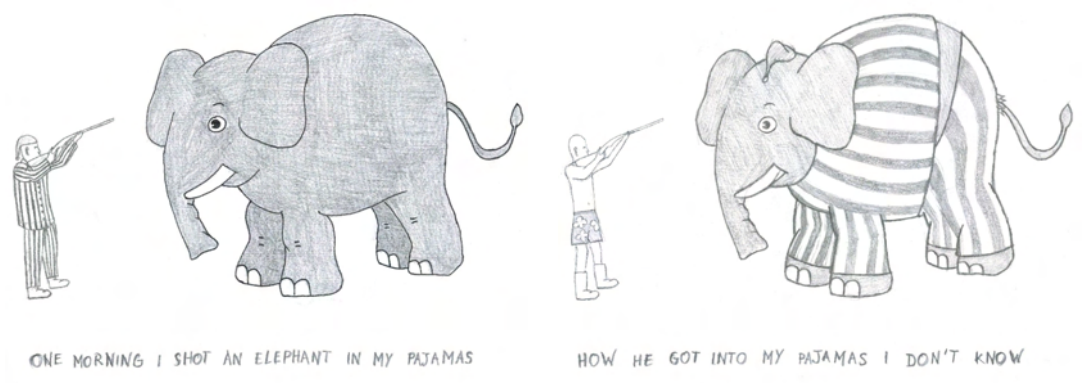


FIGURE 2.1 – Les sous-figures (a) et (b) illustrent deux interprétations possibles de la phrase “One morning I shot an elephant in my pajamas.”

par des processus cognitifs impliquant des connaissances extérieures à celles comprises dans l'énoncé. Par exemple, de l'énoncé “L'appartement est en face de l'autoroute,” un être humain déduit automatiquement que l'appartement en question se situe dans un environnement bruyant.

2.1.1 Les différents niveaux d'analyse

On distingue différents niveaux d'analyse en TAL :

- la *phonologie* analyse l'organisation des sons pour former un énoncé,
- la *morphologie* est l'étude de la forme des mots,
- la *syntaxe* s'intéresse à l'organisation des mots en structures,
- la *sémantique* étudie le sens d'un énoncé et le représente dans un langage formel,
- la *pragmatique* prend en compte le contexte pour analyser le sens d'un énoncé.

Ces niveaux d'analyse sont abordés dans la suite au travers de la présentation de différents problèmes de TAL impliqués par les interfaces en langue naturelle et des outils visant à résoudre ces problèmes.

Bien que nous pensions que les interfaces en langue naturelle ne pourront gagner les faveurs du grand public que par l'intermédiaire d'une interface vocale, la possibilité pour l'utilisateur de s'exprimer oralement n'est pour l'instant pas intégrée à notre travail ni aux autres travaux de la littérature. Les outils liés à la phonologie ne sont par conséquent pas présentés.

2.1.2 Analyse morphologique

La morphologie consiste en l'étude de la forme des mots à partir d'unité plus petites appelées *morphèmes*. Le morphème est la forme minimale du langage doué de sens. Par exemple, le mot "chiens" est composé de deux morphèmes: le stemme ou la racine ("chien") et un suffixe ("s") indiquant le pluriel. On distingue deux types de morphèmes : les *morphèmes lexicaux* ou *lexèmes* qui correspondent aux entrées du dictionnaire, et les *morphèmes grammaticaux* ou *affixes* (*préfixes*, *suffixes*, *infixes*) qui se combinent aux lexèmes.

Les phénomènes morphologiques se manifestent de deux manières : la *flexion* est un phénomène purement grammatical n'affectant pas la catégorie syntaxique, alors que la *dérivation* permet de créer de nouvelles unités lexicales. La figure 2.2 illustre une analyse morphologique possible du mot "antialcooliques".

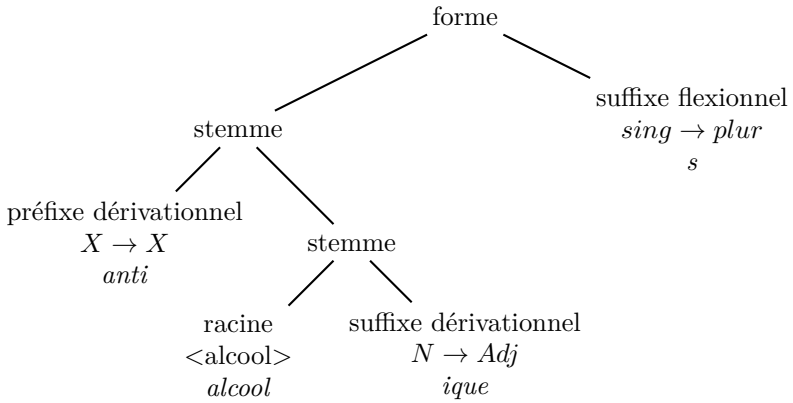


FIGURE 2.2 – Une analyse morphologique possible du mot "antialcooliques"

Deux tâches sont récurrentes dans le domaine de la morphologie :
- la segmentation du texte en unités élémentaires : *tokenization* ou *normalisation*
- la détermination des catégories grammaticales de ces unités: *POS Tagging* (*Part Of Speech Tagging*).

2.1.3 Normalisation de texte

La normalisation (ou tokenization) consiste à identifier les mots d'un énoncé et à réduire les variations dues aux conventions orthographiques ou à la morphologie. Ce traitement permet de neutraliser de nombreuses variations du langage situées au niveau du mot, parmi lesquelles : les problèmes de codage (accents, caractères spéciaux), la ponctuation (abrévation, apostrophes), l'écriture en majuscules ou

en minuscules, les pluriels, les blancs, la ponctuation, les retours à la ligne, les troncatures...

C'est une tâche bien définie pour laquelle des outils – appelés *tokenizers* – performants existent, comme par exemple le *Stanford tokenizer*[1], ou les tokenizers de la plate-forme *Gate*[2] ou de la bibliothèque *OpenNLP*[3]. Très peu d'applications considèrent la tokenization comme un objectif en soi. Il s'agit plus d'un traitement initial, indispensable aux traitements suivants.

2.1.4 Détermination des catégories grammaticales

La connaissance des catégories grammaticales des mots (ou *POS tagging* pour *Part-Of-Speech tagging*) est cruciale pour déterminer les relations entre ces mots. Malheureusement, dans la plupart des langues, un mot peut appartenir à plusieurs catégories grammaticales. Par exemple, "souris" est une forme verbale de sourire et également un nom féminin, "la" est un déterminant, un pronom personnel et également un nom masculin. Le but de ce traitement est donc de désambiguïser ces catégories grammaticales.

La *lemmatisation* est une sous-tâche du *POS tagging* qui consiste à associer à chaque mot normalisé sa forme de base, ou *lemme*, correspondant à une entrée du dictionnaire. Le lemme d'un verbe sera sa forme infinitive, alors que pour la plupart des autres mots, le lemme sera représenté par le stemme.

Les outils mettant en œuvre ce traitement, comme *Tree Tagger*, *Brill*, *Flemm* ou *MElt* [Denis and Sagot, 2009] (adapté au français), sont appelés *POS taggers*. Le listing 7 montre le résultat en sortie de *TreeTagger* pour le traitement de la phrase "Who produced the album Invincible ?". Chaque ligne contient un mot de la phrase analysée ; la première colonne reprend le mot tel qu'il apparaît dans la phrase, la deuxième donne la catégorie grammaticale identifiée (nous précisons entre parenthèses la signification complète de chaque identifiant), et la dernière colonne précise le lemme de chaque mot.

Listing 7

```
Who         WP (Wh-pronoun)              who
produced    VBD (Verb, past tense)       produce
the         DT (Determiner)              the
album       NN (Noun, singular or mas)   album
Invincible  NP (Proper noun, singular)   <unknown>
?           SENT (End of sentence)
```

1. http://nlp.stanford.edu/software/tokenizer.shtml
2. http://gate.ac.uk/
3. http://opennlp.apache.org/

2.1.5 Analyse syntaxique

Les mots se regroupent pour former des *syntagmes*. Un syntagme est un mot ou un groupe de mots consécutifs auquel on peut associer une catégorie syntaxique (sujet, verbe, complément...). Les syntagmes se combinent entre-eux pour former des *propositions* ou des *clauses*. Une proposition est constituée d'un sujet, d'un verbe et d'éventuels compléments. Une phrase est composée d'au moins une proposition.

On appelle *analyse en constituants* le traitement qui consiste à décomposer une phrase en syntagmes. Le résultat de ce traitement est un arbre syntaxique représentant l'organisation hiérarchique des syntagmes. Ce traitement est initialement fondé sur la notion de grammaire. Une grammaire définit un ensemble de règles qui permettent de déterminer si un énoncé est correct pour un langage donné. La figure 2.3 illustre une analyse en constituants de la phrase "Who produced the album Invincible ?"

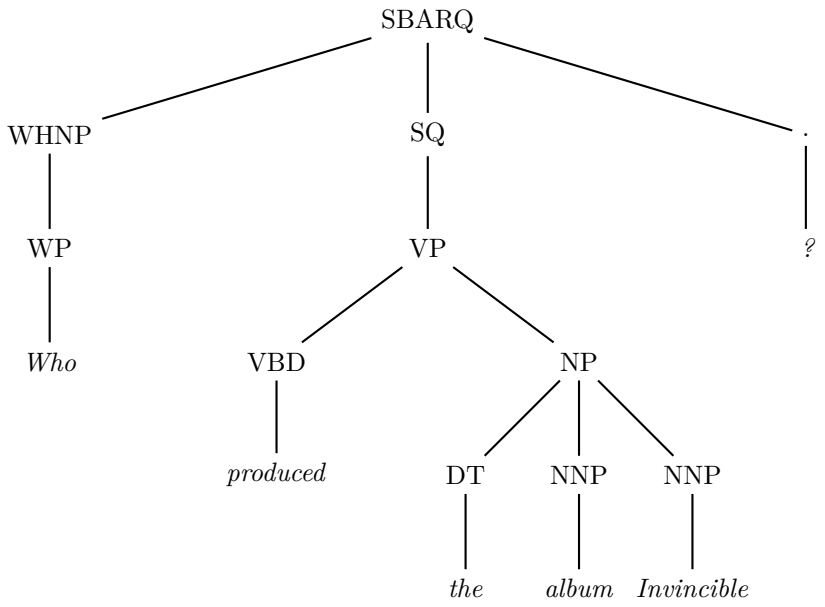


FIGURE 2.3 – Une analyse en constituants de la phrase "Who produced the album Invincible ?"

Une phrase peut également être analysée selon les fonctions grammaticales des éléments qui la constituent. La fonction grammaticale, ou dépendance syntaxique, exprime le rôle syntaxique qu'occupe un constituant par rapport à un autre. *sujet*, *objet direct*, *objet indirect* sont des exemples de dépendances syntaxiques. La figure 2.4 illustre quelques-unes des dépendances entre les constituants de la phrase "Jean Dujardin interprète le rôle de George Valentin."

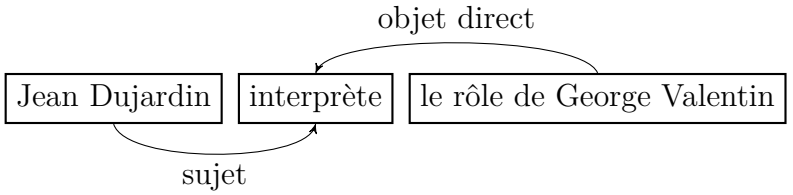


FIGURE 2.4 – Exemple de dépendances entre constituants d'une phrase

On appelle *analyse en dépendances* le traitement qui consiste à identifier les dépendances syntaxiques entre les mots d'une phrase. Le résultat de ce traitement est un arbre syntaxique représentant ces dépendances. La figure 2.5 illustre le résultat d'une analyse en dépendances réalisée sur la phrase "Jean_Dujardin interprète le rôle de George_Valentin" (le caractère souligné est utilisé pour relier deux mots appartenant à une même entité nommée ; les entités nommées sont présentées plus bas en 2.1.6).

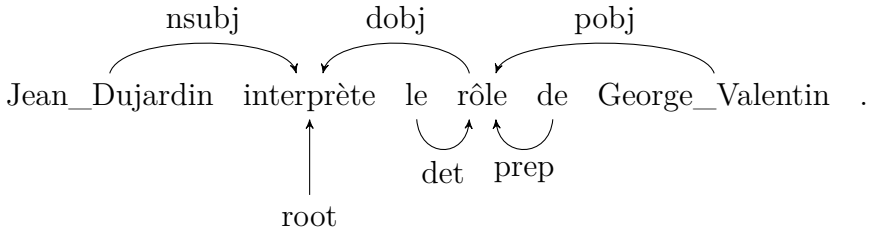


FIGURE 2.5 – Une analyse en dépendances de la phrase "Jean_Dujardin interprète le rôle de George_Valentin"

Ces deux formes d'analyse syntaxique représentent des tâches cruciales pour de nombreuses applications. Les recherches sont toujours très actives autour de ces problèmes et les outils récents, fonctionnant grâce à des techniques d'apprentissage, réalisent de bonnes performances à partir du moment où ils sont appliqués à des phrases du même type que celles du corpus avec lequel ils ont été entraînés. La plupart des systèmes disponibles ont été conçus pour analyser la langue anglaise : *Stanford parser*[4], *Charniak parser*[5], *MSTParser*[6], *Supple parser*[7] [Gaizauskas et al., 2005] ou encore *MaltParser*[8] [Nivre et al., 2007]. Notons également

4. http://nlp.stanford.edu/software/lex-parser.shtml
5. http://cs.brown.edu/~ec/
6. http://sourceforge.net/projects/mstparser/
7. http://gate.ac.uk/userguide/sec:parsers:supple
8. http://www.maltparser.org/

qu'il existe un parseur commercial pour le français, *Syntex*[9] [Bourigault and Fabre, 2000], et que l'équipe de *MaltParser* met à disposition sur son site un modèle entraîné avec un corpus en français.

2.1.6 Analyse sémantique

L'analyse sémantique du langage naturel veut construire une représentation de l'information contenue dans des expressions en langage naturel. Elle peut s'effectuer au niveau du mot, de la phrase ou du discours. La sémantique du mot, ou sémantique lexicale, a deux objectifs majeurs : l'étude du sens des mots et leurs interactions avec le contexte, et l'étude des relations paradigmatiques (hyponymie, méronymie, antonymie) entre ces mots. La sémantique de la phrase construit une représentation formelle de la phrase. La sémantique du discours veut analyser les liens entre les phrases permettant de former un discours cohérent ; nous identifions deux courants majeurs dans ce domaine : la *Théorie de Représentation du Discours* (*Discourse representation theory*, *DRT*) [Kamp, 1981; Kamp and Reyle, 1993] et la *Théorie de Représentation Segmentée du Discours* (*Segmented Discourse Representation Theory*, *SDRT*) [Asher, 1993; Lascarides, 2003].

Comme dit plus haut, la sémantique est l'étude du sens d'un énoncé. Plusieurs définitions du sens ont été proposées : le sens peut être défini en termes de relations entre les expressions d'après Quine, d'usages dans la langue d'après Wittgenstein, de preuve d'après Leibniz et Hilbert (théorie des preuves), ou encore de valeur de vérité d'après Frege et Montague (théorie des modèles).

Dans cette dernière vision, le sens d'une proposition est l'ensemble des mondes dans lesquels cette proposition est vraie. Interpréter un énoncé revient donc à définir l'ensemble des modèles exprimables dans lesquels cet énoncé est vrai. Cet ensemble peut être défini sous forme logique. C'est le but de la *sémantique compositionnelle* : ce domaine de recherche considère que le sens de la phrase est une fonction du sens de ses composants et établit des méthodes compositionnelles exploitant la syntaxe d'une phrase pour en construire une interprétation logique.

Un formalisme très utilisé dans ce domaine est le *lambda calcul*. La sémantique d'une expression est considérée comme une lambda-abstraction en attente de ses arguments. Par exemple, la sémantique du verbe "dort" est $\lambda x.dort(x)$, celle du nom commun "homme" est $\lambda x.homme(x)$, et celle du déterminant "un" est $\lambda P \lambda R.(\exists x P(x) \wedge R(x))$. Par composition, la sémantique de l'énoncé "un homme dort" est $\exists x(homme(x) \wedge dort(x))$ après béta-réduction.

Reconnaissance d'entités nommées Une entité nommée est un objet textuel (c'est-à-dire un mot ou un groupe de mots) faisant référence à une entité apparte-

9. http://www-etud.iro.umontreal.ca/~demorali/syntex/

nant à une classe considérée, telle qu'une personne, une ville, une entreprise ou un nom d'album. Le terme *gazetteer* désigne à la fois l'outil qui accomplit la tâche de reconnaissance d'entités nommées et le lexique sur lequel cet outil s'appuie pour identifier les entités nommées. L'identification des entités nommées est importante car elle permet de simplifier les étapes postérieures de traitement, en déclarant que ces entités doivent être considérées comme un tout.

Dans le cadre de travaux portant sur le web sémantique, nous nous intéressons particulièrement aux gazetteers capables de construire un lexique automatiquement à partir d'une base de connaissances et d'annoter les entités nommées avec les URI de la ressource identifiée et de sa/ses classe(s). C'est par exemple le cas du *Large KB Gazetteer*[10], exploitable depuis la plate-forme *Gate*[11]. La figure 2.6 illustre le résultat de ce traitement sur la phrase "Who produced the album In Utero ?".

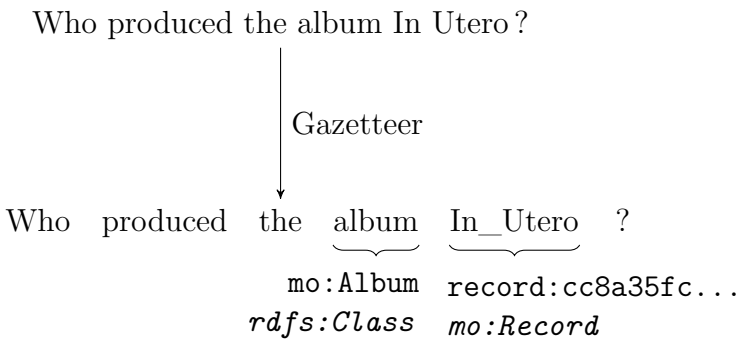


FIGURE 2.6 – Reconnaissance d'entités nommées dans la phrase "Who produced the album In Utero ?"

2.1.7 La pragmatique

La pragmatique prend en compte le contexte dans l'étude de la phrase. Elle implique l'utilisation de connaissances extra-linguistiques sur le contexte du discours (phrases précédentes, situation du locuteur...) et sur le monde en général.

Dans le cadre des interfaces en langue naturelle, il apparaît pertinent de prendre en compte certains éléments du contexte d'énonciation des requêtes :

– les requêtes précédemment énoncées par un utilisateur peuvent permettre de prédire ses centres d'intérêts et ses intentions, et ainsi d'affiner l'interprétation de la requête en cours. Bien que ces informations ne soient à notre

10. http://gate.ac.uk/userguide/sec:gazetteers:lkb-gazetteer
11. http://gate.ac.uk/

connaissance pas encore prises en compte dans des interfaces en langue naturelle, leur exploitation est courante dans d'autres systèmes d'informations. Par exemple, le système *QueRIE* [Akbarnejad et al., 2010] compare la session de l'utilisateur courant à celles des utilisateurs précédents pour suggérer de nouvelles requêtes SQL potentiellement pertinentes.
– le contexte spatiotemporel est indispensable à l'interprétation d'expressions telles que "aujourd'hui", "ce soir", "près d'ici". La prise en compte de ces informations a été expérimentée dans le domaine touristique par le projet européen QALL-ME [Ferrandez et al., 2011].

2.2 Typologie des questions

Les auteurs de [Hirschman and Gaizauskas, 2001] s'attachent à définir la notion de *question* qui est évidemment au cœur des problématiques du domaine des interfaces en langue naturelle. La classification qu'ils proposent est adoptée de façon quasi-unanime dans la littérature et nous l'avons reprise et adaptée à nos travaux.

Le *focus* est l'objet central de la question, c'est-à-dire la partie de la question qui intéresse particulièrement le locuteur et pour laquelle il attend des résultats. Par exemple, dans la question "Quels sont les acteurs qui jouent dans The Artist ?" le focus est "acteurs", car la réponse attendue sera une liste d'acteurs. Le focus est caractérisé par son type (acteur, personne, lieu, date...) et par une liste de contraintes que les éléments de ce type doivent satisfaire pour faire partie de la réponse. Dans l'exemple précédent, les éléments de la réponse doivent être de type acteur et doivent avoir joué dans le film *The Artist*.

Les questions se distinguent par le type de leur réponse :
– les questions impliquant une réponse factuelle,
– les questions impliquant l'expression d'une opinion,
– les questions impliquant un résumé.

La grande majorité des travaux sur les systèmes de question-réponse s'est pour l'instant focalisée sur la première catégorie, les question factuelles, que nous divisons en trois sous-catégories :
– les *questions dichotomiques* (*yes/no questions*) n'ont que deux réponses possible, Oui ou Non ; par exemple, "Y a-t-il un pilote dans l'avion ?".
– les *questions liste* attendent comme résultat une liste d'éléments ; par exemple, "Quels sont les acteurs qui jouent dans The Artist ?". Cette sous-catégorie englobe également les questions qui n'attendent qu'un seul résultat, comme "Quelle est la date de naissance de Jean Dujardin ?"
– les requêtes de dénombrement demandent le nombre d'éléments respectant certaines conditions ; par exemple "Combien de films ont été réalisés par Michel Hazanavicius ?"

2.3 Typologie des interfaces en langue naturelle

Dans la littérature, plusieurs dimensions ont été proposées pour permettre la classification des interfaces en langue naturelle. La première de ces dimensions est le type de question accepté en entrée par le système considéré. Par exemple dans [Moldovan et al., 2003], les interfaces en langue naturelle sont classées en cinq types de question supportés, selon une complexité croissante :

- les systèmes supportant des questions factuelles qui extraient les réponses telles quelles dans leurs sources de connaissances,
- ceux possédant des mécanismes de raisonnement basiques,
- ceux capable de fusionner les réponses de différentes sources,
- ceux mettant en place une interaction avec l'utilisateur,
- et enfin les systèmes capables de raisonnement analogique et permettant de répondre à des questions spéculatives.

Les travaux de recherche se sont jusqu'à maintenant principalement concentrés sur des approches supportant uniquement des questions factuelles. Cette première catégorie, bien qu'étant la plus simple, présente de nombreux verrous. [Hunter, 2000] y identifie des sous-types de questions particulièrement complexes : celles qui demandent des explications (typiquement des questions commençant par "Why" ou "How"), les questions commençant par "What" dont le type de la réponse n'est pas précisé, ou encore les questions définitoires.

Les interfaces en langue naturelle peuvent également être classifiées en fonction des sources qu'elles exploitent pour trouver la réponse aux questions qui leur sont posées. Sur cette dimension, trois grandes catégories se distinguent clairement et ont fait chacune l'objet de recherches à des périodes distinctes :

- les interfaces entre la langue naturelle et des bases de données,
- les interfaces entre la langue naturelle et des textes,
- les interfaces entre la langue naturelle et des bases de connaissances.

Ces catégories sont présentées dans les sections suivantes. Le travail décrit dans ce document appartenant à la troisième catégorie, celle-ci est plus détaillée, alors que les deux premières sont présentées de façon synthétique.

On peut également noter que deux autres catégories hybrides sont parfois distinguées des précédentes : les systèmes de question-réponse sur des documents semi-structurés comme des documents XML, des rapports scientifiques respectant un format donné (médecine, agriculture...), et les systèmes de question-réponse "propriétaires" exploitant des bases de données internes construites manuellement ou de façon supervisée. Ces derniers sont coûteux car ils nécessitent une main d'œuvre importante et ont le plus souvent un but commercial ; le système *START* [Katz et al., 2002], issu de la recherche académique et mis en ligne en 1993, était le premier de cette catégorie et a été suivi par des produits industriels dont

les plus connus sont *Wolfram Alpha*[12], *Powerset*[13], *True Knowledge*[14] et *Watson*[15] qui a récemment fait sensation en battant des concurrents humains au jeu télévisé *Jeopardy*. Depuis peu, le moteur de recherche *Google* lui-même dispose d'une fonctionnalité permettant de donner une réponse directement à une requête factuelle très simple, du type "Quelle est la date de naissance de Jean Dujardin ?"

2.4 Interfaces entre la langue naturelle et des bases de données

À la fin des années soixante, le besoin d'interfacer des requêtes utilisateur exprimées en langue naturelle avec des bases de données relationnelles émerge. Les recherches s'orientent alors vers la conception de systèmes capables de traduire des requêtes exprimées en langue naturelle vers des requêtes en langage formel. Ce domaine de recherche restera très actif jusqu'au début des années quatre-vingt-dix. Une veille détaillée du domaine est disponible dans [Androutsopoulos et al., 1995].

Les premières approches qui ont vu le jour étaient le plus souvent intégrées à des systèmes experts et ciblaient des domaines très spécifiques ; par exemple BASEBALL répondait à des questions factuelles simples concernant la ligue de baseball états-unienne et LUNAR visait à satisfaire la soif d'information des utilisateurs concernant l'analyse géologique de pierres lunaires. La principale limite de ces systèmes était leur forte dépendance au domaine ciblé. En effet, ils faisaient appel à des patrons lexico-syntaxiques, des grammaires, ou des règles d'association entre lexique et bases de données, qui étaient autant de connaissances implémentées "en dur" et spécifiques au domaine.

Dans un second temps, les interfaces entre la langue naturelle et les bases de données ont voulu limiter cette faiblesse en intégrant un langage de représentation intermédiaire dans leur processus d'interprétation [Martin et al., 1986; De Roeck et al., 1991; Thanisch et al., 1993]. La requête utilisateur est tout d'abord traduite en une représentation formelle intermédiaire, exprimée dans ce langage, qui représente le besoin en information de manière indépendante du domaine ciblé. De cette façon, le processus d'analyse linguistique est séparé de celui d'association entre les termes de la requête et la base de données. Cette décomposition en deux phases distinctes a en partie inspiré l'approche que nous présentons dans la partie *Contributions* de ce travail.

Plus récemment, PRECISE [Popescu et al., 2003] utilise un lexique interne

12. http://www.wolframalpha.com/
13. http://www.powerset.com/
14. http://www.evi.com/
15. http://www-05.ibm.com/fr/watson/

pour guider l'association entre les termes de la requête utilisateur et la base de données. L'approche proposée dans [Hallett et al., 2007] exploite une ontologie et assiste l'utilisateur lors de la formulation de sa requête, non pas en langue naturelle mais dans un langage formel.

Nous constatons que, malgré de très bonnes performances [Androutsopoulos et al., 1995], la grande majorité des travaux dans ce domaine n'ont pas su s'affranchir de la principale contrainte de leurs aînés, à savoir le fort couplage entre le système développé et le domaine visé. C-PHRASE [Minock, 2010] est la concrétisation des efforts les plus récents menés dans le but de corriger cette limite ; une phase de configuration est toujours nécessaire mais celle-ci est rendue la plus légère et accessible possible au travers d'une interface web permettant l'adaptation du système par un non-spécialiste à un domaine donné.

2.5 Interfaces entre la langue naturelle et des textes

Les approches proposées dans ce domaine exploitent une collection de documents comme source de connaissances. Il ne s'agit alors plus de traduire la requête dans un langage formel mais plutôt d'aller chercher directement la ou les réponses dans les documents. Les recherches visant à la résolution de cette tâche ont été largement orientées, de 1999 à 2007, par la compétition question-réponse sans restriction de domaine (*open domain question answering*) de la conférence sur l'extraction de données dans les textes (*TREC - Text REtrieval Conference*).

D'après [Hirschman and Gaizauskas, 2001], le processus d'interprétation de la grande majorité de ces systèmes se divise en deux étapes :

1. identifier le type du focus de la requête ; une bonne reconnaissance du type de l'objet est primordiale, et de nombreux travaux se sont penchés sur le problème, en définissant des hiérarchies de types de question à partir du type de la réponse attendue [Moldovan et al., 1999; Hovy et al., 2000; Wu et al., 2003; Srihari and Li, 1999] ;
2. déterminer des contraintes additionnelles sur cette entité ; ces contraintes sont le plus souvent issues des relations syntaxiques et sémantiques entre les termes de la requête en langue naturelle [Moldovan et al., 2002; Litkowski, 2000; Attardi et al., 2002].

Les systèmes actuels présentent de nombreuses fonctionnalités. Ils doivent être capables d'extraire les données du corpus de textes considéré, d'identifier le type de la question qui leur est soumise [Moldovan et al., 1999], d'identifier le type de l'objet de la question [Moldovan et al., 1999; Harabagiu et al., 2000], d'analyser la sémantique de la question et d'évaluer sa similarité avec les données ex-

traites [Moldovan et al., 2002; Litkowski, 2000; Attardi et al., 2002], de trier les réponses extraites, ou encore d'accéder à des ressources extérieures comme WordNet [16] [Harabagiu et al., 2000; Litkowski, 2000]. En comparaison aux interfaces entre la langue naturelle et les bases de données, de nombreuses tâches se complexifient du fait de la non-restriction du domaine et de la source de connaissances qui a une forme beaucoup moins contrainte et plus variable ; on peut notamment citer la reconnaissance d'entités nommées, l'extraction de relations, la résolution de coréférences et la désambiguïsation.

Un sous-ensemble de cette catégorie, apparu un peu plus tard, considère le web dans son intégralité comme corpus de documents. La tâche reste la même mais certains problèmes, comme le passage à l'échelle et l'hétérogénéité des données, prennent toute leur importance. Pour faire face à l'immensité du web, la majorité des systèmes proposés dans la littérature exploite des moteurs de recherche par mots-clés "classiques", comme Google. La requête utilisateur est d'abord transformée en une ou plusieurs requêtes compatibles avec le moteur de recherche choisi, puis ces requêtes sont exécutées sur le web, et enfin les réponses sont extraites des documents les plus pertinents renvoyés par le moteur de recherche.

Mulder [Kwok et al., 2001], par exemple, utilise WordNet pour classifier la requête et le type de l'objet, pour étendre et convertir la requête en plusieurs requêtes mots-clés qui sont ensuite exécutées sur Google. Le système extrait alors les réponses directement depuis les résumés retournés par le moteur de recherche. Pour gérer le bruit, certains sites sur le web pouvant contenir des informations incorrectes, Mulder tire parti de l'algorithme de tri de Google (évolution du PageRank) auquel il incorpore son propre système de vote pour ordonner les réponses.

Aidé par la campagne d'évaluation de TREC, ce domaine de recherche a été extrêmement actif dans les années 2000 et les systèmes développés ont obtenu d'excellents résultats. On constate cependant que la mise en place réelle de tels systèmes auprès d'utilisateurs finals n'a jamais été réalisée, si ce n'est pour des besoins de démonstration. Le développement de tels systèmes, notamment à l'échelle du web, se heurte à des verrous de taille (passage à l'échelle, gestion du bruit, identification des doublons…), et les solutions trouvées nécessitent d'importants efforts d'implémentation.

2.6 Interfaces entre la langue naturelle et des bases de connaissances

Les systèmes de question-réponse sémantiques (*semantic question answering systems*) cherchent la réponse aux questions qui leur sont posées dans une ou plu-

16. http://wordnet.princeton.edu/

sieurs bases de connaissances, le plus souvent fondées sur une ou plusieurs ontologies. Bien que rassemblant nombre des difficultés des systèmes présentés plus haut, cette catégorie d'approches est très prometteuse, de par les propriétés du format de stockage des connaissances. En effet, les auteurs de [Kaufmann and Bernstein, 2010] expliquent que les connaissances ontologiques ainsi que les connaissances assertionnelles contenues dans la base de connaissances peuvent permettre le développement de systèmes réellement indépendants du domaine ou du moins une grande facilité dans l'adaptation d'un système à un nouveau domaine.

Dans cette section, nous présentons les résultats des travaux d'utilisabilité menés jusque-là, puis nous décrivons les approches les plus marquantes de cette catégorie. Nous ne limitons pas la palette des approches décrites aux systèmes traitant des requêtes exprimées en langue naturelle ; le champ étudié est élargi à d'autres types d'approches, notamment celles traitant des requêtes mots-clés et celles proposant une interface de navigation, car celles-ci, bien qu'adoptant une stratégie différente de l'approche que nous proposons, partagent le même objectif : permettre l'interrogation d'une base de connaissances par un utilisateur final.

2.6.1 Utilisabilité

La pertinence des interfaces en langue naturelle a été mise en cause plusieurs fois dans la littérature, sans jamais pour autant mener à une conclusion claire [Chakrabarti, 2004; Dekleva, 1994; Desert, 1993; Thompson et al., 2005].

Certaines études [Dittenbach et al., 2003; Reichert et al., 2005] ont voulu analyser les préférences des utilisateurs finals et l'intérêt général des interfaces en langue naturelle. Les auteurs de [Dittenbach et al., 2003] ont développé une interface de requêtes en langue naturelle destinée à une plate-forme web dans le domaine du tourisme, et déployée durant dix jours. Les utilisateurs n'étaient soumis à aucune contrainte et pouvaient exprimer leurs requêtes comme ils le souhaitaient. 1425 requêtes uniques (les doublons ne sont comptabilisés qu'une fois) ont été recueillies (en anglais et en allemand). 57,05% des requêtes étaient des phrases complètes, exprimées en langue naturelle et grammaticalement correctes ; 21,26% étaient des fragments de questions, du type "double room for two nights in Vienna" ; 21,69% étaient des requêtes mots-clés. Les utilisateurs ont donc en majorité préféré s'exprimer en langue naturelle. De plus, les auteurs rapportent que les requêtes en langue naturelle étaient bien plus précises, bien que formulées de façon peu complexe. D'après eux, les interfaces en langue naturelle se montrent particulièrement utiles lorsque le public visé est hétérogène, comme dans le cas de la plate-forme utilisée. Les résultats de l'étude présentée dans [Reichert et al., 2005] sont différents, sans pour autant être contradictoires, les utilisateurs ayant participé appartenant cette fois à un ensemble homogène. Les auteurs ont demandé à des étudiants d'utiliser deux versions différentes d'un outil de question-réponse pour

le *e-learning*, CHESt[Linckels and Meinel, 2005]. Dans une version, les requêtes s'expriment sous forme de mots-clés ; dans l'autre, elles s'expriment en langue naturelle. Les étudiants déclarent en majorité (76%) préférer la version mots-clés mais concèdent également qu'ils utiliseraient plus volontiers la version langue naturelle si ils savaient pouvoir obtenir de meilleurs résultats. Malgré ces résultats, les auteurs pensent que le processus de formulation de requête en langue naturelle ne doit pas être considéré comme lourd et rebutant pour l'utilisateur.

Dans [Kaufmann and Bernstein, 2010], les auteurs évaluent l'utilisabilité des interfaces permettant aux utilisateurs d'accéder à des connaissances par le biais de la langue naturelle. Ils identifient trois principaux problèmes dans ce types d'interface :

- l'*ambiguïté* de la langue naturelle est le plus évident, l'*ambiguïté linguistique* –une même expression en langue naturelle peut être interprétée de différentes façons– va de pair avec la *variabilité linguistique* –une même idée peut s'exprimer de différentes façons en langue naturelle.
- la *barrière adaptative* (*adaptivity barrier*) rend les interfaces présentant de bonnes performances de recherche peu portables ; ces systèmes sont la plupart du temps spécifiques au domaine et, par conséquent, difficiles à adapter à de nouveaux contextes. Comme expliqué plus haut, l'ensemble des connaissances ontologiques contenues dans une base de connaissances est vu comme un moyen réaliste de pallier ce problème.
- le *problème d'habitabilité* (*habitability problem*) selon lequel l'utilisateur final peut se sentir perdu face à une trop grande liberté dans l'expression de sa requête. La plupart des interfaces en langue naturelle n'expliquent pas comment exprimer ses requêtes ou quel type d'information peut être demandé ; en conséquence l'utilisateur peut exprimer des requêtes au-delà des capacités du système (que ce soit du point de vue du besoin en information ou de l'expression de la requête) ou, pire encore, le système peut mal interpréter la requête, et l'utilisateur peut ne pas s'en rendre compte et considérer une réponse inappropriée comme satisfaisante.

La contribution principale de [Kaufmann and Bernstein, 2010] est inspirée de ce dernier problème : l'*hypothèse d'habitabilité* prétend qu'une interface en langue naturelle, pour présenter la meilleure utilisabilité, devrait imposer une certaine structure à l'utilisateur dans le but de l'orienter lors du processus de formulation de la requête. Ainsi le "syndrome de la feuille blanche" devrait être évité. Cependant, la contrainte structurelle ne doit pas être trop restrictive, le risque étant d'aliéner l'utilisateur. Pour soutenir cette hypothèse, les auteurs décrivent l'étude d'utilisabilité qu'ils ont conduite auprès d'utilisateurs. Pour ce faire, ils ont développé quatre interfaces de requêtes et les ont confrontées à 48 utilisateurs.

Les auteurs proposent de situer chaque interface sur un *continuum de formali-*

sation (*formality continuum*) en fonction de la nature de l'interaction avec l'utilisateur final. Les approches complètement LN sont à une extrémité de ce continuum, alors que les langages formels de requêtes sont à l'autre extrémité. Ainsi, conformément à l'hypothèse d'habitabilité, l'interface présentant la meilleure utilisabilité devrait se situer quelque-part au milieu du continuum. Les systèmes développés par les auteurs dans le cadre de cette étude d'utilisabilité sont naturellement positionnés sur ce continuum :

- *NLP-Reduce* [Kaufmann et al., 2007] est une interface en langue naturelle "naïve", indépendante du domaine, et dans laquelle les requêtes peuvent être exprimées sous forme de mots-clés, de fragments de phrases ou de phrases complètes en langue naturelle ;
- *Querix* [Kaufmann et al., 2006] est une interface en langue naturelle indépendante du domaine, dans laquelle les requêtes peuvent être exprimées sous la forme de questions en langue naturelle respectant une contrainte simple : elles doivent commencer par l'un des six débuts de question autorisés ("Which...","What...", "How many...", "How much...", "Give me...", ou "Does...") ;
- *Ginseng* [Bernstein et al., 2005] est une interface en langue naturelle contraignant l'utilisateur à exprimer sa requête sous la forme d'une phrase respectant une grammaire donnée ; cette grammaire est générée à partir de règles statiques définissant la structure générale de la phrase et de règles dynamiques issues des étiquettes de la base de connaissances cible ;
- *Semantic Cristal* est un outil graphique qui aide l'utilisateur à construire des requêtes graphe formelles.

Les résultats de cette étude montrent clairement que les utilisateurs se sentent le plus à l'aise en utilisant *Querix* et son langage de requêtes légèrement contraint, ce qui tend à corroborer l'hypothèse d'habitabilité. Cependant, il est important de noter que les retours de ces utilisateurs représentent leur perception subjective de l'exactitude des réponses retournées et non les qualités réelles de ces systèmes selon des critères objectifs. En effet, *NLP-Reduce* réalise de meilleures performances sur des critères objectifs majeurs, comme le rappel ou le temps moyen de réponse. De plus, certaines interactions complémentaires entre le système et l'utilisateur comme les pratique *Querix* (désambiguïsation explicite des termes, retour en langue naturelle) semblent créer une certaine confiance.

L'hypothèse d'habitabilité est donc en grande partie vérifiée. Mais, encore une fois, ces résultats sont issus de retours utilisateurs et non de l'observation de critères objectifs. Les utilisateurs impliqués dans cette étude sont habitués à être contraints dans leurs requêtes, notamment par l'omniprésente recherche par mots-clés.

Malgré tout cela, nous soutenons la thèse selon laquelle l'ergonomie augmente lorsque l'utilisateur peut interagir de façon plus naturelle avec l'ordinateur, notam-

ment par le biais de la voix, et nous considérons que l'interprétation de requêtes exprimées en langue naturelle est un des verrous scientifiques majeurs de ces prochaines années. Ceci se justifie par la présence importante et toujours grandissante des périphériques mobiles dans notre quotidien, et le succès des systèmes de reconnaissance vocale. Les utilisateurs s'attendent de plus en plus à demander tout type d'information à leur téléphone comme à un humain. Dans ce contexte, les interfaces entièrement orientées langue naturelle seront inévitables. Nous gardons néanmoins des travaux présentés dans cette section la nécessité d'interaction et de retour vers l'utilisateur. Ce point de vue est renforcé par les travaux récents présentés dans [Damljanović et al., 2013] qui exploitent des retours utilisateur et des fenêtres de clarification pour augmenter l'expérience de l'utilisateur.

2.6.2 Vue d'ensemble des interfaces entre utilisateur et bases de connaissances

Nous donnons ici un aperçu des types d'approches existants visant à formuler des graphes requêtes.

Le long du continuum de formalisation

Pour cela, nous suivons le modèle de classification proposé dans [Kaufmann and Bernstein, 2010], et présentons ces approches le long du continuum de formalisation, du plus formel au plus permissif, en commençant par les langages de requête eux-mêmes.

À l'extrémité de ce continuum se trouvent les langages formels de requêtes graphes comme SPARQL [17]. Ils sont la cible des systèmes que nous présentons ici. De tels langages présentent d'évidentes contraintes d'utilisabilité, ce qui les rend totalement inadaptés aux utilisateurs finals. Exprimer des requêtes formelles implique de connaître et de respecter la syntaxe du langage utilisé, de comprendre un modèle de graphes et, le plus contraignant, de connaître le schéma de la base interrogée. Le travail présenté dans [Elbassuoni et al., 2010] vise à étendre le langage SPARQL et son mécanisme d'interrogation afin de prendre en compte des mots-clés et des jokers quand l'utilisateur ne connaît pas exactement le schéma sur lequel il veut requêter ; une telle approche nécessite que l'utilisateur connaisse le langage SPARQL.

Très proche sur le continuum de formalisation, nous trouvons les approches qui assistent l'utilisateur pendant la formulation de la requête dans de tels langages.

17. http://www.w3.org/TR/rdf-sparql-query/

Des interfaces comme *Flint*[18] et *SparQLed*[19] implémentent des fonctionnalités simples comme la coloration syntaxique et l'auto-complétion. D'autres approches reposent sur des interfaces graphiques comme [Athanasis et al., 2004] pour les requêtes RQL, [Russell and Smart, 2008; Clemmer and Davies, 2011] pour SPARQL ou [Team, 2007] pour les graphes conceptuels. Même si ces interfaces graphiques sont utiles et rendent la formulation de requêtes bien moins laborieuse, elles ne permettent pas de dépasser les limites, exposées plus haut, inhérentes aux langages de requêtes graphes formels.

Sewelis [Ferré and Hermann, 2011; Ferré et al., 2011; Ferré and Hermann, 2012] introduit la recherche par facette fondée sur des requêtes (*Query-based Faceted Search*), un nouveau paradigme combinant le requêtage et la recherche par facette (paradigme en recherche d'information dans lequel l'utilisateur navigue dans le jeu de données en construisant des filtres au sein d'une interface graphique), alors que d'autres approches comme *Ginseng* [Bernstein et al., 2005] et *squall2sparql* [Ferré, 2012, 2013b] définissent un langage naturel contrôlé dont la traduction en SPARQL est directe.

D'autres travaux visent à générer automatiquement – ou semi-automatiquement – des requêtes formelles à partir de requêtes exprimées sous forme de mots-clés ou de phrases en LN. L'utilisateur exprime son besoin en information de façon intuitive, sans avoir à connaître le langage de requêtes ou bien le formalisme de représentation de connaissances utilisé dans le système. Certains travaux ont proposé de traduire des requêtes de haut niveau en requêtes formelles dans différents langages comme SeREQL [Lei et al., 2006] ou SPARQL [Zhou et al., 2007; Tran et al., 2009; Cabrio et al., 2012]. Dans ces systèmes, la génération de requêtes nécessite les étapes suivantes : (i) appariement des mots de la requête aux entités sémantiques de la base de connaissances ; (ii) construction de graphes requêtes liant les entités détectées à l'étape précédente ; (iii) classement des requêtes construites, (iv) sélection de la bonne requête par l'utilisateur. Les approches se sont pour l'instant focalisées sur des problèmes précis : optimiser l'étape d'appariement en exploitant des ressources externes comme Wordnet ou Wikipedia [Lei et al., 2006; Wang et al., 2008; Cabrio et al., 2012], optimiser l'indexation et l'exploration de la base de connaissances pour la construction de la requête graphe [Zhou et al., 2007; Tran et al., 2009], améliorer le classement des requêtes candidates [Wang et al., 2008], ou encore optimiser l'identification de relations à l'aide de patrons textuels [Cabrio et al., 2012].

Autosparql [Lehmann and Bühmann, 2011; Unger et al., 2012] étend la catégorie précédente : après une interprétation sommaire de la requête utilisateur exprimée en LN, le système échange avec l'utilisateur, lui demandant des exemples positifs

18. http://openuplabs.tso.co.uk/demos/sparqleditor
19. http://sindicetech.com/sindice-suite/sparqled/

et négatifs de réponses, afin de raffiner l'interprétation initiale qui avait été faite de la requête.

En fonction de leur (in)dépendance au domaine

Les auteurs de [Lopez et al., 2011] répartissent les interfaces entre utilisateur et bases de connaissances selon un autre axe : celui du niveau de couplage entre le système et le domaine considéré. Certains systèmes sont développés avec un domaine d'application en tête, à l'instar des interfaces entre langue naturelle et bases de données. Leur adaptation à un nouveau domaine peut se révéler coûteuse même si, dans le cas d'interfaces entre langue naturelle et bases de connaissances, l'exploitation d'ontologies peut grandement limiter ce coût. Aussi, d'après le problème de la *barrière adaptative* introduit plus haut, ces interfaces très dépendantes du domaine sont censées être les plus performantes. *QACID* [Ferrández et al., 2009] en est un exemple. Le processus d'interprétation de ce système est fondé sur un corpus de requêtes qui sont préalablement analysées et regroupées en *clusters*. Le portage du système à un nouveau domaine nécessite donc le recueil et l'analyse d'un nouveau jeu de requêtes. D'autres systèmes, comme *ORAKEL* [Cimiano, 2004; Cimiano et al., 2007] et *e-librarian* [Linckels and Meinel, 2005], sont également dépendants du domaine d'application de par l'exploitation d'un lexique spécifique à ce domaine, mais développent des efforts importants dans le but de rendre cette adaptation la plus simple et accessible possible. Les auteurs de [Cimiano et al., 2007] expliquent que *ORAKEL* a pu être adapté avec succès à un nouveau domaine par un expert de ce domaine (et non du système). Enfin, les autres systèmes sont capables de s'adapter automatiquement à un nouveau domaine, en exploitant les données de l'ontologie et de la base de connaissances ciblée. C'est le cas notamment de *Ginseng* [Bernstein et al., 2005], *Panto* [Wang et al., 2007], *NLP-Reduce* [Kaufmann et al., 2007], *Querix* [Kaufmann et al., 2006], *QuestIO* [Tablan et al., 2008] et *FREyA* [Damljanovic et al., 2010].

2.6.3 Évaluation des interfaces entre langue naturelle et bases de connaissances

Au début de notre travail, il n'existait aucune méthode formelle et objective permettant d'évaluer les interfaces entre langue naturelle et bases de connaissances. Les équipes développant ce type d'interface menaient leurs propres évaluations, et nous en avons fait de même dans un premier temps, comme nous le verrons en 7.1. Puis, le domaine de recherche gagnant en popularité et le nombre d'approches proposées augmentant régulièrement, il devenait urgent de mettre en place un cadre d'évaluation commun. Un certain nombre d'initiatives ont été lancées dans ce contexte.

La tâche *Entity Track*[20] de TREC, aujourd'hui abandonnée, voulait évaluer la recherche d'entités sur le web : pour une entité, un type et une relation donnée, les systèmes devaient retourner une liste d'entités de ce même type, reliées à cette même entité par cette même relation. Une sous-tâche ciblait plus particulièrement le web de données.

La campagne d'évaluation *INEX Linked Data Track*[21] a vu le jour en 2012. Elle comprend la tâche *Jeopardy*, nommée d'après un jeu télévisé populaire au États-Unis, dont le but est également de retourner une liste d'entités correspondant à un besoin en information exprimé de différentes manières (en langue naturelle, sous forme de mots-clés et sous forme de requêtes semi-structurées inspirées de SPARQL).

Le *Semantic Search Challenge*[22] évalue la pertinence des entités retournées par un système à partir de requêtes réellement exprimées par des utilisateurs sous forme de mots-clés.

Enfin, la compétition *Question Answering over Linked Data* [Lopez et al., 2013] (*QALD*) nous a semblé être l'initiative la plus pertinente pour évaluer notre approche. Elle fournit deux jeux de données, chacun d'entre-eux relatif à une base de connaissances (Musicbrainz[23] ou DBpedia[24]) et comportant deux jeux de requêtes rédigées en langue naturelle, un jeu d'entraînement que les participants peuvent exploiter pour développer et optimiser leurs systèmes, et un jeu de test qui est utilisé pour l'évaluation proprement dite. Les participants sont libres de considérer un seul des deux jeux de données ou les deux. Trois éditions de cette compétition ont pour l'instant eu lieu, en 2011, 2012 et 2013. Nous avons participé aux première (QALD-1[25]) et troisième (QALD-3[26]) éditions dans le but d'évaluer notre approche et de la comparer aux autres ; les résultats de ces évaluations sont décrits en 7.2 et 7.3.

2.7 Idées caractéristiques des approches existantes

Dans cette section, nous reprenons les systèmes introduits plus haut et nous nous arrêtons sur certaines de leurs spécificités.

20. http://krisztianbalog.com/files/trec2011-entity-overview.pdf
21. https://inex.mmci.uni-saarland.de/tracks/lod/
22. http://semsearch.yahoo.com/
23. http://musicbrainz.org/
24. http://dbpedia.org/
25. http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=task1&q=1
26. http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=task1&q=3

2.7.1 Langage naturel contrôlé

L'imposition d'un vocabulaire et d'une syntaxe à l'utilisateur est une alternative viable à mi-chemin entre les langages formels et une expression de requête en langue naturelle complètement libre. Par rapport à l'expression totalement libre, cette approche permet de contourner les problèmes d'ambiguïté du langage naturel. Par rapport aux langages formels, elle permet de réutiliser les capacités cognitives des utilisateurs pour la communication de connaissances et ainsi de réduire les efforts nécessaires à l'apprentissage du langage [Ferré, 2013b].

Outre *Ginseng* [Bernstein et al., 2005] présenté plus haut, *SQUALL* [Ferré, 2012, 2013b] se veut un langage de requêtes équivalent à SPARQL 1.1 en termes d'expressivité mais avec une syntaxe proche du langage naturel. L'approche supporte donc toutes les fonctionnalités de SPARQL 1.1, y compris la mise à jour de données. Cependant, le caractère naturel des phrases servant à exprimer des requêtes reste critiquable [Ferré, 2013b]. Ces phrases sont certes assez courtes (20% plus longues que leur équivalent en langue naturelle) mais ne se détachent pas réellement de la base de connaissances cible : les URI des ressources doivent être employés (un mécanisme de préfixe est néanmoins prévu) et la structure même des phrases suit celle des graphes interrogés. Par exemple, la requête "List all TelevisionEpisode-s whose series is res:The Sopranos and whose seasonNumber is 1" demande la liste des épisodes de la première saison de la série télévisée "The Sopranos". De plus, certaines des fonctionnalités avancées de SPARQL 1.1 sont prises en compte de façon assez artificielle. Par exemple, le caractère + exprimant qu'une propriété peut apparaître plusieurs fois au sein d'une chaîne de propriétés est repris à l'identique en SQUALL, comme dans la requête "Which publication-s cite+ Paper42 ?" qui demande la liste des publications qui citent *Paper42* ou citent une publication qui cite *Paper42*, etc. La traduction entre SQUALL et SPARQL étant directe, cette approche a de très loin surpassé les autres concurrents de la compétition QALD-3 (présentée en 2.6.3) sur les critères de précision et de rappel.

2.7.2 Utilisation de patrons de questions

Les travaux présentés dans [Ou et al., 2008] et [Ferrández et al., 2009] (système *QACID* développé dans le cadre du projet QALL-ME [Ferrandez et al., 2011]) semblent développés indépendamment mais présentent d'importantes similarités. Les deux utilisent des patrons de questions et exploitent le principe émergent d'implication textuelle (*textual entailment*). Le processus général proposé par les deux méthodes se décompose en deux tâches principales.

La première tâche, qui consiste à générer les patrons de questions, est plus précisément décrite dans [Ou et al., 2008]. Les auteurs établissent tout d'abord, à partir de la base de connaissances, la liste exhaustive des triplets ⟨classe, propriété,

co-domaine⟩ à partir des définitions de domaine (rdfs:domain) et de co-domaine (rds:range) d'une propriété, et des définitions de classe exprimant des restrictions de co-domaine locales (owl:allValuesFrom). Cette liste représente l'ensemble des triplets possibles de requête relatifs aux instances. Les patrons de questions obtenus sont composés chacun d'une question prédictive (*predictive question* ; ex: *What is the name of the movie which has the genre [genre_value] ?*) et du modèle de requête (*query template*) SPARQL correspondant. Les auteurs de [Ferrández et al., 2009] expliquent partir d'un ensemble de requêtes utilisateur en langue naturelle, y identifier les entités nommées, et ensuite regrouper automatiquement en cluster les requêtes sémantiquement équivalentes (sans donner de précisions sur la méthode utilisée). À chaque cluster est manuellement associée une requête SPARQL pour obtenir des patrons de questions équivalents à ceux de [Ou et al., 2008].

La deuxième tâche a pour but l'interprétation de chaque nouvelle requête utilisateur et est détaillée dans [Ferrández et al., 2009]. L'objectif est d'établir des inférences lexico-sémantiques entre une nouvelle requête et l'ensemble des phrases prédictives (*predictive sentence*) associées à chaque patron. Pour cela, les auteurs utilisent un moteur d'implication textuelle prenant en compte à la fois des règles lexicales et des règles sémantiques. Les règles lexicales utilisent, pour la plupart, des mesures de distances lexicales connues dans la littérature (algorithme de Smith-Waterman [Smith and Waterman, 1981], appariement de sous-séquences consécutives, distance de Jaro-Winkler [Jaro, 1989; Winkler, 1999], distance euclidienne, coefficient de similarité de Jaccard [Jaccard, 1902, 1912]). Une sixième règle lexicale, appelée *Wh-terms*, a été définie pour adapter le système aux spécificités dues aux requêtes ; son rôle est de vérifier que les requêtes comparées utilisent le même mot interrogatif (*Who, What, Where...*), celui-ci étant particulièrement influent sur le sens d'une question. Les règles sémantiques sont, quant-à-elles, beaucoup plus spécifiques à la tâche. Elles consistent par exemple à s'assurer que les éléments de la base de connaissances liés aux entités nommées repérées dans la requête utilisateur correspondent en nombre et en type à ceux du patron. Finalement, le patron dont la phrase est la plus proche de la requête utilisateur est utilisé pour générer la requête finale.

Nous avons identifié quelques faiblesses dans ces approches. Premièrement, la mesure de similarité entre patron et requête est faite au niveau de la phrase et il semble par conséquent difficile de prendre en compte des formulations très différentes qui expriment le même besoin en information ; la mesure de similarité ne prend pas en compte la syntaxe des requêtes. De plus, les patrons sont petits (composés le plus souvent d'un seul triplet) et ciblent des requêtes bien précises ; ce manque de généricité implique la nécessité de construire un grand nombre de patrons pour une utilisation dans un domaine donné. Les phrases prédictives sont très simples et parfois très similaires les unes aux autres, ce qui est à l'origine de

nombreuses erreurs lors de l'étape d'implication textuelle. Par exemple, le moteur d'implication textuelle associe avec la même confiance la requête "Who directed 300 ?" aux phrases prédictives "Who is the star of [Movie_name] ?", "Who is the writer of [Movie_name] ?", et "Who is the director of [Movie_name] ?" Enfin, les auteurs de [Ou et al., 2008] expliquent que certains des patrons générés n'ont pas de sens d'un point de vue utilisateur, comme par exemple le patron demandant le nom du cinéma se trouvant à des coordonnées GPS données.

Les idées décrites dans ces articles présentent quelques similarités avec celles développées dans nos travaux. Notre approche est en effet fondée sur l'exploitation de patrons de requêtes pour orienter le processus d'interprétation d'une requête utilisateur. Nous avons cependant voulu éviter les limites décrites ci-dessus en définissant des patrons plus modulaires, capables de couvrir un plus grand besoin en information, comme nous le verrons dans le chapitre 3.

2.7.3 Recours à un formalisme intermédiaire pour représenter la requête

À l'instar des interfaces entre langue naturelle et bases de données présentées en section 2.4, certaines interfaces entre langue naturelle et bases de connaissances intègrent un langage de représentation intermédiaire. C'est notamment le cas de *ORAKEL* [Cimiano, 2004; Cimiano et al., 2007] et *e-Librarian* [Linckels and Meinel, 2005].

ORAKEL produit une représentation formelle du sens d'une question par les méthodes de la sémantique compositionnelle. Le processus d'analyse utilise deux lexiques distincts : un lexique du domaine sur lequel porte la base de connaissances interrogée qui contient les expressions propre à ce domaine et les associe aux éléments de la base de connaissances, et un lexique général, indépendant du domaine, contenant les termes généralement utilisés pour construire des question (déterminants, pronoms, pronoms interrogatifs...).

De même, *e-Librarian* construit une représentation logique non ambiguë de la requête à interpréter en identifiant des triplets linguistiques composés de trois éléments présentant des dépendances syntaxiques, et en associant les termes à une ontologie pour résoudre les ambiguïtés.

Ces approches fonctionnent bien en théorie mais présupposent une analyse sémantique/syntaxique sans faille, c'est-à-dire que ni l'utilisateur (dans la formulation de sa requête) ni l'outil d'analyse sémantique/syntaxique n'ont le droit à l'erreur. On se rend compte que ces deux conditions sont rarement réunies dans des conditions réelles : les utilisateurs ne respectent pas les règles de grammaire à la lettre et, surtout, les outils de TAL sont loin d'être infaillibles pour les raisons évoquées en 2.1. Nous réexploitons cette approche en y incorporant une plus grande

souplesse. Nous définissons dans le chapitre 4 la requête pivot, une structure intermédiaire à la syntaxe stricte, mais à la sémantique plus floue, ce qui permet de pallier d'éventuels défauts dans la formulation de la phrase ou dans l'analyse automatique qui en est faite.

2.7.4 Prédiction des requêtes utilisateur

La majorité des interfaces entre langue naturelle et base de connaissances ignorent les interprétations possibles de la requête utilisateur qui n'ont pas de réponse dans la base de connaissances ciblée [Cabrio et al., 2012; Dima, 2013; He et al., 2013]. C'est une façon supplémentaire d'exploiter la base de connaissances. Les auteurs de [d'Aquin and Motta, 2011] vont plus loin dans cette direction. Ils déduisent, en effectuant une analyse formelle de concepts [Ganter et al., 1996] sur la base de connaissances, l'ensemble des requêtes susceptibles d'être posées par un utilisateur et permettent ensuite, via une interface utilisateur, la navigation entre ces requêtes possibles.

Cette heuristique présente l'avantage de limiter les possibilités d'interprétation d'une requête utilisateur mais ne se montrera, selon nous, pas toujours pertinente, car il est envisageable qu'un utilisateur pose une question à laquelle la base de connaissances n'a pas la réponse. Le principal risque est que cette question soit alors mal interprétée (assimilée à une question proche dont la base de connaissances a la réponse) et qu'un résultat inadapté soit retourné à l'utilisateur.

2.7.5 Raffinage du résultat par interaction avec l'utilisateur

Comme nous l'avons vu plus haut, *Querix* [Kaufmann et al., 2006] interagit directement avec l'utilisateur pour désambiguïser les termes de la requête, ce qui tend à instaurer une plus grande confiance. *AutoSparql* [Lehmann and Bühmann, 2011; Unger et al., 2012] utilise les retours utilisateurs comme exemples dans un algorithme d'apprentissage supervisé. L'utilisateur formule sa requête en langue naturelle ou sous forme de mots-clés, puis un module de traduction (appelé *NLP filter*) procède à une interprétation naïve de cette requête, dans le but de proposer des exemples de réponses. Parmi les exemples suggérés, l'utilisateur en signale au moins un comme faisant partie des réponses attendues à sa requête ; si aucune ou trop peu de suggestions font partie des réponses attendues par l'utilisateur, celui-ci peut en signaler de nouvelles au travers d'une boîte de recherche. Ces exemples de réponses attendues sont une base d'exemples positifs pour l'algorithme d'apprentissage utilisé par *AutoSparql*. Si, lors du déroulement de l'algorithme, le système se rend compte que cette base d'exemples n'est pas suffisante, il proposera alors

à l'utilisateur de nouveaux exemples, choisis de façon à apporter les informations manquantes, que l'utilisateur pourra déclarer positifs ou négatifs.

AutoSparql utilise un algorithme léger d'apprentissage supervisé : le *Query Tree Learner (QTL)*. Les principales étapes de cet algorithme sont les suivantes : (1) calcul de l'arbre le moins général contenant l'ensemble des éléments positifs sur le graphe RDF ; (2) si cet arbre contient au moins un des éléments négatifs, alors on considère que l'apprentissage a échoué, l'utilisateur en est informé et a la possibilité de réajuster les exemples ; (3) calcul de la généralisation maximale de l'arbre qui ne contient aucun élément négatif (algorithme *Negative Based Reduction)*. Une fois cet arbre maximal obtenu, il peut être traduit en SPARQL.

Parmi les points forts avancés par les auteurs, nous retenons la garantie d'apprendre le bon arbre de requête s'il existe et si tous les exemples sont corrects, le nombre assez restreint d'exemples nécessaires (quatre en moyenne), et la *tractabilité* de l'approche. Parmi les points faibles, nous retenons que cette approche ne supporte qu'un sous-ensemble de SPARQL ; cette limite est courante pour ce type d'interfaces mais ici, d'une façon générale, le champ des requêtes supportées est très limité : les requêtes ne doivent pas contenir de structures de type UNION, doivent pouvoir s'exprimer sous forme d'arbre et retourner une liste contenant suffisamment d'éléments pour permettre l'apprentissage (pas de liste trop courte, pas non plus de ASK, de COUNT, ou de réponse unique). Enfin, l'approche telle qu'elle est présentée ne gère pas le bruit : tous les exemples donnés par l'utilisateur doivent être corrects pour que l'apprentissage n'échoue pas.

2.7.6 Verbalisation de requêtes SPARQL

Enfin, signalons qu'un domaine a récemment pris de l'importance : celui de la verbalisation de requêtes structurées, c'est-à-dire leur traduction en langage naturel. Certains travaux ont déjà été proposés pour interpréter des requêtes SQL [Koutrika et al., 2010], mais maintenant, le besoin de verbaliser des requêtes SPARQL émerge avec la popularisation des technolgies du web sémantique.

À notre connaissance, seules deux approches partageant cet objectif ont été proposées à ce jour ; toutes deux ont pour langue cible l'anglais. *Sparticulation* [Ell et al., 2012] [27] identifie dans la requête SPARQL une entité principale et exprime en langue naturelle chacune des contraintes de la requête à partir de cette entité.

Le processus de traduction d'une requête SPARQL dans le système SPARQL2NL [Ngonga Ngomo et al., 2013] [28] se divise en quatre étapes : 1) normalisation de la requête et extraction des informations de type pour chaque variable, 2) génération d'une représentation générique de la requête, 3) application de règles de réduction

27. http://km.aifb.kit.edu/projects/spartiqulator/
28. http://sparql2nl.aksw.org/demo

et de remplacement, et 4) génération de la phrase en langue naturelle.

Ces approches, bien qu'ayant un objectif inverse de celui des interfaces en langue naturelle, peuvent se révéler très utiles pour ces dernières. En effet, nombre de ces interfaces en langue naturelle génèrent une phrase descriptive en langue naturelle représentant l'interprétation qui a été faite de la requête utilisateur originale. Ce retour utilisateur permet à ce dernier de s'assurer que sa question a bien été comprise. Chaque système proposant cette fonctionnalité implémente sa propre méthode de génération de phrase descriptive. L'exploitation d'une approche de verbalisation générique comme celles décrites plus haut permettrait de simplifier le développement de ces systèmes. De plus, une verbalisation réalisée par un système indépendant et découplé de la méthode d'interprétation garantirait une plus grande fiabilité.

2.8 Conclusion

Dans la majorité des approches mentionnées plus haut, la requête graphe finale est obtenue en explorant la base de connaissances et en reliant ses entités qui ont été identifiées dans la requête utilisateur. Cette étape, guidée par une heuristique, peut mener à une requête graphe incohérente ou dénuée de sens du point de vue de l'utilisateur. Nous proposons de définir et d'exploiter des patrons de requêtes pour guider cette étape de construction de la requête. Ces patrons construits manuellement assurent la cohérence des requêtes générées du point de vue humain. Dans le chapitre suivant, nous justifions l'intuition derrière l'idée de patrons, puis nous présentons ces patrons de façon intuitive et formelle.

Deuxième partie

Contributions scientifiques

Chapitre 3

Les patrons de requêtes

Le postulat principal sur lequel reposent nos travaux est le suivant : dans les applications réelles, les requêtes émises par les utilisateurs sont pour l'essentiel des variations autour de quelques familles typiques de requêtes. Cette intuition a été vérifiée par la suite et nous a amené à proposer une approche dans laquelle des patrons de requêtes représentent chacune de ces familles et permettent l'interprétation de requêtes exprimées en langue naturelle et leur traduction en un langage formel.

Dans la section 3.1, nous justifions notre approche par les résultats d'études publiées dans la littérature portant sur les requêtes exprimées par des utilisateurs sur des moteurs de recherche du web. Nous procédons ensuite à une présentation détaillée des patrons de requêtes, d'abord intuitivement dans la section 3.2, puis formellement dans la section 3.3. Dans la section 3.4, nous présentons l'ontologie que nous avons développée pour faciliter la construction, l'évolution, l'exploitation et le partage de patrons, puis nous donnons en section 3.5 un exemple de patron construit pour des besoins réels, avant de conclure le chapitre en section 3.6.

Les patrons de requêtes sont au cœur de nos travaux et sont par conséquent présents dans la majorité des publications liées à ce travail. Nous avons proposé une première définition dans [Pradel et al., 2012a, 2011a] ; cette définition a ensuite été augmentée avec les notions de sous-patron et de cardinalité dans [Pradel et al., 2013a].

3.1 Les requêtes d'utilisateurs

Comprendre le besoin en information des utilisateurs du web est une tâche difficile car aucun corpus n'a encore été constitué pour capturer explicitement ce besoin et donc permettre des études significatives. Les analyses qui existent reposent sur l'étude de *logs* de moteurs de recherche enregistrant la représentation

du besoin sous la forme de mots-clés ainsi que différentes informations complémentaires telles que la page web consultée à la suite de la formulation de la requête, l'adresse IP...

Les auteurs de [Spink et al., 2000] analysent des *logs* de requêtes en anglais issus d'un moteur de recherche sur le web et discutent des résultats en s'appuyant sur des études similaires antérieures [Jansen et al., 1998, 2000; Silverstein et al., 1999]. Les premières observations formulées, plutôt superficielles, ont tendance à contredire notre hypothèse. Une analyse plus poussée des requêtes utilisateur permet d'identifier des thèmes prédominants sans pour autant identifier des familles de requêtes et ainsi confirmer notre hypothèse. En effet, les thèmes identifiés sont assez généraux et ne peuvent pas être directement assimilés à des familles de requêtes. Cependant, les étude présentées dans la suite, plus récentes, vont plus loin sur cet axe d'analyse et nous confortent dans nos choix.

Les auteurs de [Jansen et al., 2007] effectuent une analyse portant sur plus d'un million et demi de requêtes formulées en 2005 par plus de 530.000 utilisateurs à partir d'un méta-moteur et comparent ces résultats à ceux obtenus lors d'études de logs extraits au début des années 2000. Certaines des observations formulées à propos de ces requêtes nous intéressent particulièrement :

- les thématiques des requêtes identifiées par les termes co-occurrents deviennent de plus en plus variées, ce qui montre une tendance des utilisateurs à utiliser le web pour une variété plus importante de tâches de recherche,
- la proportion des requêtes contenant trois termes et plus est en augmentation, tout comme le nombre de requêtes dans une session, ce qui montre que les utilisateurs ont des besoins de plus en plus précis.

Cette dernière observation est partagée par [Bendersky and Croft, 2009]. Les auteurs analysent plus de 15 millions de requêtes issues des logs de MSN search et observent que 10% des requêtes sont longues (plus de 4 mots). Ils mettent aussi en avant le fait que les moteurs de recherche actuels ont de mauvaises performances sur ce type de requêtes et que la façon la plus efficace d'exprimer les besoins en information précis que représentent ces requêtes est la formulation de questions en LN. Cette idée est également suivie dans [Dror et al., 2013], qui propose de générer automatiquement une question en LN lorsque l'utilisateur n'est pas satisfait par les réponses d'un moteur de recherche d'information classique. Cette question est ensuite posée sur des systèmes de questions-réponses communautaires tels que *Yahoo ! answer*.

De manière complémentaire, une étude récente [Hafernik and Jansen, 2013] montre également qu'une majorité de requêtes (environ 60%) issues de log du moteur d'AOL en 2006 porte sur un besoin en information spécifique caractérisé par un ensemble de 9 attributs tels qu'une requête demandant la comparaison de plusieurs entités par rapport à un critère, des informations complémentaires sur

les directions pour atteindre un lieu ou réaliser une tâche... Le fait que chaque requête spécifique ait pu être caractérisée par au moins un des attributs montre qu'une certaine regularité caractérise les requêtes spécifiques.

Une autre étude [Downey et al., 2008] analyse les performances des moteurs de recherche actuels par rapport aux besoins en information fréquents (identifés en comparant les couples <requêtes-page consultée> d'une semaine à l'autre dans une barre de recherche de navigateur) et ceux qui sont plus rares. Ils mettent en évidence qu'environ 70% des requêtes sont fréquentes et soulignent par ailleurs que pour satisfaire un besoin en information rare (qui n'a pas été ou très peu exprimé par d'autres utilisateurs), les moteurs de recherche actuels ne proposent pas de réponse directe. La meilleure façon de procéder pour un utilisateur consiste alors à poser une question plus générale puis affiner sa recherche en naviguant. Les auteurs préconisent donc l'utilisation de patrons de requêtes pour améliorer les performances des systèmes en adaptant le processus de recherche aux particularités de ces deux types de requêtes.

Les auteurs de [Hollink et al., 2013] analysent des sessions de recherche sur des logs de *Yahoo !*. Lorsqu'ils ciblent leur analyse sur les requêtes portant sur le domaine des films, ils observent que des patrons de requêtes peuvent être identifiés pour caractériser l'enchaînement des requêtes et prédire si la requête va aboutir à la satisfaction du besoin de l'utilisateur. Ces patrons sont constitués à partir de connaissances que l'on peut extraire du web de données en utilisant l'entité identifiée dans la première requête. Lorsque le patron est suivi dans une session, le besoin a de grandes chances d'être satisfait. [Hollink et al., 2010] mènent une étude similaire sur un moteur de recherche d'images et concluent sur l'intérêt pour les moteurs de recherche d'intégrer ces patrons lors de la recherche afin de fournir plus rapidement la réponse à l'utilisateur.

De nombreuses analyses mettent donc en valeur des régularités et familles de besoins en information dans les requêtes exprimées par les utilisateurs finals. La pertinence de ces analyses est certes discutable, car les requêtes étudiées sont exprimées sous formes de mots-clés, mais ces requêtes sont à ce jour les seules ressources issues d'utilisateurs disponibles en quantité suffisante. [Mazumdar et al., 2013] insiste sur la nécessité de mener d'autres analyses sur des requêtes dans lesquelles le besoin est exprimé de manière plus explicite. Nous avons mené une telle analyse à une échelle bien plus modeste et en présentons les résultats dans le chapitre 7.

De telles observations nous ont poussé à proposer un mécanisme qui permette l'expression de requêtes utilisateur en langue naturelle et leur interprétation par association à des patrons de requêtes prédéfinis.

3.2 Présentation intuitive

Les exemples 1, 3, 4 et 5 présentent des requêtes simples exprimées en langue naturelle, portant sur le domaine du cinéma et susceptibles d'être émises par un utilisateur final.

Exemple 1 *Who plays in the movie The Artist ?*

Exemple 2 *Who are the main actors of the movie The Artist ?*

Exemple 3 *Who is the director of the movie The Artist ?*

Exemple 4 *Which movies were directed by Michel Hazanavicius ?*

Exemple 5 *Which movies were directed by Coen brothers ?*

Les figures 3.1, 3.2, 3.3, 3.4 et 3.5 présentent des graphes requêtes qui sont des traductions formelles possibles des exemples précédents de requêtes en langue naturelle ; ces graphes ciblent une base de connaissances sur le domaine du cinéma dont un sous-ensemble a été présenté en 1.4.6. Le processus d'appariement de ces graphes sur cette base de connaissances (selon les mécanismes présentés en 1.5) mènerait aux réponses de chaque requête (en supposant bien évidemment que la base de connaissances contienne ces réponses). Par exemple, la requête présentée dans la sous-figure 3.1 demande la liste des acteurs jouant dans le film "The Artist". Dans cette traduction sous forme de graphe, la variable `?res` représente l'objet de la recherche ; au cours du processus d'appariement de la requête graphe à la base de connaissances ciblée, l'ensemble de toutes les valeurs prises par cette variable sera considéré comme la réponse à la requête.

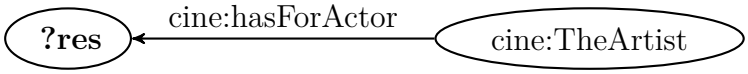


FIGURE 3.1 – Une traduction possible sous forme de requête graphe de la requête LN de l'exemple 1.

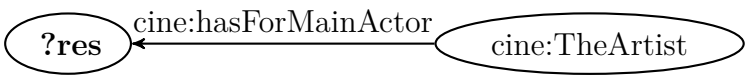


FIGURE 3.2 – Une traduction possible sous forme de requête graphe de la requête LN de l'exemple 2.

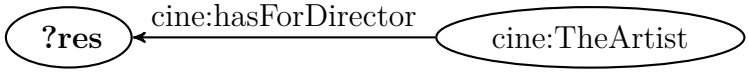


FIGURE 3.3 – Une traduction possible sous forme de requête graphe de la requête LN de l'exemple 3.

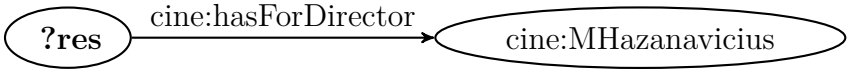


FIGURE 3.4 – Une traduction possible sous forme de requête graphe de la requête LN de l'exemple 4.

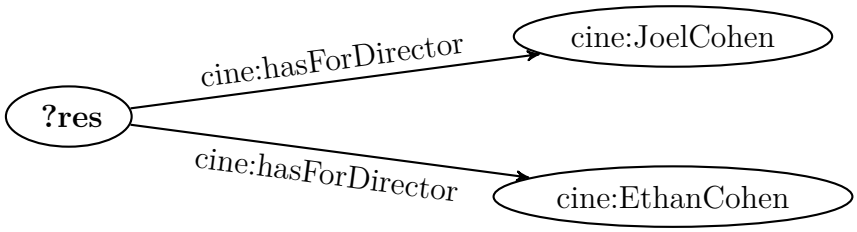


FIGURE 3.5 – Une traduction possible sous forme de requête graphe de la requête LN de l'exemple 5.

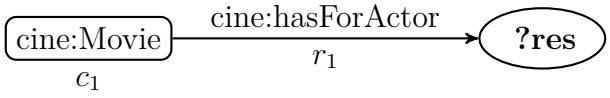
Ces requêtes semblent très familières. Chacun s'est un jour posé la question de savoir quel(le) était l'acteur ou l'actrice jouant dans tel ou tel film. Cette observation élémentaire nous a conduit à l'idée de patrons de requêtes.

Le patron présenté dans la figure 3.6 est un patron très simple couvrant justement toutes les requêtes concernant les acteurs jouant dans un film donné.

Le sommet c_1 et l'arc r_1 sont appelés des *éléments qualifiants*. Un élément qualifiant est un sommet ou un arc d'un patron qui est amené à être remplacé par une autre ressource afin de transformer le patron en requête.

c_1 cible la classe `cine:Movie` de la base de connaissances. Cela signifie que, pour obtenir le graphe requête correspondant à une requête utilisateur appartenant à cette famille de requêtes, il faut substituer cet élément qualifiant par l'instance du film concerné. Nous appelons ce processus une *instanciation de patron*. L'instanciation du patron 3.6 pour la requête de l'exemple 1 conduira, après substitution de l'élément qualifiant par la ressource `cine:TheArtist`, au graphe requête présenté dans la figure 3.1.

r_1 cible la propriété `cine:hasForActor`. Cela signifie que, lors de l'instanciation du patron, cette propriété peut être remplacée par l'une de ses sous-propriétés. Ainsi, l'instanciation du patron 3.6 pour la requête de l'exemple 2 conduira au graphe requête présenté dans la figure 3.2.



Modèle de phrase descriptive : *Who plays in* [c_1] *?*

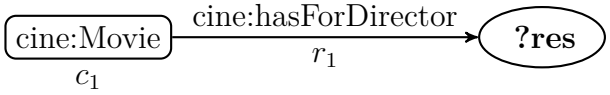
FIGURE 3.6 – Patron inspiré de la requête 3.1.

Chaque patron est également composé d'un modèle de phrase qui peut être utilisé pour générer une phrase descriptive exprimant en langue naturelle le sens de la requête obtenue après instanciation du patron. Ce modèle se présente sous la forme d'une phrase en langue naturelle dans laquelle des sous-chaînes de caractères font référence aux éléments qualifiants du graphe de patron. Ces sous-chaînes sont destinées à être remplacées par une étiquette de la ressource à laquelle se substitue l'élément qualifiant référencé lors de l'instanciation du patron. Par exemple, l'instanciation du patron 3.6 effectuée pour la requête de l'exemple 1 conduira à la phrase descriptive suivante : "Who plays in The Artist ?" (dans ce cas, nous obtenons exactement la même phrase car le modèle de phrase descriptive a été directement inspiré de la phrase de l'exemple). Le modèle de phrase descriptive peut aussi être enrichi par des structures de contrôle pour des patrons plus complexes (contenant des parties optionnelles ou répétables) comme nous le verrons plus tard.

De la même façon, nous pouvons construire les patrons de requêtes 3.7(a) et 3.7(b), respectivement inspirés des graphes requêtes 3.3 et 3.4. On remarque dans ces exemples que tous les éléments qualifiants d'un patron ne sont pas nécessairement référencés dans le modèle de phrase descriptive. Par exemple l'élément qualifiant r_1 du patron 3.7(a), ciblant la propriété `cine:hasForDirector`, n'apparaît pas dans le modèle de phrase descriptive de ce patron. Cela est volontaire de la part du concepteur du patron dans le but de rendre la phrase descriptive correspondant à une instanciation de patron moins lourde (par exemple, la phrase "Who directed the Artist ?" est plus légère et plus agréable que "Who is director of The Artist ?"). En revanche, cela implique le risque que la phrase descriptive d'une instanciation dans laquelle cette propriété est spécialisée ne reflète pas cette spécialisation (bien entendu, ce risque n'existe que si la propriété concernée a au moins une sous-propriété).

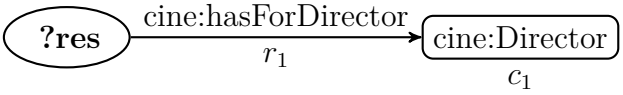
Les patrons 3.7(a) et 3.7(b) sont assez similaires ; l'un permet d'exprimer une requête concernant le(s) réalisateur(s) d'un film donné, l'autre le(s) film(s) d'un réalisateur donné. Le patron de requêtes 3.7(c) est le résultat de la fusion de ces deux patrons ; ainsi, son graphe contient deux éléments qualifiants c_1 et c_2, l'un faisant référence à la classe `cine:Movie` et l'autre à la classe `cine:Director`. Étant

donné que ce patron permet d'exprimer une requête visant des éléments de classes différentes, le modèle de phrase descriptive n'apparaît plus sous la forme d'une question ; il est en effet maintenant nécessaire d'anticiper les deux cas où soit l'instanciation de c_1, soit l'instanciation de c_2 est l'objet de la requête. L'instanciation de ce patron pour la requête de l'exemple 3 conduira au graphe requête présenté dans la figure 3.8 et la phrase descriptive générée sera : "The Artist was directed by **some director**." Une partie de cette phrase est mise en évidence syntaxiquement pour informer l'utilisateur qu'elle fait référence à l'objet de la requête



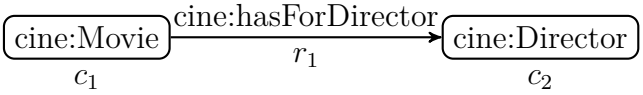
Modèle de phrase descriptive : *Who directed* [c_1] *?*

(a)



Modèle de phrase descriptive : *Which movie was directed by* [c_1] *?*

(b)



Modèle de phrase descriptive : [c_1] *was directed by* [c_2] *?*

(c)

FIGURE 3.7 – Les patrons 3.7(a) et 3.7(b) sont respectivement inspirés des requêtes 3.3 et 3.4. Le patron 3.7(c) est issu de la fusion des patrons 3.7(a) et 3.7(b).

Puis, pour permettre la génération du graphe requête 3.5, le patron 3.7(c) doit être instancié en répétant l'unique triplet composant son graphe, avec deux instances différentes pour la classe `cine:Director`. Il est possible de rendre un triplet répétable au cours du processus d'instanciation en lui assignant une cardinalité maximale n qui lui permet d'être répété autant de fois que nécessaire ; le patron obtenu est montré sur la figure 3.9. Le modèle de phrase descriptive contient alors une structure de contrôle *for* indiquant la partie du modèle de phrase qui doit

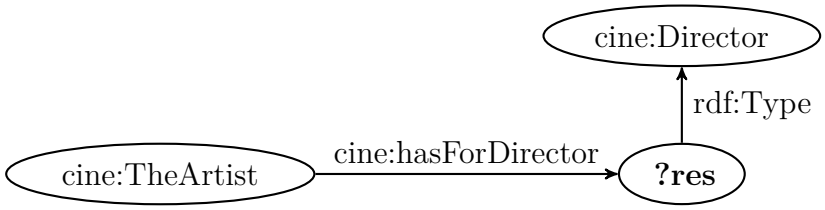
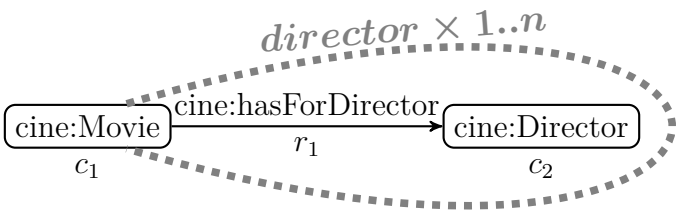


FIGURE 3.8 – Graphe requête issu de l'instanciation du patron 3.7(c) pour la requête de l'exemple 3.

elle aussi être répétée. L'instanciation de ce patron pour la requête de l'exemple 5 conduira au graphe requête présenté dans la figure 3.5 et la phrase descriptive générée sera : "**A movie** was directed by Joel Coen and Ethan Coen."



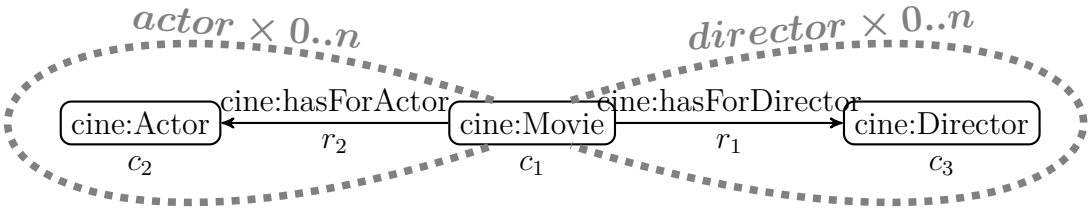
Modèle de phrase descriptive : [c_1] *was directed by* ***for_director(***[c_2]***)*** *?*

FIGURE 3.9 – Patron contenant un unique triplet répétable.

Ainsi, pour pouvoir répéter des sous-ensembles d'un patron, nous décomposons ce patron en *sous-patrons* auxquels nous attribuons des cardinalités. Ces sous-patrons peuvent être constitués de plusieurs triplets et peuvent aussi être imbriqués, comme nous verrons dans la suite.

De la même façon, il est possible d'assigner à un sous-patron une cardinalité minimale qui, quand elle a pour valeur 0, rend le sous-patron optionnel. Un sous-patron optionnel ne doit pas nécessairement apparaître dans l'instanciation du patron qui le contient. Le patron montré dans la figure 3.10 est une évolution du patron 3.9 contenant un nouveau triplet optionnel qui permet de demander la liste des acteurs qui ont joué dans un film donné (ou les films dans lesquels ont joué certains acteurs). Le modèle de phrase descriptive s'enrichit cette fois d'une structure de contrôle *if* indiquant la partie du modèle de phrase relative au sous-patron optionnel qui n'apparaîtra pas si ce sous-patron n'est pas instancié.

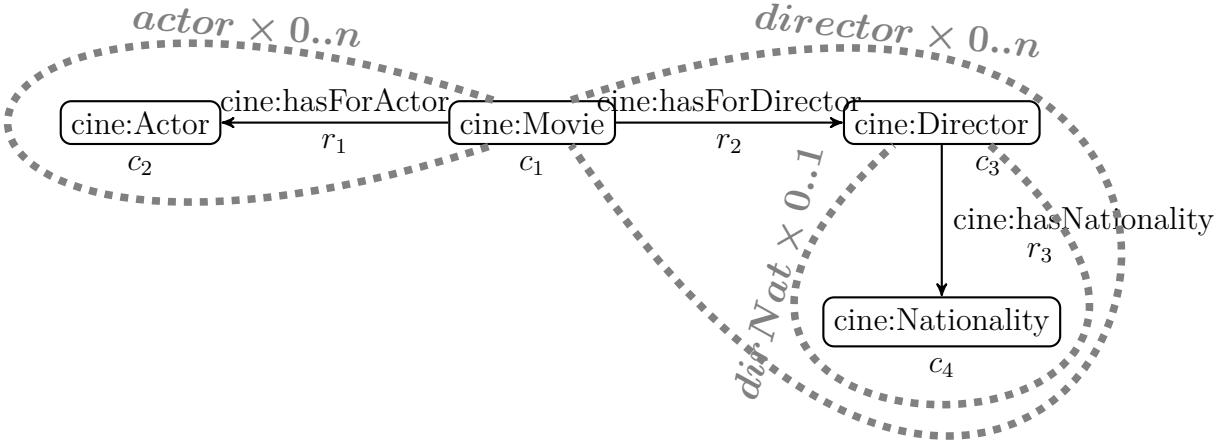
Enfin, afin de permettre une plus grande modularité, les sous-patrons imbriqués sont autorisés. Cela permet de générer à partir d'un même patron des requêtes avec des niveaux de granularité différents. Par exemple, le patron montré dans la figure 3.11 est une évolution du patron 3.10 qui permet, grâce au sous-patron



Modèle de phrase descriptive : [c_1] ***if_actor(*** *stars* ***for_actor(*** [c_2]*),)*
if_director(*was directed by* ***for_director(*** [c_3] ***))***

FIGURE 3.10 – Patron de requêtes fait de sous-patrons optionnels ou répétables.

imbriqué optionnel *dirNat*, de préciser éventuellement la nationalité du réalisateur d'un film (ou de chacun des réalisateurs si le sous-patron *director* est instancié plusieurs fois).



Modèle de phrase descriptive : [c_1] ***if_actor(*** *stars* ***for_actor(*** [c_2]*),)*
if_director(*was directed by* ***for_director(*** [c_3] ***if_dirNat(*** *which is* [c_4] ***)))***

FIGURE 3.11 – Exemple de sous-patrons imbriqués.

La section suivante généralise les exemples précédents et définit formellement ce qu'est un patron de requêtes dans notre approche.

3.3 Définition formelle

Un *patron* est un cas particulier de *sous-patron*. La notion de *sous-patron* est fondée sur celle de *sous-patron basique*. Un *sous-patron basique* est constitué de *triplets de patron*. Un *triplet de patron* est un ensemble de trois *éléments de patrons*.

Dans cette section, nous définissons toutes ces notions de façon ascendante. Nous reprenons ici les notations introduites en 1.4.6 lors de la définition de la base de connaissances considérée.

Définition 2 (Élement de patron) *$\mathcal{C}_P$ est l'ensemble de tous les éléments de patrons ciblant une classe. La fonction $targ$ associe à chaque membre $c \in \mathcal{C}_P$ la classe $targ(c) \in \mathcal{C}$ qu'il cible.*

ℓ_P est l'ensemble de tous les éléments de patrons ciblant un type de littéral. La fonction $targ$ associe à chaque membre $l \in \ell_P$ le type de littéral $targ(l) \in \ell$ qu'il cible.

$\mathcal{I}_P$ est l'ensemble de tous les éléments de patrons ciblant une instance. La fonction $targ$ associe à chaque membre $i \in \mathcal{I}_P$ l'instance $targ(i) \in \mathcal{I}$ qu'il cible.

$\mathcal{L}_P$ est l'ensemble de tous les éléments de patrons ciblant un littéral. La fonction $targ$ associe à chaque membre $L \in \mathcal{L}_P$ le littéral $targ(L) \in \mathcal{L}$ qu'il cible.

$\mathcal{P}_{o_P}$ est l'ensemble de tous les éléments de patrons ciblant une propriété d'objet. La fonction $targ$ associe à chaque membre $p_o \in \mathcal{P}_{o_P}$ la propriété d'objet $targ(p_o) \in \mathcal{P}_o$ qu'il cible.

$\mathcal{P}_{d_P}$ est l'ensemble de tous les éléments de patrons ciblant une propriété de données. La fonction $targ$ associe à chaque membre $p_d \in \mathcal{P}_{d_P}$ la propriété de données $targ(p_d) \in \mathcal{P}_d$ qu'il cible.

$\mathcal{E}_p = \mathcal{C}_P \cup \ell_P \cup \mathcal{I}_P \cup \mathcal{L}_P \cup \mathcal{P}_{o_P} \cup \mathcal{P}_{d_P}$ est l'ensemble de tous les éléments de patron.

$\mathcal{E}_{qual} = \mathcal{C}_P \cup \ell_P \cup \mathcal{P}_{o_P} \cup \mathcal{P}_{d_P}$ est l'ensemble de tous les éléments de patrons potentiellement qualifiants.

Exemple 6 *Le patron de la figure 3.11 contient sept élements :*

$$c_1,c_2,c_3,c_4 \in \mathcal{C}_P, \text{ avec } targ(c_1) = \texttt{cine:Movie}, targ(c_2) = \texttt{cine:Actor},$$
$$targ(c_3) = \texttt{cine:Director} \text{ et } targ(c_4) = \texttt{cine:Nationality}$$
$$r_1,r_2,r_3 \in \mathcal{P}_{o_P}, \text{ avec } targ(r_1) = \texttt{cine:hasForActor}, targ(r_2) = \texttt{cine:hasForDirector},$$
$$\text{et } targ(r_3) = \texttt{cine:hasNationality}$$

Un *triplet de patron* est une liste de trois *éléments de patron* (un sujet, un prédicat, un objet) respectant les contraintes exprimées dans la figure 3.12. Le sujet est un élément de patron ciblant soit une classe, soit une instance. Le prédicat est un élément de patron ciblant soit une propriété d'objet, soit une propriété de données. Si le prédicat cible une propriété d'objet, alors l'objet est un élément de patron ciblant soit une classe, soit une instance. Si le prédicat cible une propriété de données, alors l'objet est un élément de patron ciblant soit un type de littéral, soit un littéral.

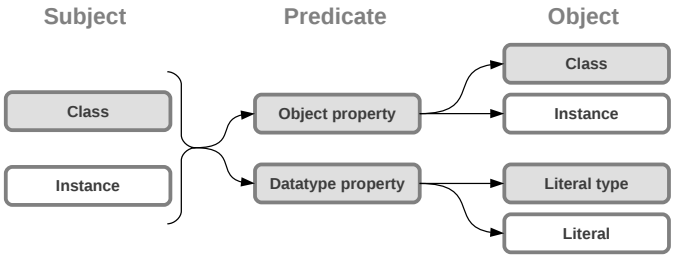


FIGURE 3.12 – Contraintes sur les éléments constituant un triplet de patron. Les éléments à fond gris peuvent être utilisés comme éléments qualifiants

Définition 3 (Triplet de patron) *L'ensemble de tous les triplets de patron se note* $\mathcal{T}_P$ *et correspond à l'ensemble suivant :*

$$\mathcal{T}_P = (\mathcal{C}_P \cup \mathcal{I}_P) \times \begin{pmatrix} \left(\mathcal{P}_{o_P} \times (\mathcal{C}_P \cup \mathcal{I}_P)\right) \\ \cup \\ \left(\mathcal{P}_{d_P} \times (\ell_P \cup \mathcal{L}_P)\right) \end{pmatrix}$$

Une fonction elem associe à chaque triplet de patron $t \in \mathcal{T}_P$ *l'ensemble des éléments de patrons qui le constituent :*

$$\forall t = (e_{p1},e_{p2},e_{p3}) \in \mathcal{T}_\mathcal{P},\ elem(t) = \{e_{p1},\ e_{p2},\ e_{p3}\}$$

Exemple 7 *Le patron de la figure 3.11 contient trois triplets de patron :* $t_1,\ t_2,\ t_3 \in \mathcal{T}_P$

$$t_1 = (c_1,\ r_1,\ c_2)$$
$$t_2 = (c_1,\ r_2,\ c_3)$$
$$t_3 = (c_3,\ r_3,\ c_4)$$

Un sous-patron basique est un ensemble de triplets de patron.

Définition 4 (Sous-patron basique) *L'ensemble de tous les sous-patrons basiques se note* $\mathcal{B}_p$ *et correspond à l'ensemble suivant :*

$$\mathcal{B}_p = 2^{\mathcal{T}_P}$$

Une fonction elem associe à chaque sous-patron basique $b \in \mathcal{B}_p$ *l'ensemble des éléments de patrons qui le constituent :*

$$\forall b \in \mathcal{B}_p, elem(b) = \bigcup_{t \in b} elem(t)$$

Exemple 8 *Le patron de la figure 3.11 est composé de trois sous-patrons basiques (contenant chacun un unique triplet de patron) :* b_1,b_2,$b_3 \in \mathcal{B}_p$

$$b_1 = \{t_1\}$$
$$b_2 = \{t_2\}$$
$$b_3 = \{t_3\}$$

Définition 5 (Sous-patron) *L'ensemble de tous les sous-patrons se note* $\mathcal{S}_p$. *Si le 4-uplet* $s = (S, Q, c_{min}, c_{max}) \in 2^{\mathcal{S}_p \cup \mathcal{B}_p} \times 2^{\mathcal{Q}} \times \mathbb{N} \times \mathbb{N}^*$ *est un sous-patron, on note*

- $cont(s) = S$ *l'ensemble des sous-patrons (basiques) contenus dans* s,
- $qual(s) = Q$ *son ensemble d'éléments qualifiants,*
- $card_{min}(s) = c_{min}$ *et* $card_{max}(s) = c_{max}$ *ses cardinalités minimale et maximale respectivement,*
- *et* $elem(s) = \cup_{s' \in S} elem(s')$ *l'ensemble des éléments de patrons qui le constituent.*

Un sous-patron est défini récursivement de façon suivante :

1. $\begin{cases} si & b \in \mathcal{B}_p,\ b \text{ est connexe},\ Q \in 2^{\mathcal{E}_{qual}},\ Q \subset elem(b), \\ & c_{min} \in \mathbb{N},\ c_{max} \in \mathbb{N}^* \text{ et } c_{min} \leq c_{max}, \\ alors & (b, Q, c_{min}, c_{max}) \in \mathcal{S}_p; \end{cases}$

2. $\begin{cases} si & s_0 \in \mathcal{B}_p \text{ } (s_0 \text{ éventuellement égal à } \emptyset),\ s_1, \ldots, s_n \in \mathcal{S}_p, n \geq 1, \\ & \bigcup_{0 \leq i \leq n} cont(s_i) \text{ est connexe}, \\ & Q \in 2^{\mathcal{E}_{qual}},\ Q \subset \bigcup_{0 \leq i \leq n} elem(s_i),\ \bigcup_{1 \leq i \leq n} qual(s_i) \cap Q = \emptyset, \\ & c_{min} \in \mathbb{N},\ c_{max} \in \mathbb{N}^*,\ c_{min} \leq c_{max}, \\ & \forall\ 0 \leq i \leq n,\ 1 \leq j \leq n,\ i \neq j,\ qual(s_i) \cap elem(s_j) = \emptyset, \\ & et\ \forall\ 1 \leq i \leq n,\ \exists v \in elem(s_i) \text{ tel que } v \text{ est un sommet} \\ & d'articulation\ de\ \bigcup cont(s_i)\ (c'est\text{-}à\text{-}dire\ \bigcup cont(s_i) \setminus v \text{ n'est pas} \\ & connexe\ et\ admet\ cont(s_i) \setminus v \text{ comme composante connexe}), \\ alors & (\bigcup_{0 \leq i \leq n} s_i, Q, c_{min}, c_{max}) \in \mathcal{S}_p; \end{cases}$

Autrement dit, un sous-patron s_p est soit :

- un sous-patron basique connexe, associé à un sous-ensemble de ses éléments considéré comme l'ensemble des éléments qualifiants de s_p, et deux valeurs entières portant les informations de cardinalités, la cardinalité minimale étant logiquement inférieure à la cardinalité maximale,
- un ensemble de sous-patrons (dont éventuellement un sous-patron basique), associé à un sous-ensemble des éléments de ces sous-patrons considéré comme l'ensemble des éléments qualifiants de s_p, et deux valeurs entières portant les informations de cardinalités, tels que

– le graphe formé par l'union de l'ensemble des graphes des sous-patrons est connexe,
– pour chaque sous-patron inclus, ses éléments qualifiants n'apparaissent dans aucun autre sous-patron inclus ni dans l'ensemble des éléments qualifiants du sous-patron s_p formé,
– chaque sous-patron inclus contient un sommet d'articulation du graphe de s_p ; cette dernière condition permet de garantir que la suppression d'un sous-patron optionnel conduira toujours à un graphe requête connexe, et que la répétition d'un sous-patron sera faite de façon cohérente autour de cet élément qui sera un point d'ancrage au reste du graphe requête,
– la cardinalité minimale est inférieure à la cardinalité maximale.

Graphe requête connexe Certaines des contraintes énoncées dans la définition d'un sous-patron permettent d'assurer que le graphe de n'importe quelle requête générée à partir d'un patron sera connexe. Cette contrainte est un choix de notre part ; elle n'est pas imposée par SPARQL car un motif de graphe SPARQL peut très bien ne pas être connexe. On constate cependant que la très grande majorité des besoins en information se traduisent par des graphes requêtes connexes. Le résultat de l'exécution d'une requête formée d'un motif de graphe non connexe étant égal au produit cartésien des résultats de l'exécution des requêtes formées de chacun des sous-graphes connexes du motif de graphe initial, les graphes requêtes non connexes sont le plus souvent liés à des erreurs ou à des besoins très spécifiques. Nous avons choisi de ne pas prendre en compte les graphes non connexes pour ces raisons.

Exemple 9 *Le patron de la figure 3.11 est composé de trois sous-patrons : actor, director, dirNat* $\in \mathcal{S}_p$*. actor et dirNat sont construits à partir d'un sous-patron basique, alors que director est construit à partir de deux sous-patrons dont un basique :*

$$\begin{aligned} actor &= \Big(\{b_1\},\, \{r_1,\, c_2\},\, 0,\, n\Big) \\ dirNat &= \Big(\{b_3\},\, \{r_3,\, c_4\},\, 0,\, 1\Big) \\ director &= \Big(\{b_2,\, dirNat\},\, \{r_2,\, c_3\},\, 0,\, n\Big) \end{aligned}$$

Définition 6 (Patron de requêtes) *Un patron de requêtes est un sous-patron qui n'est contenu dans aucun autre sous-patron.*

Si p est un patron, alors $card_{min}(p) = card_{max}(p) = 1$*.*

$\mathcal{P}_P$ *est l'ensemble de tous les patrons de requêtes.*

Exemple 10 *Le patron movieDesc* $\in \mathcal{P}_P$ *de la figure 3.11 se note de la façon suivante :*

$$movieDesc = (\{actor, director\}, \{c_1\}, 1, 1) \in \mathcal{P}_P$$

3.4 Une ontologie pour modéliser les patrons

Dans cette section, nous proposons l'ontologie *patterns*[1], permettant de représenter dans les langages du web sémantique les patrons eux-mêmes, à l'image de ce qui est fait dans [Dietrich and Elgar, 2005] pour les patrons de conception issus du génie logiciel. Outre le fait d'homogénéiser les connaissances que nous manipulons, cela nous permet de partager, de comparer, de différencier et de faire évoluer les patrons qui deviennent eux-mêmes interrogeables et manipulables au moyen de requêtes SPARQL. Nous pouvons ainsi constituer des entrepôts de patrons, ces entrepôts pouvant être répartis sur le Web.

3.4.1 Qualité et bonnes pratiques

Durant la conception de cette ontologie, nous nous sommes efforcé d'appliquer les techniques couramment recommandées dans la communauté. Nous avons notamment exploité des patrons de conception d'ontologies. Le plus important de ces patrons est le patron de conception *normalization* introduit dans [Rector, 2003] ; cette méthode permet de construire des ontologies modulaires et réutilisables en définissant les classes par les propriétés que doivent vérifier leurs instances. Le concepteur de l'ontologie n'a ainsi pas à se soucier des propriétés de subsomption : la taxonomie est inférée automatiquement par le moteur d'inférences (une classe A subsume une classe B si les instances de B présentent au moins toutes les propriétés des instances de A).

Parmi les autres bonnes pratiques mises en œuvre, citons l'alignement de notre ontologie avec des ontologies de référence (comme FOAF présentée dans [Brickley and Miller, 2007]) et la définition pour chaque propriété d'objet d'une propriété inverse facilitant ainsi les manipulations.

Enfin, l'ontologie ayant été décrite en OWL2, nous avons tiré parti des nouveautés de ce langage en termes d'expressivité présentées dans [Golbreich and Wallace, 2012]. Nous avons notamment défini des chaînes de propriétés permettant de plus

1. téléchargeable à l'adresse `http://swip.univ-tlse2.fr/SwipWebClient/welcome.html`

grandes capacités d'inférence sans avoir recours à un langage consacré à l'expression de règles, et utilisé la possibilité de considérer une ressource avec deux visions différentes (méthode dite du *punning* présentée en 3.4.3).

Notre ontologie étant construite selon les principes du patron de conception *normalization* introduit plus haut, les classes sont donc définies à l'aide des propriétés. Nous présentons dans un premier temps les propriétés, puis les classes principales de l'ontologie.

3.4.2 Les propriétés

La Figure 3.13 représente la hiérarchie des principales propriétés d'objet de notre ontologie, caractérisées par leurs domaines et co-domaines. À chaque propriété correspond une propriété inverse.

La propriété `makesUp` est une relation de méronymie générique ; elle est spécialisée par différentes sous-propriétés choisies en fonction de la nature de la partie et du tout comme `isSubjectOf`, `isPropertyOf`, `isObjectOf`, `makesUpPattern`, `isSentenceOf` et `isSubsentenceOf`.

`isSentenceOf` permet d'associer le modèle de phrase descriptive à un patron. Les propriétés `isMadeUpOfList`, `isSubsentenceOf`, `sstTargetsPc` et `ssitHasValue` sont impliquées dans la construction du modèle de phrase descriptive d'un patron, comme décrit plus bas.

La propriété `targetsKBElement` exprime la relation entre un élément de patron et l'élément (classe, propriété ou type de données) de l'ontologie cible auquel cet élément de patron fait référence ; elle permet ainsi de spécifier les possibilités d'appariement. L'utilisation d'une de ses quatre sous-propriétés, `targetsClass`, `targetsProperty`, `targetsInstance` et `targetsLiteralType`, dépend de la nature de l'élément ciblé. Le co-domaine de chacune de ces propriétés ne peut être exprimé car cela nécessiterait l'utilisation d'une partie du vocabulaire réservé (`owl:Class`, `owl:Property`, `owl:Literal`).

Les propriétés de type de données `hasCardinalityMin` et `hasCardinalityMax` permettent de spécifier les cardinalités des sous-patrons.

Enfin, les propriétés `patternHasAuthor` (sous-propriété de `dbo:author`) et `targetsOntology` permettent de préciser respectivement le ou les auteurs d'un patron de requêtes et la ou les ontologies ciblées par ce patron.

3.4.3 Les classes

La Figure 3.14 présente la hiérarchie des classes principales de notre ontologie, également reliées par la relation de méronymie.

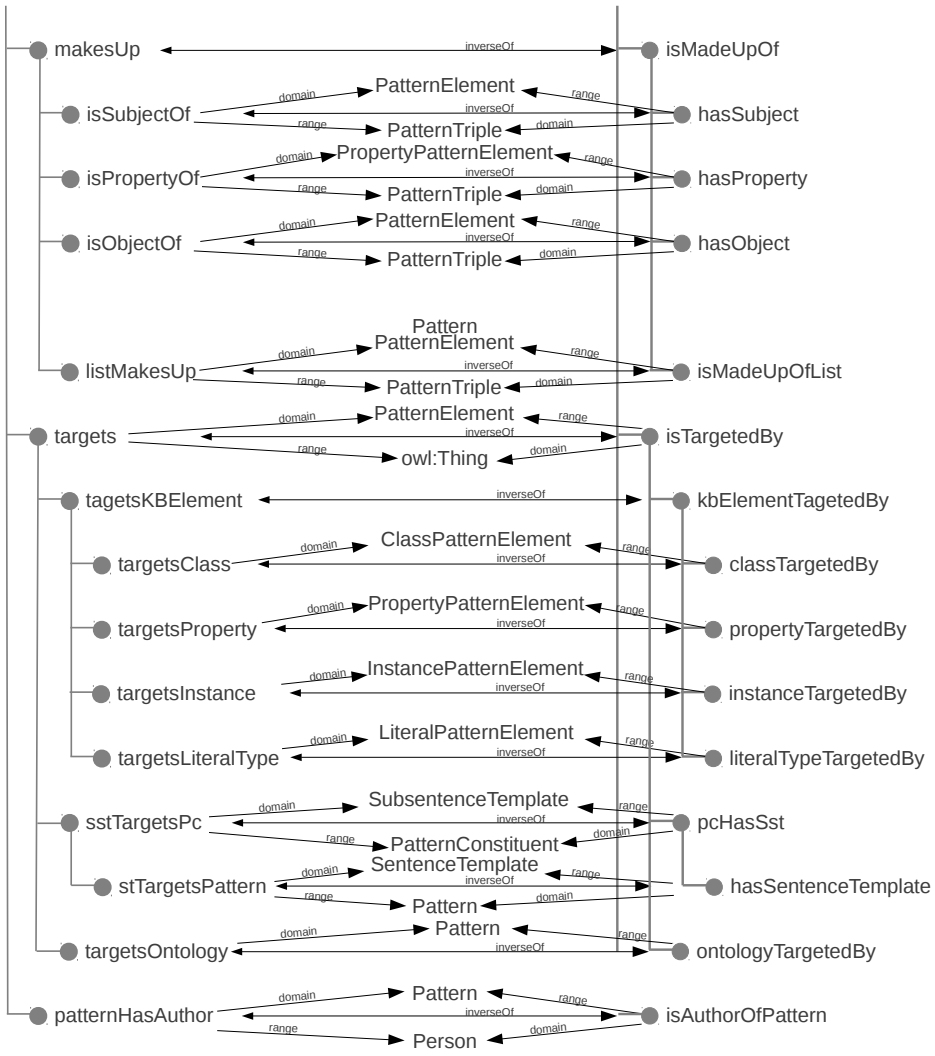


FIGURE 3.13 – Les propriétés d'objet de l'ontologie SWIP.

Sous-patrons et patron Les sous-patrons sont représentés par la classe `Subpattern`, qui subsume les classes `NecessarySubpattern`, `ContingentSubpattern` et `SimpleSubpattern`. La classe `ContingentSubpattern` représente les sous-patrons optionnels (dont la cardinalité minimale est nulle), et la classe `NecessarySubpattern` représente les sous-patrons qui doivent obligatoirement apparaître dans l'instanciation du patron qui les contient (leur cardinalité minimale est supérieure ou égale à un). Un patron (classe `Pattern`) est une sorte de sous-patron et a une cardinalité minimale égale à un ; il est donc aussi une sorte de sous-patron nécessaire.

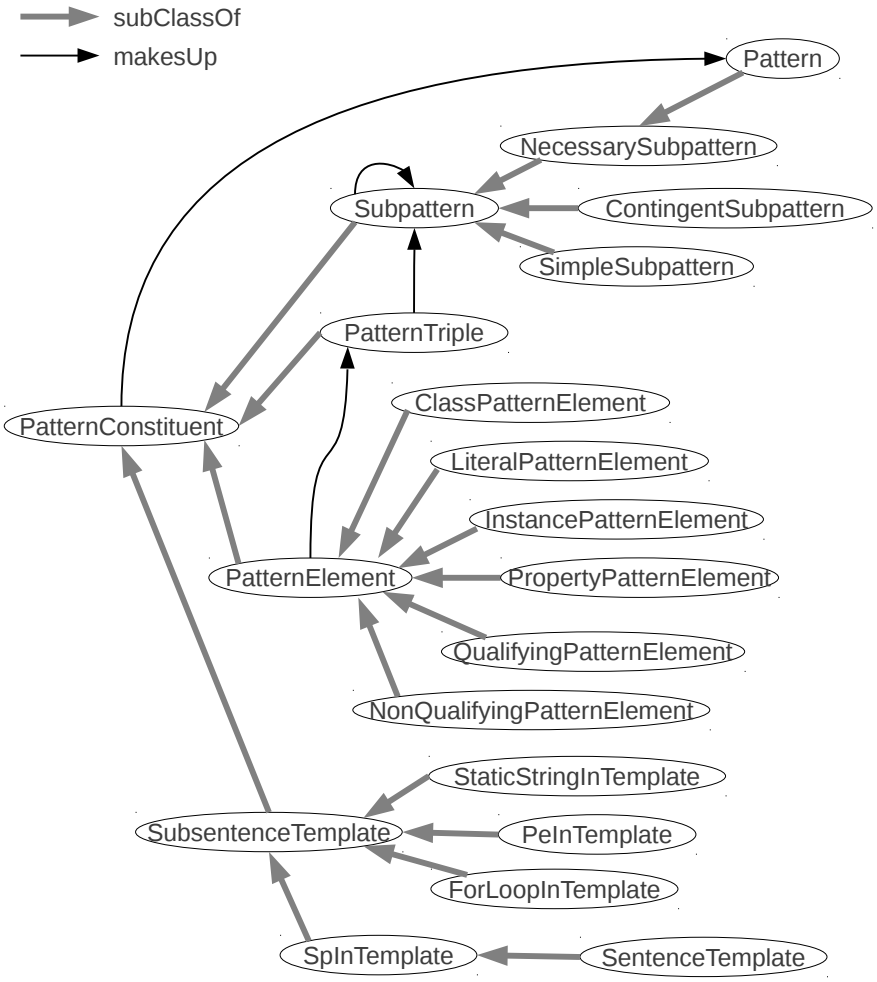


FIGURE 3.14 – Les principales classes permettant de décrire des patrons, les relations de taxonomie qui les hiérarchisent, ainsi que les relations de méronymie possibles entre les instances de ces classes.

Triplets et éléments de patrons Une instance de la classe `PatternTriple` est un triplet du graphe d'un patron. Un triplet est caractérisé par son sujet (relation `hasSubject`), sa propriété (relation `hasProperty`) et son objet (relation `hasObject`). Les valeurs de ces propriétés sont des instances de la classe `PatternElement`, qui sont les composants atomiques du patron. Un élément de patron fait référence à un type de littéral (`LiteralPatternElement`), une classe (`ClassPatternElement`), une instance (`InstancePatternElement`) ou une pro-

priété (`PropertyPatternElement`) de l'ontologie. Par exemple, une instance de `ClassPatternElement` impliquée dans un patron sera reliée par la relation `targets KBElement` à une classe de l'ontologie cible ; cette classe sera alors, dans ce contexte, considérée comme une instance. Cette méthode, appelée *punning*, est autorisée en OWL 2. Cependant, son utilisation est impossible à modéliser dans une ontologie OWL 2 car elle nécessiterait l'utilisation d'une partie du vocabulaire réservé (`owl:Class`, `owl:Property`, `owl:Literal`). `owl:Thing` est utilisé en lieu et place de chacune de ces entités proscrites, ce qui ne permet pas de différencier les trois sous-classes de `PatternElement` autrement qu'en les déclarant distinctes.

Phrase descriptive Les classes `StaticStringInTemplate`, `PeInTemplate`, `SpInTemplate`, `ForLoopInTemplate` et `SentenceTemplate`, toutes sous-classes de `SubsentenceTemplate`, permettent de décrire le modèle de phrase descriptive d'un patron. Chacune des instances de ces classes permet de décrire la façon dont doit être générée la partie de la phrase descriptive relative à un composant de patron. Le modèle de phrase descriptive d'un sous-patron (classe `SpInTemplate`) est relié par la propriété `sstTargetsPc` au sous-patron concerné et par la propriété `isMadeUpOfList` à une liste RDF dont les éléments décrivent de façon séquentielle l'extrait de phrase descriptive à générer. Chaque élément de cette liste est :

- soit une ressource représentant une chaîne de caractères statique : une instance de la classe `StaticStringInTemplate` reliée à la valeur de cette chaîne de caractères par la propriété de type de données `ssitHasValue`.
- soit une ressource faisant référence à un élement qualifiant du patron : une instance de la classe `PeInTemplate` reliée à l'élément qualifiant en question par la propriété d'objet `sstTargetsPc`.
- soit une ressource faisant référence à un autre sous-patron, compris dans le premier sous-patron : une instance de la classe `SpInTemplate` reliée au sous-patron inclus par la propriété `sstTargetsPc` et à une nouvelle liste par la propriété `isMadeUpOfList`.
- soit une ressource signalant la partie de la phrase descriptive à répéter en cas d'instanciation multiple d'un sous-patron répétable : une instance de la classe `ForLoopInTemplate` reliée elle aussi au sous-patron concerné par la propriété `sstTargetsPc` et à une nouvelle liste par la propriété `isMadeUpOfList`.

De la même façon qu'un patron est une sorte de sous-patron, le modèle de phrase descriptive d'un patron (classe `SentenceTemplate`) est une sorte de modèle de phrase descriptive de sous-patron. C'est pourquoi `SentenceTemplate` est une sous-classe de `SubsentenceTemplate` et le modèle de phrase descriptive d'un patron est relié par la propriété `sstTargetsPc` au patron concerné et par la propriété `isMadeUpOfList` à liste RDF construite comme décrit ci-dessus (à ceci près qu'elle ne peut contenir d'instance de `ForLoopInTemplate`, un patron ayant une cardi-

nalité minimale et maximale de un ; cette dernière contrainte n'est pas exprimée dans l'ontologie).

3.4.4 Mise en œuvre et validation

Une grammaire BNF permettant d'exprimer les patrons et l'ensemble de leurs composants a été définie ; elle est consultable dans l'annexe B. Lors de leur conception, les patrons sont représentés dans un fichier texte qui respecte cette grammaire. Ces patrons au format texte sont alors automatiquement traduits en RDF, en s'appuyant sur l'ontologie présentée plus haut. Cette traduction est fournie via un service web RESTfull [2] exploitable au moyen d'un simple formulaire HTML [3].

Cette méthode a été mise en place afin de court-circuiter l'étape de définition des patrons directement en RDF, rebutante et source d'erreurs. De plus, elle est pour nous une alternative viable et rapide à l'utilisation d'une interface destinée à l'édition et l'évolution de patrons, dont le développement, plus long et plus coûteux, apparaît dans la liste des travaux futurs.

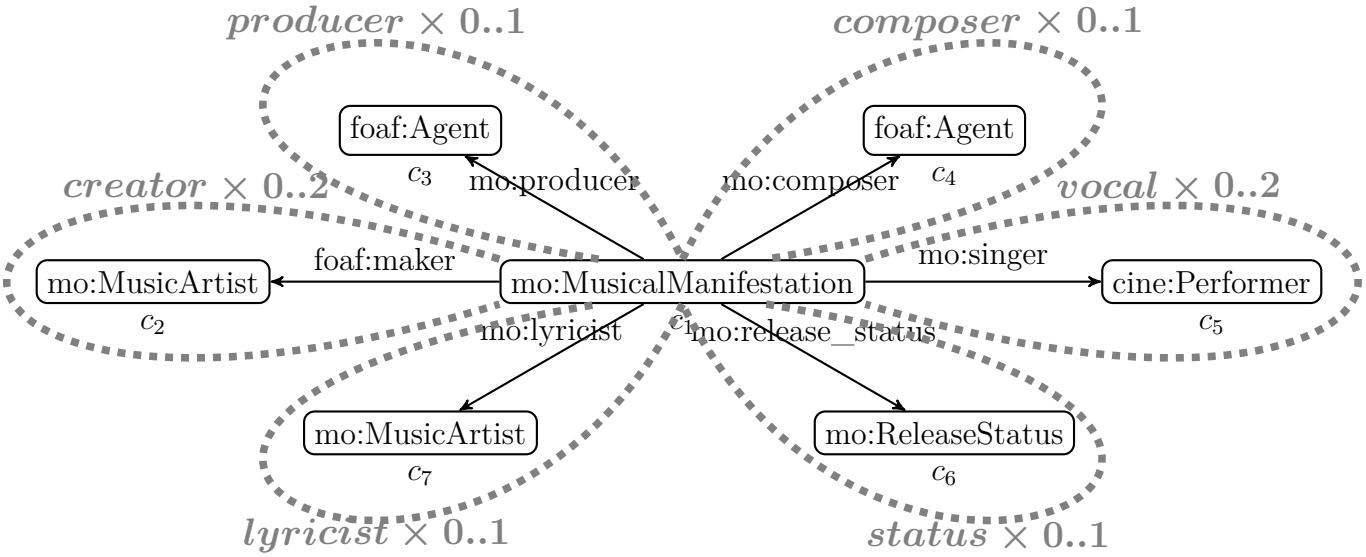
La validité des patrons générés par rapport au modèle défini dans l'ontologie peut être vérifiée à l'aide du validateur de contraintes d'intégrité de Pellet, présenté dans [Tao et al., 2010]. Cet outil, en contradiction avec les exigences portant sur le comportement d'un agent du Web sémantique, fait l'hypothèse du monde fermé (*closed world assumption*) et de l'identifiant unique (*unique name assumption*), et permet ainsi de considérer OWL comme un langage de validation pour RDF et donc de valider la conformité d'un ensemble de données RDF à une ontologie OWL.

Les axiomes exprimés dans l'ontologie peuvent ainsi être mis à profit pour vérifier la cohérence de ces patrons. Par exemple, dans l'ontologie *patterns*, un triplet de patron est défini comme une ressource ayant exactement un sujet, exactement une propriété et exactement un objet ; les contraintes sur le type de ces éléments montrées dans la figure 3.12 sont également exprimées.

3.5 Exemple détaillé

Dans cette section, nous décrivons un patron réel construit à l'occasion de notre participation à la compétition *QALD-3*, décrite dans le chapitre 7. Le patron présenté dans la figure 3.15 permet d'obtenir tout type d'information concernant une instance de "manifestation musicale" (classe `mo:MusicalManifestation`). D'après la spécification de l'ontologie *Music* [4], cette classe est liée à l'édition, la production

2. http://swip.univ-tlse2.fr/Patterns/PatternsWS/rest/patternsTextToRdf
3. http://swip.univ-tlse2.fr/Patterns/patternsTextToRdf.html
4. http://musicontology.com/



Modèle de phrase descriptive : $[c_1]$ ***if_status(*** *whose status is* $[c_6]$, ***)***
if_creator(*created by* ***for_creator(*** $[c_2]$, ***))*** ***if_producer(*** *produced by* $[c_3]$, ***)***
if_composer(*composed by* $[c_4]$, ***)*** ***if_vocal(*** *vocalised by* ***for_vocal(*** $[c_5]$, ***))***
if_lyricist(*whose lyricist is* $[c_7]$, ***)***

FIGURE 3.15 – Représentation graphique du patron *cd_info*

ou la publication d'une expression musicale. Un enregistrement (classe `mo:Record`) ou une piste (classe `mo:Track`) sont des exemples de sous-classes de manifestation musicale. Ce patron permet par exemple de demander qui est ou quels sont les artistes chantant sur une piste donnée, ou encore quels sont les enregistrements produits par une personne donnée et dont les paroles ont été écrites par un artiste donné.

Nous pouvons formuler quelques remarques liées à la physionomie de ce patron. Celui-ci est en effet en forme d'étoile, centré autour de la classe `mo:Musical Manifestation`. Une majorité des patrons que nous avons construits présentent cette configuration : un patron s'attache le plus souvent à permettre le requêtage des informations autour d'une classe importante de la base de connaissances. Dans certains patrons, des variations peuvent apparaître dans la longueur des branches de l'étoile, notamment lorsque la base de connaissances utilise des nœuds vides. Enfin, bien que le formalisme décrit plus haut le permette, il n'a jamais été néces-

saire lors des phases d'évaluation de construire des graphes de patrons présentant des cycles.

Nous constatons également que, dans l'exemple, la cardinalité minimale de chaque sous-patron est toujours de zéro. Cela se vérifie également dans la grande majorité des patrons construits jusqu'ici. La cardinalité maximale, quant à elle, n'est jamais supérieure à deux, afin de limiter le nombre d'associations possibles lors de la phase de formalisation de la requête pivot, présentée dans le chapitre 5. Au cours des évaluations, nous n'avons jamais rencontré de requêtes utilisateur nécessitant plus de deux occurrences d'un sous-patron.

Le listing suivant présente ce même patron, exprimé selon la syntaxe introduite plus haut :

Listing 8

```
prefixes
 foaf: "http://xmlns.com/foaf/0.1/"
   mo: "http://purl.org/ontology/mo/"
end prefixes

pattern cd_info
 query
  [1_mo:MusicalManifestation 2_foaf:maker          3_mo:MusicArtist; ]creator:0..2
  [1                         6_mo:producer         7_foaf:Agent;     ]producer:0..1
  [1                         8_mo:composer         9_foaf:Agent;     ]composer:0..1
  [1                        10_mo:singer          11_mo:Performer;   ]vocal:0..2
  [1                        12_mo:release_status  13_mo:ReleaseStatus;]status:0..1
  [1                        14_mo:lyricist        15_mo:MusicArtist; ]lyricist:0..1
 end query
 sentence
  -1- -status-["␣whose␣status␣is␣"-13-",␣"]
      -creator-["␣created␣by␣" -for-creator-[-3-",␣"]]
      -producer-["␣produced␣by␣"-7-",␣"]
      -composer-["␣composed␣by␣"-9-",␣"]
      -vocal-["␣vocalised␣by␣"-for-vocal-[-11-",␣"]]
      -lyricist-["␣whose␣lyricist␣is␣"-15-]
 end sentence
end pattern
```

La traduction de ce patron au format RDF par le service web décrit plus haut donne un graphe trop gros pour être représenté graphiquement. Nous considérons donc pour illustrer le résultat de cette traduction un patron plus simple, constitué d'un sous-ensemble du patron précédent. Le patron décrit dans le listing ci-dessous contient uniquement les sous-patrons *creator* et *producer* ; son modèle de phrase descriptive a été adapté en conséquence. La figure 3.16 montre le graphe RDF issu de la traduction de ce patron.

Listing 9

```
prefixes
 foaf: "http://xmlns.com/foaf/0.1/"
   mo: "http://purl.org/ontology/mo/"
end prefixes
```

```
pattern cd_info
 query
  [1_mo:MusicalManifestation 2_foaf:maker  3_mo:MusicArtist;] creator:0..2
  [1                         6_mo:producer 7_foaf:Agent;     ] producer:0..1
 end query
 sentence
  -1- -creator-["␣created␣by␣" -for-creator-[-3- ",␣" ] ]
      -producer-["␣produced␣by␣" -7- ",␣" ]
 end sentence
end pattern
```

3.6 Conclusion

Les patrons de requêtes que nous proposons sont modulaires dans la mesure où il est possible de définir des sous-patrons imbriqués, optionnels (qui peuvent ne pas apparaître dans la requête SPARQL générée) ou répétables (qui peuvent apparaître plusieurs fois dans la requête SPARQL générée). Ainsi, un seul patron couvre à lui seul un large spectre de besoins en information, et un nombre limité de patrons suffit à couvrir un domaine donné, comme nous le verrons dans le chapitre 7 qui présente les évaluations que nous avons réalisées.

Nous montrerons dans le chapitre 5 comment les patrons sont exploités pour interpréter une requête pivot et la traduire en SPARQL. Avant cela, nous expliquons dans le chapitre suivant ce qu'est une requête pivot et comment celle-ci est obtenue à partir d'une requête utilisateur en langue naturelle.

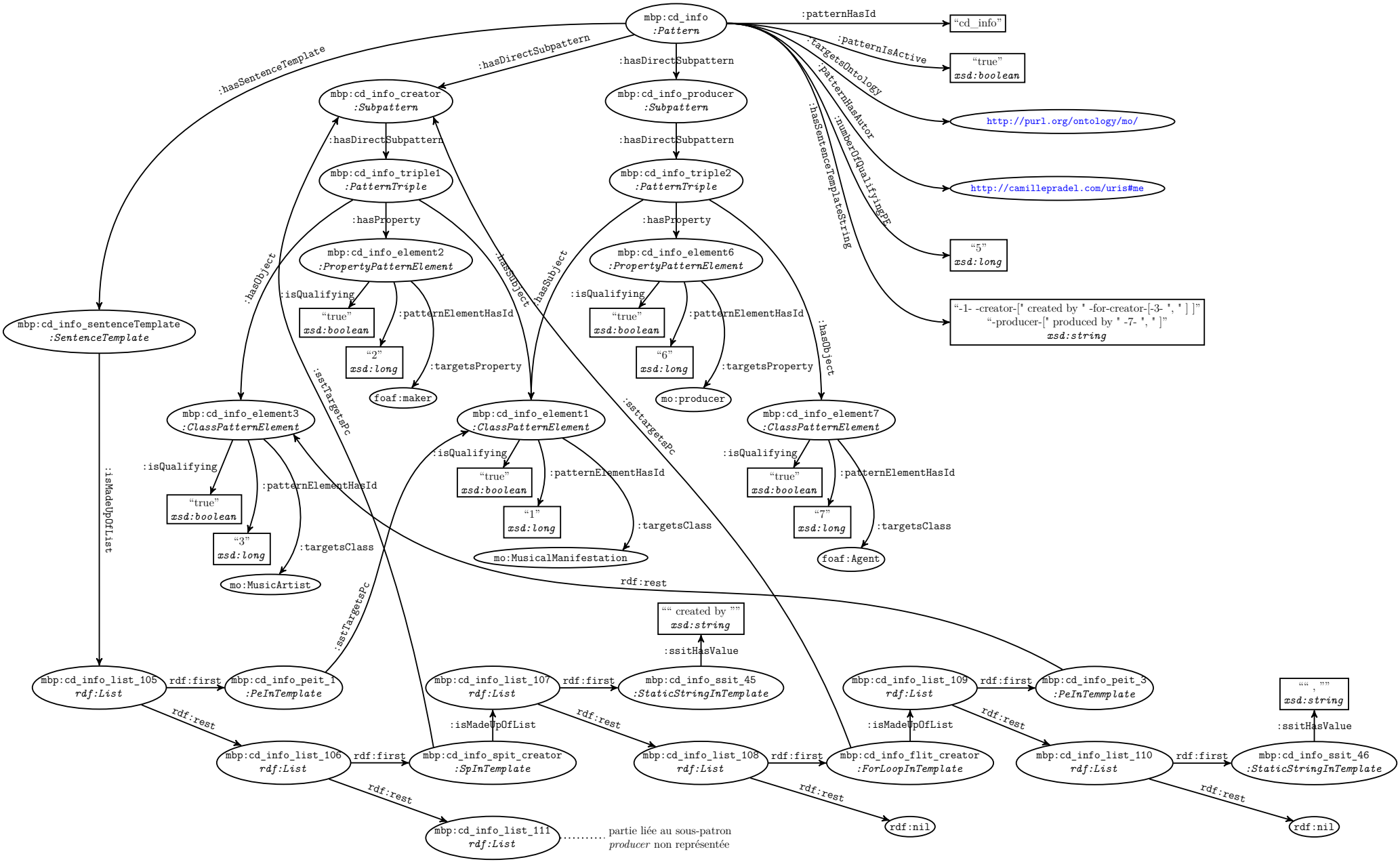


FIGURE 3.16 – Traduction RDF d'un sous-ensemble du patron *cd_info*

Chapitre 4

La requête pivot

Nous avons vu dans le chapitre 2 que certaines interfaces entre langue naturelle et bases de données intègrent un langage de représentation intermédiaire entre la langue naturelle et le langage formel ciblé. Nous avons également vu en 2.7.3 que certaines interfaces entre langue naturelle et base de connaissances ont repris cette idée mais sont par conséquent fortement dépendantes d'une analyse syntaxique de qualité. Notre approche s'inspire de cette méthode mais veut y incorporer plus de souplesse afin de pallier d'éventuels erreurs dans l'analyse syntaxique (qui sont assez courants en pratique). La structure intermédiaire que nous proposons est appelée *requête pivot*. Elle vise à stocker les résultats de la première étape d'interprétation. Brièvement, une requête pivot est composée des mots-clés de la requête originale en langue naturelle et permet d'exprimer les relations identifiées entre ces mots-clés ; elle peut être exprimée au moyen d'un langage que nous avons défini, appelé *langage pivot*.

Nous commençons ce chapitre par justifier notre choix en section 4.1. Puis nous donnons une définition formelle d'une requête pivot en section 4.2 et décrivons en section 4.3 le langage que nous avons conçu pour exprimer les requêtes pivot. Enfin, en section 4.4, nous détaillons le processus permettant de traduire une requête utilisateur de la langue naturelle vers le langage pivot, avant de conclure le chapitre en section 4.5.

La requête pivot a été introduite dans [Pradel et al., 2012a, 2011a], et le langage pivot dans [Pradel et al., 2011b]. La traduction de la requête LN vers le la requête pivot a été esquissée dans [Pradel, 2012], puis approfondie dans [Pradel et al., 2013b,a]

4.1 Justification

Nous utilisons la requête pivot pour deux raisons principales. Elle permet de faciliter la mise en œuvre du multilinguisme au moyen d'un format intermédiaire commun : un module spécifique de traduction de la langue naturelle vers le langage pivot doit être écrit pour chaque langue, mais l'étape de formalisation de la requête pivot reste la même. Pour le moment, nous avons adapté notre système à l'anglais et au français.

De plus, la requête pivot permet d'appréhender les relations entre les termes de la phrase qui ont pu être extraites par l'analyse en dépendance de la requête utilisateur. Ces informations sont à ce jour ignorées par les principales autres approches alors qu'elles peuvent être décisives dans l'interprétation qui est faite d'une requête LN. Par exemple, si l'on considère la requête "Dans quel film réalisé par Sofia Coppola joue Bill Murray ?", un système "naïf" se contentant d'extraire les mots-clés de la requête pourrait demander la liste des films réalisés par Bill Murray et dans lesquels joue Sofia Coppola (et ce, même s'il procède à une vérification du type des instances et des domaine et co-domaine des propriétés, Sofia Coppola et Bill Murray étant tous deux réalisateurs et acteurs). Il est dans ce cas indispensable d'identifier que "réalisé" établit un lien entre "film" et "Sofia Coppola", et "joue" entre "film" et "Bill Murray". Le langage pivot permet d'exprimer ce lien.

4.2 Définition formelle

Une requête pivot est constituée d'une ou plusieurs sous-requêtes ; une sous-requête est constituée d'un, deux ou trois éléments de requête. Dans cette section, nous définissons toutes ces notions de façon ascendante.

L'élément atomique d'une requête pivot est appelé un *élément de requête*. Il peut être soit un mot-clé (par exemple `actor`), soit un type de littéral (par exemple `date`), soit un littéral typé (par exemple *date<2013-04-05>*).

Définition 7 (Élement de requête) *$\mathcal{K}_q$ est un dictionnaire contenant l'ensemble de tous les mots-clés. La fonction val associe à chaque membre $k \in \mathcal{K}_q$ sa chaîne de caractères $val(k)$.*

ℓ_q est l'ensemble de tous les éléments de requête faisant référence à un type de littéral. La fonction val associe à chaque membre $l \in \ell_q$ le type de littéral de la base de connaissances correspondant $val(l) \in \ell$. La fonction text associe à chaque membre $l \in \ell_q$ sa représentation sous forme de chaine de caractères.

$\mathcal{L}_q$ est l'ensemble de tous les éléments de requête faisant référence à un littéral. La fonction val associe à chaque membre $L \in \mathcal{L}_q$ le littéral de la base de connaissances correspondant $val(L) \in \mathcal{L}$, et ainsi $type(val(L)) \in \ell$ identifie le type de littéral correspondant.

$\mathcal{E}_q = \mathcal{K}_q \cup \ell_q \cup \mathcal{L}_q$ *est l'ensemble de tous les éléments de requête.*

Les éléments de requête sont ordonnés selon des règles spécifiques par ensemble de un, deux ou trois pour former des sous-requêtes.

Définition 8 (Sous-requête) $\mathcal{S}_{q1} = \mathcal{E}_q$ *(respectivement* $\mathcal{S}_{q2} = \mathcal{K}_q \times \mathcal{E}_q$, $\mathcal{S}_{q3} = \mathcal{K}_q \times \mathcal{K}_q \times \mathcal{E}_q$*) est l'ensemble des sous-requêtes composées de un (respectivement deux, trois) élément(s) de requête appelées* sous-requêtes unaires *(respectivement* binaire*,* ternaire*).*

$\mathcal{S}_q = \mathcal{S}_{q1} \cup \mathcal{S}_{q2} \cup \mathcal{S}_{q3}$ *est l'ensemble de toutes les sous-requêtes.*

Définition 9 (Type de requête) $\mathcal{T}_q = \{list, count, dichotomous\}$ *est l'ensemble des types de requête.*

Notre approche prend en compte trois types de requêtes qui se définissent par le type du résultat attendu:

- les *requêtes liste* attendent comme résultat une liste de ressources respectant certaines conditions et se traduisent en SPARQL par une requête formée d'une clause `SELECT` ;
- les *requêtes de dénombrement* demandent le nombre de ressources respectant certaines conditions et se traduisent en SPARQL par une requête formée d'une clause `SELECT` et d'un agrégat `COUNT` comme attribut de projection ;
- les *requêtes dichotomiques* n'ont que deux réponses possible, Oui ou Non, et se traduisent en SPARQL par une requête utilisant l'opérateur `ASK`.

Enfin, une *requête pivot* est un ensemble fini de sous-requêtes associé à un ensemble d'*éléments objet* de la requête et un type de requête donné. Les éléments objet d'une requête pivot représentent l'information demandée par l'utilisateur et doivent évidemment apparaître dans cette requête ; tous les éléments de requête faisant référence à un type de littéral doivent être des éléments objet et aucun élément de requête faisant référence à une valeur de littéral ne peut être un élément objet ; une *requête liste* doit contenir au moins un élément objet.

Définition 10 (Requête pivot) $q = (S,F,T) \in 2^{\mathcal{S}_q} \times 2^{\mathcal{K}_q \cup \ell} \times \mathcal{T}_q$ *est une requête pivot si et seulement si:*

- $F \subseteq \bigcup_{s \in S} s$,
- $F \supseteq \bigcup_{s \in S} s \cap \ell_q$,
- $F \cap \mathcal{L}_q = \emptyset$,
- $|F| \geq 1$ *si* $T = list$.

$\mathcal{Q}_q$ *est l'ensemble de toutes les requêtes pivot.*

$\forall q = (S,F,T) \in \mathcal{Q}_q$, $focus(q) = F \in 2^{\mathcal{K}_q \cup \ell}$ dénote l'ensemble des éléments objet de la requête q, et $elem(q) = \bigcup_{s \in S} s$ dénote l'ensemble des éléments de requête présents dans q.

Exemple Dans la requête pivot issue de la requête LN "When was the album Abbey Road released ?" "Abbey Road", "album" et "release" apparaîtront en tant que mots-clés, et "When" fait référence au type de littéral `date`. De plus, le mot-clé "album" caractérise "Abbey Road", et "release" établit un lien entre "Abbey Road" et le littéral `date`. La représentation formelle de cette traduction sera donc la requête q telle que :

- $k_1, k_2, k_3 \in \mathcal{K}_q$, $val(k_1)$ ="Abbey Road", $val(k_2)$ ="album" et $val(k_3)$ ="release",
- $l_1 \in \ell$, correspond au type `xsd:date`,
- $q = \{ \{(k_1,k_2),(k_1,k_3,l_1)\}, \{l_1\}, list \}$

4.3 Syntaxe du langage pivot

Bien que la requête pivot soit un résultat intermédiaire, nous avons défini un langage permettant de formuler des requêtes pivot dans le but de faciliter la communication et l'interaction au cours du processus d'interprétation. La grammaire de ce langage est disponible en annexe C. Nous donnons ici simplement une présentation intuitive de ses fonctionnalités.

Le langage pivot que nous proposons peut être vu comme une extension d'un langage de mots-clés classique qui permet en sus de déclarer explicitement des relations entre les mots-clés. Le point d'interrogation optionnel devant un mot-clé ou entre les chevrons qui succèdent à un type de littéral signifie que ce mot-clé ou ce type de littéral est objet de la requête, c'est-à-dire que l'utilisateur veut obtenir des résultats spécifiques lui correspondant. Il peut y avoir plusieurs objets à une requête.

Comme indiqué dans la section 4.2, une requête pivot est composée d'une conjonction de sous-requêtes qui peuvent être :

- des sous-requêtes unaires qui ne contiennent qu'un seul élément, comme par exemple `?"actor"` qui demande la liste des acteurs présents dans la base de connaissances ;
- des sous-requêtes binaires qui qualifient un élément par l'intermédiaire d'un autre, comme par exemple la requête `?"actor": "married"` qui demande la liste des acteurs mariés ;
- les sous-requêtes ternaires qui qualifient à l'aide d'un mot-clé la relation entre deux éléments, comme par exemple la requête `?"actor": "married to"= "Penelope Cruz"` qui demande quel(s) acteur(s) est/a été/ont été marié(s)

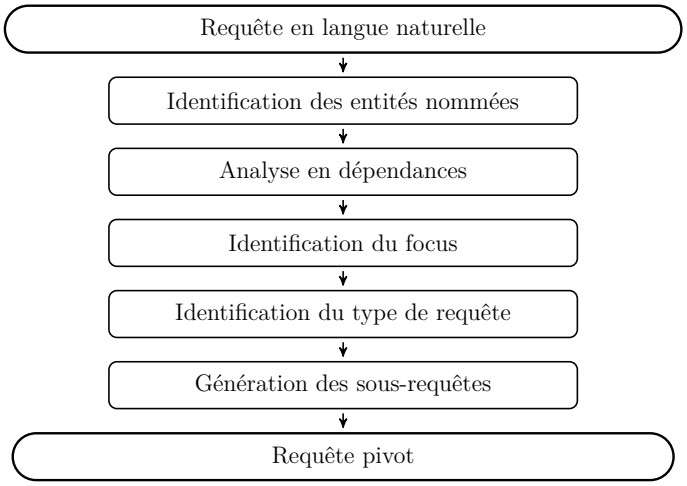


FIGURE 4.1 – Sous-étapes de l'interprétation de la requête LN.

à Penelope Cruz.

Le langage pivot permet également de "factoriser" certaines sous requêtes, en utilisant une syntaxe inspirée de Turtle, comme illustré dans l'exemple suivant.

Le type de requête est exprimé à la fin de cette requête, par un mot-clé réservé en majuscule : `COUNT` (resp. `ASK`) indique une requête de dénombrement (resp. dichotomique), l'absence de mot-clé indique une requête liste.

Exemple La requête pivot de l'exemple précédent, correspondant à la requête LN "When was the album Abbey Road released ?", peut se traduire en langage pivot `?"Abbey Road": "album". "Abbey Road": "release"= date<?>` (ou dans une forme plus condensée `?"Abbey Road": "album"; "release"= date<?>`).

4.4 Traduction d'une requête LN en une requête pivot

La traduction d'une requête langue naturelle en une requête pivot se fonde sur l'utilisation d'un analyseur de dépendances syntaxiques qui produit un arbre dans lequel les sommets correspondent aux termes de la phrase et les arêtes aux dépendances grammaticales entre les mots. Comme illustré sur la figure 4.1, la traduction se déroule en plusieurs étapes.

4.4.1 Identification des entités nommées

Avant d'analyser la requête, une première étape identifie dans la phrase les entités correspondant aux ressources de la base de connaissances. Ces entités sont alors considérées comme un tout et ne sont pas séparées par l'analyseur dans l'étape suivante. Par exemple, dans la phrase "Who are the actors of Underworld: Awakening ?" "Underworld: Awakening" sera considéré comme une entité nommée puisqu'il s'agit d'une étiquette d'une instance de film dans la base de connaissances. Cette étape est particulièrement cruciale lorsque l'on interroge des bases de connaissances contenant de longues étiquettes, comme des noms de groupes ou des titres de films composés de plusieurs mots, voire d'une phrase entière.

Ce traitement repose sur l'utilisation de gazetteers construits en cohérence avec la base de connaissances considérée.

Exemple 11 *La figure 4.2 montre le résultat obtenu après identification des entités nommées sur le requête NL "Who plays George Valentin in The Artist ?" Ce même exemple est poursuivi dans la suite pour illustrer les autres étapes du processus.*

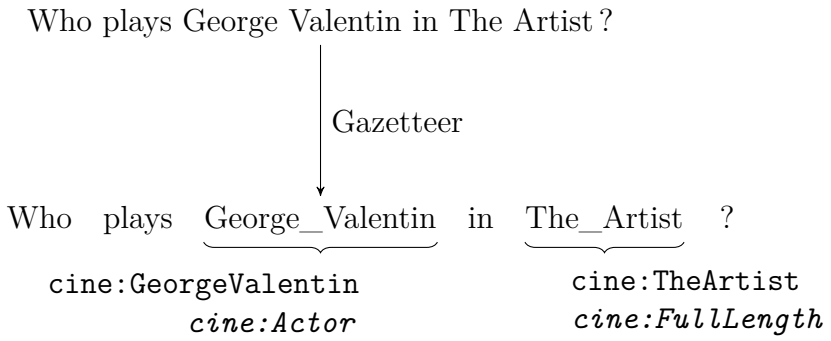


FIGURE 4.2 – Identification des entités nommées dans la phrase "Who plays George Valentin in The Artist ?"

4.4.2 Analyse en dépendances

Un analyseur syntaxique est ensuite utilisé pour générer un arbre de dépendances de la requête utilisateur tout en prenant en compte les entités nommées identifiées précédemment. Les étapes suivantes du processus sont mises en œuvre par application de règles sur cet arbre de dépendances.

Exemple 12 *La figure 4.3 montre le résultat d'une analyse en dépendances (prenant en compte les entités nommées préalablement identifiées) sur la requête NL "Who plays George_Valentin in The_Artist ?"*

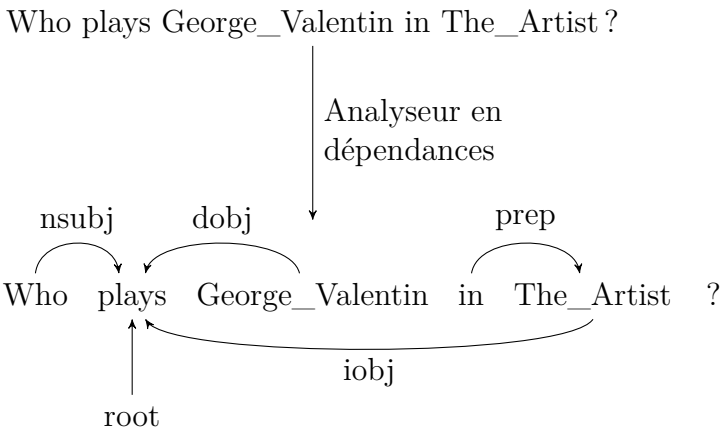


FIGURE 4.3 – Analyse en dépendances de la phrase “Who plays George_Valentin in The_Artist ?”

4.4.3 Identification du focus de la requête

Cette étape a pour but d'identifier le focus de la requête, c'est-à-dire les éléments de la requête pour lesquels l'utilisateur attend des résultats (les éléments correspondant aux variables de projection qui seront attachées à la clause SPARQL `SELECT`).

Le focus de la requête est cherché en parcourant l'arbre de dépendances à la recherche d'un mot interrogatif. Dans le cas où un mot interrogatif est trouvé et qu'une arête le relie à un nom, le lemme correspondant au nom est considéré comme étant le focus de la requête.

Exemple 13 *Pour la requête “Which films did Ted_Hope produce ?”, le focus de la requête sera “film”.*

Si une telle arête n'existe pas, nous positionnons le focus comme étant le type de la réponse attendue par le mot interrogatif (“who” identifie une personne, “when” une date, “where” un lieu).

Exemple 14 *Pour la requête “When did Charles_Chaplin die ?”, le focus sera “date”. Et pour la requête des exemples prédédents, “Who plays George_Valentin in The_Artist ?”, le focus sera “person”.*

Si aucun mot interrogatif n'est trouvé dans le graphe, nous recherchons des expressions spécifiques introduisant des requêtes listes, qui sont des requêtes qui attendent des listes de ressources remplissant certaines conditions ; de telles requêtes commencent par “List”, “List all”, “I am looking for”, “What are”, “Give me”... Dans ce cas, le focus de la requête sera le lemme du nom identifié comme étant le focus direct du verbe dans la clause dans laquelle l'expression a été trouvée.

Exemple 15 *Dans la requête "List all movies directed by Clint_Eastwood", le focus sera "movie".*

4.4.4 Identification du type de requête

Si aucun focus n'a été identifié, nous considérons la requête comme dichotomique ce qui signifie qu'elle n'admet que deux réponses : "oui" ou "non".

Exemple 16 *C'est le cas de la requête "Was Jean Dujardin involved in the film The Artist ?" pour laquelle aucune des règles énoncées plus haut n'a permis l'identification d'un focus.*

Si au moins un focus a été identifié, nous cherchons dans la phrase une expression spécifique introduisant une requête de dénombrement, comme par exemple "How many" ou "Give me the number of." Si une telle expression est repérée, la requête est alors considérée comme une requête de dénombrement.

Exemple 17 *Ainsi, la requête "How many movies were directed by Clint_Eastwood ?" est considérée comme une requête de dénombrement.*

Sinon, il s'agit d'une requête liste.

4.4.5 Génération des sous-requêtes

Le graphe de dépendances est une fois de plus parcouru afin d'identifier les sous-requêtes potentielles de la requête pivot à partir des différentes propositions de la phrase. Une liste contenant les pronoms interrogatifs et les mots composant les expressions introduisant les requêtes listes est utilisée. Les sommets pour lesquels le lemme apparaît dans la liste ou pour lesquels la catégorie grammaticale indique qu'il s'agit d'un déterminant ou d'une préposition sont ignorés.

Les sommets restants sont parcourus en appliquant les règles suivantes :

1. si trois sommets correspondant à un triplet sujet-verbe-complément sont identifiés, une sous-requête ternaire est construite à partir du lemme de chaque sommet, selon le schéma `sujet: verbe= complément`.

   Exemple 18 *L'application de cette règle à la requête NL "Did Michel_Hazanavicius direct The_Artist ?" donne la requête pivot*
   `"Michel Hazanavicius": "direct"= "The Artist". ASK.`

2. si l'un de ces sommets est un mot interrogatif, il est remplacé par le mot identifiant le type de la réponse attendue selon les mêmes règles que celles utilisées lors de l'identification du focus ("who" devient `"person"`, "when" devient `"date"`, "where" devient `"place"`).

Exemple 19 *Pour la requête "Who directed The_Artist ?", la requête ternaire correspondante est* `?"person": "direct"= "The Artist"`.

3. dans le cas où le sommet correspondant au sujet ou au complément est un pronom relatif, le mot-clé utilisé dans la requête pivot sera le lemme du sommet référencé par le pronom.

   Exemple 20 *Dans la requête "Who played in a film in which Jean Dujardin also played ?", la sous-requête correspondant à la seconde proposition est* `"film": "play"= "Jean Dujardin"`, *la sous-requête générée pour la première est* `?"person": "play"= "film"`. *Remarquons que l'utilisation du même mot-clé* `"film"` *dans les deux sous-requêtes indique que le même film doit être considéré dans chacune d'entre-elles.*

4. si deux sommets font partie d'un syntagme nominal identifié et qu'une arête les reliant exprime une dépendance de tête ou d'expansion, une sous-requête binaire est construite à partir du lemme de chacun des sommets.

   Exemple 21 *Pour la requête "Give me all the films of Jean Dujardin", la sous-requête générée est* `?"film": "Jean Dujardin"`. *Et dans la requête "Who directed the movie The_Artist ?" le syntagme nominal "the movie The_Artist" donnera l'extrait de requête pivot* `"The Artist": "movie"`.

Chaque règle est testée pour chaque sommet. Ainsi, d'une requête LN peuvent être générées plusieurs sous-requêtes pivot, et un même élément peut apparaître dans plusieurs sous-requêtes. Par exemple pour la question "Which actor played in the movie The_Artist ?", les deux sous-requêtes `"The Artist": "movie"` et `"actor": "play"= "The Artist"` sont générées.

Exemple 22 *Si l'on considère l'arbre de dépendances de la figure 4.3, on identifie deux triplets sujet-verbe-complément : ("Who", "plays", "George_Valentin") et ("Who", "plays", "The_Artist"). En exploitant les résultats des étapes précédentes (identification du type et du focus de la requête) et en applicant les règles de transformation 1 et 2 ("Who" devient* `person`*), on obtient la requête pivot suivante :* `"person": "play"= "George Valentin", "The Artist"`.

Enfin, pour chaque mot n'appartenant pas à la liste des mots ignorés et n'ayant été impliqué dans aucune application de règle, une sous-requête unaire contenant ce mot est construite, ceci afin de ne pas perdre le mot dans la traduction en langage pivot même s'il n'a pas été possible de le rattacher à d'autres mots-clés.

Ces règles peuvent paraître simples, mais nous avons observé que la structure des requêtes exprimées par les utilisateurs finals est généralement basique, et dans le cas de requêtes complexes, le résultat de l'analyse en dépendances n'est pas assez fiable pour appliquer des règles plus élaborées. Nous avons développé d'autres

règles plus complexes exploitant finement l'arbre de dépendances syntaxiques, mais les premières expériences ont vite montré que leur application rendait l'approche moins robuste aux erreurs de l'analyseur en dépendances.

4.5 Conclusion

La requête pivot est obtenue par application de règles sur l'arbre de dépendances de la requête originale en langue naturelle. Nous nous limitons à l'application de règles simples afin d'extraire les principales relations entre les mots-clés de la requête. Ces relations sont ensuite prises en compte dans la deuxième grande étape du processus d'interprétation qui consiste en la formalisation de la requête pivot à l'aide de patrons de requêtes. Cette étape est décrite dans le chapitre suivant.

Chapitre 5

Formalisation de la requête pivot en requête graphe

Nous présentons ici la traduction en requête graphe qui est faite d'une requête pivot $q \in \mathcal{Q}_q$ par l'intermédiaire d'un patron de requêtes $p \in \mathcal{P}_p$. La figure 5.1 illustre toutes les étapes de ce processus.

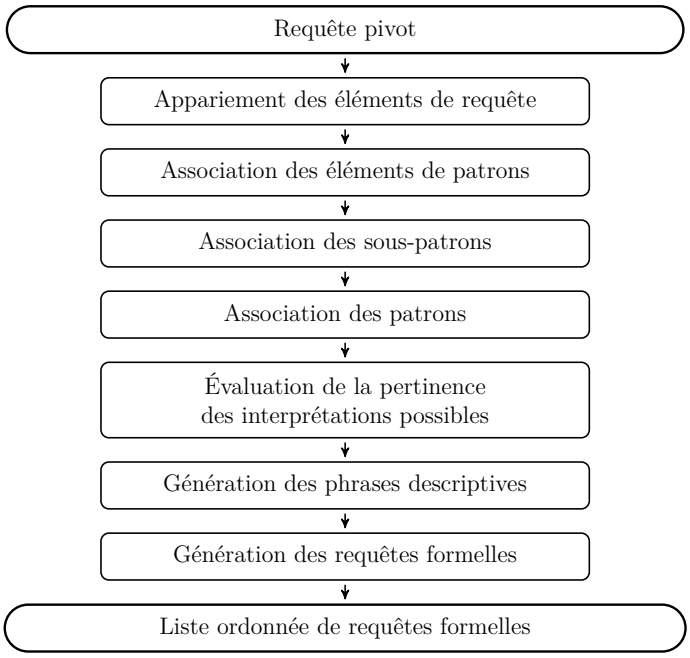


FIGURE 5.1 – Étapes de l'interprétation de la requête pivot.

Dans un premier temps, des liens sont établis entre les éléments de la requête utilisateur et les éléments de la base de connaissances (instances, classes, propriétés et types de données).

Les étapes suivantes consistent à associer chaque patron de requêtes à la requête pivot à traiter, de façon conforme aux liens préalablement établis. Cette opération est réalisée de façon ascendante : tout d'abord sont générées des associations entre éléments de patron et éléments de requête, puis entre sous-patron et requête pivot, en commençant par les sous-patrons basiques (ceux qui ne contiennent aucun autre sous-patron), et en élargissant progressivement aux sous-patrons englobants, jusqu'à obtenir les associations du patron lui-même. Chaque *association de patron* obtenue est une interprétation potentielle de la requête pivot (et par transitivité de la requête utilisateur).

Dans le but de présenter à l'utilisateur les interprétations les plus pertinentes en priorité, une *note de pertinence* est calculée pour chaque association de patron. La note de pertinence reflète la fiabilité supposée de l'association de patron concernée, et donc de l'interprétation qu'elle incarne. Les associations sont ensuite ordonnées selon cette note avant d'être soumises à l'utilisateur.

Enfin, la génération pour chaque association de patron de la *phrase descriptive* (la phrase présentée à l'utilisateur représentant le sens la requête) et de la *requête SPARQL* (la requête exécutée sur la base de connaissances si la phrase descriptive associée est sélectionnée par l'utilisateur) correspondantes est triviale, grâce au modèle de phrase descriptive et au graphe de chaque patron.

Les sections 5.1 à 5.7 détaillent chacune de ces sous-étapes : l'appariement des éléments de requête, l'association des éléments de patrons, l'association des sous-patrons, l'association des patrons, l'évaluation de la pertinence des interprétations possibles, et la génération des phrases explicatives et des requêtes formelles, respectivement. La section 5.8 commente la complexité algorithmique de notre approche et la section 5.9 conclut le chapitre.

Ces sections exploitent les définitions et notations d'une base de connaissances (introduites en 1.4.6), d'un patron (introduites en 3.3) et d'une requête pivot (introduites en 4.2).

L'étape de formalisation de la requête pivot étant centrale dans notre approche, elle est abordée dans la grande majorité des publications liées à ce travail, et décrite de la façon la plus détaillée dans [Pradel et al., 2011b].

5.1 Appariement des éléments de le requête aux éléments de la base de connaissances

Cette première étape consiste à apparier chacun des éléments de la requête utilisateur avec des entités de la base de connaissances (classes, propriétés, instances et types de littéraux). Un élément de requête peut être apparié à plusieurs entités. Nous associons une valeur de confiance à chaque appariement, correspondant à

la qualité supposée de l'appariement, calculée différemment selon la nature des éléments appariés (par mesure de similarité entre chaînes de caractères pour les classes, propriétés et instances, et par compatibilité de types pour les littéraux).

Définition 11 (Similarité entre deux chaînes de caractères) *Nous considérons une fonction* $sim : \mathcal{K}_q^2 \rightarrow [0;1]$ *définissant une mesure de similarité entre chaînes de caractères, telle que* $\forall k \in \mathcal{K}_q, sim(k,k) = 1$.

Définition 12 (Similarité entre une ressource et une chaîne de caractères) *La fonction* sim^{KB} *retourne, dans le contexte d'une base de connaissances* KB, *pour une ressource* $r \in \mathcal{E}$ *et une chaîne de caractères* $k \in \mathcal{K}$, *un couple contenant la mesure de similarité la plus élevée entre toutes les étiquettes de* r *et* k, *ainsi que l'étiquette concernée :*

$$\begin{array}{rll} sim^{KB} : \mathcal{E} \times \mathcal{K}_q & \rightarrow [0;1] \times \mathcal{K}_q & \\ (r,\, k) & \mapsto (s,\, l) \text{ tel que } s & = sim(l,\, k) \\ & & = Max_{label(r,\, l') \in KB} sim(l',\, k) \end{array}$$

Définition 13 (Appariement d'un élément de requête à la base de connaissances) *La fonction* $match^{KB}$ *appariant un élément de requête* $e_q \in \mathcal{E}_q$ *à une base de connaissances* KB, *avec un seuil* $\sigma \in]0;1]$ *se définit ainsi :*

$$\begin{array}{rll} match^{KB} : \mathcal{E}_q \times]0;1] & \rightarrow 2^{\mathcal{E} \times]0;1] \times (\mathcal{K}_q \cup \ell \cup \mathcal{L})} & \\ (e_q, \sigma) & \mapsto \begin{cases} \{(r,\, s,\, l) | (s,\, l) = sim^{KB}(r,\, val(e_q)) \text{ et } s \geq \sigma\} & \text{si } e_q \in \mathcal{K}_q \\ \{(val(e_q),\, 1,\, val(e_q))\} & \text{si } e_q \in \mathcal{L}_q \cup \ell \end{cases} \end{array}$$

Exemple 23 *La requête en langue naturelle "Which actors play in the movie The Artist ?" peut se traduire par la requête pivot suivante :* `?actor: play= The_Artist. The_Artist: movie.` *La représentation de cette requête utilisant les notations introduites dans le chapitre 4 est la requête* q *telle que :*

- $actor, play, theArtist, movie \in \mathcal{K}_q$, $val(actor) =$"actor", $val(play) =$"play", $val(theArtist) =$"The Artist" et $val(movie) =$"movie",
- $q = \{\ \{(actor, play, theArtist), (theArtist, movie)\}, \{actor\}, list\ \}$

Si l'on considère la distance de Jaro [Jaro, 1989] comme mesure de similarité entre deux chaînes de caractères, une base de connaissances Cine sur le domaine du cinéma (dont un sous ensemble a été présenté en 1.4) et une valeur de seuil de

0.5*, les résultats du processus d'appariement de chacun des mots-clés de q sont les suivants :*

$$match_{0.5}^{Cine}(actor) = \left\{ \begin{array}{c} (\texttt{cine:Actor}, 1, \texttt{"actor"}), \\ (\texttt{cine:LesActeurs}, 0.53, \texttt{"Les Acteurs"}) \end{array} \right\}$$

$$match_{0.5}^{Cine}(play) = \left\{ (\texttt{cine:playsIn}, 0.83, \texttt{"plays in"}) \right\}$$

$$match_{0.5}^{Cine}(theArtist) = \left\{ \begin{array}{c} (\texttt{cine:TheArtist}, 1, \texttt{"The Artist"}), \\ (\texttt{cine:Actor}, 0.78, \texttt{"artist"}) \end{array} \right\}$$

$$match_{0.5}^{Cine}(movie) = \left\{ \begin{array}{c} (\texttt{cine:Movie}, 1, \texttt{"movie"}), \\ (\texttt{cine:Movie43}, 0.875, \texttt{"Movie 43"}) \end{array} \right\}$$

5.2 Association des éléments de patron

Cette étape consiste à déterminer pour chaque sommet qualifiant de patron toutes les associations possibles à des éléments de requêtes – appelées associations d'éléments – et leurs valeurs de confiance respectives. Ces associations seront ensuite combinées pour créer des associations pour chacun des sous-patrons de p et enfin pour p.

Lorsque l'on mène cette étape à bien, plusieurs cas peuvent se présenter :

1. l'élement qualifiant du patron fait référence à une classe : il peut y avoir une association entre cet élément et tout mot-clé de la requête si ce mot-clé a été précédemment apparié à une ressource de la base de connaissances qui est soit une instance de la classe référencée, soit une sous-classe de cette classe (ce cas inclut celui où les deux classes sont équivalentes).
2. l'élement qualifiant du patron fait référence à une propriété (d'objet ou de type de données) : il peut y avoir une association entre cet élément et tout mot-clé de la requête si ce mot-clé a été précédemment apparié à une propriété de la base de connaissances qui est une sous-propriété de la propriété référencée (ce cas inclut celui où les deux propriétés sont équivalentes).
3. l'élement qualifiant du patron fait référence à un type de données : il peut y avoir une association entre cet élément et tout littéral de la requête qui a été apparié au type de données référencé.

Dans tous les cas, une valeur de confiance s est assignée à chaque association. Elle est égale à celle de l'appariement impliqué dans l'association.

Définition 14 (Association d'un élément de patron à un élément de requête) *La fonction $mapEE^{KB}$ associant un élément de patron $e_p \in \mathcal{E}_{qual}$ à un élément de*

requête $e_q \in \mathcal{E}_q$*, avec un seuil* $\sigma \in]0;1]$ *dans le contexte d'une base de connaissances* KB *se définit ainsi :*

$$mapEE^{KB} : \mathcal{E}_{qual} \times \mathcal{E}_q \times]0;1] \rightarrow 2^{\mathcal{E}_{qual} \times \mathcal{E}_q \times \mathcal{E} \times]0;1] \times \mathcal{K}}$$

$$(e_p,e_q,\sigma) \mapsto \begin{cases} \left\{ (e_p,e_q,r,s,l) \;\middle|\; \begin{array}{l}(r,s,l) \in match_\sigma^{KB}(e_q) \\ et\ instanceOf(r,val(e_p)) \in KB\end{array} \right\} \\ \qquad si\ e_p \in \mathcal{C}_P,\ e_q \in \mathcal{K}_q,\ et\ r \in \mathcal{I} \\ \left\{ (e_p,e_q,r,s,\text{'some '} + l) \;\middle|\; \begin{array}{l}(r,s,l) \in match_\sigma^{KB}(e_q) \\ et\ subclassOf(r,val(e_p)) \in KB\end{array} \right\} \\ \qquad si\ e_p \in \mathcal{C}_P,\ e_q \in \mathcal{K}_q,\ et\ r \in \mathcal{C} \\ \left\{ (e_p,e_q,r'',s,l'') \;\middle|\; \begin{array}{l}(r,s,l) \in match_\sigma^{KB}(e_q), \\ subpropertyOf(r,r') \in KB, \\ \left(\begin{array}{c} r' = r'' \\ ou \\ inverseOf(r',r'') \in KB \end{array}\right), \\ subpropertyOf(r'',val(e_p)) \in KB, \\ l'' \in label(r'')\ et\ l'' = l\ si\ r'' = r\end{array} \right\} \\ \qquad si\ e_p \in \mathcal{P}_{oP} \cup \mathcal{P}_{dP}\ et\ e_q \in \mathcal{K}_q, \\ \left\{ \big(e_p,e_q,r,s,text(e_q)\big) \;\middle|\; \begin{array}{l}(r,s) \in match_\sigma^{KB}(e_q) \\ et\ type(r) = val(e_p)\end{array} \right\} \\ \qquad si\ e_p \in \ell_P\ et\ e_q \in \mathcal{L}_q, \\ \emptyset\ sinon \end{cases}$$

Le résultat de cette fonction est un ensemble d'associations d'élément. Une association d'élément associe un élément qualifiant de patron à un élément de requête en fonction des appariements préalablement établis. Quand $(e_p,e_q,r,s,l) \in mapEE_\sigma^{KB}(e_p,e_q)$, on dit que l'élément de patron e_p peut être associé à l'élément de requête e_q au travers de la ressource r et avec la note de confiance s ; l est la représentation textuelle de l'association qui sera utilisée lors de la génération de la phrase descriptive.

Exemple 24 *On considère le patron de requêtes, introduit en 3.2, dont on ramène la cardinalité maximale des sous-patrons répétables qu'il contient à deux, afin de simplifier les calculs qui vont suivre. La représentation graphique de ce patron est donnée dans la figure 5.2 et sa représentation formelle est donnée ci-dessous :*

– *le patron :*

$movieDesc = (\{actor,director\},\{c_1\},1,1) \in \mathcal{P}_P$

– *les sous-patrons :*

$$\begin{aligned} actor &= (\{b_1\},\{r_1,c_2\},0,2) \in \mathcal{S}_p \\ dirNat &= (\{b_3\},\{r_3,c_4\},0,1) \in \mathcal{S}_p \\ director &= (\{b_2,dirNat\},\{r_2,c_3\},0,2) \in \mathcal{S}_p \end{aligned}$$

– *les sous-patrons basiques :*

$$\begin{aligned} b_1 &= \{t_1\} \in \mathcal{B}_p \\ b_2 &= \{t_2\} \in \mathcal{B}_p \\ b_3 &= \{t_3\} \in \mathcal{B}_p \end{aligned}$$

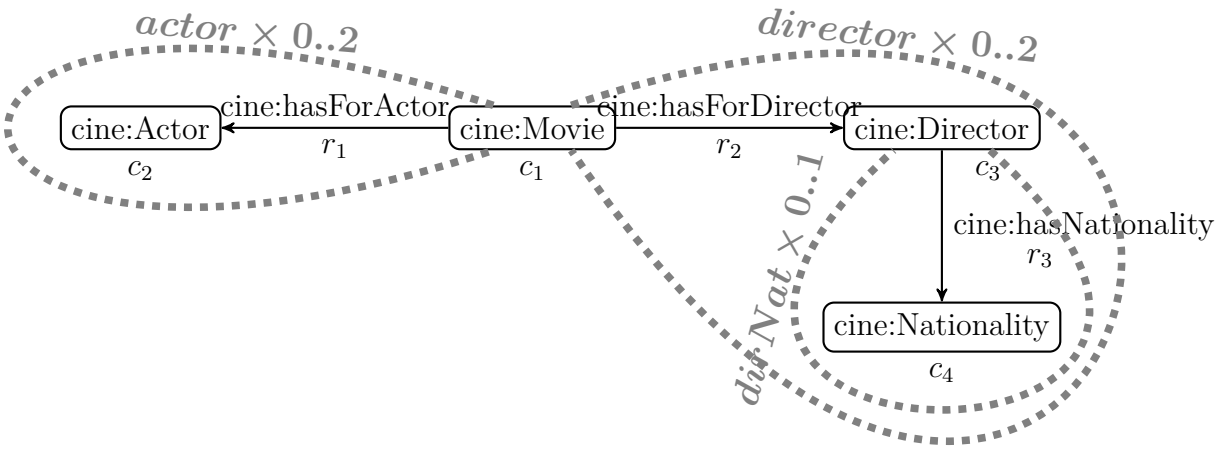
– *les triplets de patrons :*

$$\begin{aligned} t_1 &= (c_1,r_1,c_2) \in \mathcal{T}_P \\ t_2 &= (c_1,r_2,c_3) \in \mathcal{T}_P \\ t_3 &= (c_3,r_3,c_4) \in \mathcal{T}_P \end{aligned}$$

– *les éléments de patrons :*

$$\begin{aligned} c_1,c_2,c_3,c_4 \in \mathcal{C}_P,\ avec\ targ(c_1) &= \texttt{cine:Movie}, \\ targ(c_2) &= \texttt{cine:Actor}, \\ targ(c_3) &= \texttt{cine:Director} \\ et\ targ(c_4) &= \texttt{cine:Nationality} \\ r_1,r_2,r_3 \in \mathcal{P}_{o_P},\ avec\ targ(r_1) &= \texttt{cine:Movie}, \\ targ(r_2) &= \texttt{cine:Actor} \\ et\ targ(r_3) &= \texttt{cine:Nationality} \end{aligned}$$

Les associations d'éléments obtenues à partir des appariements de l'exemple précédent sont les suivantes :



Modèle de phrase descriptive : $[c_1]$ ***if_actor(*** *stars* ***for_actor(*** $[c_2]$***),*** ***if_director(*** *was directed by* ***for_director(*** $[c_3]$ ***if_dirNat(*** *which is* $[c_4]$ ***))*** ***)***

FIGURE 5.2 – Patron de requêtes utilisé dans nos exemples

$$mapEE^{KB}_{0.5}(c_1,\ actor) = \Big\{(c_1,\ actor,\ \texttt{cine:LesActeurs},\ 0.53,\ \texttt{"Les Acteurs"})\Big\}$$
$$car \quad (\texttt{cine:LesActeurs},\ 0.53,\ \texttt{"Les Acteurs"}) \in match^{KB}_{0.5}(actor)$$
$$et\ instanceOf(\texttt{cine:LesActeurs},\ val(c_1)) \in KB$$

$$mapEE^{KB}_{0.5}(c_1,\ theArtist) = \Big\{(c_1,\ theArtist,\ \texttt{cine:TheArtist},\ 1,\ \texttt{"The Artist"})\Big\}$$
$$car \quad (\texttt{cine:TheArtist},\ 1,\ \texttt{"The Artist"}) \in match^{KB}_{0.5}(theArtist)$$
$$et\ instanceOf(\texttt{cine:TheArtist},\ val(c_1)) \in KB$$

$$mapEE^{KB}_{0.5}(c_1,\ movie) = \left\{ \begin{array}{c} (c_1,\ movie,\ \texttt{cine:Movie},\ 1,\ \texttt{"movie"}), \\ (c_1,\ movie,\ \texttt{cine:Movie43},\ 0.875,\ \texttt{"Movie 43"}) \end{array} \right\}$$
$$car \quad (\texttt{cine:Movie},\ 1,\ \texttt{"movie"}) \in match^{KB}_{0.5}(movie),$$
$$subclassOf(\texttt{cine:Movie},\ val(c_1)) \in KB,$$
$$(\texttt{cine:Movie43},\ 0.875,\ \texttt{"Movie 43"}) \in match^{KB}_{0.5}(movie)$$
$$et\ instanceOf(\texttt{cine:Movie43},\ val(c_1)) \in KB$$

$$mapEE^{KB}_{0.5}(c_2,\ actor) = \Big\{(c_2,\ actor,\ \texttt{cine:Actor},\ 1,\ \texttt{"actor"})\Big\}$$
$$car \quad (\texttt{cine:Actor},\ 1,\ \texttt{"actor"}) \in match^{KB}_{0.5}(actor)$$
$$et\ subclassOf(\texttt{cine:Actor},\ val(c_2)) \in KB$$

$$mapEE^{KB}_{0.5}(c_2,\ theArtist) = \Big\{(c_2,\ theArtist,\ \texttt{cine:Actor},\ 0.78,\ \texttt{"artist"})\Big\}$$
$$car \quad (\texttt{cine:Actor},\ 0.78,\ \texttt{"artist"}) \in match^{KB}_{0.5}(actor)$$
$$et\ subclassOf(\texttt{cine:Actor},\ val(c_2)) \in KB$$

$$mapEE^{KB}_{0.5}(r_1,play) = \Big\{(r_1,\ play,\ \texttt{cine:hasForActor},\ 0.83,\ \texttt{"has for actor"})\Big\}$$
$$car \quad (\texttt{cine:playsIn},\ 0.83,\ \texttt{"plays in"}) \in match^{KB}_{0.5}(play)$$
$$et\ inverseOf(\texttt{cine:playsIn},\ val(r_1)) \in KB$$

Pour toutes les autres paires $(e_p,e_q) \in elem(movieDesc) \times elem(q)$*, aucune association ne peut être établie :* $mapEE^{KB}_{0.5}(e_p,e_q) = \emptyset$

Dans les exemples qui suivent, nous utiliserons ces notations :

$$m_1 = (c_1,\ actor,\ \texttt{cine:LesActeurs},\ 0.53,\ \texttt{"Les Acteurs"})$$
$$m_2 = (c_1,\ theArtist,\ \texttt{cine:TheArtist},\ 1,\ \texttt{"The Artist"})$$
$$m_3 = (c_1,\ movie,\ \texttt{cine:Movie},\ 1,\ \texttt{"movie"})$$
$$m_4 = (c_1,\ movie,\ \texttt{cine:Movie43},\ 0.875,\ \texttt{"Movie 43"})$$
$$m_5 = (c_2,\ actor,\ \texttt{cine:Actor},\ 1,\ \texttt{"actor"})$$
$$m_6 = (c_2,\ theArtist,\ \texttt{cine:Actor},\ 0.78,\ \texttt{"artist"})$$
$$m_7 = (r_1,\ play,\ \texttt{cine:hasForActor},\ 0.83,\ \texttt{"has for actor"})$$

Un dernier type d'association d'élément peut être produit sur la base des précédents. Ces associations, appelées *associations d'élément instance-classe* sont issues de l'observation de la manière dont les utilisateurs, quand ils s'expriment en langue naturelle, spécifient souvent un terme faisant référence à une instance par un autre terme faisant référence à une classe à laquelle appartient cette instance. On trouve de nombreux exemples dans les requêtes en langue naturelle de la compétition QALD-3 [1] : "the band Dover", "the album In Utero", "the song Hardcore Kids"...

D'après la quatrième des règles présentées en 4.4.5 permettant de générer les sous-requêtes de la requête pivot, ce type de formulation est traduite en une sous-requête binaire, composée du mot-clé faisant référence à l'instance qualifié par le mot-clé faisant référence à la classe ; par exemple, "the album In Utero" devient `"In Utero": "album"`. Nous utilisons un type particulier d'association d'élément pour prendre en compte ce cas ; une association de ce type, appelée *association d'élément instance-classe*, associe un élément de patron à deux mots-clés. Sa valeur de confiance est égale à la somme des valeurs de confiance des deux appariements pris en compte.

1. http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=task1&q=3

Définition 15 (Associations d'élément instance-classe) *La fonction* $mapIC^{KB}$ *donnant l'ensemble des associations d'élément instance-classe d'un élément de patron* $e_p \in \mathcal{E}_{qual}$ *à une requête pivot* $q \in \mathcal{Q}_q$*, avec le seuil* $\sigma \in]0;1]$ *et dans le contexte d'une base de connaissances* KB *se définit ainsi :*

$$mapIC^{KB} : \mathcal{E}_{qual} \times \mathcal{Q}_q \times]0;1] \rightarrow 2^{\mathcal{E}_{qual} \times \mathcal{E}_q \times \mathcal{E}_q \times \mathcal{I} \times \mathcal{C} \times]0;2] \times \mathcal{K}}$$

$$(e_p,q,\sigma) \mapsto \left\{ \left(\begin{array}{c} e_p, \\ e_{qi},\ e_{qc}, \\ i,\ c, \\ s_i + s_c, \\ l_i +' (' + l_c + ')' \end{array} \right) \middle| \begin{array}{l} (e_p,\ e_{qi},\ i,\ s_i,\ l_i) \in \bigcup \Big(mapEE^{KB}_{\sigma}(e_p,\ e_q) | e_q \in elem(q) \Big), \\ (e_p,\ e_{qc},\ c,\ s_c,\ l_c) \in \bigcup \Big(mapEE^{KB}_{\sigma}(e_p,\ e_q) | e_q \in elem(q) \Big), \\ e_{qi} \neq e_{qc} \\ et\ instanceOf(i,\ c) \in KB \end{array} \right\}$$

où l'opérateur + fait référence à la concaténation de chaînes de caractères.

Exemple 25 *Chacun des couples d'associations d'éléments* $(m_1,\ m_3)$ *et* $(m_2,\ m_3)$ *de l'exemple précédent permet de contruire une association d'élément instance-classe. Tous concernent l'élément de patron* c_1*.*

$$mapIC^{KB}_{0.5}(c_1,\ q) = \left\{ \begin{array}{c} \left(\begin{array}{c} c_1,\ actor,\ movie, \\ \texttt{cine:LesActeurs},\ \texttt{cine:Movie}, \\ 1.53,\ \texttt{"Les Acteurs (movie)"} \end{array} \right), \\ \left(\begin{array}{c} c_1,\ theArtist,\ movie, \\ \texttt{cine:TheArtist},\ \texttt{cine:Movie}, \\ 2,\ \texttt{"The Artist (movie)"} \end{array} \right) \end{array} \right\}$$

Le couple d'associations d'éléments $(m_4,\ m_3)$ *ne convient pas dans ce cas car ces deux associations impliquent le même élément de requête movie. Dans les exemples qui suivent, nous utiliserons ces notations :*

$$m_8 = \left(\begin{array}{c} c_1,\ actor,\ movie, \\ \texttt{cine:LesActeurs},\ \texttt{cine:Movie}, \\ 1.53,\ \texttt{"Les Acteurs (movie)"} \end{array} \right)$$

$$m_9 = \left(\begin{array}{c} c_1,\ theArtist,\ movie, \\ \texttt{cine:TheArtist},\ \texttt{cine:Movie}, \\ 2,\ \texttt{"The Artist (movie)"} \end{array} \right)$$

La traduction sous forme de graphe d'une requête LN peut utiliser des éléments (classes, instances ou propriétés) qui ne sont pas explicitement référencés dans cette requête LN (par exemple, la requête "Give me the list of movies from Tim Burton" implique la propriété `cine:hasForDirector` sans l'exprimer explicitement). Il sera donc nécessaire plus tard dans le processus d'interprétation de prendre en compte

des associations de patrons dans lesquelles tous les élements de patron n'ont pas été associés. C'est pourquoi nous introduisons dès cette étape, pour chaque élément de patron, une association d'élément vide signifiant que l'élément concerné n'est pas associé.

Définition 16 (Association d'élément vide) *Nous considérons également, pour chaque élément de patron qualifiant e_p, une* association d'élément vide

$$\emptyset_M^{e_p} = (e_p,\emptyset,\emptyset,0,l) \text{ avec } l = \begin{cases} l' \in label(targ(e_p)) & si\ targ(e_p) \in \mathcal{I} \cup \mathcal{R}_o \cup \mathcal{R}_d \\ "some\ " + l' \in label(targ(e_p)) & si\ targ(e_p) \in \mathcal{C} \\ "some\ " + text(targ(e_p)) & si\ targ(e_p) \in \ell \end{cases}$$

si la ressource $targ(e_p)$ a plusieurs étiquettes, une seule est sélectionnée au hasard

Dans une telle association, l'élément de patron considéré n'est associé à rien.

Définition 17 (Association d'élément de patron à une requête pivot) *La fonction $mapEQ^{KB}$ associant un élément de patron $e_p \in \mathcal{E}_{qual}$ à une requête pivot $q \in \mathcal{Q}_q$, avec le seuil $\sigma \in]0;1]$ dans le contexte d'une base de connaissances KB se définit ainsi :*

$$\begin{aligned} mapEQ^{KB} : \mathcal{E}_{qual} \times \mathcal{Q}_q \times]0;1] &\rightarrow 2^{\mathcal{E}_{qual} \times \mathcal{E}_q \times \mathcal{E} \times]0;1] \times \mathcal{K}} \\ (e_p,q,\sigma) &\mapsto \begin{array}{l} \bigcup \left(mapEE_\sigma^{KB}(e_p,e_q) | e_q \in elem(q) \right) \\ \cup mapIC_\sigma^{KB}(e_p,\ q) \\ \cup \emptyset_M^{e_p} \end{array} \end{aligned}$$

Exemple 26 *Les associations entre les éléments du patron movieDesc et la requête q obtenues à partir des résultats de l'exemple précédent sont les suivantes :*

$$\begin{aligned} mapEQ_{0.5}^{KB}(c_1,\ q) &= \left\{ \begin{array}{c} (c_1,\ actor,\ \texttt{cine:LesActeurs},\ 0.53,\ \texttt{"Les Acteurs"}), \\ (c_1,\ theArtist,\ \texttt{cine:TheArtist},\ 1,\ \texttt{"The Artist"}), \\ (c_1,\ movie,\ \texttt{cine:Movie},\ 1,\ \texttt{"movie"}), \\ (c_1,\ movie,\ \texttt{cine:Movie43},\ 0.875,\ \texttt{"Movie 43"}), \\ \left(\begin{array}{c} c_1,\ actor,\ movie, \\ \texttt{cine:LesActeurs},\ \texttt{cine:Movie}, \\ 1.53,\ \texttt{"Les Acteurs (movie)"} \end{array} \right), \\ \left(\begin{array}{c} c_1,\ theArtist,\ movie, \\ \texttt{cine:TheArtist},\ \texttt{cine:Movie}, \\ 2,\ \texttt{"The Artist (movie)"} \end{array} \right), \\ (c_1,\emptyset,\emptyset,0,\texttt{"some movie"}) \end{array} \right\} \\ &= \left\{ m_1,\ m_2,\ m_3,\ m_4,\ m_8,\ m_9,\ \emptyset_M^{c_1} \right\} \end{aligned}$$

$$mapEQ^{KB}_{0.5}(c_2,\, q) = \left\{ \begin{array}{c} (c_2,\, actor,\, \texttt{cine:Actor},\, 1,\, \texttt{"actor"}), \\ (c_2,\, theArtist,\, \texttt{cine:Actor},\, 0.78,\, \texttt{"artist"}), \\ (c_2,\emptyset,\emptyset,0,\texttt{"some actor"}) \end{array} \right\}$$
$$= \left\{ m_5,\, m_6,\, \emptyset^{c_2}_M \right\}$$

$$mapEQ^{KB}_{0.5}(c_3,\, q) = \left\{ (c_3,\emptyset,\emptyset,0,\texttt{"some director"}) \right\}$$
$$= \left\{ \emptyset^{c_3}_M \right\}$$

$$mapEQ^{KB}_{0.5}(c_4,\, q) = \left\{ (c_4,\emptyset,\emptyset,0,\texttt{"some nationality"}) \right\}$$
$$= \left\{ \emptyset^{c_4}_M \right\}$$

$$mapEQ^{KB}_{0.5}(r_1,\, q) = \left\{ \begin{array}{c} (r_1,\, play,\, \texttt{cine:hasForActor},\, 0.83,\, \texttt{"has for actor"}), \\ (r_1,\emptyset,\emptyset,0,\texttt{"has for actor"}) \end{array} \right\}$$
$$= \left\{ m_7,\, \emptyset^{r_1}_M \right\}$$

$$mapEQ^{KB}_{0.5}(r_2,\, q) = \left\{ (r_2,\emptyset,\emptyset,0,\texttt{"is directed by"}) \right\}$$
$$= \left\{ \emptyset^{r_2}_M \right\}$$

$$mapEQ^{KB}_{0.5}(r_3,\, q) = \left\{ (r_3,\emptyset,\emptyset,0,\texttt{"has nationality"}) \right\}$$
$$= \left\{ \emptyset^{r_3}_M \right\}$$

5.3 Association de sous-patrons

Nous pouvons ensuite générer pour chaque sous-patron toutes les associations possibles à la requête utilisateur – appelées associations de sous-patrons. Les associations de sous-patrons sont construites de façon incrémentale (ce qui se traduit par la fonction récursive définie plus bas), en commençant par les sous-patrons basiques et, dans le cas de sous-patrons imbriqués, en élargissant progressivement aux sous-patrons dont le contenu a déjà été associé.

Dans cette étape, les associations précédemment établies sont combinées en accord avec la structure imposée par les patrons, ce qui assure que les éléments

de la base de connaissances identifiés par les éléments de la requête sont combinés de façon à exprimer un réel besoin en information. Ainsi, le graphe requête généré est cohérent et a du sens du point de vue de l'utilisateur. Les éléments de la requête graphe générée qui ne sont pas exprimés dans la requête utilisateur sont directement tirés des patrons (c'est le rôle des associations d'éléments vides).

Toutes les associations de sous-patrons possibles sont générées, y compris celles qui ignorent les sous-patrons optionnels, et celles qui répètent les sous-patrons répétables.

Définition 18 (Association d'un sous-patron à une requête pivot) *Une association d'un sous-patron s_p consiste en un ensemble d'associations d'élément pour chacun de ces éléments qualifiants $qual(s_p)$, ainsi qu'un ensemble d'associations de sous-patron pour chacun des sous-patrons qu'il contient $cont(s_p)$. $\mathcal{M}$ est l'ensemble de toutes les associations de sous-patrons. La fonction récursive $mapSQ^{KB}$ associant un sous-patron $s_p \in \mathcal{S}_P$ à une requête pivot $q \in \mathcal{Q}_q$, avec un seuil $\sigma \in]0;1]$ dans le contexte d'une base de connaissances KB se définit ainsi :*

$$
\begin{array}{l}
mapSQ^{KB} : \mathcal{S}_P \times \mathcal{Q}_q \times]0;1] \rightarrow \mathcal{M} \\
(s_p,q,\sigma) \mapsto \ \prod_{e_p \in qual(s_p)} mapEQ^{KB}_{\sigma}(e_p,q) \\
\qquad \times \ \bigcup_{s'_p \in cont(s_p)} \left(\begin{array}{c} \bigcup \\ \left(\begin{array}{c} max(c_{min},1) \\ \leq c \leq \\ min\left(c_{max}, \left|mapSQ^{KB}_{\sigma}(s'_p,q)\right|\right) \end{array} \right) \end{array} \mathcal{P}_c\Big(mapSQ^{KB}_{\sigma}(s'_p,q)\Big) \right)
\end{array}
$$

où $\prod$ et $\times$ font référence au produit Cartésien et $\mathcal{P}_k(S)$ est l'ensemble des k-combinaisons des éléments d'un ensemble S.

Exemple 27 *Le sous-patron $dirNat$ ne contient aucun sous-patron (la deuxième partie du produit cartésien de la formule ci-dessus est donc nulle) et aucun de ses sommets qualifiants (r_3 et c_4) n'a pu être associé dans l'étape précédente :*

$$
\begin{array}{rcl}
mapSQ^{KB}_{0.5}(dirNat,q) = & mapEQ^{KB}_{0.5}(r_3,\, q) \times mapEQ^{KB}_{0.5}(c_4,\, q) \\
= & \{\emptyset^{r_3}_M\} \times \{\emptyset^{c_4}_M\} \\
= & \left\{(\emptyset^{r_3}_M, \emptyset^{c_4}_M)\right\}
\end{array}
$$

Le sous-patron director contient le sous-patron dirNat (avec une cardinalité minimale nulle et une cardinalité maximale égale à un) mais aucun de ses sommets qualifiants (r_2 et c_3) n'a pu être associé dans l'étape précédente :

$$\begin{aligned} mapSQ_{0.5}^{KB}(director,q) = \; & mapEQ_{0.5}^{KB}(r_2,\, q) \times mapEQ_{0.5}^{KB}(c_3,\, q) \\ & \times \mathcal{P}_1\Big(mapSQ_{\sigma}^{KB}(dirNat,q)\Big) \\ = \; & \{\emptyset_M^{r_2}\} \times \{\emptyset_M^{c_3}\} \times \mathcal{P}_1\Big(\Big\{(\emptyset_M^{r_3},\emptyset_M^{c_4})\Big\}\Big) \\ = \; & \{\emptyset_M^{r_2}\} \times \{\emptyset_M^{c_3}\} \times \Big\{\Big\{(\emptyset_M^{r_3},\emptyset_M^{c_4})\Big\}\Big\} \\ = \; & \Big\{\Big(\emptyset_M^{r_2},\emptyset_M^{c_3},\{\emptyset_M^{r_3},\emptyset_M^{c_4}\}\Big)\Big\} \end{aligned}$$

On remarque que, dans l'unique association de sous-patron obtenue, le dernier élément du n-uplet qui le représente n'est pas simplement une association d'élément, mais un ensemble de deux associations d'éléments. Un membre d'une association de sous-patron peut en effet être une autre association de sous-patron. Cette situation se présente lorsque le second membre de la formule donnée ci-dessus n'est pas nul, c'est-à-dire lorsque l'on procède à la composition d'associations de sous-patrons imbriqués.

Enfin, le sous-patron actor ne contient aucun sous-patron (la deuxième partie du produit cartésien de la formule ci-dessus est donc nulle) et chacun de ses sommets qualifiants (r_1 et c_2) est impliqué dans des associations autres que l'association d'élément vide :

$$\begin{aligned} mapSQ_{0.5}^{KB}(actor,q) = \; & mapEQ_{0.5}^{KB}(r_1,\, q) \times mapEQ_{0.5}^{KB}(c_2,\, q) \\ = \; & \{m_7,\, \emptyset_M^{r_1}\} \times \{m_5,\, m_6,\, \emptyset_M^{c_2}\} \\ = \; & \left\{ \begin{array}{l} (m_7,\, m_5),\, (m_7,\, m_6),\, (m_7,\, \emptyset_M^{c_2}),\; \\ (\emptyset_M^{r_1},\, m_5),\, (\emptyset_M^{r_1},\, m_6),\, (\emptyset_M^{r_1},\, \emptyset_M^{c_2}) \end{array} \right\} \end{aligned}$$

Définition 19 *Pour une association de sous-patron donnée m_{sp}, nous notons $elemMap(m_{sp})$ l'ensemble de toutes les associations d'élément contenues dans m_{sp} (peu importe le niveau d'imbriquation des associations d'éléments dû à l'éventuelle composition de sous-patrons imbriqués).*

Exemple 28 *Nous donnons ci-dessous les ensembles des associations d'élément*

contenues dans quelques-unes des associations de sous-patrons trouvées plus haut :

$$elemMap\Big((\emptyset_M^{r_3},\emptyset_M^{c_4})\Big) = \Big\{\emptyset_M^{r_3},\emptyset_M^{c_4}\Big\}$$
$$elemMap\Big(\big(\emptyset_M^{r_2},\emptyset_M^{c_3},\{\emptyset_M^{r_3},\emptyset_M^{c_4}\}\big)\Big) = \Big\{\emptyset_M^{r_2},\emptyset_M^{c_3},\emptyset_M^{r_3},\emptyset_M^{c_4}\Big\}$$
$$elemMap\Big((m_7,\, m_5)\Big) = \Big\{m_7,\, m_5\Big\}$$

Définition 20 (Association de sous-patron vide) *L'association d'un sous-patron s_p à une requête q ne contenant que des associations d'élément vides est appelée* association de sous-patron vide *et se note $\emptyset_M^{s_p}$.*

Exemple 29 *Les associations de sous-patrons trouvées précédemment peuvent donc se noter ainsi :*

$$mapSQ_{0.5}^{KB}(dirNat,q) = \Big\{\emptyset_M^{dirNat}\Big\}$$
$$mapSQ_{0.5}^{KB}(director,q) = \Big\{\emptyset_M^{director}\Big\}$$
$$mapSQ_{0.5}^{KB}(actor,q) = \left\{ \begin{array}{c} (m_7,\, m_5),\, (m_7,\, m_6),\, (m_7,\, \emptyset_M^{c_2}),\\ (\emptyset_M^{r_1},\, m_5),\, (\emptyset_M^{r_1},\, m_6),\, \emptyset_M^{actor} \end{array} \right\}$$

5.4 Association de patron

Les patrons de requêtes étant des sous-patrons, ils sont associés de la même façon que ces derniers. Le résultat de cette étape est un ensemble d'associations de patrons. Une association de patron m_p associe un patron p à la requête pivot considérée q et constitue une interprétation possible de la requête utilisateur.

Exemple 30 *Le patron movieDesc contient directement les sous-patrons actor et director (tous deux avec une cardinalité minimale nulle et une cardinalité maximale égale à deux) et son sommet qualifiant c_1 est impliqué dans quatre associations*

en plus de l'association d'élément vide :

$$
\begin{aligned}
mapSQ^{KB}_{0.5}(movieDesc,q) = \; & mapEQ^{KB}_{0.5}(c_1,\, q) \\
& \times \begin{pmatrix} & \mathcal{P}_1\Big(mapSQ^{KB}_{\sigma}(actor,q)\Big) \\ \cup & \mathcal{P}_2\Big(mapSQ^{KB}_{\sigma}(actor,q)\Big) \end{pmatrix} \\
& \times \mathcal{P}_1\Big(mapSQ^{KB}_{\sigma}(director,q)\Big) \\
= \; & \{m_1,\, m_2,\, m_3,\, m_4,\, m_8,\, m_9,\, \emptyset^{c_1}_M\} \\
& \times \begin{pmatrix} & \mathcal{P}_1\left(\left\{ \begin{array}{c} (m_7,\, m_5),\, (m_7,\, m_6),\, (m_7,\, \emptyset^{c_2}_M), \\ (\emptyset^{r_1}_M,\, m_5),\, (\emptyset^{r_1}_M,\, m_6),\, \emptyset^{actor}_M \end{array} \right\}\right) \\ \cup & \mathcal{P}_2\left(\left\{ \begin{array}{c} (m_7,\, m_5),\, (m_7,\, m_6),\, (m_7,\, \emptyset^{c_2}_M), \\ (\emptyset^{r_1}_M,\, m_5),\, (\emptyset^{r_1}_M,\, m_6),\, \emptyset^{actor}_M \end{array} \right\}\right) \end{pmatrix} \\
& \times \mathcal{P}_1\Big(\Big\{\emptyset^{director}_M\Big\}\Big) \\
= \; & \ldots
\end{aligned}
$$

Le calcul ne peut être détaillé ici car il est trop volumineux. En effet, le résultat est constitué de 147 associations de patron. Nous donnons ci-dessous quelques-uns de ces résultats qui sont utilisés dans les exemples suivants :

$$
\begin{aligned}
M_1 &= \Big(m_1,\, \Big\{(m_7,\, m_6),(\emptyset^{r_1}_M,\, m_5)\Big\},\, \Big\{\emptyset^{director}_M\Big\}\Big) \\
M_2 &= \Big(m_8,\, \Big\{(\emptyset^{r_1}_M,\, m_5)\Big\},\, \Big\{\emptyset^{director}_M\Big\}\Big) \\
M_3 &= \Big(m_9,\, \Big\{(m_7,\, m_5)\Big\},\, \Big\{\emptyset^{director}_M\Big\}\Big) \\
M_4 &= \Big(\emptyset^{c_1}_M,\, \Big\{\emptyset^{actor}_M\Big\},\, \Big\{\emptyset^{director}_M\Big\}\Big) = \emptyset^{movieDesc}_M
\end{aligned}
$$

Dans une association de patron, les associations de sous-patron vides relatives à un sous-patron optionnel (dont la cardinalité minimale est nulle) sont ignorées dans la suite du processus, c'est-à-dire qu'elles ne seront pas prises en compte dans l'évaluation de la pertinence de l'association de patron, ni lors de la génération du graphe requête et de la phrase descriptive correspondante. Nous introduisons ici la notion d'*association d'élément effective* utilisée dans la suite.

Définition 21 *Pour une association de sous-patron donnée m_{sp}, nous notons $effElemMap(m_{sp})$ l'ensemble de toutes les associations d'élément contenues dans m_{sp} et qui ne font pas partie d'une association de sous-patron vide, appelées associations d'élément effectives.*

Exemple 31 *Nous donnons ci-dessous les ensembles des associations d'élément effectives contenues dans quelques-unes des associations de sous-patrons trouvées plus haut :*

$$effElemMap(M_1) = \{m_1,\, m_7,\, m_6,\, m_5\}$$
$$effElemMap(M_2) = \{m_8,\, \emptyset_M^{r_1},\, m_5\}$$
$$effElemMap(M_3) = \{m_9,\, m_7,\, m_5\}$$
$$effElemMap(M_4) = \emptyset$$

Notons que l'ensemble des associations d'éléments effectives d'une association de patron m_p ne consiste pas simplement en l'ensemble des associations d'élément de m_p privé de ces associations d'élément vides. En effet, certaines associations d'éléments vides apparaissent dans des associations de sous-patron non vides et ne sont donc pas ignorées dans la suite.

5.5 Evaluation de la pertinence des interprétations possibles

Dans le but de soumettre en priorité à l'utilisateur les interprétations de requête qui semblent les plus pertinentes, une *note de pertinence* R est calculée pour chaque association de patron M_i. Cette note est obtenue à partir de plusieurs notes partielles, chaque note partielle prenant en compte certains paramètres que nous avons jugés pertinents.

Définition 22 (Note d'association d'élément) *La* note d'association d'élément R_{map} *représente la confiance que nous avons en chaque association d'élément impliquée dans l'association de patron considérée :*

$$R_{map}(M_p) = \frac{\sum_{(e_p,e_q,r,t,l)\in effElemMap(m_q)} t}{|effElemMap(m_q)|}$$

Cette note représente la moyenne des scores des appariements impliqués dans les associations d'éléments effectives de l'interprétation évaluée, et permet ainsi d'avantager les interprétations exploitant des appariements de bonne qualité et de pénaliser celles qui comportent beaucoup d'associations d'élément vides.

Exemple 32

$$R_{map}(M_1) = \frac{0.53 + 0.83 + 0.78 + 1}{4} = 0.785$$
$$R_{map}(M_2) = \frac{1.53 + 0 + 1}{3} = 0.843$$
$$R_{map}(M_3) = \frac{2 + 0.83 + 1}{3} = 1.277$$
$$R_{map}(M_4) = 0$$

Définition 23 (Note de couverture de requête) *La* note de couverture de requête R_{Qcov} *prend en compte la proportion d'éléments de la requête pivot initiale qui sont impliqués dans l'association de patron considérée :*

$$R_{Qcov}(M_p) = \frac{|\{e_q \in elem(q) \mid \exists(e_p,e_q,r,t,l) \in elemMap(m_q)\}|}{|elem(q)|}$$

Cette note représente la proportion des éléments de la requête pivot initiale qui ont été exploités pour fournir l'interprétation évaluée. Nous rappelons que certaines associations de patron peuvent ignorer des éléments de la requête pivot. Cette note permet de pénaliser de telles associations.

Exemple 33

$$R_{Qcov}(M_1) = \frac{3}{4} = 0.75 \qquad \textit{(movie n'est pas pris en compte)}$$
$$R_{Qcov}(M_2) = \frac{2}{4} = 0.5 \qquad \textit{(theArtist et play ne sont pas pris en compte)}$$
$$R_{Qcov}(M_3) = \frac{4}{4} = 1 \qquad \textit{(m_9 implique à la fois theArtist et movie)}$$
$$R_{Qcov}(M_4) = \frac{0}{4} = 0$$

Définition 24 (Note de couverture du patron) *La* note de couverture du patron R_{Pcov} *prend en compte la proportion d'éléments qualifiants du patron qui sont impliqués dans une association d'élément non nulle (sans considérer les sous-patrons optionnels impliqués dans une association de sous-patron vide) :*

$$R_{Pcov}(M_p) = \frac{|\{e_p \in qual(p) \mid \exists(e_p,e_q,r,t,l) \in effElemMap(m_q) \text{ et } e_q \neq \emptyset\}|}{|effElemMap(m_q)|}$$

Cette note est l'équivalent pour les patrons de la note de couverture de requête. Elle représente la proportion des éléments qualifiants du patron exploités pour fournir l'interprétation évaluée. Seuls les éléments de patrons qui ne sont pas compris dans des associations de sous-patron vides sont pris en compte, car nous ne voulons pas pénaliser le fait qu'un sous-patron optionnel ait été ignoré.

Exemple 34

$$R_{Pcov}(M_1) = \frac{|\{c_1,r_1,c_2\}|}{|\{c_1,r_1,c_2\}|} = \frac{3}{3} = 1$$
$$R_{Pcov}(M_2) = \frac{|\{c_1,\ c_2\}|}{|\{c_1,\ r_1,\ c_2\}|} = \frac{2}{3} = 0.666$$
$$R_{Pcov}(M_3) = \frac{|\{c_1,\ r_1,\ c_2\}|}{|\{c_1,\ r_1,\ c_2\}|} = \frac{3}{3} = 1$$
$$R_{Pcov}(M_4) = \frac{|\emptyset|}{|\{c_1\}|} = \frac{0}{1} = 0$$

Définition 25 (Note de respect de la structure de la requête pivot) *La* note de respect de la structure de la requête pivot R_{struct} *prend en compte la similarité entre la structure de la requête pivot et celle de la requête graphe (dont la construction est décrite plus bas) issue de l'interprétation de cette requête pivot :*

$$R_{struct}(M_p) = 1 + n\alpha$$

où $\alpha \in]0;1]$ *et n représente le nombre de paires d'éléments de requêtes appartenant à une même sous-requête pivot auxquels ont été associés des éléments de patron appartenant à un même triplet de patron.*

Cette dernière note permet d'avantager les interprétations dont la requête graphe est proche de la structure donnée par la requête pivot. Elle constitue une version dégradée d'un véritable algorithme de comparaison de graphe. Plus il y a de paires d'élements de la requête pivot qui "se retrouvent" sur un même triplet de la requête graphe obtenue, plus la note sera élevée.

Exemple 35 *En prenant* $\alpha = 0{,}1$

$$R_{struct}(M_1) = 1 + \left| \left\{ \begin{array}{c} (actor,\ play), \\ (play,\ theArtist), \\ (actor,\ theArtist) \end{array} \right\} \right| \times 0{,}1 = 1{,}3$$
$$R_{struct}(M_2) = 1 + \emptyset \times 0{,}1 = 1$$
$$R_{struct}(M_3) = 1 + \left| \left\{ \begin{array}{c} (actor,\ play), \\ (play,\ theArtist), \\ (actor,\ theArtist) \\ (theArtist,\ movie) \end{array} \right\} \right| \times 0{,}1 = 1{,}4$$
$$R_{struct}(M_4) = 1 + |\emptyset| \times 0{,}1 = 1$$

Définition 26 (Note de pertinence) *La* note de pertinence R *est finalement obtenue à partir des notes précédentes :*

$$R(M_p) = \Big(R_{map}(m_q)\Big)^{\alpha_{map}} \cdot \Big(R_{Qcov}(m_q)\Big)^{\alpha_{Qcov}} \cdot \Big(R_{Pcov}(m_q)\Big)^{\alpha_{Pcov}} \cdot \Big(R_{struct}(m_q)\Big)^{\alpha_{struct}}$$
$$avec\ \alpha_{map},\alpha_{Qcov},\alpha_{Pcov},\alpha_{struct} \in \mathbb{R}$$

Exemple 36 *En prenant* $\alpha_{map} = \alpha_{Qcov} = \alpha_{Pcov} = \alpha_{struct} = 1$

$$R(M_1) = 0{,}785 \times 0{,}75 \times 1 \times 1{,}3 = 0.765$$
$$R(M_2) = 0{,}843 \times 0{,}5 \times 0{,}666 \times 1 = 0.281$$
$$R(M_3) = 1{,}277 \times 1 \times 1 \times 1{,}4 = 1.788$$
$$R(M_4) = 0 \times 0 \times 0 \times 1 = 0$$

5.6 Génération des phrases descriptives

La dernière étape consiste à présenter les résultats à l'utilisateur dans le but d'interroger la base de connaissances. Pour cela, nous générons pour chaque association de patron une phrase en langue naturelle qui décrit la requête représentée par l'asssociation, et nous présentons ces phrases à l'utilisateur par ordre de pertinence décroissant. De cette façon, en parcourant les phrases explicatives, l'utilisateur peut facilement comprendre le sens de chaque requête et choisir dès qu'il la rencontre celle correspondant à son besoin en information. De l'association sélectionnée, le système déduit la requête formelle exprimée en SPARQL. Ces deux opérations, la génération de la phrase descriptive et la formulation de la requête

graphe, sont rendues triviales par le modèle de phrase descriptive et le graphe de chaque patron.

Pour une association de patron donnée, la phrase descriptive est obtenue en adaptant le modèle de phrase rattaché au patron concerné par l'association. Cette adaptation est réalisée de manière descendante : on considère d'abord le modèle de phrase dans sa globalité, puis on modifie progressivement les parties du modèle de phrase liées aux sous-patrons imbriqués en fonction des associations de ces sous-patrons.

L'adaptation du modèle de phrase d'un sous-patron (et du patron lui-même) s'effectue selon ces étapes :

1. si le sous-patron considéré est répétable et instancié plusieurs fois dans l'association concernée, alors le contenu du modèle de phrase situé à l'intérieur de la structure *for* relative à ce sous-patron est répété autant de fois que nécessaire ;
2. pour chaque élément qualifiant du sous-patron considéré, l'élément du modèle de phrase descriptive le référençant est remplacé par la représentation textuelle de l'association d'élément impliquant cet élément qualifiant ;
   nous rappelons que cette représentation textuelle, mise à part dans le cas des associations d'éléments vides, a été construite à l'aide de l'étiquette qui a permis l'appariement concerné ; ainsi, la phrase générée est la plus proche possible de la requête LN de l'utilisateur et celui-ci a plus de facilité à l'interpréter ;
   si l'élément de patron considéré est associé au focus de la requête, alors sa représentation textuelle est syntaxiquement mise en exergue de façon à ce que l'utilisateur puisse vérifier que l'interprétation est fidèle à son besoin du point de vue du focus (et par conséquent du point de vue de l'attribut de projection) ;
3. enfin, pour chaque sous-patron inclus dans le sous-patron considéré, la partie du modèle de phrase le concernant est soit supprimée si le sous-patron inclus est optionnel et impliqué dans une association de sous-patron vide, soit adaptée à/aux association(s) de sous-patron le concernant.

Exemple 37 *La figure 5.3 illustre les différentes étapes de tranformation du modèle de phrase descriptive du patron movieDesc pour l'adapter à l'association de patron* M_3.

5.7 Génération des requêtes formelles

L'étape de formulation de la requête en SPARQL pour l'association sélectionnée n'est pas plus complexe. La méthode d'obtention du graphe requête est très proche

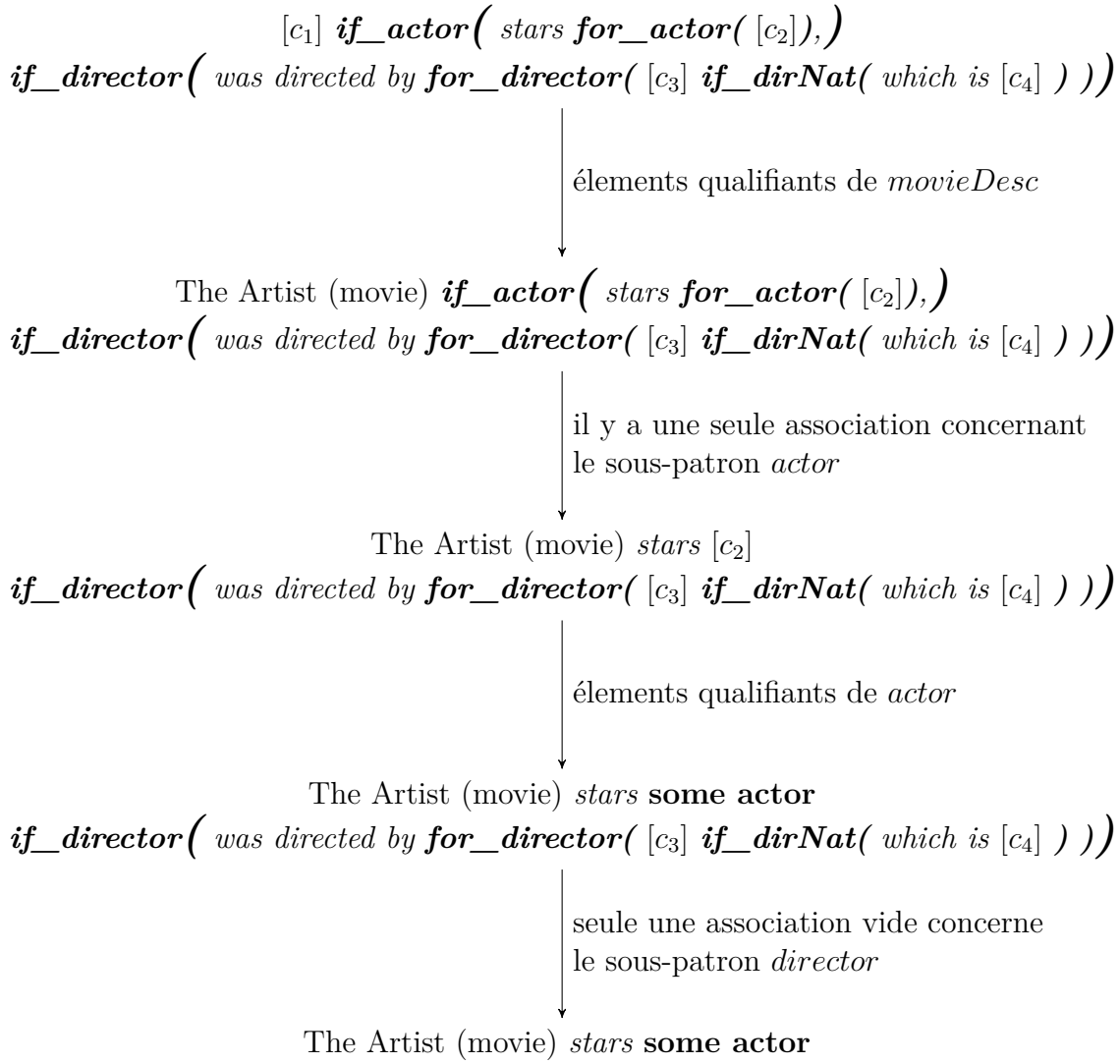


FIGURE 5.3 – Étapes de génération de la phrase descriptive de l'association M_3

de celle employée pour la génération de la phrase descriptive, mais c'est cette fois le graphe du patron de requêtes qui est adapté à l'association de requête considérée.

L'adaptation du graphe d'un sous-patron (et du patron lui-même) s'effectue de la manière suivante :

1. si le sous-patron considéré est répétable et instancié plusieurs fois dans l'association concernée, alors le sous-graphe qui le concerne est répété autant de fois que nécessaire, à l'exception du sommet d'articulation du sous-patron qui sert de "point d'ancrage" des différentes instances de ce sous-patron au

reste du graphe requête,

2. chaque élément qualifiant du sous-patron est remplacé :
   - si il référence une classe (respectivement un type de littéral) et l'élément de la base de connaissances apparié de l'association d'élément concernant cet élément qualifiant est une classe (respectivement un type de littéral), par une variable déclarée comme ayant pour type cette classe (respectivement ce type de littéral) appariée,
   - sinon, directement par l'élément de la base de connaissances apparié de l'association d'élément concernant cet élément qualifiant (dans le cas d'une association d'élément instance-classe, c'est l'instance appariée qui est utilisée),
3. enfin, pour chaque sous-patron inclus dans le sous-patron considéré, le sous-graphe le concernant est soit supprimé si le sous-patron inclus est optionnel et impliqué dans une association de sous-patron vide, soit adapté à/aux association(s) de sous-patron le concernant.

Le graphe requête est donc obtenu après application de chacune de ces étapes, de façon descendante, à tous les sous-patrons concernés par une association de patron, et sa sérialisation en Turtle donne le motif de graphe (le contenu de la clause `WHERE`) de la requête SPARQL recherchée. Si la requête pivot est de type liste, la requête SPARQL sera de type `SELECT` et sa variable de projection la variable introduite lors de la modification du graphe du patron pour prendre en compte l'association d'élément impliquant le focus de la requête pivot. Si il s'agit d'une requête de dénombrement, l'opérateur `COUNT` prenant comme argument la ou les variables de projection est ajouté après le `SELECT`. Enfin, si la requête traitée est dichotomique, on introduit simplement le mot-clé `ASK` devant le motif de graphe.

Exemple 38 *La requête SPARQL obtenue après adaptation du graphe du patron movieDesc pour l'association de patron M_1 est la suivante :*

Listing 10

```
PREFIX cine: <http://www.irit.fr/ontologies/cinema#>
SELECT ?actor
WHERE
{
  cine:TheArtist cine:hasForActor ?actor.
}
```

5.8 Quelques mots sur la combinatoire

Du point de vue de la complexité, une phase critique de l'approche que nous proposons se situe à l'étape d'association des sous-patrons, présentée en 5.3. En

effet, la formule proposée pour générer ces associations laisse clairement apparaître une faille potentielle de l'approche.

Cependant, nous avons constaté lors de nos expérimentations que ce problème est rarement pénalisant car, le plus souvent, ce n'est qu'un petit sous-ensemble des éléments d'un patron qui est "concerné" par une requête, les autres éléments n'étant impliqués dans rien d'autre qu'une association d'élément vide (c'est d'ailleurs la situation que l'on peut observer dans l'exemple précédent).

La faille reste néanmoins réelle et la combinatoire élevée se manifeste particulièrement lorsque le nombre d'appariements pour chaque mot-clé augmente. Le problème pourrait donc prendre beaucoup plus d'importance si l'approche était déployée à très grande échelle sur le web. Nous énonçons ici deux pistes d'optimisation :

- l'amélioration de la fiabilité de l'appariement des mots-clés nous permettrait d'augmenter le seuil σ des appariements considérés, autrement dit de ne considérer que les k meilleurs appariements pour chaque mot-clé (k étant le plus faible possible) tout en s'assurant de ne pas mettre de côté le bon appariement,
- le dévelopement d'une heuristique permettant d'orienter la génération des associations de sous-patrons éviterait de générer des associations non pertinentes.

5.9 Conclusion

Dans cette section, nous avons montré comment nous exploitons les patrons de requêtes (décrits dans le chapitre 3) pour interpréter une requête pivot (générée à partir de la requête LN utilisateur comme décrit dans le chapitre 4) pour obtenir une liste d'associations de patrons ordonnées par leurs pertinences supposées, chaque association de patron étant une candidate à l'interprétation de la requête pivot. Ceci conclut la partie décrivant les apports scientifiques de notre travail.

La dernière partie présente les apports d'un point de vue pratique : l'implémentation de notre approche est décrite dans le chapitre 6 et les évaluations que nous avons menées sont présentées dans le chapitre 7.

Troisième partie

Contributions pratiques

Chapitre 6

Implémentation

Un prototype de notre approche a été implémenté sous forme d'application web pour des besoins d'évaluation et de démonstration. Il peut être testé à cet URL : http://swip.univ-tlse2.fr/SwipWebClient. Nous avons nommé le système *Swip*, pour *Semantic Web Interface using Patterns*.

À l'instar de l'approche présentée dans ce travail, le système se divise en deux modules principaux. Chaque module est chargé d'une des deux grandes étapes du processus d'interprétation d'une requête utilisateur, comme illustré dans la figure 6.1. Le premier module est chargé d'interpréter la requête utilisateur exprimée en langue naturelle et de la traduire en une requête pivot. Le second associe les patrons de requête à cette requête pivot pour obtenir des interprétations possibles de la requête utilisateur et ordonne ces interprétations par leur pertinence supposée avant de les soumettre à l'utilisateur sous forme de phrase descriptive.

Le premier module est architecturé de façon assez classique pour une application web : des services web effectuent les sous-tâches basiques du processus d'interprétation présentées dans le chapitre 4 et sont exploités par un *workflow*, lui-même accessible sous forme de service web. Chaque service exploite en entrée le résultat du ou des services situés en amont dans le *workflow*. Ce premier module et l'ensemble des services web qui le composent sont présentés dans la section 6.1 de ce chapitre.

L'implémentation du second module est plus originale et présente selon nous une réelle nouveauté. En effet, dans cette implémentation, les traitements permettant les appariements et associations décrits dans le chapitre 5 sont entièrement réalisés sur une base de triplets RDF par l'intermédiaire de requêtes de mise à jour SPARQL. Cette approche permet d'éviter l'implémentation de fonctions de manipulation et d'appariement de graphes et d'exploiter les capacités d'un moteur SPARQL qui est justement conçu et optimisé pour apparier des graphes. Ce second module est présenté dans la section 6.2 où sont également détaillées les premières étapes du processus de formalisation de la requête pivot afin de donner une idée

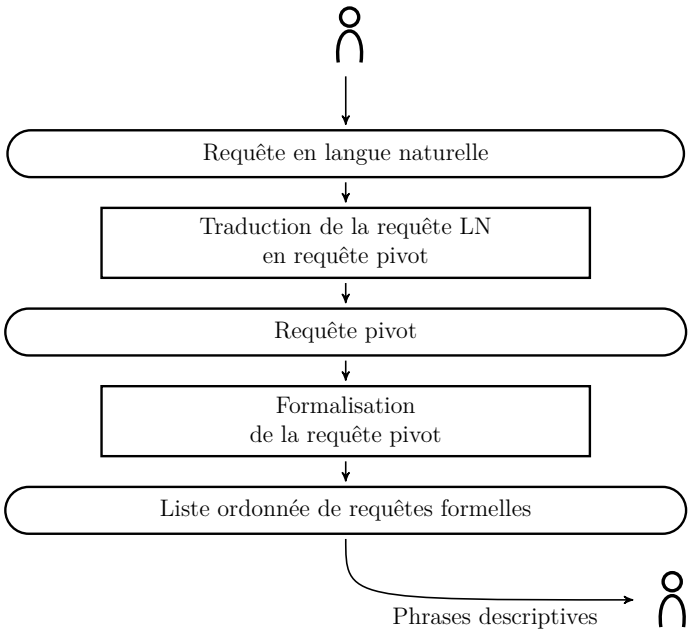


FIGURE 6.1 – Aperçu du processus d'interprétation d'une requête LN dans le système *Swip*

précise de son implémentation.

La section 6.3 discute des avantages et inconvénients de cette approche et la section 6.4 présente brièvement l'interface développée pour tester l'approche et visualiser les patrons de requêtes. Enfin, la section 6.5 conclut ce chapitre.

6.1 Interprétation de la requête en langue naturelle

La première grande étape du processus, liée au traitement automatique du langage naturel, est implémentée dans un ensemble de services web, chacun réalisant une des sous-étapes introduites en section 4.4 et présentées dans la figure 4.1. L'identification des entités nommées [1] est réalisée à l'aide de l'outil *Large Knowledge Base (LKB) gazetteer* [2]. Ce gazetteer présentait pour nous certains avantages : il est intégré à la plate-forme GATE [Cunningham, 2002], il est capable de créer un index directement à partir d'une base de triplets RDF et, surtout, il passe très bien à l'échelle.

1. http://swip.univ-tlse2.fr/NlToPivotGazetteer/testForm-ext.html
2. http://gate.ac.uk/userguide/sec:gazetteers:lkb-gazetteer

L'analyse en dépendances de la requête LN [3] consiste en fait en deux traitements successifs : la détermination des catégories grammaticales, effectuée par *TreeTagger* [Schmid, 1994], et l'analyse en dépendances en elle-même, réalisée par *MaltParser* [Nivre et al., 2007]. Nous avons choisi *MaltParser* principalement parce qu'il est facile de se procurer des modèles pré-entraînés pour différentes langues.

Les dernières sous-étapes consistant à traduire l'arbre de dépendances en une requête pivot (identification du type de requête, identification des objets de la requête, et génération des sous-requêtes) sont réalisées par un seul et même service web [4], implémenté en *Java*.

Enfin, le *workflow* [5] interface l'ensemble de ces services web et permet leur exécution successive, les résultats de l'un étant exploités par le suivant.

6.2 Interprétation de la requête pivot tout en SPARQL

La deuxième grande étape du processus (formalisation de la requête pivot) est réalisée en exploitant *Fuseki* [6], un serveur SPARQL intégrant le moteur de requêtes *ARQ* [7]. Dans cette section, nous détaillons l'implémentation de cette étape de traduction qui présente de notre point de vue une nouveauté réelle : l'interprétation de la requête pivot est réalisée en exploitant directement les capacités d'un moteur SPARQL. En effet, la fonctionnalité de base de ce type de moteur est l'appariement de graphe, ce qui est exactement l'objectif de cette étape d'interprétation. Ainsi, cette tâche est entièrement réalisée via des requêtes SPARQL, que nous présentons dans la suite de cette section. Le résultat de chaque étape est systématiquement enregistré dans la base de connaissances, via les fonctionnalités de mise à jour de SPARQL (*SPARQL updates*), ce qui les rend directement exploitables dans l'étape suivante.

La sous-section 6.2.1 présente les ontologies OWL que nous avons développées afin de constituer un cadre pour les triplets générés. Les sous-sections 6.2.2, 6.2.3 et 6.2.4 détaillent les premières étapes du processus. Par souci de concision et de simplicité, les étapes suivantes sont présentées plus grossièrement dans la sous-section 6.2.5. La figure 6.2 montre toutes les étapes de cette implémentation, chaque étape correspondant à une requête `UPDATE` ou `ASK` (qui permet d'émuler une boucle, comme expliqué plus bas en 6.2.5) ; ces requêtes sont données en annexe D

3. http://swip.univ-tlse2.fr/NlToPivotParser/testForm-ext.html
4. http://swip.univ-tlse2.fr/NlToPivotRules/testForm-ext.html
5. http://swip.univ-tlse2.fr/SwipWorkflow/testForm-ext.html
6. http://jena.apache.org/documentation/serving_data/
7. http://openjena.org/ARQ/

et sur la page web de présentation du système *Swip*[8].

6.2.1 Ontologies de patrons et de requêtes

Les triplets manipulés et générés lors du processus d'implémentation exploitent le vocabulaire de deux ontologies construites par nos soins et accessibles depuis la page d'accueil du système *Swip*.

L'ontologie *patterns*, présentée en section 3.4, permet de modéliser des patrons de requête et facilite ainsi la gestion, le partage et l'évolution de ces patrons. Les données RDF représentant les patrons de requêtes au format spécifié par cette ontologie trouvent ici une nouvelle utilité étant donné que, une fois insérées dans la base de triplets, elles sont exploitées telles quelles pour interpréter la requête utilisateur.

L'ontologie *queries* permet quant à elle de modéliser des requêtes pivot et définit les structures communes pour représenter les résultats intermédiaires et finals du processus d'interprétation. Nous ne la détaillons pas ici, étant donné qu'elle a été conçue selon les mêmes principes que l'ontologie *patterns* et se révèle plus simple que cette dernière. Comme nous l'avons expliqué dans la section 4.2, une requête pivot est un ensemble de sous-requêtes, chaque sous-requête étant un 1/2/3-uplet d'éléments de requête. Les classes et propriétés de l'ontologie *queries* reflètent logiquement cette structure ; leur noms sont explicites, et un exemple de requête pivot et une partie des résultats de son interprétation exprimés en RDF et fondés sur cette ontologie est montré plus bas.

Cette ontologie intègre également des axiomes permettant l'inférence de nouveaux triplets. Cependant, ces possibilités d'inférence ne sont ici pas exploitées pour des raisons de performances : contrairement à la description des patrons qui est statique (mis à part dans les cas exceptionnels et ponctuels d'évolution ou d'ajout de patrons), les triplets générés au cours de l'interprétation évoluent en continu, et il n'est donc pas possible d'inférer les connaissances liées à l'ontologie *queries* en pré-traitement. Or, il est très coûteux d'activer un raisonneur sur un jeu de données contenu dans un serveur SPARQL, particulièrement lorsque ce jeu de données fait l'objet de nombreuses mises à jour. C'est pourquoi dans notre implémentation, l'ensemble des triplets exploités dans la suite du processus sont générés explicitement à chaque étape, et ce même si ils sont redondants et pourraient être obtenus à partir d'autres triplets et des connaissances ontologiques.

8. http://swip.univ-tlse2.fr/SwipWebClient/welcome.html

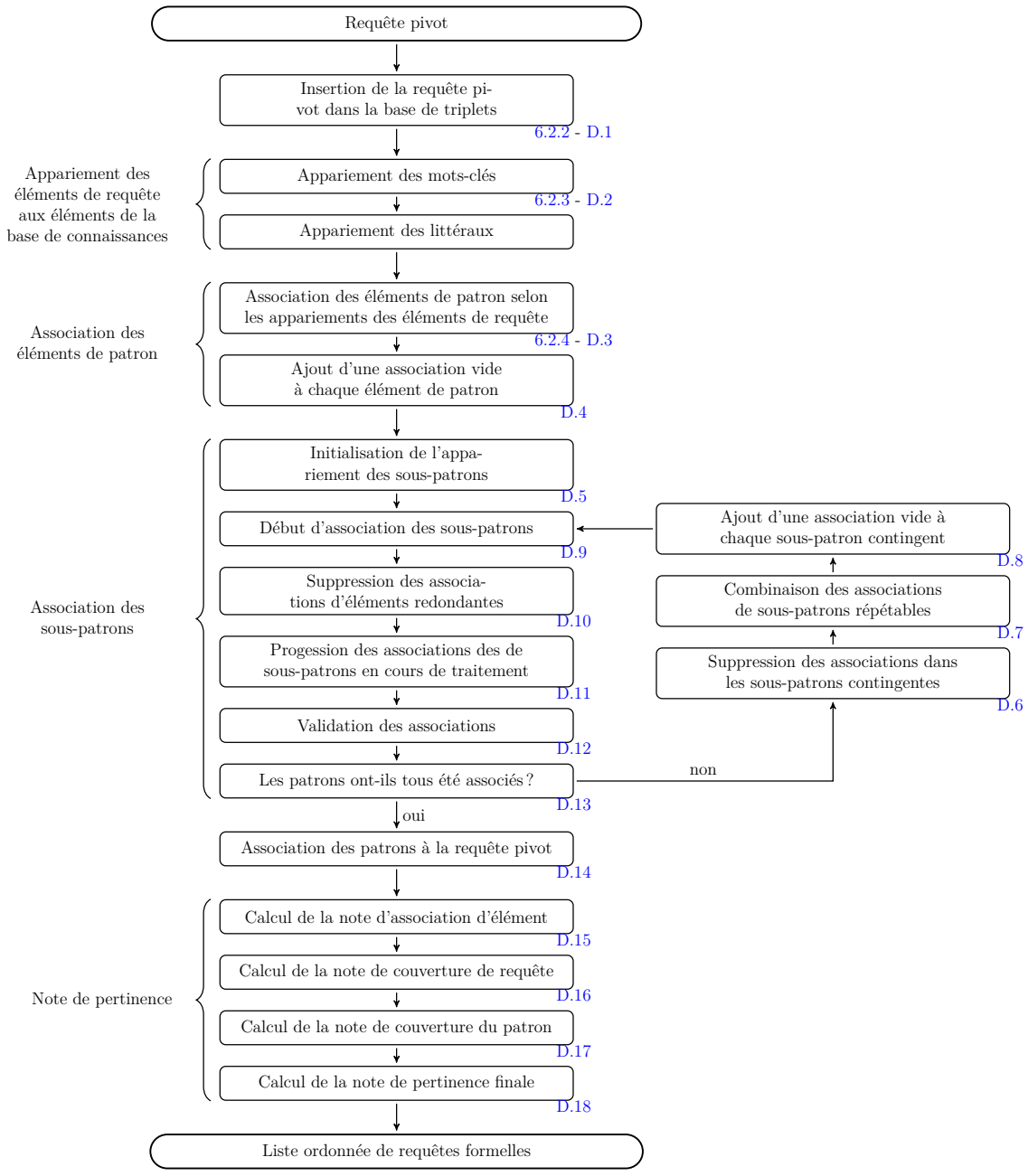


FIGURE 6.2 – Détail des étapes de formalisation de la requête pivot.

6.2.2 Insertion de la requête pivot dans la base de connaissances

Dans un premier temps, le système génère un URI qui est unique pour chaque ensemble de requêtes pivot équivalentes. Si cet URI existe déjà dans la base de connaissances, la requête précédente a déjà été traitée et les résultats sauvegardés peuvent être exploités tels quels. Sinon, le système génère un graphe RDF (fondé sur l'ontologie *queries* introduite plus haut) représentant la requête pivot et l'insère au moyen d'une requête de mise à jour SPARQL (`INSERT DATA`) dans la base de connaissances avant de commencer l'interprétation de la requête. Le graphe RDF produit pour la requête `?"person": "produce"= "In Utero". "In Utero": "album"` (qui est la requête pivot obtenue par l'interprétation de la requête en langue naturelle "Who produced the album In Utero ?" tiré du jeu d'entraînement de la compétition QALD-3) est montré dans la partie gauche de la figure 6.3. Cette figure montre les données RDF initiales ainsi que les résultats des mises à jour SPARQL qui sont effectuées au cours de l'étape d'appariement (cf. sous-section 6.2.3). Les ressources RDF sont représentées dans des nœuds ovales, les littéraux dans des rectangles, et les propriétés sont logiquement matérialisées par des arcs étiquetés. Pour des besoins de lisibilité, les types des littéraux ne sont pas montrés et les classes auxquelles appartiennent les ressources ne sont montrées que lorsque cela aide à la compréhension ; l'URI de la classe d'un nœud est alors indiquée en dessous de ce nœud.

6.2.3 Appariement des éléments de la requête à la base de connaissances.

La première étape de l'interprétation de la requête pivot consiste à apparier chaque élément de la requête pivot aux entités de la base de connaissances (classes, propriétés et instances) ou aux types de littéraux, en associant à chaque appariement une note de confiance qui représente la qualité supposée de l'appariement et sa vraisemblance.

L'*appariement des mots-clés* est effectué en calculant une mesure de similarité entre les mots-clés de la requête pivot et les étiquettes des ressources de la base de connaissances. Pour cela, nous utilisons *LARQ*[9], une extension de SPARQL proposée par le moteur SPARQL *ARQ*[10] qui permet d'exploiter les capacités d'indexation et de recherche dans du texte du moteur de recherche *Apache Lucene*[11]. Cette extension introduit un nouvel élément de syntaxe qui permet de déterminer

9. http://jena.apache.org/documentation/larq/
10. http://jena.apache.org/documentation/query/
11. http://lucene.apache.org/

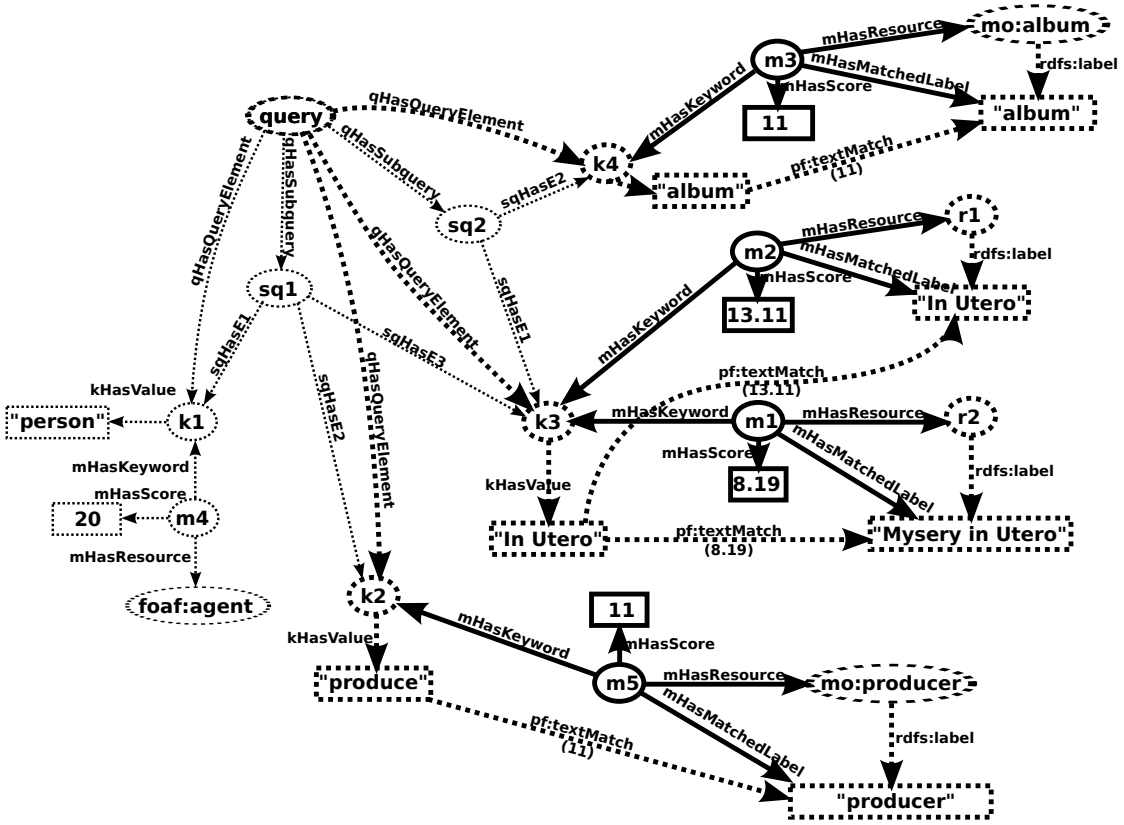


FIGURE 6.3 – L'étape d'appariement sur notre requête pivot exemple.

```
INSERT
{
  ?matchUri a queries:Matching;
            queries:matchingHasKeyword ?keyword;
            queries:matchingHasResource ?r;
            queries:matchingHasScore ?score;
            queries:matchingHasMatchedLabel ?l.
  ?keyword queries:keywordAlreadyMatched "true"^^xsd:boolean.
}
WHERE
{
  <[queryUri]> queries:queryHasQueryElement ?keyword.
  ?keyword a queries:KeywordQueryElement;
           queries:queryElementHasValue ?keywordValue.
  FILTER NOT EXISTS { ?keyword queries:keywordAlreadyMatched "true"^^xsd:boolean. }
  (?l ?score) pf:textMatch (?keywordValue 6.0 5).
  ?r rdfs:label ?l.
  BIND (UUID() AS ?matchUri)
}
```

FIGURE 6.4 – Mise à jour SPARQL utilisée pour apparier les mots-clés de la requête pivot.

les littéraux ressemblant à une chaîne de caractères donnée et une valeur représentant le niveau de ressemblance, appelée *score Lucene* : le "triplet" `(?lit ?score) pf:textMatch '+text'` lie à la variable `?lit` tous les littéraux qui sont similaires à la chaîne de caractères `text` et à la variable `?score` le *score Lucene* correspondant. La requête SPARQL intégrale utilisée pour mettre en œuvre cette étape d'appariement est montrée dans la figure 6.4.

La figure 6.3 montre un sous-ensemble des appariements obtenus par l'éxécution de cette requête et les instances d'appariements (classe `queries:Matching`) générées. Le graphe avant l'exécution de la requête est tracé en pointillés ; la partie de ce graphe qui est appariée au motif de graphe de la clause `WHERE` est en gras, et les ressources et triplets insérés dans la base de connaissances sont en traits pleins. Les figures suivantes utilisent les mêmes conventions graphiques.

Il est important de noter que, bien que cette étape telle que nous l'avons décrite soit mise en œuvre à l'aide d'une extension (non standard) de SPARQL qui la rend peu portable, une alternative peut facilement être implémentée en utilisant des fonctions standard de SPARQL telles que les fonctions de chaînes de caractères `REGEX` et `CONTAINS`, ou encore une simple comparaison de chaînes de caractères. Cette version standard présenterait néanmoins des performances dégradées, l'appariement entre chaînes de caractères étant établi de façon exacte et le temps de réponse des moteurs SPARQL actuels étant très long pour ce type de fonction.

6.2.4 Association des éléments de patron aux éléments de requête

Avant de pouvoir associer l'intégralité des patrons à la requête utilisateur, une première tâche consiste à déterminer pour chaque élément de patron toutes les associations possibles aux éléments de la requête utilisateur et leur note de confiance respective. Ces associations sont appelées *associations d'élément*.

Cette étape consiste à créer une association entre un élément de requête et un élément de patron quand cet élément de requête a été apparié à une ressource qui est liée d'une façon ou d'une autre à l'élément de patron ciblé. Nous définissons plusieurs cas permettant d'établir un lien entre une ressource appariée r et un élément de patron ciblé t :

1. r est une sous-classe de t (ce cas comprend celui où r est la classe t elle-même),
2. r est une sous-propriété de t (ce cas comprend celui où r est la propriété t elle-même),
3. r est une instance de t (ce cas comprend celui des associations d'élément instance-classe, entroduites en 5.2 et dont l'implémentation est décrite un peu plus loins),
4. r fait référence au même type de littéral que t.

À chaque association d'élément est assignée une note de confiance qui est la même que celle de l'appariement impliqué. La figure 6.5 montre l'instanciation de quelques appariements via une requête de mise à jour SPARQL. L'élément de patron `cd_info_element5` qui cible la classe `mo:Record` est associé deux fois au mot-clé `k1` (`"In Utero"`) qui a été préalablement apparié à deux instances de cette même classe. L'élément de patron `cd_info_element4` qui cible la propriété `mo:producer` est associé au mot-clé `k2` (`"produce"`) qui a été préalablement apparié à cette même propriété.

Le cas de propriétés "*Type" L'instanciation de l'association d'élément `elemMap5` est issue d'une extension du premier cas établi plus haut. Cette extension est due à l'observation d'un choix de modélisation récurrent fait par certains développeurs d'ontologies, qui consiste à classifier les instances d'une classe c en définissant une propriété d'objet avec pour domaine c, une classe (énumérée) c' qui représente le co-domaine de cette propriété, et des instances de c' qui représentent les différentes façon de classifier les instances de c. Par exemple, dans l'ontologie *music* [Raimond et al., 2007], les instances de la classe `mo:Record` peuvent être impliquées en tant que sujet dans un triplet avec pour prédicat `mo:releaseType` et pour objet une instance de la classe (`mo:Album`, `mo:Single`,

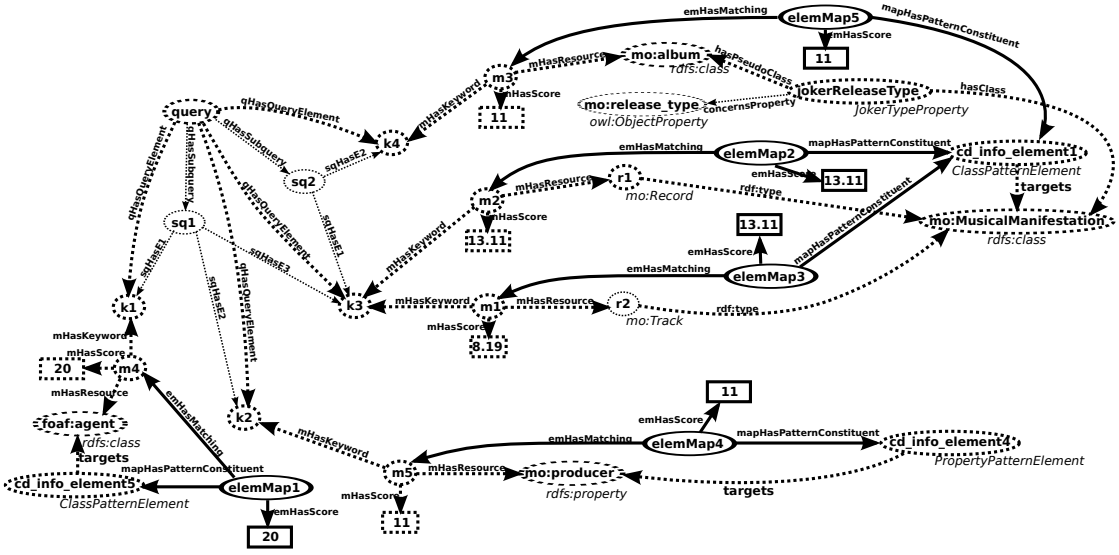


FIGURE 6.5 – L'étape d'association d'élément sur notre requête pivot exemple.

`mo:Live`, `mo:Soundtrack`...). Il nous semble que ce choix, bien qu'il ait probablement été guidé par certaines exigences, n'est pas pertinent car il ignore le mécanisme de classification proposé par RDFS et OWL (à savoir le typage d'instances à l'aide de classes) pour exprimer des connaissances qui sont en fait bel et bien une classification. Nous appuyons notre critique en avançant deux éléments que nous considérons comme symptomatiques de ce travers et que l'on retrouve dans notre exemple :

- deux termes qui ont le même statut dans une phrase en langue naturelle (comme par exemple les termes "songs" et "soundtracks" qui sont utilisés de la même façon dans les requêtes "Give me all songs by Aretha Franklin" et "Give me all soundtracks composed by John Williams," toutes deux extraites du jeu de données de la compétition QALD-3) sont modélisés de manière différente dans l'ontologie, le premier par une classe, le second par une instance de la classe `mo:ReleaseType` ;
- la façon dont les developpeurs de l'ontologie eux-mêmes ont nommé la propriété d'objet mise en cause trahit la nature réelle de cette propriété ; en effet, une propriété dont le nom finit par `Type` sera très probablement utilisée pour typer des instances et devrait donc en tant que telle être une sous-propriété de `rdf:type`, or ce n'est pas le cas ; cette seconde observation est à l'origine du nom que nous avons donné à ce type de propriétés : les propriétés "*Type" (*wildcard type properties* en anglais, d'où l'URI de la classe utilisée plus bas).

Notre approche permet de pallier ce manque de généricité en identifiant explicitement ces cas. Dans la figure 6.5, l'instance `wildcardReleaseType` de type `WildcardTypeProperty` exprime le fait que la ressource `mo:album` doit être considérée durant le processus d'association comme une sous-classe de `mo:Record`, ce qui permet l'association de l'élément qualifiant `cd_info_element1` au mot-clé `k4` ; elle spécifie aussi que la propriété de typage correspondante (qui devra être utilisée dans la requête SPARQL finale) est dans ce cas `mo:release_type` au lieu du classique `rdf:type`.

Les associations d'élément instance-classe L'implémentation que nous proposons gère de façon simple le cas particulier des associations d'élément instance-classe, présenté en 5.2. La figure 6.6 poursuit le même exemple et illustre l'instanciation d'une association d'élément instance-classe. `elemMap1` et `elemMap4` instanciées préalablement associent `cd_info_element1` respectivement à `k3` (`"In Utero"`) et `k4` (`"album"`), et la ressource appariée par `k3` est une instance (lorsque l'on considère la remarque précédente sur les propriétés "*Type") de la ressource appariée par `k4` ; cela permet d'instancier une nouvelle association d'élément `elemMap6` associant l'élément de patron considéré aux deux éléments de requête. Son score est la somme des notes de confiance des deux associations d'élément originellement impliquées.

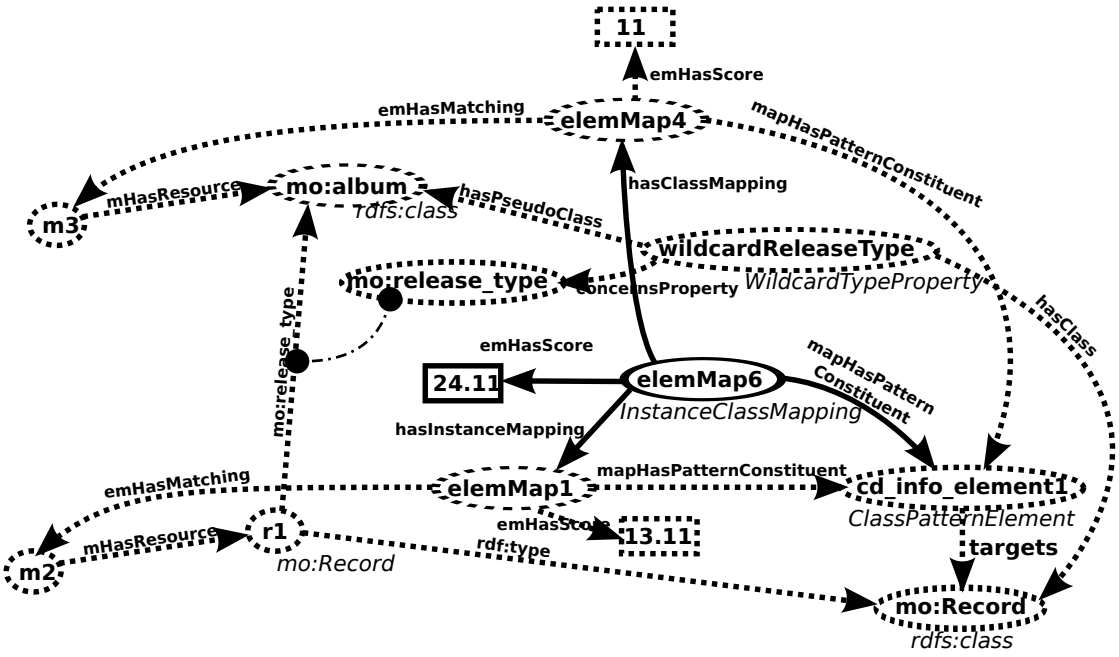


FIGURE 6.6 – Une association d'élément instance-classe sur notre requête pivot exemple

6.2.5 Étapes suivantes

Nous ne détaillons pas autant les étapes suivantes du processus d'interprétation par souci de brièveté. Encore une fois, les requêtes SPARQL utilisées et l'ontologie organisant les triplets générée sont disponibles sur la page d'accueil du système *Swip*.

Comme nous l'avons présenté en 5.3, l'étape d'appariement des sous-patrons implique des opérations complexes, comme des combinaisons d'éléments d'un ensemble ou des produits cartésiens entre des ensembles dont le nombre ne peut être déterminé à l'avance. De plus, l'association d'un patron nécessite de commencer par associer les sous-patrons les plus simples qui ne contiennent aucun autre sous-patron, puis les sous-patrons qui les contiennent directement, et ainsi de suite jusqu'à avoir associé le patron lui-même. Ces besoins sont traditionnellement résolus en programmation impérative par une structure de contrôle de type *boucle* (*tant que*), elle-même permise par une fonctionnalité de saut conditionnel (*si*).

Une simple succession de mises à jour SPARQL ne peut répondre à ces besoins. Nous avons donc dû ajouter la possibilité d'effectuer un saut conditionnel dans notre implémentation. Ce saut conditionnel est mis en œuvre de manière très simple au travers d'une requête SPARQL `ASK` : deux embranchements sont possibles à l'issue de l'exécution d'une telle requête et celui réellement emprunté dépend du résultat de cette exécution. Comme on peut le voir sur la figure 6.2, cette méthode est utilisée à la fin de l'étape d'association des sous-patrons, au niveau de la requête (`ASK`) "*Les patrons ont-ils tous été associés ?*" Si la réponse est non (`FALSE`), l'embranchement suivi renvoit à l'exécution de requêtes en amont dans le processus pour revenir au test plus tard. Si la réponse est oui (`TRUE`), alors on passe aux étapes suivantes. Ainsi, une boucle est créée permettant d'assurer que tous les sous-patrons ont été associés avant de passer à la suite.

6.3 Critique de l'implémentation

6.3.1 Avantages

L'architecture décrite plus haut présente pour nous de nombreux avantages. L'un des plus évidents est la facilité d'utilisation d'un système de cache. En effet, étant donné que le résultat de chaque étape de traitement est inséré dans l'entrepôt RDF, il est alors très simple de réexploiter les données générées précédemment. Par exemple, comme expliqué dans la sous-section 6.2.2, le système Swip se rend compte qu'une requête entrante a déjà été traitée lorsque son URI est déjà présent dans la base de triplets et le résultat de son interprétation peut être directement retourné. De même, l'appariement d'un mot-clé donné n'est réalisé qu'une seule fois pour l'ensemble des requêtes contenant ce même mot-clé.

De plus, cette architecture permet un asynchronisme total et naturel entre le client et le serveur : une fois que l'URI de la requête est construit, il est retourné au client qui peut alors mettre à jour les résultats intermédiaires de l'interprétation en cours progressivement et indépendamment du serveur. Pour cela, il requête directement à l'aide de SPARQL l'entrepôt RDF où sont enregistrés ces résultats.

Cette approche est également homogène et cohérente dans la mesure où les données d'entrée, les données intermédiaires et les données de sortie sont enregistrées et manipulées via des standards. Même la configuration du système et les ajustements sont réalisés de cette manière ; par exemple, les cas identifiés de "*TypeProperties" (présentés plus haut) et certains appariements utiles (c'est-à-dire des appariements de mots-clés au niveau de l'ontologie qui ne pourraient pas être obtenus par mesure de similarité entre chaînes de caractères, comme par exemple entre le mot-clé `"husband"` et la propriété `rel:spouseOf`) sont directement exprimés en RDF, insérés dans la base de triplets et exploités tels quels au cours du processus d'interprétation.

Nous attirons également l'attention du lecteur sur le fait que, bien que la première grande étape du processus d'interprétation (traduction de la langue naturelle vers le langage pivot) ait été implémentée d'une façon plus "traditionnelle," en utilisant des services web, cela pourrait maintenant être fait différemment en exploitant la récente initiative *NLP2RDF* [Hellmann et al., 2013] qui veut fournir un format commun pour les données de sortie des outils de traitement automatique du langage les plus populaires, ce format appelé *NIF (NLP Interchange Format)* étant totalement intégré au cadre du web sémantique.

Enfin, bien que l'implémentation ne le permette pas dans son état actuel, l'architecture utilisée serait facilement adaptable afin de déployer *Swip* de façon distribuée, ce qui faciliterait le passage du système à grande échelle. En effet, pour fonctionner, *Swip* a simplement besoin d'avoir accès via SPARQL aux données à interroger et aux patrons de requêtes. Grace aux récentes fonctionnalités de fédération de SPARQL (*federated SPARQL* [Prud'hommeaux and Buil-Aranda, 2013]), les données peuvent être regroupées sur un seul serveur SPARQL ou réparties sur plusieurs, et les patrons peuvent eux aussi être regroupés ou répartis, rassemblés avec la base de connaissances qu'ils concernent ou isolés. Cette architecture exploitant des standards est donc très souple et permet de nombreuses variations. De plus, ce sont les serveurs SPARQL qui effectuent les traitements, ce qui rend le programme initial (celui qui émet le requêtes SPARQL) très léger. On peut ainsi imaginer que ce programme, exécuté côté client, met en branle, via la fédération de SPARQL, de nombreux serveurs répartis sur le web et les orchestre pour traiter l'interprétation d'une requête.

6.3.2 Inconvénients

Nous retenons également deux inconvénients majeurs qui viennent ternir le tableau. Le premier est le manque de contrôle sur l'exécution des requêtes SPARQL et par conséquent sur les performances générales du système. Le serveur SPARQL est utilisé comme une boîte noire et son efficacité influence directement celle du processus d'interprétation. L'expérience a montré que le serveur ARQ n'est pas performant pour traiter de grosses requêtes (plus de vingt triplets répartis dans plusieurs sous-requêtes) et que la division de ces requêtes en une série équivalente de requêtes successives permet une nette amélioration des performances.

De plus, SPARQL est encore une recommandation relativement jeune qui, malgré les nouvelles fonctionnalités apportées par SPARQL 1.1, propose un panel de fonctions assez limité. Par exemple, il y a très peu de fonctions arithmétiques ; seules les plus basiques sont supportées, ce qui n'est pas le cas de la fonction puissance. En conséquence, une solution dégradée a dû être trouvée pour le calcul de la note de pertinence finale qui impliquait initialement cette fonction.

6.4 Présentation de l'interface

Nous avons également implémenté une interface permettant d'exploiter le prototype présenté dans ce chapitre. Elle est composée de deux parties principales : l'interface de requête et l'interface de visualisation des patrons.

L'interface de requête [12] permet de tester le système en lui-même, en lui soumettant (soit en langue naturelle, soit en langage pivot) des requêtes à interpréter. Cette interface n'est pas destinée à un utilisateur final, mais plutôt à observer le comportement du prototype car elle permet d'afficher les résultats de chacune des sous-étapes du processus d'interprétation (l'interface interroge régulièrement le serveur SPARQL acteur de l'interprétation pour se tenir au courant de l'avancemenent du traitement et pour afficher les derniers résultats disponibles). On peut aussi connaître les URI de chacune des ressources produites par l'interprétation et effectuer des requêtes plus précises sur le endpoint [13] SPARQL associé. Seule la partie affichant les phrases descriptives est proche de ce que pourrait voir un utilisateur dans un système final. Une capture d'écran de cette interface est montrée dans la figure 6.7.

L'interface de visualisation des patrons [14] est destinée à devenir une véritable interface d'administration, au sein de laquelle il serait possible non seulement de visualiser mais aussi d'éditer des patrons, de vérifier leur cohérence et de régler les

12. http://swip.univ-tlse2.fr/SwipWebClient/
13. http://swip.univ-tlse2.fr:8080/control-panel.tpl
14. http://swip.univ-tlse2.fr/SwipWebClient/patternViewer.html

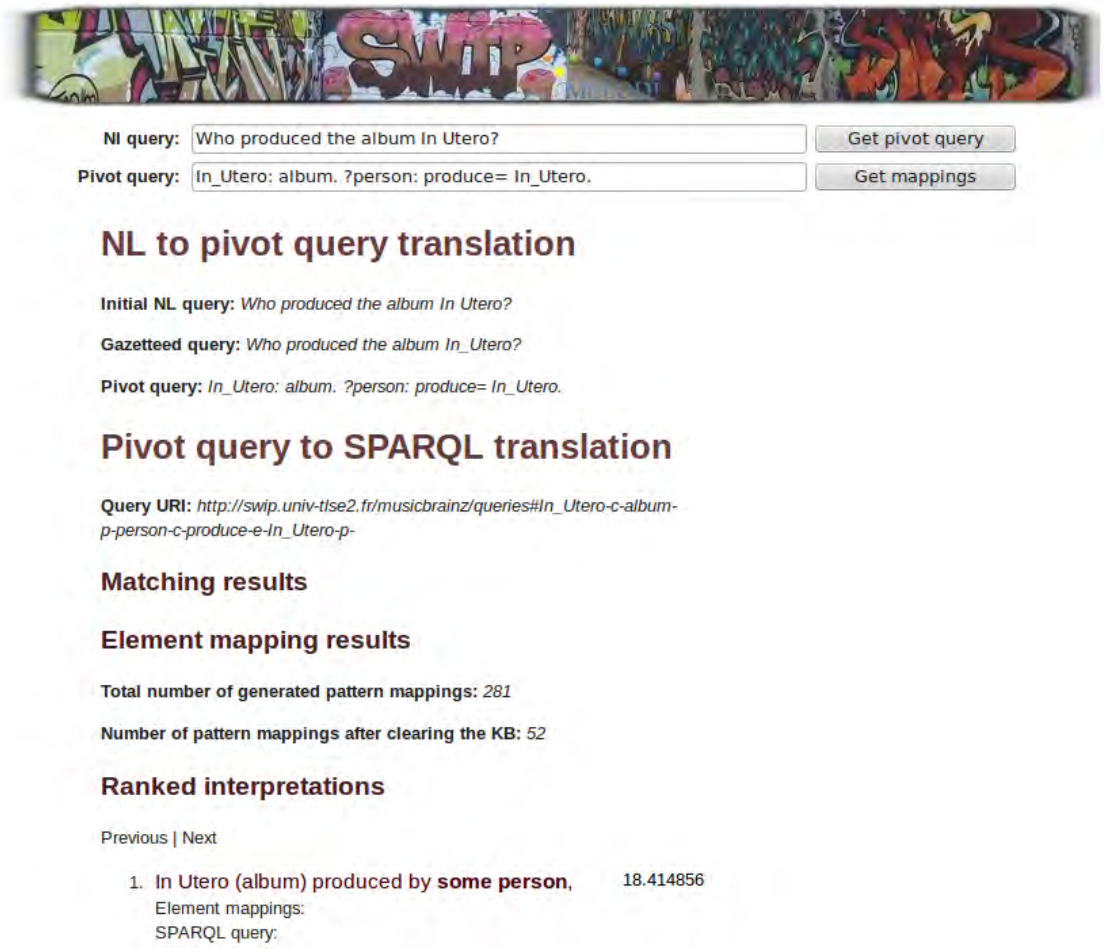


FIGURE 6.7 – Interface de requête

autres paramètres du système (appariements utiles, propriétés "*type", cache, etc.) Le développement de cette partie n'est pour l'instant que peu avancé et l'interface ne permet à ce jour que de visualiser les patrons, comme le montre la figure 6.8.

6.5 Conclusion

Ce chapitre a présenté le prototype implémentant notre approche. Celui-ci adopte une architecture novatrice pour effectuer l'interprétation de la requête pivot, dans le sens où l'ensemble des traitements de cette étape sont ramenés à des opérations d'appariements de graphes, qui sont effectuées par un serveur SPARQL et dont les résultats (temporaires et finals) sont enregistrés sur ce même serveur.

Ce prototype a été utilisé pour mener les expérimentations décrites dans le

chapitre suivant.

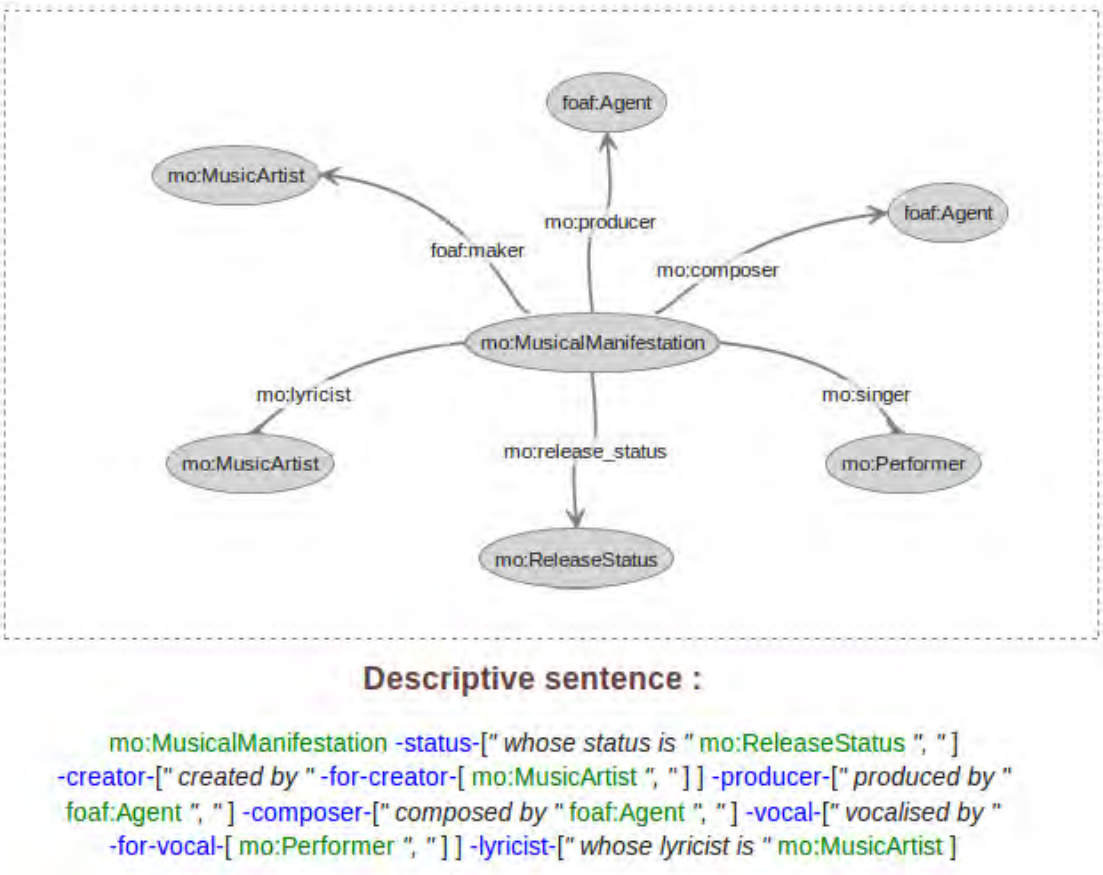


FIGURE 6.8 – Interface de visualisation des patrons

Chapitre 7

Évaluations

Nous avons mené différentes expérimentations dans le but de valider notre approche. Les résultats de ces expérimentations sont exposés dans ce chapitre. Les sections 7.1, 7.2 et 7.3 présentent les trois principales séries d'expérimentations menées au cours de notre travail : une évaluation maison sur le domaine du cinéma et les compétitions QALD-1 et QALD-3. La section 7.4 conclut ce dernier chapitre.

Les deux premières évaluations sont régulièrement utilisées dans nos publications pour appuyer notre démarche. Les résultats obtenus à la compétition QALD-3 font l'objet d'une description détaillée dans [Pradel et al., 2013b].

7.1 Évaluations "maison" sur le domaine du cinéma

Au début de notre travail, nous avons dû faire face à l'absence de cadre d'évaluation pour tester notre système. Nous avons donc construit notre propre cadre d'évaluation. 160 requêtes distinctes concernant le cinéma d'un point de vue culturel et artistique ont été recueillies auprès de 24 personnes, chaque requête étant composée d'un ensemble de mots-clés et d'une phrase en langage naturel exprimant le besoin en information. Nous avons également construit une ontologie sur le domaine du cinéma [Pradel et al., 2012b] que nous avons peuplée de façon à obtenir une base de connaissances contenant la réponse aux requêtes recueillies.

Les résultats présentés ici ont été obtenus avec un premier prototype du système *Swip* [Pradel et al., 2011a,b, 2012a]. Celui-ci n'intégrait pas encore l'étape de traduction de la requête LN en requête pivot (l'utilisateur devait obligatoirement exprimer sa requête en langage pivot) et les patrons de requêtes n'intégraient pas encore la notion de sous-patron (ni celles de cardinalité et de sous-patron optionnel ou répétable).

Parmi les 160 requêtes à notre disposition, 120 ont été sélectionnées aléatoirement et utilisées pour construire manuellemment six patrons de requêtes. L'évaluation du système a porté sur les 40 requêtes restantes.

L'enjeu principal de ces premières expérimentations était la confirmation ou l'infirmation de l'hypothèse à l'origine de nos travaux, hypothèse selon laquelle les requêtes exprimées par des utilisateurs sur un domaine donné peuvent être regroupées en un nombre restreint de grandes familles de requêtes (incarnées dans notre approche par les patrons de requêtes). Nous avons constaté que 37 des 40 requêtes de test sont couvertes par l'ensemble de patrons construit, c'est-à-dire que leur graphe requête peut être généré à partir du graphe d'au moins un des six patrons de l'ensemble. Nous considérions alors ce taux de couverture des requêtes de test par les patrons de 92.5% comme suffisament élevé pour corroborer notre hypothèse et nous pousser à poursuivre nos travaux dans cette direction.

Pour évaluer les performances du système (c'est-à-dire l'évaluation de la qualité des interprétations), trois utilisateurs différents ont tenté de formuler chacune de ces requêtes en utilisant le langage pivot. Nous avons ainsi obtenu trois formulations de chaque requête qui présentaient le plus souvent quelques différences, liées à l'aspect subjectif de l'utilisation du langage pivot. Nous avons ensuite calculé pour chaque requête la valeur $\frac{1}{r}$, où r correspond à la position de la requête idéale dans les résultats retournés par le système (mesure appelée *Mean Reciprocal Rank*), puis nous avons réalisé la moyenne de ces valeurs. En effectuant ce test sur chacun des 120 exemples (3 formulations pour chacune des 40 requêtes sélectionnées), nous avons obtenu un score moyen de 0,81, ce qui était sensiblement meilleur que certaines des approches faisant référence à ce moment-là [Zhou et al., 2007; Tran et al., 2009]. De plus, la requête idéale apparaissait dans une des trois premières propositions pour 89% des cas.

Dans ces expérimentations, les résultats sur la couverture des requêtes utilisateur par les patrons sont particulièrement significatifs. En effet, le jeu de données utilisé comporte un nombre de requêtes significatif, obtenues auprès de nombreuses personnes. De plus, hormis le domaine (qui était volontaire vague), aucune contrainte portant sur la formulation des requêtes n'était spécifiée. Les jeux de données utilisés pour les expériences suivantes ne présentent pas une telle variété.

En revanche, la pertinence de l'évaluation des performances du système peut être mise en doute, le jeu de données cible ayant été construit par nos soins et sur la base des questions recueillies. La compétition QALD nous offre un cadre d'évaluation objectif et rigoureux pour évaluer les performances de *Swip* et les comparer à celles d'autres systèmes. Nous avons donc par la suite participé aux première et troisième éditions de cette compétition.

7.2 La compétition QALD-1

Nous avons donc pris part à la première édition de la compétition QALD [Lopez et al., 2013] en 2011 : QALD-1 [1]. Nous avons uniquement concouru au sous-ensemble de la compétition concernant un export RDF de la base de connaissances Musicbrainz [2], sur le domaine de la musique et comportant plus de 50 millions de triplets. Nous avons construit les patrons de requêtes à partir de 50 requêtes d'entraînement, fournies par les organisateurs sous la forme de requêtes en langue naturelle et leurs traductions en SPARQL, puis nous avons, sur la base de ces patrons, procédé à la traduction automatique de 50 requêtes de test, différentes des précédentes et fournies uniquement en langue naturelle.

Pour chaque requête de test, le classement de l'interprétation que nous considérions comme la meilleure parmi les autres interprétations générées par *Swip* était communiqué aux organisateurs mais n'a pas été pris en compte dans l'évaluation, cette information étant spécifique à notre système. Nous nous sommes cependant interdit de considérer les interprétations au-delà des trois premières retournées par le système.

La méthode d'évaluation était définie par les organisateurs de la compétition. Elle consiste à calculer, pour chaque requête du jeu de test, la précicion, le rappel et la F-mesure de la traduction SPARQL retournée pour chaque système participant, en la comparant à une requête de référence rédigée à la main (*gold standard*). Les résultats sont résumés dans le tableau suivant qui présente, pour *Swip* et pour *FREyA*, l'autre système ayant concouru sur le même jeu de données, le nombre total de requêtes que les systèmes devaient interpréter, le nombre de requêtes effectivement traitées, le nombre de requêtes correctement interprétées (les requêtes dont la traduction SPARQL a donné exactement les même résultats que la requête SPARQL de référence), le nombre de requêtes mal interprétées (toutes les autres requêtes, y compris celles qui étaient partiellement bien interprétées), et les moyennes des valeurs de rappel, de précision et de F-mesure.

	total	processed	right	wrong	recall	precision	f-measure
FREyA	50	41	30	11	0.6	0.73	0.66
Swip	50	35	28	7	0.56	0.8	0.66

On observe que *Swip*, avec un rappel plus faible et une plus grande précision, obtenait la même valeur de F-mesure que son concurrent. Le faible rappel était principalement dû à un nombre important de requêtes non supportées (comme celles impliquant des agrégats). Nous ne décrivons pas plus ces résultats car la section suivante détaille les résultats de notre participation en 2013 à la troisième

1. http://www.sc.cit-ec.uni-bielefeld.de/qald-1
2. http://musicbrainz.org/

édition de ce même challenge, plus pertinents car plus récents et impliquant un prototype du système *Swip* qui implémente l'intégralité de l'approche décrite dans ce travail (y compris la traduction de la langue naturelle vers le langage pivot et la prise en compte des sous-patrons).

7.3 La compétition QALD-3

Notre participation à la compétition QALD-3 [3][Cabrio et al., 2013] représente à ce jour l'évaluation la plus récente et la plus complète de notre approche. Nous avons concouru dans le cadre de la tâche 1 de la compétition (cette édition introduisant une nouvelle tâche sur la lexicalisation d'ontologies à laquelle nous n'avons pas pris part). Nous avons cette fois considéré les deux jeux de données proposés par les organisateurs : l'export RDF de Musicbrainz [4] intégrant l'ontologie *music* [Raimond et al., 2007] (ce qui n'était pas le cas dans QALD-1), et *DBpedia* [5], la base de triplets la plus populaire du web de données. Bien que cette tâche proposait des questions en langue naturelle en plusieurs langues, nous avons considéré uniquement les questions exprimées en anglais. La qualité des résultats varie en fonction de la base de connaissances ciblée.

7.3.1 Aperçu des résultats

Le jeu de données test de Musicbrainz était constitué de 50 requêtes en langue naturelle. Sur les 50, nous en avons traité 33. Le système *Swip* en a interprété correctement 24, n'a trouvé qu'une réponse partielle pour 2 d'entre elles, et a échoué sur les autres. Les valeurs moyennes de précision, de rappel et de F-mesure, calculées par les organisateurs de la compétition, sont toutes égales à 0,51. Il est difficile d'évaluer ces performances étant donné qu'aucun autre participant ne s'est attaqué au jeu de données Musicbrainz. Cependant, ces résultats sont assez bons quand on les compare à ceux obtenus par les autres participants sur le jeu de données DBpedia. En effet, pour ce jeu de données, le système *squall2sparql* [Ferré, 2012, 2013b,a] surpasse de très loin les autres participants, mais, comme expliqué au chapitre 2, ce système n'accepte pas directement des questions en langue naturelle. Si l'on exclut *squall2sparql*, la meilleure valeur de F-mesure est seulement de 0,36. Bien sûr, la pertinence d'une comparaison aussi directe est très critiquable, étant donné que les deux bases de connaissances présentent des propriétés signi-

3. http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=task1&q=3
4. http://musicbrainz.org/
5. http://dbpedia.org

ficativement différentes et que notre système, pour des raisons qui sont exposées plus tard, a obtenu des résultats faibles sur DBpedia.

Les résultats obtenus sur le jeu de données DBpedia, composé de 100 questions de test, sont bien moins satisfaisants. Nous avons traité 21 questions parmi lesquelles 14 ont été interprétées correctement et 2 partiellement. Les valeurs de précision (0,16), de rappel (0,16) et de F-mesure (0,16) sont faibles.

7.3.2 Évolution depuis QALD-1

Ces résultats peuvent être comparés à ceux que nous avions obtenus deux ans auparavant, lors de la participation à la première édition de la compétition QALD (exclusivement sur le jeu de données Musicbrainz). Cette fois encore, nous ne pouvons pas comparer directement les chiffres pour deux raisons :

- la méthode d'évaluation était alors légèrement différente de celle mise en place pour QALD-3,
- et l'export RDF de Musicbrainz n'intégrait pas encore l'ontologie *music*, comme il le fait actuellement, mais un schéma qui était une traduction basique du schéma relationnel utilisé par la base de données Musicbrainz.

Si l'on applique la nouvelle méthode d'évaluation aux résultats obtenus par *Swip* à QALD-1, on obtient une valeur moyenne de précision (resp. rappel, F-mesure) de 0,59 (resp. 0,59, 0,58). Les nouveaux résultats peuvent donc paraître plus décevants que les précédents, mais il est important de prendre en compte le fait que l'étape de traduction de la langue naturelle vers le langage pivot décrit en 4.4 est maintenant automatique, et non plus réalisée à la main comme c'était le cas lors de notre participation à QALD-1. Étant donné que la tâche d'interprétation de la langue naturelle est très complexe, nous considérons que ces nouveaux résultats sont très encourageants.

7.3.3 Construction et couverture des patrons de requêtes

Les patrons de requêtes ciblant Musicbrainz utilisés pour cette évaluation ont été construits en adaptant (pour les accorder avec la nouvelle ontologie) et mettant à jour (pour prendre en compte les 50 nouvelles questions d'entraînement) ceux créés pour QALD-1. Nous avons ainsi obtenu cinq patrons de requêtes qui peuvent être visualisés via l'interface utilisateur de *Swip*. Parmi les 50 questions de test, seulement deux nécessitaient des graphes requêtes qui n'auraient pas pu être générés par l'un des patrons, sans prendre en compte les requêtes classées hors champ (*out of scope*) par les organisateurs, auxquelles la base de connaissances ne pouvait de toute façon pas répondre. Ce taux de couverture élevé montre que notre approche est adaptée pour des bases de connaissances portant sur un domaine fermé et dont les données sont consistantes avec le schéma.

Pour DBpedia, les patrons ont également été construits à partir des requêtes d'entraînement de la compétition. Cependant, seulement 19% des requêtes de test étaient couvertes par ces patrons. On est loin des 96% obtenus pour le jeu de données Musicbrainz.

Le principal problème que nous avons rencontré était le non respect des schémas selon lesquels les données sont censées être organisées. En effet, la plus grande partie de la base de données DBpedia est générée automatiquent à partir d'informations éditées par des milliers de contributeurs, et non pas exportée depuis une base de données consistante comme Musicbrainz. Par conséquent, nous avons dû faire face à l'hétérogénéité et l'inconsistance de la base de données cible que *Swip* n'a pas été capable de surmonter. Par exemple, le surnom d'une entité peut être exprimé dans la base de connaissances par quatre propriétés distinctes (`dbo:nickname`, `dbo:nicknames`, `dbp:nickname` et `dbp:nicknames`). Autre exemple, dans la traduction SPARQL de la question "Which states of Germany are governed by the Social Democratic Party ?" présentée dans le listing 11, la valeur de la propriété `dbp:rulingParty` peut être soit un URI faisant référence au parti social démocrate allemand, soit un littéral simple.

Listing 11

```
SELECT DISTINCT ?uri
WHERE {
  ?uri rdf:type yago:StatesOfGermany .
  { ?uri dbp:rulingParty 'SPD'@en. }
   UNION
  { ?uri dbp:rulingParty
      res:Social_Democratic_Party_of_Germany. }
}
```

7.3.4 Requêtes non supportées

Comme annoncé plus haut, 17 requêtes visant Musicbrainz et 81 visant DBpedia n'ont pu être traitées. Nous avons identifié certaines catégories dont l'expressivité dépasse celle qui est gérée actuellement par notre approche. La majorité des requêtes non supportées contiennent des agrégats, comme "Which bands recorded more than 50 albums ?" Pour l'instant, l'unique fonction d'agrégat supportée par *Swip* est le comptage du nombre total de résultats d'une requête, dans "How many singles did the Scorpions release ?", car c'est un cas facile à identifier dans la question en langue naturelle et il semblerait que ce soit l'agrégat le plus fréquent dans les requêtes utilisateur (du moins dans celles de la compétition QALD).

De plus, *Swip* n'est pas non plus capable de gérer les requêtes qui nécessitent des inférences réalisées en amont de la requête et faisant appel à des connaissances externes, dans "Which artists turned 60 on May 15, 2011 ?", ni les requêtes impliquant des comparaisons de chaînes de caractères, dans "Give me all bands whose name starts with Play."

Bien sûr, de l'ajout du support de ces catégories résulterait une amélioration significative de nos résultats à la compétition, et la conception de ces nouvelles fonctionnalités s'inscrit naturellement dans la liste de nos travaux futurs, à commencer par le support des agrégats qui nous paraissent être les plus importants dans l'expression des besoins utilisateurs.

7.3.5 Les requêtes dont l'interprétation a échoué

Nous détaillons ici quelques requêtes dont l'interprétation a échoué. Nous expliquons pourquoi elles ont échoué et identifions quelques défauts dans l'implémentation actuelle du système *Swip* ainsi que des imperfections dans les jeux de données de la compétition.

Musicbrainz

Pour traduire en SPARQL les questions 15 "Who composed the song Coast to Coast ?" et 48 "Give me all soundtracks composed by the Pet Shop Boys," *Swip* a fait usage de la propriété `mo:composer` qui, dans l'absolu, nous paraît traduire le besoin en information de la requête utilisateur de façon plus appropriée que la propriété `foaf:maker` utilisée dans la requête de référence. Cependant, la requête SPARQL ainsi générée ne retourne aucun résultat et est donc considérée comme fausse.

Un autre cas de requêtes échouées qui ont en fait été correctement interprétées par le système sont les questions 19 "With whom did Phil Collins work together ?", 50 "On which singles did Robbie Williams collaborate with Nicole Kidman ?" et 51 "Did Kylie Minogue ever collaborate with Mariah Carey ?" Ces requêtes impliquent des parties de la base de connaissances qui ne sont pas consistantes avec le vocabulaire concerné (*Relationship*[6]). En effet, l'export RDF de Musicbrainz (et par conséquent, les requêtes de référence) emploie à mauvais escient la propriété `rel:collaboratesWith` qui, conformément au vocabulaire *Relationship*, n'est pas supposée être utilisée avec un nœud anonyme intermédiaire (c'est une propriété symétrique avec pour domaine et co-domaine `foaf:Person`).

Pour la question 46 "Give me the titles of all singles by Phil Collins," *Swip* a généré une requête SPARQL qui retourne les ressources elles-mêmes et non leur titre. En effet, *Swip* ignore les références aux étiquettes (ou toute autre représentation lexicale d'une ressource, comme les titres ou les noms) pour deux raisons principales :

– nous pensons que les valeurs de ces propriétés sont naturellement utilisées en langue naturelle sans spécifier la nature de la propriété (on utilise par

6. http://vocab.org/relationship/collaboratesWith.html

exemple plus volontiers “Michael Jackson” que “somebody whose name is Michael Jackson”) ;
– l'interface utilisateur de *Swip* utilise de toute façon les étiquettes pour afficher les résultats à l'utilisateur (après tout, n'est-ce pas le rôle des étiquettes ?)

L'interprétation des questions 32 “How many pieces of work did Mozart create ?” et 39 “Does the song Peggy Sue appear on Buddy Holly's Greatest Hits compilation ?” a également échoué à cause de l'inexactitude des résultats de l'étape d'appariement. Dans la requête 32, le mot-clé `Mozart` a été apparié à la ressource ayant cette même chaîne de caractères comme étiquette, et la ressource ayant pour étiquette “Wolfgang Amadeus Mozart” a été ignorée car les chaînes de caractères ont été considérées trop éloignées l'une de l'autre. Dans la question 39, l'intégralité de la chaîne de caractères “Buddy Holly's Greatest Hits” a été reconnue comme une entité nommée et a été appariée à la ressource ayant cette même chaîne de caractères comme étiquette, ce qui a mené finalement à une requête SPARQL ne retournant aucun résultat ; la requête de référence impliquait une ressource ayant pour étiquette “Greatest Hits” et pour valeur de `foaf:maker` une autre ressource ayant pour étiquette “Buddy Holly.” Pour dépasser ces ambiguïtés, nous devons travailler à améliorer cette étape d'appariement qui devrait se montrer plus flexible et pourrait éventuellement prendre en compte la popularité des ressources (ainsi un grand et célébrissime compositeur pourrait prendre l'ascendant sur un obscur groupe de musique).

L'interprétation de la question 11 “How long is Louder Than Words by Against All Authority ?” a échoué à cause d'une simple coquille dans le patron concerné. En effet, le patron utilisait la propriété `duration` de l'ontologie *music*, au lieu de celle de l'ontologie *timeline* qui est celle qui est utilisée dans la base de connaissances. Cette erreur a été introduite lors de l'adaptation des patrons de requêtes à la nouvelle ontologie. La question a en fait été interprétée correctement mais la requête SPARQL générée faisait appel à une propriété qui n'existait pas. Ce type d'erreur pourrait être facilement évité en étendant l'outil de visualisation des patrons avec des fonctionnalités d'édition et de vérification.

Enfin, l'interprétation de la question 5 “How many tracks does Erotica have ?” a été considérée comme correcte, car en accord avec la requête SPARQL de référence montrée dans le listing 12, mais en fait ni la requête générée ni la requête de référence ne sont correctes. En effet, chacune de ces requêtes retourne 40, le nombre total de pistes dans tous les albums nommés “Erotica”. Dans ce cas, plusieurs instances du même album de Madonna sont considérées et chaque piste est donc comptabilisée plusieurs fois. De plus, les pistes d'un autre album nommé “Erotica” sont aussi prises en compte. Tous ces problèmes deviennent évidents lorsqu'on exécute la requête du listing 13. Cette requête retourne (logiquement) 40 résultats, impliquant 40 URI distincts pour la variable `?track`, mais l'on se rend compte que

certaines pistes ont le même titre et ne sont pas issues du même album.

Listing 12

```
SELECT COUNT(DISTINCT ?track)
WHERE {
  ?album dc:title 'Erotica' .
  ?album mo:track ?track .
}
```

Listing 13

```
SELECT *
WHERE {
  ?album dc:title 'Erotica' .
  ?album mo:track ?track .
  OPTIONAL {?album foaf:maker ?maker .
            ?maker foaf:name ?makerName .}
  OPTIONAL {?track dc:title ?trackTitle .}
}
```

DBpedia

En ce qui concerne le jeu de donnée de DBPedia, nous nous contentons ici d'exprimer quelques observations à propos des requêtes dont l'interprétation a échoué. La traduction SPARQL de la question "Who is the mayor of Berlin ?" ne retourne aucun résultat car elle fait appel à la propriété `dbo:mayor` au lieu de `dbo:leader` qui est utilisé dans la requête SPARQL de référence. Ici encore, *Swip* a utilisé une propriété qui est plus proche du besoin utilisateur exprimé dans la requête en langue naturelle mais qui n'est malheureusement pas employée dans les données RDF.

La requête SPARQL issue de l'interprétation de "Who is the husband of Amanda Palmer" ne retourne aucun résultat car elle utilise la propriété `dbo:spouse` et non pas `dbp:spouse` qui est la seule propriété à donner des résultats pour cette requête.

Amélioration des résultats

Parmi les questions des jeux d'entraînement et de test de QALD-3, certaines sont hors champ (*out of scope*), c'est-à-dire qu'on ne peut y répondre en exploitant uniquement la base de connaissances concernée. Notre système ne gére pas directement la détection des requêtes hors champ, mais lorsque toutes les interprétations retournées pour une requête donnée ont une note de pertinence basse, cela donne un indice fort à l'utilisateur, dont le doute est immédiatement renforcé par les phrases descriptives générées qui ne représentent très certainement pas le sens de sa requête originale.

Lors de l'évaluation, nous n'avons pas essayé de répondre aux questions hors champ. Or, selon les organisateurs, l'interprétation correcte d'une telle question

est simplement une requête SPARQL qui ne retourne aucun résultat. Il aurait donc été facile pour nous de valider les questions hors champ des jeux de tests (en considérant par exemple la première interprétation retournée par le système) et d'augmenter ainsi le score de notre système. Certes, cette augmentation est artificielle et ne change rien aux qualités et limites réelles de l'approche, mais elle permet certainement une meilleure mise en perspective des performances de *Swip* par rapport à celles des autres systèmes, ces derniers ayant eux aussi probablement été adaptés aux règles fixées par la compétition.

Nous avons réévalué les interprétations des requêtes portant sur Musicbrainz (grâce à l'outil d'évaluation [7] laissé à disposition par les organisateurs), en répondant cette fois à toutes les requêtes, y compris les requêtes hors champ, afin de connaître les résultats que nous aurions obtenus en prenant en compte ces caractéristiques de la méthode d'évaluation. Nous avons ainsi pu valider les quatre requêtes hors champ du jeu de test. Nous avons également corrigé au préalable la coquille dans le patron qui a empêché l'interprétation correcte de la requête 11, cet impair relevant plus d'une maladresse de notre part que d'une réelle faiblesse de l'approche. Après ces modifications, nous obtenons des valeurs moyennes de précision, de rappel et de F-mesure de 0,61.

7.4 Conclusion

Les expérimentations décrites dans ce chapitre avaient deux objectifs : valider empiriquement la pertinence de l'idée d'utiliser des patrons pour représenter des grandes familles de besoins en information utilisateur et comparer les performances de notre approche à celle des autres systèmes.

Les résultats des analyses menées sur des jeux de requêtes recueillies auprès d'utilisateurs finals ou issues des données de la compétition QALD montrent que notre hypothèse d'origine est cohérente : il est effectivement possible, pour un domaine donné, de couvrir la grande majorité des besoins utilisateur avec un nombre assez restreint de patrons.

De plus, le système *Swip* a réalisé de bonnes performances sur des bases de connaissances consistantes, spécifiques à un domaine et respectant les schémas selon lesquels elles sont organisées. Notre approche fondée sur des patrons de requêtes est par conséquent bien adaptée à ce type de bases de connaissances. En revanche, les résultats observés sur DBpedia, une base de connaissances contenant beaucoup d'inconsistances et transgressant régulièrement les schémas qui l'accompagnent, sont faibles. L'approche devrait donc évoluer pour s'adapter à ce type de bases de connaissances.

7. http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=evaltool&q=3

Conclusion

Le web sémantique émerge peu à peu du web documentaire. Une quantité toujours plus importante de connaissances, organisées selon des standards adaptés au traitement par des machines, est rendue accessible, mais le problème de l'accès à ces connaissances pour l'utilisateur final n'est toujours pas résolu. Notre travail est une contribution à la résolution de ce problème.

Ce mémoire de thèse détaille l'approche que nous avons proposée dans le but de traduire automatiquement des requêtes LN en SPARQL. La principale originalité de cette approche est l'exploitation de patrons de requêtes modulaires qui assurent que l'interprétation qui est faite de la requête est cohérente du point de vue de l'utilisateur. Des détails sont donnés sur l'implémentation et une évaluation rigoureuse a montré la pertinence de l'approche.

Les apports de cette thèse se trouvent tant au niveau scientifique qu'au niveau pratique. Nous avons, par l'étude de la littérature analysant les besoins en information d'utilisateurs finals, montré le bien-fondé de l'utilisation de patrons de requêtes pour représenter les familles de requêtes liées à un domaine et identifier le besoin en information exprimé dans des requêtes LN. Nous avons également donné une définition de patrons de requêtes modulaires, formés de sous-patrons optionnels ou répétables, et défini une ontologie OWL permettant de les représenter au format RDF dans le but de faciliter leur construction, leur évolution, leur partage et leur exploitation.

Nous avons introduit et défini la requête pivot, résultat intermédiaire de notre processus d'interprétation, qui permet de faciliter la mise en œuvre du multilinguisme dans notre approche, de prendre en compte les relations exprimées entre les termes de la requête utilisateur et de rendre l'approche plus résiliente à des outils d'analyse syntaxique faillibles.

Les deux grandes étapes du processus de notre approche ont été détaillées. La première exploite des outils de traitement automatique des langues pour extraire de la requête LN exprimée par l'utilisateur les mots-clés et les relations entre ces mots-clés, puis traduit ces résultats en une requête pivot. La seconde associe des patrons de requêtes préétablis à cette requête pivot en tirant le meilleur parti de la modularité de ces patrons.

Le prototype *Swip* (*Semantic Web Interface using Patterns*) constitue la mise en œuvre de cette approche. Le module principal de ce prototype présente un fonctionnement innovant qui révèle le potentiel de la représentation des patrons en RDF. L'intégralité des calculs nécessaires à l'interprétation d'une requête est déléguée à un serveur SPARQL contenant les données à interroger et les patrons de requêtes au format RDF. L'ensemble du processus est donc ramené à une succession d'opérations d'appariement de graphes, traduites en requêtes de mise à jour SPARQL et traitées par ce serveur.

Les résultats des expérimentations menées attestent nos choix et montrent les qualités du système. Nous avons en effet pu observer qu'un nombre restreint de patrons permet de couvrir la grande majorité des besoins en information formulés par un utilisateur final. De plus, le système *Swip* est fonctionnel et a réalisé de bonnes performances lors de nos participations aux première et troisième éditions de la compétition QALD.

Nous prévoyons d'étendre notre travail dans différentes directions. À court terme, nous voulons expérimenter les capacités du système à s'adapter à différentes langues. Pour cela, nous avons prévu de participer à la prochaine édition de la compétition QALD au moins en français et en anglais, le passage de la langue naturelle au langage pivot ayant déjà été implémenté dans ces deux langues. Nous développons également un partenariat avec l'*Institut national de recherche en sciences et technologies pour l'environnement et l'agriculture* (*IRSTEA*) dans le but de mettre en place un cadre d'application réel visant à rendre compte des observations sur le développement des plantes et plus particulièrement sur les attaques dont elles sont victimes.

Notre approche et son implémentation mériteraient quelques ajouts et améliorations de fonctionnalités. Nous voudrions notamment améliorer l'étape d'appariement des éléments de la requête à la base de connaissances, afin d'augmenter leur pertinence et de limiter leur nombre. Les appariements de termes faisant référence à des propriétés échouent régulièrement parce que les propriétés n'ont pas d'étiquettes associées dans les ontologies ciblées ou parce qu'elles sont exprimées dans les requêtes LN de façon complètement différente. À notre connaissance, le travail présenté dans [Cabrio et al., 2012] est à ce jour le seul à aborder ce problème.

Le développement d'heuristiques permettant d'orienter la génération des associations de sous-patrons et de diminuer la complexité de cette étape serait également profitable au système. En effet, la grande majorité des très nombreuses associations de requêtes obtenues à la fin du processus d'interprétation ne présente pas d'intérêt et obtient d'ailleurs le plus souvent une note de pertinence très faible. Nous voudrions établir certains critères permettant de tronquer l'ensemble des résultats généré et éviter ainsi au maximum la génération d'associations non pertinentes.

Comme expliqué en conclusion du chapitre 7, pour améliorer nos résultats à la compétition QALD et, d'une manière plus générale, pour améliorer les performances de notre système sur le web de données, nous devrions faire évoluer l'approche de façon à la rendre plus proche des données et moins dépendante de l'ontologie. Nous avons également commencé à travailler sur le support des agrégats [Amarger et al., 2013] et voulons continuer sur cette voie. Peu de travaux ont encore abordé ce dernier point (du moins en ce qui concerne les interfaces entre langue naturelle et bases de connaissances) ; les auteurs de [Unger et al., 2012] sont à notre connaissance les seuls à détailler leur démarche.

Une des principales limites de notre approche réside dans le fait qu'à l'heure actuelle les patrons de requêtes sont construits à la main. Cette étape préliminaire de conception des patrons représente une quantité de travail importante et doit de plus être répétée à chaque fois que l'on veut porter le système sur une nouvelle base de connaissances. C'est pourquoi nous voudrions, dans la suite, explorer des méthodes de génération automatique ou assistée de patrons de requêtes. Nous avons déjà énoncé quelques pistes de recherches dans [Pradel, 2012], et pourrons nous inspirer de travaux récents visant à construire des patrons à l'aide de *logs* de requêtes SPARQL [Lorey and Naumann, 2013] ou via du *crowdsourcing* [Demartini et al., 2013]

Nous prévoyons également de nous affranchir de la nécessité de définir un modèle de phrase descriptive pour chaque patron. Nous pensons en effet que les recherches autour de la verbalisation de requêtes SPARQL vont s'intensifier et, au vu des premiers résultats [Ell et al., 2012; Ngonga Ngomo et al., 2013], que la quantité des systèmes implémentant ce genre d'approche sera bientôt suffisante pour générer des phrases compréhensibles et exprimant précisément le besoin en information d'une requête SPARQL.

En marge des travaux présentés dans ce mémoire, nous avons commencé à travailler sur une approche, intégrée au cadre fixé par la méthodologie de construction d'ontologies *NeOn* [Suárez-Figueroa et al., 2012a] et présentée dans [Suárez-Figueroa et al., 2012b], visant à exploiter le système *Swip* pour interpréter automatiquement des questions de compétence [Grüninger and Fox, 1995] (sortes d'exigences caractérisant une ontologie et formulées sous forme de questions) et vérifier ainsi qu'une ontologie répond bien aux besoins pour lesquels elle est construite. Nous prévoyons d'approfondir ces travaux par la suite.

Enfin, nous aimerions également explorer les nouvelles pistes ouvertes par l'implémentation que nous avons réalisée. Nous pensons en effet que l'approche que nous avons utilisée pour l'implémentation de la formalisation de la requête pivot, décrite en 6.2, peut être généralisée et utilisée pour d'autres applications : il serait ainsi possible d'implémenter tous types d'algorithmes, et en particulier des algorithmes manipulant des graphes. De plus, pour les raisons exposées en 6.3.1, cette

approche semble parfaitement adaptée au développement d'applications web et pourrait très bien s'adapter aux cadres proposés visant à intégrer les API web au web de données, comme *RDF-REST* [Champin, 2013]. Nous voudrions donc formaliser cette nouvelle forme de programmation exploitant les mises à jour SPARQL et la comparer à des paradigmes proches, comme la programmation dirigée par les données (*data-driven programming*), ou des approches plus pratiques exploitant des bases de données, comme PL/SQL. Ces travaux nous mèneraient certainement à proposer une extension à SPARQL permettant le saut conditionnel dans le but de faire de ce langage un langage de programmation particulièrement adapté à l'implémentation d'algorithmes impliquant des graphes.

Annexe A

Préfixes

identifiant	valeur
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
owl	url
xsd	http://www.w3.org/2001/XMLSchema#
dc	http://purl.org/dc/elements/1.1/
dbo	http://dbpedia.org/ontology/
dbp	http://dbpedia.org/property/
foaf	http://xmlns.com/foaf/0.1/
bio	http://purl.org/vocab/bio/0.1/
mo	http://purl.org/ontology/mo/
rel	http://purl.org/vocab/relationship/
res	http://dbpedia.org/resource/
cine	http://www.irit.fr/ontologies/cinema#
record	http://zitgist.com/music/record/
	url
pf	http://jena.hpl.hp.com/ARQ/property#
patterns	http://swip.univ-tlse2.fr/ontologies/Patterns#
queries	http://swip.univ-tlse2.fr/ontologies/Queries#

Annexe B

Grammaire des patrons

```
patterns ::= prefixes
             (pattern {$ps.add($pattern.p);})+

prefixes ::= PREF
             prefix+
             ENDPREF

prefix ::= ID COLON CITE

pattern ::= PAT ID
            QUER
            (subpattern)+
            ENDQUER
            sentenceTemplate[p]
            ENDPAT

subpattern ::= patterntriple
             | subpatterncollection

patterntriple ::= e1 e2 e3 ';'

e1 ::= classPe

e2 ::= propertyPe

e3 ::= classPe
     | literalPe

subpatterncollection ::= '['(subpattern)+']' (ID (COLON mio=INT (TWOPOINTS (mao=
    ↪ INT|N))? (SLASH pe=INT)?)?)?

classPe ::= (MINUS)? INT(UNDERSCORE qname)?

propertyPe ::= (MINUS)? id=INT(UNDERSCORE qname)?('('ref=INT (','ref2=INT)*')' )?

qname ::= p=ID COLON s=ID

literalPe ::= INT UNDERSCORE LOWERTHAN qname GREATERTHAN

sentenceTemplate ::= SENT
                     (subsentenceTemplate[p])+
                     ENDSENT
```

```
subsentenceTemplate ::= peInTemplate[p]
                      | spcInTemplate[p]
                      | staticStringInTemplate
                      | forLoopInTemplate[p]

peInTemplate ::= '-'INT'-'
    ;

spcInTemplate ::= '-' ID '-['(subsentenceTemplate[p])+']'

staticStringInTemplate ::= CITE

forLoopInTemplate ::= '-for-' ID '-['(subsentenceTemplate[p])+']'

ID ::= ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*;

INT ::= '0'..'9'+;

CITE ::= '"' (~('"'))+ '"';

COMMENT ::= '//' ~('\n'|'\r')* '\r'? '\n'
          | '/*' ( options {greedy=false;} : . )* '*/'

WS  :   (' '|'\t'|'\n'|'\r')+
```

Annexe C

Grammaire du langage pivot

```
query ::= subquerySet ('.' subquerySet)* ('.')?
subquerySet ::= e1q1 | e1q23 ':' q23End (';' q23End)*
e1q1 ::= keyword
e1q23 ::= keyword
q23End ::= e2q2 | e2q3 '=' e3q3 (',' e3q3)*
e2q2 ::= keyword | literal
e2q3 ::= keyword
e3q3 ::= keyword | literal
keyword ::= ('?' )? ('$'intValue'_')? keywordValue
keywordValue ::= '"' ( <any source character except "\"
        or newline or the quote> | '\\' | '\"' )+ '"'
literal ::= integerLit | dateLit
integerLit ::= 'integer<' intValue | '?' '>'
dateLit ::= 'date<' (year '-' month '-' day)
        | (year '-' month) | (year) | '?' '>'
year ::= intValue
month ::= '1'..'12'
day ::= '1'..'31'
intValue ::= '0'..'9'+
```

Annexe D

Requêtes SPARQL de l'interprétation de la requête pivot

This appendix presents the SPARQL queries involved in the pivot query interpretation process.

For readability reasons, in each query, the URI of the currently processed query is replaced by the string "[queryUri]" and used prefixes are always the same ; their values are given here:

```
PREFIX patterns:   <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:    <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:    <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX pf:    <http://jena.hpl.hp.com/ARQ/property#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
```

D.1 Commit a pivot query.

SPARQL Update used to commit the pivot query in the KB.

```
# commit query [queryUri]
PREFIX queries:    <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:    <http://swip.univ-tlse2.fr:8080/musicbrainz/>
INSERT DATA
{
  GRAPH graph:queries
  {
    <[queryUri]> a queries:PivotQuery;
                 queries:queryHasQueryElement graph:person;
                 queries:queryHasQueryElement graph:produce;
                 queries:queryHasQueryElement graph:album;
                 queries:queryHasQueryElement graph:Dookie;
                 queries:queryHasSubquery <[queryUri]/sq1>;
                 queries:queryHasSubquery <[queryUri]/sq2>.
```

```
    graph:person a queries:KeywordQueryElement;
                 queries:queryElementHasValue "person".
    graph:produce a queries:KeywordQueryElement;
                  queries:queryElementHasValue "produce".
    graph:album a queries:KeywordQueryElement;
                queries:queryElementHasValue "album".
    graph:Dookie a queries:KeywordQueryElement;
                 queries:queryElementHasValue "Dookie".
    <[queryUri]/sq1> a queries:Q2;
                     queries:subqueryHasE1 <[queryUri]/Dookie>;
                     queries:subqueryHasE2 <[queryUri]/album>.
    <[queryUri]/sq2> a queries:Q3;
                     queries:subqueryHasE1 <[queryUri]/person>;
                     queries:subqueryHasE2 <[queryUri]/produce>;
                     queries:subqueryHasE3 <[queryUri]/Dookie>.
  }
}
```

D.2 Match query elements to KB resources.

SPARQL Update executed in order to find out and store the matched KB elements for each keyword of the query.

```
# matching keywords to KB labels
PREFIX queries:   <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:   <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX pf:   <http://jena.hpl.hp.com/ARQ/property#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
INSERT
{
  GRAPH graph:queries
  {
    ?matchUri a queries:Matching;
              queries:matchingHasKeyword ?keyword;
              queries:matchingHasResource ?r;
              queries:matchingHasScore ?s;
              queries:matchingHasMatchedLabel ?l.
   ?keyword queries:keywordAlreadyMatched "true"^^xsd:boolean.
  }
}
WHERE
{
  {
    # subquery with DISTINCT to solve unidentified problem apparently due to larq
    SELECT DISTINCT * WHERE
    {
      GRAPH graph:queries
      {
        <[queryUri]> queries:queryHasQueryElement ?keyword.
        ?keyword a queries:KeywordQueryElement;
                 queries:queryElementHasValue ?keywordValue.
        FILTER NOT EXISTS { ?keyword queries:keywordAlreadyMatched "true"^^xsd:
            ↪ boolean. }
      }
      {
        GRAPH graph:ontologies
        {
```

```
          (?l ?score) pf:textMatch (?keywordValue 6.0 5).
          ?r rdfs:label ?l.
          BIND ((?score * 2) AS ?s)
        }
      }
      UNION
      {
        GRAPH graph:instances
        {
          (?l ?score) pf:textMatch (?keywordValue 6.0 5).
          ?r rdfs:label ?l.
          BIND ((?score * 1) AS ?s)
        }
      }
    }
  }
  BIND (UUID() AS ?matchUri)
}
```

D.3 Map pattern elements according to query element matches

SPARQL Update used to

```
# mapping KB pattern elements to keywords
PREFIX patterns:   <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:    <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:    <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
INSERT
{
  GRAPH graph:queries
  {
    ?emUri a queries:ElementMapping;
           queries:emHasPatternElement ?pe;
           queries:mappingHasPatternConstituent ?pe;
           queries:emHasMatching ?matching;
           queries:emHasScore ?score;
           queries:mappingHasQuery <[queryUri]>;
           queries:hasDescriptiveSubsentence ?descSentUri.
    ?descSentUri a queries:DescriptiveSubsentence;
                 queries:hasStringValue ?stringValue.
  }
}
WHERE
{
  GRAPH graph:queries
  {
    <[queryUri]> queries:queryHasQueryElement ?keyword.
    ?matching queries:matchingHasKeyword ?keyword;
              queries:matchingHasResource ?r;
              queries:matchingHasScore ?score;
              queries:matchingHasMatchedLabel ?l.
  }
  GRAPH graph:patterns
  {
    ?p patterns:patternHasPatternElement ?pe.
```

```
    ?pe patterns:targetsKBElement ?kbe.
  }
  {
    GRAPH graph:ontologies
    {
      {?r rdfs:subClassOf ?kbe.
      BIND ("some " AS ?stringValuePrefix)}
      UNION
      {?r rdfs:subPropertyOf ?kbe.}
    }
  }
  UNION
  {
    GRAPH graph:instances
    {
      ?r a ?c.
    }
    GRAPH graph:ontologies
    {
      ?c rdfs:subClassOf ?kbe.
    }
  }
  BIND (UUID() AS ?emUri)
  BIND (UUID() AS ?descSentUri)
  BIND (if (BOUND(?stringValuePrefix), CONCAT(?stringValuePrefix, ?l), STR(?l)) AS
      ↪ ?stringValue)
}
```

D.4 Add an empty mapping to all pattern elements

SPARQL Update used to

```
# add an empty mapping to all pattern elements
PREFIX patterns:  <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:  <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:  <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
INSERT
{
  GRAPH graph:queries
  {
    ?emUri a queries:EmptyElementMapping;
           a queries:ElementMapping;
           queries:emHasPatternElement ?pe;
           queries:mappingHasPatternConstituent ?pe;
           queries:mappingHasQuery <[queryUri]>;
           queries:emHasScore "0"^^xsd:float;
           queries:hasDescriptiveSubsentence ?descSentUri.
    ?descSentUri a queries:DescriptiveSubsentence;
                 queries:hasStringValue ?l.
    ?pe queries:toConsiderInMappingQuery <[queryUri]>.
  }
}
WHERE
{
```

```
  GRAPH graph:patterns
  {
    ?pe a patterns:PatternElement;
        patterns:targetsKBElement ?kbe.
    OPTIONAL { ?kbe rdfs:label ?label. }
  }
  BIND (UUID() AS ?emUri)
  BIND (UUID() AS ?descSentUri)
  BIND (IF( BOUND(?label), CONCAT( "(a/an)", ?label), CONCAT( "no_label_found_for_
      ↪ ", STR(?kbe))) AS ?l)
}
```

D.5 Initialize subpattern collection mappings

SPARQL Update ...

```
# specify for all subpatterns that they have not yet been mapped to the current
    ↪ query
PREFIX patterns:  <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:  <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:  <http://swip.univ-tlse2.fr:8080/musicbrainz/>
INSERT
{
  GRAPH graph:queries
  {
    ?spc queries:isNotMappedToQuery <[queryUri]>.
  }
}
WHERE
{
  GRAPH graph:patterns
  {
    ?spc a patterns:SubpatternCollection.
  }
}
```

D.6 Remove mappings of contingent subpattern collections containing only empty mappings

SPARQL Update ...

```
# remove mappings of contingent subpattern collections containing only empty
    ↪ mappings
# an empty mapping for the contingent subpattern collection itself will be added
    ↪ later
PREFIX patterns:  <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:  <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:  <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
DELETE
{
  GRAPH graph:queries
  {
```

```
    ?m ?a ?b.
  }
}
WHERE
{
  GRAPH graph:patterns
  {
    ?spc patterns:hasCardinalityMin "0"^^xsd:int.
  }
  GRAPH graph:queries
  {
    ?spc queries:toConsiderInMappingQuery <[queryUri]>.
    ?m queries:mappingHasQuery <[queryUri]>;
       queries:mappingHasPatternConstituent ?spc;
       ?a ?b.
    FILTER NOT EXISTS
    {
      ?m queries:mappingContainsMapping+ ?em.
      ?em queries:emHasMatching ?matching.
    }
  }
}
```

D.7 Combine mappings

SPARQL Update ...

```
# combine mappings of repeatable subpattern collections
PREFIX patterns:   <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:    <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:    <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
INSERT
{
  GRAPH graph:queries
  {
    ?mappingUri a queries:SubpatternCollectionMapping;
                queries:mappingHasPatternConstituent ?spc;
                queries:mappingContainsMapping ?m1, ?m2;
                queries:mappingHasQuery <[queryUri]>.
    ?spc queries:allreadyCombinedForQuery <[queryUri]>.
    # process the descriptive sentence
    ?mappingUri queries:hasDescriptiveSubsentence ?descSubsent.
    ?descSubsent a queries:DescriptiveSubsentence;
                 queries:isMadeUpOfList ?list;
                 queries:isMadeUpOfList-for ?listFor1, ?listFor2.
  }
}
WHERE
{
  GRAPH graph:patterns
  {
    SELECT DISTINCT ?spc WHERE
    {
      ?spc patterns:hasCardinalityMax "2"^^xsd:int.
    }
  }
  GRAPH graph:queries
```

```
  {
    ?spc queries:toConsiderInMappingQuery <[queryUri]>.
    FILTER NOT EXISTS
    {
      ?spc queries:allreadyCombinedForQuery <[queryUri]>.
    }
    ?m1 queries:mappingHasPatternConstituent ?spc.
    ?m2 queries:mappingHasPatternConstituent ?spc.
    FILTER ( STR(?m1) < STR(?m2) )
    # process the descriptive sentence
    ?m1 (queries:hasDescriptiveSubsentence/queries:isMadeUpOfList) ?list.
    ?m1 (queries:hasDescriptiveSubsentence/queries:isMadeUpOfList-for) ?listFor1.
    ?m2 (queries:hasDescriptiveSubsentence/queries:isMadeUpOfList-for) ?listFor2.
  }
  BIND (UUID() AS ?mappingUri)
  BIND (UUID() AS ?descSubsent)
}
```

D.8 Add an empty mapping to contingent subpattern collections

SPARQL Update ...

```
# add an empty mapping to ContingentSubpatternCollections to be considered in next
    ↪ mappings
PREFIX patterns:   <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:    <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:    <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
INSERT
{
  GRAPH graph:queries
  {
    ?mappingUri a queries:EmptySubpatternCollectionMapping;
                a queries:SubpatternCollectionMapping;
                queries:mappingHasPatternConstituent ?spc;
                queries:mappingHasQuery <[queryUri]>;
                queries:hasDescriptiveSubsentence ?descSentUri.
    ?descSentUri a queries:DescriptiveSubsentence;
                   queries:hasStringValue "(EmptySubpatternCollectionMapping)".
  }
}
WHERE # select all ContingentSubpatternCollections which don't have yet an
    ↪ EmptySubpatternCollectionMapping
{
  GRAPH graph:queries
  {
    ?spc queries:toConsiderInMappingQuery <[queryUri]>.
    FILTER NOT EXISTS
    {
      ?mapping a queries:EmptySubpatternCollectionMapping;
                queries:mappingHasPatternConstituent ?spc;
                queries:mappingHasQuery <[queryUri]>.
    }
  }
  GRAPH graph:patterns
```

```
  {
    SELECT DISTINCT ?spc WHERE
    {
      ?spc a patterns:SubpatternCollection;
           patterns:hasCardinalityMin "0"^^xsd:int.
    }
  }
  BIND (UUID() AS ?mappingUri)
}
```

D.9 Start mapping of SubpatternCollections whose all contained components are already mapped

SPARQL Update ...

```
# start mapping of SubpatternCollections whose all contained components are
    ↪ already mapped
PREFIX patterns:  <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:  <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:  <http://swip.univ-tlse2.fr:8080/musicbrainz/>
DELETE
{
  GRAPH graph:queries
  {
    ?spc queries:isNotMappedToQuery <[queryUri]>.
  }
}
INSERT
{
  GRAPH graph:queries
  {
    ?spc queries:isBeingMappedToQuery <[queryUri]>.
  }
}
WHERE
{
  {
    # get all SubpatternCollections which are not mapped but whose contained
        ↪ SubpatternCollections are 'to be considered'
    SELECT ?spc WHERE
    {
      # all SubpatternCollections
      GRAPH graph:patterns
      {
        ?spc a patterns:SubpatternCollection.
      }
      # which are not mapped
      GRAPH graph:queries
      {
        ?spc queries:isNotMappedToQuery <[queryUri]>.
      }
      # whose contained SubpatternCollections are all 'to be considered'
      MINUS
      {
```

```
        # get all SubpatternCollections whose at least one contained
            ↪ SubpatternCollection is being mapped not mapped
        SELECT ?spc WHERE
        {
          GRAPH graph:patterns
          {
            ?spc a patterns:SubpatternCollection;
                 patterns:hasDirectSubpattern ?spc2.
            ?spc2 a patterns:SubpatternCollection;
          }
          GRAPH graph:queries
          {
            ?spc2 (queries:isNotMappedToQuery | queries:isBeingMappedToQuery) <[
                ↪ queryUri]>.
          }
        }
      }
    }
  }
}
```

D.10 Prevent redundant element mappings

SPARQL Update ...

```
# prevent from adding in a subpattern collection mapping a element mapping
# relative to a pattern element appearing outside of this subpattern collection
PREFIX patterns:   <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:    <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:    <http://swip.univ-tlse2.fr:8080/musicbrainz/>
DELETE
{
  GRAPH graph:queries
  {
    ?pc queries:toConsiderInMappingQuery <[queryUri]>.
  }
}
INSERT
{
  GRAPH graph:queries
  {
    ?pc queries:toConsiderNextStepInMappingQuery <[queryUri]>.
  }
}
WHERE
{
  SELECT DISTINCT ?pc WHERE
  {
    GRAPH graph:queries
    {
      ?spc queries:isBeingMappedToQuery <[queryUri]>.
      ?pc queries:toConsiderInMappingQuery <[queryUri]>.
    }
    GRAPH graph:patterns
    {
      ?spc patterns:isMadeUpOf ?pc.
      OPTIONAL
      {
        ?spc2 patterns:isMadeUpOf ?pc.
```

```
        FILTER ( !sameTerm(?spc, ?spc2) )
      }
      FILTER NOT EXISTS
      {
        { ?spc patterns:isMadeUpOf ?spc2. }
        UNION
        { ?spc2 patterns:isMadeUpOf ?spc. }
      }
    }
  }
}
```

D.11 Make progress the mappings of currently processed SubpatternCollections

SPARQL Update ...

```
# make progress the mappings of currently processed SubpatternCollections
PREFIX patterns:  <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:  <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:  <http://swip.univ-tlse2.fr:8080/musicbrainz/>
DELETE
{
  GRAPH graph:queries
  {
    ?pc queries:toConsiderInMappingQuery <[queryUri]>.
    ?pc queries:allreadyCombinedForQuery <[queryUri]>.
    # to not consider these mappings anymore
    ?mPc queries:mappingHasPatternConstituent ?pc.
    ?mSpc queries:mappingHasPatternConstituent ?spc.
  }
}
INSERT
{
  GRAPH graph:queries
  {
    ?pc queries:isMappedToQuery <[queryUri]>.
    ?mappingUri a queries:SubpatternCollectionMapping;
                queries:mappingHasPatternConstituent ?spc;
                queries:mappingContainsMapping ?mSpc, ?mPc;
                queries:mappingHasQuery <[queryUri]>.
    # used for descriptive sentence generation
    ?mPc queries:mappingHadPatternConstituent ?pc.
    ?mSpc queries:mappingHadPatternConstituent ?spc.
  }
}
WHERE
{
  {
    # select a pair (?spc, ?pc) such as
    # - ?spc is a SubpatternCollection made of ?pc
    # - ?spc is being mapped
    # - ?pc is to be considered
    SELECT ?spc ?pc WHERE
    {
      GRAPH graph:queries
      {
```

```
        ?pc queries:toConsiderInMappingQuery <[queryUri]>.
        ?spc queries:isBeingMappedToQuery <[queryUri]>.
      }
      GRAPH graph:patterns
      {
        ?spc patterns:isMadeUpOf ?pc.
      }
    } ORDER BY ?spc ?pc LIMIT 1
  }
  GRAPH graph:queries
  {
    OPTIONAL
    {
      ?mSpc queries:mappingHasPatternConstituent ?spc;
            queries:mappingHasQuery <[queryUri]>.
    }
    ?mPc queries:mappingHasPatternConstituent ?pc;
         queries:mappingHasQuery <[queryUri]>.
  }
  BIND (UUID() AS ?mappingUri)
}
```

D.12 Validate mappings

SPARQL Update ...

```
# validate mappings of SubpatternCollections which are being mapped and whose
    ↪ contained SubpatternCollections are all mapped or to be mapped in next step
# change the status of toConsiderNextStepInMappingQuery SubpatternCollections into
    ↪ toConsiderInMappingQuery
PREFIX patterns:  <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:  <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:  <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
DELETE
{
  GRAPH graph:queries
  {
    ?spc queries:isBeingMappedToQuery <[queryUri]>.
    ?comp queries:toConsiderNextStepInMappingQuery <[queryUri]>.
  }
}
INSERT
{
  GRAPH graph:queries
  {
    ?spc queries:toConsiderInMappingQuery <[queryUri]>.
    ?comp queries:toConsiderInMappingQuery <[queryUri]>.
    # process the descriptive sentence
    ?spcm queries:hasDescriptiveSubsentence ?descSubsent.
    ?descSubsent a queries:DescriptiveSubsentence;
                 ?madeUpRelation ?listSerialized .
    ?oneListSerialized rdf:first ?oneElementSerializedSsit ;
                       rdf:first ?oneElementSerializedFlit ;
                       rdf:first ?oneElementSerializedOther ;
                       rdf:firsttt ?plop ;
                       rdf:rest ?nextListSerialized .
    ?oneElementSerializedSsit queries:hasStringValue ?ssitValue.
    ?oneElementSerializedFlit queries:insertForSubsentenceOf ?spc.
```

```
  }
}
WHERE
{
  {
    # get all SubpatternCollections which are being mapped and whose contained
        ↪ SubpatternCollections are all mapped or to be mapped in next step
    SELECT ?spc WHERE
    {
      # all SubpatternCollections
      GRAPH graph:patterns
      {
        ?spc a patterns:SubpatternCollection.
      }
      # which are being mapped
      GRAPH graph:queries
      {
        ?spc queries:isBeingMappedToQuery <[queryUri]>.
      }
      # whose contained SubpatternCollections are all mapped or to be mapped in
          ↪ next step
      MINUS
      {
        # get all SubpatternCollections whose at least one contained component is
            ↪ not mapped
        SELECT DISTINCT ?spc WHERE
        {
          GRAPH graph:patterns
          {
            ?spc a patterns:SubpatternCollection;
                 patterns:isMadeUpOf ?comp.
          }
          GRAPH graph:queries
          {
            ?comp queries:toConsiderInMappingQuery <[queryUri]>.
          }
        }
      }
    }
  }
  # change the status of toConsiderNextStepInMappingQuery SubpatternCollections
      ↪ into toConsiderInMappingQuery
  OPTIONAL
  {
    GRAPH graph:patterns
    {
      ?spc patterns:isMadeUpOf ?comp.
    }
    GRAPH graph:queries
    {
      ?comp queries:toConsiderNextStepInMappingQuery <[queryUri]>.
    }
  }
  # process the descriptive sentence
  GRAPH graph:queries
  {
    ?spcm queries:mappingHasPatternConstituent ?spc.
  }
  GRAPH graph:patterns
  {
    ?sst (patterns:sstTargetsPc|^patterns:hasSentenceTemplate) ?spc ;
         patterns:isMadeUpOfList ?list ;
```

```
    # when dealing with spc containing a for loop, two subsentences are generated
        ↪ (one for the inner for loop, one for the whole spc)
    OPTIONAL {
      ?sst a patterns:ForLoopInTemplate.
      BIND (queries:isMadeUpOfList-for AS ?madeUpRelation).
    }
    OPTIONAL {
      ?sst a patterns:SpcInTemplate.
      BIND (queries:isMadeUpOfList AS ?madeUpRelation).
    }
    ?list rdf:rest* ?oneList .
    ?oneList rdf:first ?oneElement ;
             rdf:rest ?nextList .
  }
  # oneElement is a static string
  OPTIONAL
  {
    GRAPH graph:patterns { ?oneElement patterns:ssitHasValue ?ssitValue.
                           BIND (UUID() AS ?oneElementSerializedSsit) }
  }
  # oneElement is a ForLoopInTemplate
  OPTIONAL
  {
    GRAPH graph:patterns { ?oneElement a patterns:ForLoopInTemplate.
                           BIND (UUID() AS ?oneElementSerializedFlit) }
  }
  # oneElement is a PeInTemplate or a SpcInTemplate
  OPTIONAL
  {
    GRAPH graph:patterns { {?oneElement a patterns:PeInTemplate.} UNION {?
        ↪ oneElement a patterns:SpcInTemplate.}
                           ?oneElement patterns:sstTargetsPc ?pc. }
    GRAPH graph:queries { ?spcm queries:mappingContainsMapping+ ?sspcm.
                          ?sspcm queries:mappingHadPatternConstituent ?pc;
                                 queries:hasDescriptiveSubsentence ?
                                     ↪ oneElementSerializedOther.
                          FILTER NOT EXISTS # because of combined mappings which
                              ↪ maps same spc as their contained mappings
                          # FIXME: this part makes the query very slow (for
                              ↪ something that looks simple)
                          {
                            ?spcm queries:mappingContainsMapping+ ?sspcm2.
                            ?sspcm2 queries:mappingHadPatternConstituent ?pc;
                                    queries:mappingContainsMapping ?sspcm.
                          }}
    BIND (<SpcInTemplate> AS ?plop)
  }
  BIND (IRI(CONCAT(str(?spcm), "_descSubsent")) AS ?descSubsent)
  BIND (IRI(CONCAT(str(?spcm), str(?list))) AS ?listSerialized)
  BIND (IRI(CONCAT(str(?spcm), str(?oneList))) AS ?oneListSerialized)
  BIND (IRI(CONCAT(str(?spcm), str(?nextList))) AS ?nextListSerialized)
}
```

D.13 Are all patterns mapped ?

SPARQL Ask used to

```
# are all patterns mapped to the current query?
PREFIX patterns:   <http://swip.univ-tlse2.fr/ontologies/Patterns#>
```

```
PREFIX queries:   <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:   <http://swip.univ-tlse2.fr:8080/musicbrainz/>
ASK
{
  GRAPH graph:patterns
  {
    ?p a patterns:Pattern.
  }
  FILTER NOT EXISTS
  {
    GRAPH graph:queries { ?p queries:toConsiderInMappingQuery <[queryUri]>. }
  }
}
```

D.14 Mapping patterns to the pivot query

```
# perform pattern mapping
PREFIX patterns:   <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:   <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:   <http://swip.univ-tlse2.fr:8080/musicbrainz/>
DELETE
{
  GRAPH graph:queries
  {
    ?p queries:toConsiderInMappingQuery <[queryUri]>.
  }
}
INSERT
{
  GRAPH graph:queries
  {
    ?p queries:isMappedToQuery <[queryUri]>.
    ?pm a queries:PatternMapping.
  }
}
WHERE
{
  GRAPH graph:queries
  {
    ?pm queries:mappingHasPatternConstituent ?p;
        queries:mappingHasQuery <[queryUri]>.
    FILTER NOT EXISTS {?pm2 queries:mappingContainsMapping ?pm.}
  }
  GRAPH graph:patterns
  {
    ?p a patterns:Pattern.
  }
}
```

D.15 Element mapping relevance mark

```
# process the element mapping relevance mark
# step 1: process the average score of involved matchings
PREFIX patterns:   <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:   <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:   <http://swip.univ-tlse2.fr:8080/musicbrainz/>
```

```
INSERT
{
  GRAPH graph:queries
  {
    ?pm queries:hasEmAvg ?avgscore;
  }
}
WHERE
{
  SELECT ?pm (AVG(?score) AS ?avgscore) WHERE
    {
    GRAPH graph:queries
    {
      ?pm a queries:PatternMapping;
           queries:mappingHasQuery <[queryUri]>;
           queries:mappingContainsMapping+ ?em.
      ?em queries:emHasScore ?score.
    }
  } GROUP BY ?pm
};
# step 2: disavantage query mappings involving several times a same keyword
INSERT
{
  GRAPH graph:queries
  {
    ?pm queries:hasEmFactor ?factor;
  }
}
WHERE
{
  SELECT ?pm (COUNT(distinct ?kw) / COUNT(distinct ?em) AS ?factor ) WHERE
  {
    GRAPH graph:queries
    {
      ?pm a queries:PatternMapping;
           queries:mappingHasQuery <[queryUri]>;
           queries:mappingContainsMapping+ ?em.
      FILTER NOT EXISTS {?em a queries:EmptyElementMapping.}
      ?em (queries:emHasMatching / queries:matchingHasKeyword) ?kw.
    }
  } GROUP BY ?pm
};
INSERT
{
  GRAPH graph:queries
  {
    ?pm queries:hasEmrMark ?emrmark;
  }
}
WHERE
{
  GRAPH graph:queries
  {
    ?pm queries:hasEmAvg ?avgscore.
    ?pm queries:hasEmFactor ?factor.
    BIND (?factor * ?avgscore AS ?emrmark)
  }
}
```

D.16 Query coverage relevance mark

```
# process the query coverage relevance mark
PREFIX patterns:  <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:  <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:  <http://swip.univ-tlse2.fr:8080/musicbrainz/>
INSERT
{
  GRAPH graph:queries
  {
    ?pm queries:hasQcrMark ?qcrmark;
  }
}
WHERE
{
  {
    SELECT (COUNT(?qe) AS ?nbqe) WHERE
    {
      GRAPH graph:queries
      {
        <[queryUri]> queries:queryHasQueryElement ?qe.
      }
    }
  }
  {
    SELECT ?pm (COUNT(DISTINCT ?qe) AS ?nbmappedqe) WHERE
    {
      GRAPH graph:queries
      {
        ?pm a queries:PatternMapping;
            queries:mappingHasQuery <[queryUri]>;
            queries:mappingContainsMapping+ ?em.
        ?em (queries:emHasMatching / queries:matchingHasKeyword) ?qe.
      }
    } GROUP BY ?pm
  }
  BIND ( ?nbmappedqe / ?nbqe AS ?qcrmark )
}
```

D.17 Pattern coverage relevance mark

```
# process the pattern coverage relevance mark
# in two step probably because of a bug in ARQ
# step 1: find out for each query mapping the number of element mapping with
    ↪ distinct pattern element
PREFIX patterns:  <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:  <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:  <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
INSERT
{
  GRAPH graph:queries
  {
    ?pm queries:numberOfSignificantElementMappings ?nbmappedpe;
  }
}
WHERE
{
  select ?pm (count(distinct ?pe) as ?nbmappedpe) where
```

```
  {
      GRAPH graph:queries
      {
        ?pm a queries:PatternMapping;
            queries:mappingHasQuery <[queryUri]>;
            queries:mappingHasPatternConstituent ?p;
            queries:mappingContainsMapping+ ?em.
        ?em queries:emHasPatternElement ?pe.
        FILTER NOT EXISTS { ?em a queries:EmptyElementMapping. }
      }
  } group by ?pm
};
# step 2: find out for each query mapping the total number of element mappings
PREFIX patterns:  <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:  <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:  <http://swip.univ-tlse2.fr:8080/musicbrainz/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
INSERT
{
  GRAPH graph:queries
  {
    ?pm queries:numberOfElementMappings ?nbmappedpe;
  }
}
WHERE
{
  select ?pm (count(distinct ?pe) as ?nbmappedpe) where
  {
      GRAPH graph:queries
      {
        ?pm a queries:PatternMapping;
            queries:mappingHasQuery <[queryUri]>;
            queries:mappingHasPatternConstituent ?p;
            queries:mappingContainsMapping+ ?em.
        ?em queries:emHasPatternElement ?pe.
      }
  } group by ?pm
};
# step 3:
INSERT
{
  GRAPH graph:queries
  {
    ?pm queries:hasPcrMark ?pcrmark;
  }
}
WHERE
{
  GRAPH graph:queries
  {
    ?pm queries:mappingHasQuery <[queryUri]>;
        queries:numberOfElementMappings ?nbem;
        queries:numberOfSignificantElementMappings ?nbsem.
  }
  BIND (?nbsem / ?nbem AS ?pcrmark)
}
```

D.18 Final relevance mark

```
# process final relevance mark
```

```
PREFIX patterns:   <http://swip.univ-tlse2.fr/ontologies/Patterns#>
PREFIX queries:    <http://swip.univ-tlse2.fr/ontologies/Queries#>
PREFIX graph:    <http://swip.univ-tlse2.fr:8080/musicbrainz/>
INSERT
{
  GRAPH graph:queries
  {
    ?pm queries:hasRelevanceMark ?rmark;
  }
}
WHERE
{
  GRAPH graph:queries
  {
    ?pm a queries:PatternMapping;
        queries:mappingHasQuery <[queryUri]>;
        queries:hasEmrMark ?emrmark;
        queries:hasQcrMark ?qcrmark;
        queries:hasPcrMark ?pcrmark.
  }
  BIND ( ?emrmark * ?qcrmark * ?pcrmark AS ?rmark )
}
```

D.19 Clear KB

```
# clear all mappings whose rank is over a given threshold
```

Bibliographie

J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman. SQL QueRIE recommendations: a query fragment-based approach. *Proceedings of the VLDB Endowment*, 3(1-2):1597–1600, 2010. URL http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/vldb_2010_workshop/PersDB_2010/resources/PersDB2010_4.pdf.

Fabien Amarger, Ollivier Haemmerlé, Nathalie Hernandez, and Camille Pradel. Taking SPARQL 1.1 extensions into account in the SWIP system. In *Conceptual Structures for STEM Research and Education*, page 75–89, Bombay, Inde, 2013. Springer. URL http://link.springer.com/chapter/10.1007/978-3-642-35786-2_7.

I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Natural Language Engineering*, 1(01):29–81, 1995. URL http://journals.cambridge.org/abstract_S135132490000005X.

Nicholas Asher. *Reference to abstract objects in discourse*. Kluwer Academic (Dordrecht and Boston), 1993. URL http://www.getcited.org/pub/103083242.

Manuel Atencia, Jérôme David, and François Scharffe. Keys and pseudo-keys detection for web datasets cleansing and interlinking. In *Knowledge Engineering and Knowledge Management*, page 144–153. Springer, 2012. URL http://link.springer.com/chapter/10.1007/978-3-642-33876-2_14.

Nikos Athanasis, Vassilis Christophides, and Dimitris Kotzinos. Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL). In *The Semantic Web–ISWC 2004*, page 486–501. Springer, 2004. URL http://link.springer.com/chapter/10.1007/978-3-540-30475-3_34.

Giuseppe Attardi, Antonio Cisternino, Francesco Formica, Maria Simi, and Alessandro Tommasi. PiQASso: pisa question answering system. *NIST special publication*, page 633–641, 2002. URL http://cat.inist.fr/?aModele=afficheN&cpsidt=14439940.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, page 722–735. Springer, 2007. URL http://link.springer.com/chapter/10.1007/978-3-540-76298-0_52.

Dave Beckett and Jeen Broekstra. SPARQL query results XML format (second edition), March 2013. URL http://www.w3.org/TR/rdf-sparql-XMLres/.

Michael Bendersky and W. Bruce Croft. Analysis of long queries in a large scale search log. In *Proceedings of the 2009 workshop on Web Search Click Data*, page 8–14, 2009. URL http://dl.acm.org/citation.cfm?id=1507511.

Tim Berners-Lee. Semantic web on XML, December 2000. URL http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html.

Tim Berners-Lee, Mark Fischetti, and Michael L. Foreword By-Dertouzos. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. HarperInformation, 2000. URL http://dl.acm.org/citation.cfm?id=556560.

Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):28–37, 2001. URL http://isel2918929391.googlecode.com/svn-history/r347/trunk/RPC/Slides/p01_theSemanticWeb.pdf.

Abraham Bernstein, Esther Kaufmann, and Christian Kaiser. Querying the semantic web with ginseng: A guided input natural language search engine. In *In: 15th Workshop on Information Technologies and Systems, Las Vegas, NV*, 2005. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.90.7212.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009a. URL http://www.igi-global.com/article/linked-data-story-far/37496.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia-A crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009b. URL http://www.sciencedirect.com/science/article/pii/S1570826809000225.

Willem Nico Borst. *Construction of engineering ontologies for knowledge sharing and reuse*. Universiteit Twente, 1997. URL http://doc.utwente.nl/17864.

Didier Bourigault and Cécile Fabre. Approche linguistique pour l'analyse syntaxique de corpus. *Cahiers de grammaire*, 25:131–151, 2000. URL http://w3.erss.univ-tlse2.fr/textes/publications/CDG/25/CG25-8-Bourigault.pdf.

Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive science*, 9(2):171–216, 1985. URL http://www.sciencedirect.com/science/article/pii/S0364021385800148.

Dan Brickley and Libby Miller. FOAF vocabulary specification, May 2007. URL http://xmlns.com/foaf/spec/20070524.html.

Elena Cabrio, Julien Cojan, Alessio Palmero Aprosio, Bernardo Magnini, Alberto Lavelli, and Fabien Gandon. QAKiS: an open domain QA system based on relational patterns. In *International Semantic Web Conference (Posters & Demos)*, volume 914, 2012. URL http://www.academia.edu/download/30335457/paper_24.pdf.

Elena Cabrio, Philipp Cimiano, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Christina Unger, and Sebastian Walter. QALD-3: multilingual question answering over linked data. Valencia, Spain, September 2013. URL http://www.clef-initiative.eu/documents/71612/57dbe6e4-fe94-4cf9-963c-10f1dbbe3ef9.

Soumen Chakrabarti. Breaking through the syntax barrier: Searching with entities and relations. In *Knowledge Discovery in Databases: PKDD 2004*, page 9–16. Springer, 2004. URL http://link.springer.com/chapter/10.1007/978-3-540-30116-5_3.

Pierre-Antoine Champin. RDF-REST: a unifying framework for web APIs and linked data. 2013. URL http://liris.cnrs.fr/Documents/Liris-6050.pdf.

Michel Chein and Marie-Laure Mugnier. Conceptual graphs: Fundamental notions. In *Revue d'intelligence artificielle*, 1992. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.848.

Philipp Cimiano. ORAKEL: a natural language interface to an f-logic knowledge base. In *Natural Language Processing and Information Systems*, page 401–406. Springer, 2004. URL http://link.springer.com/chapter/10.1007/978-3-540-27779-8_38.

Philipp Cimiano. Ontology learning from text. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*, page

19–34, 2006. URL http://link.springer.com/content/pdf/10.1007/978-0-387-39252-3_3.pdf.

Philipp Cimiano, Peter Haase, Jörg Heizmann, and Matthias Mantel. Orakel: A portable natural language interface to knowledge bases. 2007. URL http://pub.uni-bielefeld.de/download/2497310/2525113.

Grant Clark. SPARQL protocol for RDF, January 2008. URL http://www.w3.org/TR/rdf-sparql-protocol/.

Aaron Clemmer and Stephen Davies. Smeagol: A "Specific-to-General" semantic web query interface paradigm for novices. In *Database and Expert Systems Applications*, page 288–302, 2011. URL http://link.springer.com/chapter/10.1007/978-3-642-23088-2_21.

Allan M. Collins and M. Ross Quillian. Retrieval time from semantic memory. *Journal of verbal learning and verbal behavior*, 8(2):240–247, 1969. URL http://www.sciencedirect.com/science/article/pii/S0022537169800691.

Hamish Cunningham. GATE, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002. URL http://link.springer.com/article/10.1023/A:1014348124664.

Danica Damljanović, Milan Agatonović, Hamish Cunningham, and Kalina Bontcheva. Improving habitability of natural language interfaces for querying ontologies with feedback and clarification dialogues. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2013. URL http://www.sciencedirect.com/science/article/pii/S157082681300005X.

Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham. Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. In *The Semantic Web: Research and Applications*, page 106–120. Springer, 2010. URL http://link.springer.com/chapter/10.1007/978-3-642-13486-9_8.

Mathieu d'Aquin and Enrico Motta. Extracting relevant questions to an RDF dataset using formal concept analysis. In *Proceedings of the sixth international conference on Knowledge capture*, page 121–128, 2011. URL http://dl.acm.org/citation.cfm?id=1999698.

A. N. De Roeck, C. J. Fox, B. G. T. Lowden, Ray Turner, and Bryan Walls. A natural language system based on formal semantics. In *Proceedings of the International Conference on Current Issues in Computational Linguistics, Pengang, Malaysia*, 1991.

Sasa M. Dekleva. Is natural language querying practical ? *ACM SIGMIS Database*, 25(2):24–36, 1994. URL http://dl.acm.org/citation.cfm?id=190745.

Gianluca Demartini, Beth Trushkowsky, Tim Kraska, and Michael J. Franklin. CrowdQ: crowdsourced query understanding. In *CIDR*, 2013. URL http://cs.brown.edu/people/kraskat/pub/CIDR13_CrowdQ.pdf.

Pascal Denis and Benoît Sagot. Coupling an annotated corpus and a morpho-syntactic lexicon for state-of-the-art POS tagging with less human effort. In *PACLIC*, page 110–119, 2009. URL http://alpage.inria.fr/~sagot/pub/paclic09tagging.pdf.

Sheilla E. Desert. WESTLAW is natural v. boolean searching: a performance study. *Law Library Journal*, 84(4):713–742, 1993. URL http://works.bepress.com/aallcallforpapers/45/.

Jens Dietrich and Chris Elgar. A formal description of design patterns using OWL. In *Software Engineering Conference, 2005. Proceedings. 2005 Australian*, page 243–250, 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1402019.

Corina Dima. Intui2: A prototype system for question answering over linked data. Valencia, Spain, September 2013. URL http://www.clef-initiative.eu/documents/71612/e2ceddb7-db03-46a2-832f-b9ddfb4ff842.

Michael Dittenbach, Dieter Merkl, and Helmut Berger. A natural language query interface for tourism information. 2003. URL http://195.130.87.21:8080/dspace/handle/123456789/840.

Doug Downey, Susan Dumais, Dan Liebling, and Eric Horvitz. Understanding the relationship between searchers' queries and information goals. In *Proceedings of the 17th ACM conference on Information and knowledge management*, page 449–458, 2008. URL http://dl.acm.org/citation.cfm?id=1458143.

Gideon Dror, Yoelle Maarek, Avihai Mejer, and Idan Szpektor. From query to question in one click: suggesting synthetic questions to searchers. In *Proceedings of the 22nd international conference on World Wide Web*, page 391–402, 2013. URL http://dl.acm.org/citation.cfm?id=2488423.

Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, and Gerhard Weikum. Searching rdf graphs with SPARQL and keywords. *IEEE Data Engineering Bulletin*, 33(1), 2010. URL ftp://ftp.research.microsoft.com/pub/debull/a10mar/weikum-paper.pdf.

Basil Ell, Denny Vrande\vcić, and Elena Simperl. Labels in the web of data. In *The Semantic Web–ISWC 2011*, page 162–176. Springer, 2011. URL http://link.springer.com/chapter/10.1007/978-3-642-25073-6_11.

Basil Ell, Denny Vrandecic, and Elena Simperl. Spartiqulation: Verbalizing sparql queries. In *Proceedings of ILD Workshop, ESWC2012*, 2012. URL http://ceur-ws.org/Vol-913/04_ILD2012.pdf.

Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, 2007. URL http://books.google.fr/books?hl=fr&lr=&id=qYVpA2t2EtQC&oi=fnd&pg=PA1&dq=Ontology+Matching&ots=67tFOHj6kG&sig=1s9koEIHdC44EsLz204YQUoejXc.

Sébastien Ferré. SQUALL: a controlled natural language for querying and updating RDF graphs. In *Controlled Natural Language*, page 11–25. Springer, 2012. URL http://link.springer.com/chapter/10.1007/978-3-642-32612-7_2.

Sébastien Ferré. squall2sparql: a translator from controlled english to full SPARQL 1.1. Valencia, Spain, September 2013a. URL http://www.clef-initiative.eu/documents/71612/2ce1b6ce-bdf2-4c25-8822-899aed299d8c.

Sébastien Ferré. SQUALL: a controlled natural language as expressive as SPARQL 1.1. In *Natural Language Processing and Information Systems*, page 114–125. Springer, 2013b. URL http://link.springer.com/chapter/10.1007/978-3-642-38824-8_10.

Sébastien Ferré and Alice Hermann. Semantic search: reconciling expressive querying and exploratory search. In *The Semantic Web–ISWC 2011*, page 177–192. Springer, 2011. URL http://link.springer.com/chapter/10.1007/978-3-642-25073-6_12.

Sébastien Ferré and Alice Hermann. Reconciling faceted search and query languages for the semantic web. *International Journal of Metadata, Semantics and Ontologies*, 7(1):37–54, 2012. URL http://inderscience.metapress.com/index/XP8X8052065J53N1.pdf.

Sébastien Ferré and Sebastian Rudolph. Advocatus Diaboli–Exploratory enrichment of ontologies with negative constraints. In *Knowledge Engineering and Knowledge Management*, page 42–56. Springer, 2012. URL http://link.springer.com/chapter/10.1007/978-3-642-33876-2_7.

Sébastien Ferré, Alice Hermann, and Mireille Ducassé. Combining faceted search and query languages for the semantic web. In *Advanced Information Systems*

Engineering Workshops, page 554–563, 2011. URL http://link.springer.com/chapter/10.1007/978-3-642-22056-2_57.

Oscar Ferrandez, Christian Spurk, Milen Kouylekov, Iustin Dornescu, Sergio Ferrandez, Matteo Negri, Ruben Izquierdo, David Tomas, Constantin Orasan, and Guenter Neumann. The QALL-ME framework: A specifiable-domain multilingual question answering architecture. *Web semantics: Science, services and agents on the world wide web*, 9(2):137–145, 2011. URL http://www.sciencedirect.com/science/article/pii/S1570826811000126.

O. Ferrández, R. Izquierdo, S. Ferrández, and J. L. Vicedo. Addressing ontology-based question answering with collections of user queries. *Information Processing & Management*, 45(2):175–188, 2009. URL http://www.gise.cse.iitb.ac.in/wiki/images/3/38/Sdarticle.pdf.

Robert Gaizauskas, Mark Hepple, Horacio Saggion, Mark A. Greenwood, and Kevin Humphreys. SUPPLE: a practical parser for natural language engineering applications. In *Proceedings of the Ninth International Workshop on Parsing Technology*, page 200–201, 2005. URL http://dl.acm.org/citation.cfm?id=1654521.

Bernhard Ganter, Gerd Stumme, and Rudolf Wille. *Formal concept analysis*. Springer Heidelberg, 1996. URL http://dl.kr.org/dl2008/Ganter.pdf.

Paul Gearon, Alexandre Passant, and Axel Polleres. SPARQL 1.1 update, March 2013. URL http://www.w3.org/TR/2013/REC-sparql11-update-20130321/.

Christine Golbreich and Peter F. Wallace. OWL 2 web ontology language new features and rationale (second edition), December 2012. URL http://www.w3.org/TR/owl2-new-features/.

Michael Grüninger and Mark S. Fox. Methodology for the design and evaluation of ontologies. 1995. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.8723.

W3C OWL Working Group. OWL 2 web ontology language document overview (second edition), December 2012. URL http://www.w3.org/TR/owl2-overview/.

W3C SPARQL Working Group. SPARQL 1.1 overview, March 2013. URL http://www.w3.org/TR/sparql11-overview/.

Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993. URL

http://secs.ceas.uc.edu/~mazlack/ECE.716.Sp2011/Semantic.Web.Ontology.Papers/Gruber.93a.pdf.

Carolyn Theresa Hafernik and Bernard J. Jansen. Understanding the specificity of web search queries. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, page 1827–1832, 2013. URL http://dl.acm.org/citation.cfm?id=2468684.

Catalina Hallett, Donia Scott, and Richard Power. Composing questions through conceptual authoring. *Computational Linguistics*, 33(1):105–133, 2007. URL http://dl.acm.org/citation.cfm?id=1245140.

Sanda M. Harabagiu, Dan I. Moldovan, Marius Pasca, Rada Mihalcea, Mihai Surdeanu, Razvan C. Bunescu, Roxana Girju, Vasile Rus, and Paul Morarescu. FALCON: boosting knowledge for answer engines. In *TREC*, volume 9, page 479–488, 2000. URL http://trec.nist.gov/pubs/trec9/papers/smu.pdf.

Shizhu He, Shulin Liu, Yubo Chen, Guangyou Zhou, Kang Liu, and Jun Zhao. CASIA@QALD-3: a question answering system over linked data. Valencia, Spain, September 2013. URL http://www.clef-initiative.eu/documents/71612/d035fda1-5c46-478a-ac95-fd114986f574.

Sebastian Hellmann, Jens Lehmann, Sören Auer, and Martin Brümmer. Integrating NLP using linked data. 2013. URL http://svn.aksw.org/papers/2013/ISWC_NIF/public.pdf.

L. Hirschman and R. Gaizauskas. Natural language question answering: the view from here. *Natural Language Engineering*, 7(4):275–300, 2001. URL http://dl.acm.org/citation.cfm?id=973891.

Pascal Hitzler, Markus Krotzsch, and Sebastian Rudolph. *Foundations of semantic web technologies*. Chapman and Hall/CRC, 2011. URL http://books.google.fr/books?hl=fr&lr=&id=BdzL24RqcGIC&oi=fnd&pg=PP1&dq=foundations+of+semantic+web+technologies&ots=EeESVwgz3G&sig=2ed1YzRaRO7VlqAT_tFXBLb2hs4.

Laura Hollink, Peter Mika, and Roi Blanco. Web usage mining with semantic analysis. In *Proceedings of the 22nd international conference on World Wide Web*, page 561–570, 2013. URL http://dl.acm.org/citation.cfm?id=2488438.

Vera Hollink, Theodora Tsikrika, and Arjen De Vries. Semantic vs term-based query modification analysis. *Information Foraging Lab*, page 39, 2010. URL http://www.academia.edu/download/30907593/proceedings_dir2010.pdf#page=41.

Eduard Hovy, Laurie Gerber, Ulf Hermjakob, Michael Junk, and Chin-yew Lin. Question answering in webclopedia. In *In Proceedings of the Ninth Text REtrieval Conference (TREC-9*, 2000. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.1195.

A. Hunter. Natural language database interfaces. *Knowl. Manage*, 2000.

American Film Institute. AFI's 100 years... 100 movie quotes, June 2005. URL http://www.afi.com/Docs/tvevents/pdf/quotes100.pdf.

Paul Jaccard. *Lois de distribution florale dans la zone alpine*. Corbaz, 1902.

Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912. URL http://onlinelibrary.wiley.com/doi/10.1111/j.1469-8137.1912.tb05611.x/abstract.

Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: a study of user queries on the web. In *ACM SIGIR Forum*, volume 32, page 5–17, 1998. URL http://dl.acm.org/citation.cfm?id=281253.

Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the web. *Information processing & management*, 36(2):207–227, 2000. URL http://www.sciencedirect.com/science/article/pii/S0306457399000564.

Bernard J. Jansen, Amanda Spink, and Sherry Koshman. Web searcher interaction with the dogpile. com metasearch engine. *Journal of the American Society for Information Science and Technology*, 58(5):744–755, 2007. URL http://onlinelibrary.wiley.com/doi/10.1002/asi.20555/full.

Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989. URL http://www.tandfonline.com/doi/abs/10.1080/01621459.1989.10478785.

Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *The knowledge engineering review*, 18(1):1–31, 2003. URL http://journals.cambridge.org/production/action/cjoGetFulltext?fulltextid=183300.

Hans Kamp. A theory of truth and semantic representation. *Formal semantics-the essential readings*, page 189–222, 1981. URL http://books.google.fr/books?hl=fr&lr=&id=ptgUWREtAkMC&oi=fnd&pg=PA189&dq=A+theory+of+truth+and+semantic+representation&ots=Yw7KaYbCG7&sig=EsZlW_c3uuGfRwkcquxua-S8RAA.

Hans Kamp and Uwe Reyle. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*. Number 42. Springer, 1993. URL http://books.google.fr/books?hl=fr&lr=&id=I5ESOZVryc0C&oi=fnd&pg=PA1&dq=from+discourse+to+logic&ots=X75LbU-T-t&sig=ugJCCpvIZJE4MN_2C2djX58NTco.

Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy J. Lin, Gregory Marton, Alton Jerome McFarland, and Baris Temelkuran. Omnibase: Uniform access to heterogeneous data for question answering. In *Proceedings of the 6th International Conference on Applications of Natural Language to Information Systems-Revised Papers*, page 230–234, 2002. URL http://dl.acm.org/citation.cfm?id=666145.

Esther Kaufmann and Abraham Bernstein. Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8 (4):377–393, 2010. URL http://www.sciencedirect.com/science/article/pii/S1570826810000582.

Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *In: 5th ISWC*, 2006. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.94.6633.

Esther Kaufmann, Abraham Bernstein, and Lorenz Fischer. NLP-Reduce: a "naıve" but domain-independent natural language interface for querying ontologies. 2007. URL http://gate.ac.uk/sale/dd/related-work/Kaufmann_nlp+reduce_ESWC2007.pdf.

Daniel Kayser. La représentation des connaissances. 1997. URL http://hal.archives-ouvertes.fr/hal-00091603/.

Michael Kifer and Harold Boley. RIF overview (second edition), February 2013. URL http://www.w3.org/TR/rif-overview/.

Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): concepts and abstract syntax, February 2004. URL http://www.w3.org/TR/rdf-concepts/.

Georgia Koutrika, Alkis Simitsis, and Yannis E. Ioannidis. Explaining structured queries in natural language. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, page 333–344, 2010. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5447824.

Cody Kwok, Oren Etzioni, and Daniel S. Weld. Scaling question answering to the web. *ACM Transactions on Information Systems (TOIS)*, 19(3):242–262, 2001. URL http://dl.acm.org/citation.cfm?id=502117.

Alex Lascarides. *Logics of conversation*. Cambridge University Press, 2003. URL http://books.google.fr/books?hl=fr&lr=&id=VD-8yisFhBwC&oi=fnd&pg=PR13&dq=logics+of+conversation&ots=iQe0v-ITly&sig=d4ICCf0xw3_lNR5BB7MxoKIiWbo.

Jens Lehmann and Lorenz Bühmann. AutoSPARQL: let users query your knowledge base. In *The Semantic Web: Research and Applications*, page 63–79. Springer, 2011. URL http://link.springer.com/chapter/10.1007/978-3-642-21034-1_5.

Yuangui Lei, Victoria Uren, and Enrico Motta. Semsearch: A search engine for the semantic web. In *Managing Knowledge in a World of Networks*, page 238–245. Springer, 2006. URL http://link.springer.com/chapter/10.1007/11891451_22.

Serge Linckels and Christoph Meinel. A simple solution for an intelligent librarian system. In *IADIS AC*, page 495–503, 2005. URL https://www.hpi.uni-potsdam.de/fileadmin/hpi/FG_ITS/papers/Web-University/2005_Linckels_AC.pdf.

Kenneth C. Litkowski. Syntactic clues and lexical resources in question-answering. In *TREC*, 2000. URL http://trec.nist.gov/pubs/trec9/papers/clresearch00.pdf.

Vanessa Lopez, Victoria Uren, Marta Sabou, and Enrico Motta. Is question answering fit for the semantic web ?: a survey. *Semantic Web*, 2(2):125–155, 2011. URL http://iospress.metapress.com/index/Q43K6J0118720542.pdf.

Vanessa Lopez, Christina Unger, Philipp Cimiano, and Enrico Motta. Evaluating question answering over linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 21:3–13, August 2013. ISSN 1570-8268. doi: 10.1016/j.websem.2013.05.006. URL http://www.sciencedirect.com/science/article/pii/S157082681300022X.

Johannes Lorey and Felix Naumann. Detecting SPARQL query templates for data prefetching. In *The Semantic Web: Semantics and Big Data*, page 124–139. Springer, 2013. URL http://link.springer.com/chapter/10.1007/978-3-642-38288-8_9.

Paul Martin, Douglas E. Appelt, Barbara J. Grosz, and Fernando Pereira. TEAM: an experimental transportable natural-language interface. In *Proceedings of 1986 ACM Fall joint computer conference*, page 260–267, 1986. URL http://dl.acm.org/citation.cfm?id=324576.

Suvodeep Mazumdar, Daniela Petrelli, and Fabio Ciravegna. Exploring user and system requirements of linked data visualization through a visual dashboard approach. *Semantic Web*, 2013. URL http://iospress.metapress.com/index/211118653W5756W3.pdf.

Deborah L. McGuiness and Frank van Harmelen. OWL web ontology language overview, February 2004. URL http://www.w3.org/TR/owl-features/.

Michael Minock. C-phrase: A system for building robust natural language interfaces to databases. *Data & Knowledge Engineering*, 69(3): 290–302, 2010. URL http://www.sciencedirect.com/science/article/pii/S0169023X09001499.

Dan Moldovan, Marius Pa\csca, Sanda Harabagiu, and Mihai Surdeanu. Performance issues and error analysis in an open-domain question answering system. *ACM Transactions on Information Systems (TOIS)*, 21(2):133–154, 2003. URL http://dl.acm.org/citation.cfm?id=763694.

Dan I. Moldovan, Sanda M. Harabagiu, Marius Pasca, Rada Mihalcea, Richard Goodrum, Roxana Girju, and Vasile Rus. LASSO: a tool for surfing the answer net. In *TREC*, volume 8, page 65–73, 1999. URL http://trec.nist.gov/pubs/trec8/papers/smu.pdf.

Dan I. Moldovan, Sanda M. Harabagiu, Roxana Girju, Paul Morarescu, V. Finley Lacatusu, Adrian Novischi, Adriana Badulescu, and Orest Bolohan. LCC tools for question answering. In *TREC*, 2002. URL http://trec.nist.gov/pubs/trec11/papers/lcc.moldovan.pdf.

Le Monde. Pour ses 15 ans, google présente son nouvel algorithme, September 2013. URL http://www.lemonde.fr/technologies/article/2013/09/27/pour-ses-15-ans-google-presente-son-nouvel-algorithme_3486330_651865.html.

Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 web ontology language profiles (second edition), December 2012. URL http://www.w3.org/TR/owl2-profiles/.

Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber. SPARQL2NL: verbalizing sparql queries. In *Proceedings of*

the 22nd international conference on World Wide Web companion, page 329–332, 2013. URL http://dl.acm.org/citation.cfm?id=2487936.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. MaltParser: a language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007. URL http://journals.cambridge.org/production/action/cjoGetFulltext?fulltextid=1012772.

Natasha Noy and Alan L. Rector. Defining n-ary relations on the semantic web, April 2006. URL http://www.w3.org/TR/swbp-n-aryRelations/.

S. Ou, C. Orasan, D. Mekhaldi, and L. Hasler. Automatic question pattern generation for ontology-based question answering. In *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference. Menlo Park, CA: AAAI Press*, 2008. URL http://clg.wlv.ac.uk/papers/ou-FLAIRS-08.pdf.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, page 149–157, 2003. URL http://dl.acm.org/citation.cfm?id=604070.

María Poveda-Villalón, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Validating ontologies with OOPS! In *Knowledge Engineering and Knowledge Management*, page 267–281. Springer, 2012. URL http://link.springer.com/chapter/10.1007/978-3-642-33876-2_24.

C. Pradel, O. Haemmerlé, and N. Hernandez. Expressing conceptual graph queries from patterns: how to take into account the relations. In *Proceedings of the 19th International Conference on Conceptual Structures, ICCS'11, Lecture Notes in Artificial Intelligence #6828*, pages 234–247, Derby, GB, July 2011a. Springer.

C. Pradel, O. Haemmerlé, and N. Hernandez. A semantic web interface using patterns: the SWIP system. In *Proceedings of GKR 2011*, pages 172–187, Barcelona, Spain, July 2011b. Croitoru et al.

Camille Pradel. Allowing end users to query graph-based knowledge bases. In *Knowledge Engineering and Knowledge Management*, page 8–15, Galway City, Ireland, October 2012. URL http://www.springerlink.com/index/G4841WL5650857M1.pdf.

Camille Pradel, Ollivier Haemmerlé, and Nathalie Hernandez. Expression de requêtes SPARQL à partir de patrons: prise en compte des relations. In *22es Journées Francophones d'Ingénierie des Connaissances*, page 771–787, Chambéry,

France, May 2012a. Pierre Antoine Champin, Jean Charlet, Nathalie Hernandez, Raphaël Troncy. URL http://hal.archives-ouvertes.fr/hal-00746737/.

Camille Pradel, Nathalie Hernandez, Mouna Kamel, and Bernard Rothenburger. Une ontologie du cinéma pour évaluer les applications du web sémantique. In *Atelier Ontologies et Jeux de Données pour évaluer le web sémantique (OJD) à IC 2012*, Paris, France, June 2012b. Nathalie Aussenac-Gilles, Jean Charlet, Mouna Kamel, Nathalie Hernandez, Camille Pradel. URL http://ontologies.alwaysdata.net/site_media/OJD_4.pdf.

Camille Pradel, Ollivier Haemmerlé, and Nathalie Hernandez. Natural language query interpretation into SPARQL using patterns. In *COLD@ISWC2013*, Sydney (Australia), October 2013a.

Camille Pradel, Ollivier Haemmerlé, and Nathalie Hernandez. SWIP at QALD-3: results, criticisms and lesson learned. Valencia, Spain, September 2013b. URL http://www.clef-initiative.eu/documents/71612/d035fda1-5c46-478a-ac95-fd114986f574.

Eric Prud'hommeaux and Carlos Buil-Aranda. SPARQL 1.1 federated query, March 2013. URL http://www.w3.org/TR/sparql11-federated-query/.

Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF, January 2008. URL http://www.w3.org/TR/rdf-sparql-query/.

M. Ross Quillian. Semantic memory. Technical report, DTIC Document, 1966. URL http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0641671.

Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001. URL http://link.springer.com/article/10.1007/s007780100057.

Yves Raimond, Samer A. Abdallah, Mark B. Sandler, and Frederick Giasson. The music ontology. In *ISMIR*, page 417–422, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.173.5403&rep=rep1&type=pdf.

Alan L. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In *Proceedings of the 2nd international conference on Knowledge capture*, page 121–128, 2003. URL http://dl.acm.org/citation.cfm?id=945664.

Monique Reichert, Serge Linckels, Christoph Meinel, and Thomas Engel. Student's perception of a semantic search engine. In *CELDA*, page

139–147, 2005. URL http://www.hpi.uni-potsdam.de/fileadmin/hpi/FG_ITS/papers/Web-University/2005_Reichert_CELDA.pdf.

Alistair Russell and Paul Smart. Nitelight: A graphical editor for sparql queries. 2008. URL http://eprints.soton.ac.uk/266258/.

Stefan Schlobach and Craig A. Knoblock. Dealing with the messiness of the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:1, 2012.

Helmut Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of international conference on new methods in language processing*, volume 12, page 44–49, 1994. URL http://www.stttelkom.ac.id/staf/imd/Riset/POS%20Tagging/Using%20Decision%20Tree.pdf.

Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. In *Journal on Data Semantics IV*, page 146–171. Springer, 2005. URL http://link.springer.com/chapter/10.1007/11603412_5.

C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. In *ACm SIGIR Forum*, volume 33, page 6–12, 1999. URL http://dl.acm.org/citation.cfm?id=331405.

Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

John F. Sowa. Conceptual graphs for a data base interface. *IBM Journal of Research and Development*, 20(4):336–357, 1976. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5391080.

John F. Sowa. Conceptual structures: information processing in mind and machine. 1984. URL http://www.osti.gov/energycitations/product.biblio.jsp?osti_id=5673179.

A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic. Searching the web: The public and their queries. *Journal of the American society for information science and technology*, 52(3):226–234, 2000. URL http://onlinelibrary.wiley.com/doi/10.1002/1097-4571(2000)9999:9999%3C::AID-ASI1591%3E3.0.CO;2-R/full.

Rohini Srihari and Wei Li. Information extraction supported question answering. Technical report, DTIC Document, 1999. URL http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA460042.

Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1): 161–197, 1998. URL http://www.sciencedirect.com/science/article/pii/S0169023X97000566.

Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Enrico Motta, and Aldo Gangemi. *Ontology engineering in a networked world*. Springer, 2012a. URL http://books.google.fr/books?hl=fr&lr=&id=wdYrJTInTS4C&oi=fnd&pg=PR7&dq=Ontology+Engineering+in+a+Networked+World&ots=zYgRxlHlc_&sig=s57cNwwC_pfrdBJsyJeit-3ctUs.

Mari Carmen Suárez-Figueroa, Camille Pradel, and Nathalie Hernandez. Verifying ontology requirements with SWIP (poster), October 2012b. URL http://ekaw2012.ekaw.org/sites/ekaw2012.ekaw.org/files/ekaw2012pd_submission_14%20(3).pdf. EKAW 2012, Galway City, Ireland.

Valentin Tablan, Danica Damljanovic, and Kalina Bontcheva. A natural language query interface to structured information. In *The Semantic Web: Research and Applications*, page 361–375. Springer, 2008. URL http://link.springer.com/chapter/10.1007/978-3-540-68234-9_28.

Jiao Tao, Evren Sirin, Jie Bao, and Deborah L. McGuinness. Integrity constraints in OWL. In *AAAI*, 2010. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/download/1931@misc/2229.

CoGui Team. CoGui, 2007. URL http://www.lirmm.fr/cogui/.

P. Thanisch, I. Androutsopoulos, and G. D. Ritchie. Masque/sql-an efficient and portable natural language query interface for relational databases. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Proceedings of the Sixth International Conference Held in Edinburgh, Scotland, June 1-4, 1993*, page 327, 1993.

Craig W. Thompson, Paul Pazandak, and Harry R. Tennant. Talk to your semantic web. *Internet Computing, IEEE*, 9(6):75–78, 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1541951.

T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, page 405–416, 2009. URL http://people.aifb.kit.edu/dtr/papers/keywordtopk.pdf.

Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. URL http://www.jstor.org/stable/10.2307/2251299.

Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over RDF data. In *Proceedings of the 21st international conference on World Wide Web*, page 639–648, 2012. URL http://dl.acm.org/citation.cfm?id=2187923.

Victoria Uren, Philipp Cimiano, José Iria, Siegfried Handschuh, Maria Vargas-Vera, Enrico Motta, and Fabio Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: science, services and agents on the World Wide Web*, 4(1): 14–28, 2006. URL http://www.sciencedirect.com/science/article/pii/S1570826805000338.

Chong Wang, Miao Xiong, Qi Zhou, and Yong Yu. Panto: A portable natural language interface to ontologies. In *The Semantic Web: Research and Applications*, page 473–487. Springer, 2007. URL http://link.springer.com/chapter/10.1007/978-3-540-72667-8_34.

Haofen Wang, Kang Zhang, Qiaoling Liu, Thanh Tran, and Yong Yu. Q2semantic: A lightweight keyword interface to semantic search. In *The Semantic Web: Research and Applications*, page 584–598. Springer, 2008. URL http://link.springer.com/chapter/10.1007/978-3-540-68234-9_43.

William E. Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.4336.

Min Wu, Xiaoyu Zheng, Michelle Duan, Ting Liu, and Tomek Strzalkowski. Question answering by pattern matching, web-proofing, semantic form proofing. In *NIST Special Publication, in: The 12th Text Retrieval Conference (TREC), 2003*, page 578–585, 2003. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.9753.

Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. SPARK: adapting keyword query to semantic search. In *The Semantic Web*, page 694–707. Springer, 2007. URL http://link.springer.com/chapter/10.1007/978-3-540-76298-0_50.