Modèle d'acquisition de données du LOD et génération automatique de requêtes

Céline Alec*, Chantal Reynaud-Delaître*, Brigitte Safar*

*LRI, Univ. Paris-Sud, CNRS, Université Paris-Saclay, Orsay, F-91405
prenom.nom@lri.fr

Résumé. Le LOD (Linked Open Data) représente aujourd'hui une source prometteuse pour de très nombreuses applications du Web Sémantique à condition toutefois de développer des techniques d'acquisition adaptées. Le travail présenté dans ce papier traite de ce problème. L'objectif est de peupler une ontologie OWL avec des assertions de propriétés en tenant compte de l'existence, dans le LOD, de propriétés multiples, équivalentes, multi-valuées, de propriétés absentes mais pouvant toutefois être calculées, de propriétés sans valeurs. Les correspondances à établir étant complexes, nous proposons un modèle pour les spécifier. Nous définissons également un modèle pour spécifier des chemins d'accès alternatifs à des propriétés en cas de valeurs manquantes. Enfin, les valeurs de propriétés auxquelles le LOD donne accès après application des modèles de correspondance et d'accès peuvent nécessiter des traitements complémentaires de façon à satisfaire les contraintes de l'ontologie à peupler. Nous proposons des mécanismes adaptés. Puis, nous montrons comment les différents modèles proposés sont utilisés pour générer automatiquement des requêtes SPARQL et ainsi faciliter l'interrogation. Cette génération automatique est déroulée sur deux exemples.

1 Introduction

Un des faits marquants de l'évolution de l'ingénierie des connaissances est d'avoir diversifié les sources de connaissances utilisées dans les systèmes de traitement d'information intelligents, permettant ainsi de tirer profit de leur complémentarité (Aussenac-Gilles et al., 2014). Le web des données liées est aujourd'hui une source de connaissances utilisable, tout comme les documents numériques exploités en complément d'experts spécialistes du domaine, les données dont sont extraites des connaissances, les modèles, ressources, thesaurus, représentations structurées comme les ontologies. Les données liées réfèrent à un style de publication et d'interconnexion de données structurées sur le web, basé sur le modèle RDF. Elles ont l'avantage de fournir un mécanisme d'accès unique et normalisé au lieu de s'appuyer sur différents formats d'interface et de résultat. Les sources de données peuvent être ainsi facilement explorées par les moteurs de recherche. Par définition, elles ont des liens avec d'autres sources de données. Le nombre de données publiées selon ce principe croît rapidement (on parle aujourd'hui de dizaines de milliards de triplets publiés sur Internet). Cette masse de données représente

une source prometteuse utilisée dans de très nombreuses applications du web sémantique, à condition toutefois de développer des techniques de collecte de données adaptées. En effet, des études (Zaveri et al., 2014) ont mis en évidence des problèmes de qualité tels que l'incomplétude, la présence d'informations redondantes, l'inexactitude de certaines données. Le travail présenté dans ce papier traite du problème spécifique d'acquisition de données du LOD (Linked Open Data).

Notre travail se situe dans le contexte d'une application basée sur une ontologie qui combine différentes méthodes pour résoudre une tâche complexe. L'une de ces méthodes a pour objectif de découvrir des définitions de concepts recouvrant des ensembles d'individus dont les assertions de propriétés sont supposées connues. Cette méthode (Alec et al., 2016a,b) met en œuvre des techniques d'apprentissage automatique et la qualité des résultats du processus de découverte dépend des données fournies en entrée du processus, en particulier des assertions de propriétés décrivant les individus considérés. Il est donc nécessaire de peupler au préalable l'ontologie du domaine de l'application avec toutes les assertions de propriétés concernées (ou un maximum). Ce peuplement est réalisé en partie via un processus d'extraction de connaissances à partir de textes. En complément, les valeurs des propriétés concernées non citées dans les textes analysés sont recherchées dans le LOD. Le concepteur sélectionne les jeux de données les plus pertinents pour ces propriétés et indique ensuite les propriétés à rechercher dans ces sources. Le nombre de ces propriétés étant supposé ne pas être trop grand, cette tâche peut donc être réalisée manuellement. Toutefois les termes utilisés pour dénoter ces propriétés pouvant être différents dans l'ontologie et dans les jeux de données du LOD, des correspondances doivent être établies et elles peuvent être complexes, comme le montrent les exemples suivants.

Le premier exemple illustre une correspondance de type $1 : n$. Une propriété $prop$ caractérisant un concept c de l'ontologie $\mathcal{O}$ peut être associée à plusieurs propriétés équivalentes dans un jeu de données du LOD, avec des noms syntaxiquement différents et des valeurs pouvant être exprimées dans différentes unités de mesure. Ainsi, la propriété precipitationMm du mois de janvier d'une destination bien précise est représentée dans DBpedia (Auer et al., 2008), entre autres, par dbp:janPrecipitationMm, dbp:janRainMm, dbp:janRainInch. Le second exemple est encore une correspondance de type $1 : n$ mais dans ce cas, les propriétés ne sont pas équivalentes et toutes les valeurs des propriétés sont nécessaires pour calculer la valeur de $prop$. Par exemple, la température dans un lieu donné et un mois donné peut être obtenue dans DBpedia en calculant la moyenne entre la température la plus haute et la température la plus basse dans ce lieu ce mois-là. Le troisième exemple concerne des valeurs manquantes. En effet, bien que la base de connaissances DBpedia soit gigantesque, elle est aussi incomplète. Beaucoup de propriétés n'ont pas de valeurs. Par exemple, une destination donnée peut ne pas avoir de valeur pour la température la plus haute un mois donné. Nous nommons ces correspondances des "correspondances complexes" car celles-ci ne sont pas de simples correspondances $1 : 1$. Par ailleurs, elles peuvent être établies entre des propriétés qui ne sont pas forcément équivalentes (propriétés intervenant dans un calcul ou une transformation).

Le peuplement d'une ontologie à l'aide d'assertions de propriétés doit tenir compte de l'ensemble de ces situations, propriétés multiples, valeurs manquantes, tout en tenant compte du fait que les propriétés peuvent aussi être multi-valuées. L'obtention de données les plus complètes possibles est très important dans notre contexte pour obtenir les résultats les meilleurs possibles en sortie du processus applicatif complet mis en œuvre. Notre travail porte donc sur la collecte de données du LOD compte tenu du fait que l'information qui y est explicitement re-

présentée ne correspond pas forcément à la structure et au format de l'information recherchée, que les données comportent des redondances, et qu'elles peuvent, en plus, être incomplètes. Ceci nécessite de spécifier précisément les traitements portant sur les données multiples disponibles dans le LOD qui vont permettre d'obtenir l'information recherchée. Ces traitements peuvent être complexes, complexifiant d'autant l'interrogation des données en SPARQL, le langage standard d'interrogation des données RDF (Harris et al., 2013). Afin de faciliter ce processus, et d'éviter au concepteur la phase d'écriture des requêtes en SPARQL, nous proposons un modèle de correspondances et d'accès aux données du LOD à partir d'une ontologie. Ce modèle, complété par des mécanismes de traitement des valeurs de propriétés acquises pour satisfaire les contraintes de l'ontologie à peupler, est adapté à la spécification de traitements complexes. Il est utilisé comme support à la fois, pour la collecte de données et pour la génération automatique de requêtes. Nous considérons en effet que la structure des requêtes SPARQL peut être déterminée par le modèle de correspondance et d'accès proposé, et ceci de façon indépendante du domaine d'application. L'originalité de notre travail porte sur la prise en compte de traitements complexes permettant d'accéder à des informations non explicitement représentées dans le LOD, de la spécification de ces traitements à leur exécution, et ce d'une façon totalement transparente, un aspect qui, à notre connaissance, n'a pas encore été traité par les travaux du domaine.

Ce papier est structuré de la façon suivante. Dans un premier temps, nous présentons des exemples de traitements complexes dont la mise en œuvre motive notre travail. Puis, nous présentons des travaux proches effectués dans différents cadres comme la médiation de données, l'accès facilité aux données liées ou l'établissement de correspondances complexes. Nous présentons ensuite le modèle d'acquisition des valeurs de propriétés du LOD à partir d'une ontologie OWL. Ensuite, nous expliquons comment ce modèle sert de support à la génération automatique de requêtes SPARQL. La dernière partie déroule des exemples illustrant l'intérêt du modèle proposé et le processus de génération des requêtes. Enfin, nous concluons et énonçons quelques perspectives.

2 Cas d'utilisation illustrant nos objectifs

Dans cette section, nous donnons des exemples de cas d'utilisation ayant motivé notre travail.

Exemple 1. Supposons que l'on recherche, dans DBpedia, la valeur de la propriété supPopulationKm2 d'une ontologie myOnto, correspondant à la densité maximale de la population au km^2 d'un lieu. Une analyse des données de DBpedia montre (cf. figure 1) qu'un lieu peut avoir une propriété exprimant la densité de population avec éventuellement plusieurs valeurs et également d'autres propriétés, comme la superficie et le nombre d'habitants, qui peuvent aussi permettre de calculer la densité de la population. Si on veut connaître la valeur de la densité maximale de la population au km^2 d'un lieu, il faut utiliser toutes ces données. supPopulationKm2 de myOnto correspond donc à Max(Max(populationDensity),Max(populationTotal)/Min(areaTotal)) dans DBpedia. L'obtention de la valeur de supPopulationKm2 nécessite de collecter les valeurs de populationTotal et areaTotal en ne retenant que la valeur maximale et minimale respectivement, d'extraire la densité maximale donnée par populationDensity puis celle calculée à partir du rapport entre Max(populationTotal) et Min(areaTotal), puis de ne retenir que la valeur maximale de ces

deux densités. Cela passe par l'écriture d'une requête SPARQL avec des requêtes imbriquées, des sous-requêtes reliées par l'opérateur UNION, l'usage d'opérateurs d'agrégation et de l'opérateur BIND pour spécifier les calculs ainsi que les changements de format nécessaires. Cette requête, dont un exemple appliqué au Canada est présenté ci-dessous, n'est pas simple. On supposera connue la correspondance entre l'individu représentant le Canada dans myOnto, noté myOnto:Canada, et celui de DBpedia, noté dbr:Canada.

```
CONSTRUCT { myOnto:Canada myOnto:supPopulationKm2 ?val0 }
WHERE{
   {
      SELECT (MAX(?val2) AS ?val1)
      WHERE {
         {
            SELECT (MAX(?val3) AS ?val2)
            WHERE { dbr:Canada dbo:populationDensity ?val3. }
         } UNION {
            {
               SELECT (MIN(?val4) AS ?val3a)
               WHERE { dbr:Canada <dbo:PopulatedPlace/areaTotal> ?val4. }
            }
            {
               SELECT (MAX(?val4) AS ?val3b)
               WHERE { dbr:Canada dbo:populationTotal ?val4. }
            }
            BIND (?val3b/?val3a AS ?val2)
         }
      }
   }
   BIND (xsd:double(?val1) AS ?val0)
}
```

Exemple 2. La recherche porte dans ce second cas sur la valeur de precipitationDays, une propriété fonctionnelle dans myOnto qui représente le nombre de jours pluvieux caractérisant la classe Weather. Les instances de la classe Weather correspondent aux données météorologiques d'un lieu donné pour un mois donné. Ainsi, myOnto:WeatherJanuarySarnia est un exemple d'instance de Weather correspondant aux données météorologiques de Sarnia en janvier. Cette instance est caractérisée par les deux propriétés suivantes représentées sous forme de triplets : <myOnto:Sarnia myOnto:hasWeather myOnto:WeatherJanuarySarnia> <myOnto:WeatherJanuarySarnia myOnto:concernMonth myOnto:January>. Supposons que la recherche porte sur la valeur du nombre de jours pluvieux à Sarnia en janvier, soit la valeur de PrecipitationDays de l'instance myOnto:WeatherJanuarySarnia. Une analyse des données de DBpedia (cf. Figures 2 et 3) montre que la propriété janPrecipitationDays caractérise les lieux décrits dans DBpedia. Toutefois, cette propriété est parfois multi-valuée, et dans ce cas on pourra être intéressé par la moyenne des valeurs indiquées. En généralisant, on peut donc établir une correspondance entre precipitationDays avec la contrainte sur son domaine restreignant cette propriété aux instances liées par la relation concernMonth au mois de janvier dans myOnto et la moyenne des valeurs de janPrecipitationDays dans DBpedia de façon à ne collecter qu'une seule valeur puisque la propriété est fonctionnelle. Dans ce cas, la complexité de la mise en correspondance provient de l'expression de la contrainte associée à la propriété dans l'ontologie pour laquelle on cherche la valeur, à laquelle s'ajoute le calcul de moyenne. L'expression de contraintes s'explique par le fait que les modèles dans l'ontologie et dans les sources de données, comme DBpedia, sont différents.

FIG. 1 – *Propriétés de Canada dans DBpedia*

FIG. 2 – *Propriétés de Sarnia dans DBpedia*

FIG. 3 – *Propriétés de Juneau dans DBpedia*

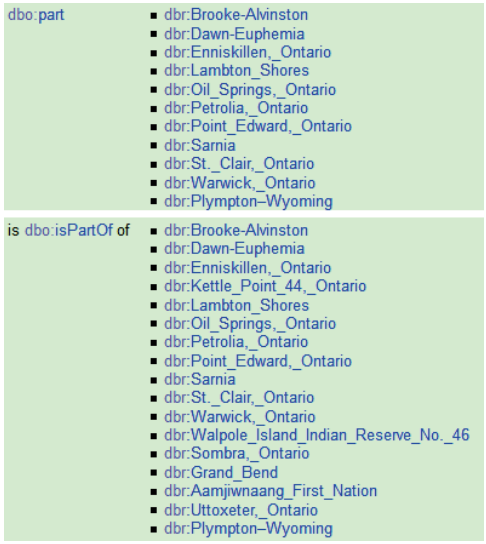

FIG. 4 – *Propriétés de Lambton County dans DBpedia*

Notons que cette correspondance concernant precipitationDays décrite ci-dessus ne retournera le résultat attendu que si janPrecipitationDays est valuée dans DBpedia. Or, les lieux décrits dans DBpedia n'ont pas tous une valeur pour cette propriété. Nous souhaitons, dans ce cas, trouver des valeurs de substitution qui, bien qu'approximatives, pourraient être utiles (par exemple, le nombre de jours pluvieux d'une sous-partie géographique de Lambton County cf. Figure 4). Ce processus de substitution va toutefois compliquer l'écriture des requêtes (cf. requête ci-dessous correspondant à l'exemple cité).

```
CONSTRUCT { myOnto:WeatherJanuaryLambtonCounty myOnto:precipitationDays ?val0 }
WHERE{
   {
      SELECT (AVG(?val2) AS ?val1)
      WHERE{

         SELECT ?ind1 (AVG(?val3) AS ?val2)
         WHERE {
            {?ind1 dbo:isPartOf dbr:Lambton_County} UNION
            {dbr:Lambton_County dbo:part ?ind1}
            ?ind1 dbp:janPrecipitationDays ?val3.
         }
         GROUP BY ?ind1
      }
   }
   BIND (xsd:double(?val1) AS ?val0)
}
```

3 Travaux proches

Notre travail peut être rapproché de travaux de médiation de données qui concernent la ré-écriture de requêtes SPARQL de façon à pouvoir les exécuter sur différents jeux de données du LOD.

Correndo et al. (2010) proposent un algorithme de ré-écriture de requêtes qui repose sur l'utilisation de règles de ré-écriture de motifs de graphe et exploite l'alignement entre les ontologies associées aux jeux de données. Les règles de ré-écriture peuvent faire état de matchings non triviaux (par exemple, l'association has-author correspond à la composition creator-Info o hasCreator). L'originalité de ce travail réside dans l'expression de contraintes qui conditionnent l'usage des règles. Ces contraintes peuvent porter sur les unités de mesure dans lesquelles sont exprimées les données, ce qui doit déclencher un processus de conversion au moment de la ré-écriture de la requête, ou sur le format des données (adresse correspondant à une seule valeur ou à plusieurs : adresse rue, code postal et ville). Ces contraintes correspondent donc à des traitements à effectuer sur les données, même si ces derniers restent relativement simples. Notre travail les considère également.

Makris et al. (2012) présentent une méthode générique de ré-écriture de requêtes SPARQL qui rend le processus d'interrogation des données liées totalement transparent pour l'utilisateur. Cette ré-écriture, dont la correction et la complétude ont été prouvées formellement, est faite par rapport à un ensemble de mappings entre ontologies, pré-définis et représentés en OWL (Makris et al., 2010). L'accent est mis sur la richesse des types de mappings considérés, ceux-ci étant établis entre des expressions de classes, de propriétés ou d'individus à l'aide de relations d'équivalence ou de subsomption. Notre travail est proche du fait de la représentation en OWL des mappings considérés et donc de la possibilité d'exprimer également des contraintes OWL sur les éléments mis en correspondance, mais nous nous intéressons à des mappings plus complexes qui peuvent être exprimés à l'aide de plusieurs agrégats appliqués successivement.

Notre travail peut également être rapproché de ceux réalisés pour faciliter l'accès aux données liées, via le développement d'interfaces qui exploitent l'expressivité du modèle de données sous-jacent et du langage de requêtes, tout en cachant leur complexité.

Les systèmes de questions-réponses sont des solutions permettant à un utilisateur d'interroger le web des données liées en formulant des requêtes en langage naturel de façon assez intuitive (Kaufmann et Bernstein, 2010). Dans beaucoup de ces systèmes, les questions sont associées à des représentations sous forme de triplets RDF. Des sous-graphes sont alors extraits de jeux de données RDF après application d'heuristiques et de mesures de similarité. Cette solution fonctionne assez bien pour des requêtes facilement compréhensibles pour lesquelles la structure de la question peut aisément et automatiquement être associée à une représentation sous forme de triplets. Des correspondances de vocabulaires entre les termes de la question et ceux des jeux de données sont supposées exister. Le système de question-réponse décrit dans (Unger et al., 2012) considère des situations plus complexes. Il s'agit de questions en anglais comprenant des comparateurs comme more than ou the most, qui nécessiteront une clause `HAVING` ou `ORDER BY` en SPARQL. L'approche proposée consiste à analyser la question, via des techniques de traitement automatique de la langue, pour produire un modèle de requête SPARQL reflétant la structure de la question, indépendant du domaine d'application, qui sera ensuite instancié. Ces travaux sont intéressants car ils visent à simplifier l'accès aux données du LOD. Ils supposent toutefois que, modulo des mappings donnés ou découverts, l'informa-

tion recherchée est explicitement représentée (ou peut être facilement calculée par application d'une fonction agrégation), qu'elle ne résulte pas d'un traitement à réaliser portant sur plusieurs données différentes, et qu'elle peut donc être obtenue par une seule requête. L'idée de la construction d'un modèle de requête SPARQL par analyse linguistique des questions, proposée dans (Unger et al., 2012), est toutefois intéressante. De façon analogue, nous proposons de construire des requêtes SPARQL à partir de patrons qui sont ensuite instanciés. Nos patrons sont basés sur les modèles de correspondance et d'accès aux données qui intégrent la spécification de traitements complexes.

D'autres travaux, comme celui décrit dans (Shekarpour et al., 2011), se donnent pour objectif de faciliter la construction de requêtes SPARQL pour interroger le web des données liées à partir de requêtes mots-clefs, moins ambiguës que des requêtes en langage naturel, et dont le traitement peut être plus efficace. Il s'agit de proposer des réponses les plus pertinentes possibles alors que les relations entre les mots clefs dans la question posée ne sont pas précisées. L'accent est mis sur la capacité à rechercher les tuples du LOD satisfaisant la requête mots-clefs quand ils existent. Des patrons de requêtes SPARQL sont associés aux requêtes mots-clefs en fonction du type de recherche : recherche de valeurs de propriétés d'individus, d'individus partageant des propriétés, ou d'associations/compositions d'associations entre individus. Le processus est efficace car les requêtes ne donnent pas lieu à des calculs complexes. Nous nous différencions de cette deuxième catégorie de travaux par le fait que nous nous intéressons, en plus, à l'obtention d'informations non explicitement représentées, nécessitant la recherche au préalable de données sur lesquelles des traitements complexes doivent être effectués. Les données exploitées peuvent ne pas être équivalentes ou similaires aux données recherchées mais correspondre à des données intervenant dans un calcul. La requête à écrire pour obtenir les données recherchées issues de traitements complexes doit bien souvent comprendre plusieurs sous-requêtes imbriquées, qui ne peuvent être dérivées de l'analyse du contenu des questions posées.

Enfin notre travail peut également être rapproché de travaux en alignement d'ontologies. Beaucoup de travaux ont été réalisées dans ce domaine ces dernières années (Euzenat et Shvaiko, 2013) mais la plupart des outils ne génèrent que des correspondances simples entre entités atomiques. Dans cet état de l'art, nous ne nous intéresserons qu'aux travaux concernant les correspondances complexes. Des patrons de correspondance, définis dans (Scharffe, 2009) et (Scharffe et al., 2014), sont présentés comme des outils pour faciliter la définition de correspondances complexes. Certains permettent de spécificier les conditions qu'une entité d'une ontologie doit satisfaire pour être mise en correspondance avec une entité d'une autre ontologie. D'autres indiquent des conversions d'unités de mesure ou une modification du type d'une donnée. Ce travail est vu comme une première étape dans la définition des correspondances complexes. Les patrons aident à affiner des correspondances simples préalablement calculées. Les travaux dont l'objectif est de trouver des correspondances complexes le font en général à partir de correspondances simples. C'est le cas, par exemple, dans (Pereira Nunes et al., 2013) qui porte sur la recherche de correspondances complexes entre propriétés. Celles-ci sont établies à partir de correspondances simples obtenues par application de techniques d'analyse des instances des propriétés. Enfin, nous citerons le travail de Gillet et al. (2013), qui définit la notion de correspondances complexes pour la ré-écriture de patrons de requêtes. Ces correspondances sont des associations de type $1 : n$ ou $n : 1$ ou $n : m$ exprimées à l'aide des constructeurs de la logique de description. Les relations modélisées sont les relations Equi-

valence, Plus général, Plus spécifique. Pour les expérimentations réalisées, les correspondances complexes ont été établies manuellement sur la base de patrons proposés dans la littérature, dont la couverture a été jugée suffisante. La ré-écriture est appliquée à un ensemble de patrons de requêtes. Des règles ont été définies pour traduire une correspondance complexe formalisée en un graph pattern RDF et guider le processus de ré-écriture.
Dans tous les travaux de cette troisième catégorie, nous constatons que les correspondances complexes sont toujours définies à partir de correspondances simples. C'est sur ce point que nous nous différencions. En effet lorsque l'entité cible avec laquelle l'entité source est associée est le résultat d'un traitement complexe mettant en œuvre successivement différents agrégats, la correspondance ne peut pas être dérivée de correspondances simples. Les entités faisant l'objet de tels traitements complexes ne peuvent pas être découvertes par un système d'alignement.

4 Modèle d'acquisition de valeurs de propriétés du LOD à partir d'une ontologie OWL

On considère une ontologie source O_s et une ontologie cible O_t. O_s est une ontologie OWL. O_t une ontologie fournissant un accès à des sources RDF, telles qu'un jeu de données du LOD. Chaque ontologie est définie comme un tuple ($\mathcal{C}$, $\mathcal{P}$, $\mathcal{I}$, $\mathcal{A}$) où $\mathcal{C}$ est l'ensemble des classes, $\mathcal{P}$ l'ensemble des propriétés (datatype, objet et annotation) caractérisant les classes, $\mathcal{I}$ un ensemble d'instances et d'assertions de propriété, et $\mathcal{A}$ un ensemble d'axiomes.

Cette section définit un modèle d'acquisition des propriétés de O_t utiles pour valuer des propriétés de l'ontologie source $\mathcal{O}_s$ supposées connues. Remplir cette tâche impose de résoudre deux problèmes : (1) trouver l'individu i_t de $\mathcal{O}_t$ correspondant à un individu i_s de O_s dont on veut valuer une propriété, (2) trouver les propriétés O_t, dites propriétés cibles, correspondant à la propriété de l'ontologie source $\mathcal{O}_s$, dite propriété source, à valuer. Cet article ne porte que sur le second point. On supposera qu'une première étape préalablement effectuée a conduit à un ensemble $Ind_{s,t}$ de couples (i_s,i_t) où un individu de l'ontologie source correspond à un individu de l'ontologie cible. La seconde étape prend appui sur un modèle d'acquisition de valeurs de propriétés d'un jeu de données du LOD composé de trois sous-parties :

- d'un modèle qui établit une correspondance entre une propriété source et une propriété cible,
- d'un modèle qui définit l'accès aux valeurs des propriétés recherchées,
- de mécanismes de traitement des valeurs finales acquises, permettant entre autres d'agréger de différentes manières les informations obtenues via les modèles de correspondance et d'accès.

4.1 Modèle de correspondance

Le modèle de correspondance définit des correspondances (PE_s,PE_t) entre une Expression de Propriété source (PE_s) et une Expression de Propriété cible (PE_t). Une correspon-

dance est applicable pour tout couple $(i_s, i_t) \in Ind_{s,t}$ tel que :

$$\begin{cases} i_s \in & domaine(PE_s) \\ i_t \in & domaine(PE_t) \end{cases}$$

Dans ce qui suit, on appellera *propriété datatype*, une propriété dont le co-domaine est un littéral, et *propriété objet*, une propriété dont le co-domaine est une expression de classes. Définissons tout d'abord la notion d'expression de propriétés (PE_s) dans $\mathcal{O}_s$.

Définition 1. *Une expression de propriété dans $\mathcal{O}_s$ (PE_s), est une propriété objet (op) ou une propriété datatype (dp), sur le domaine desquelles on peut exprimer une restriction (notée $PE_s.Restrict(d)$), en OWL DL.*
PE_s = *op* | *dp* | $PE_s.Restrict(d)$

Exemple 3. La propriété objet country est une PE_s. La propriété datatype latitude est une PE_s. La propriété precipitationMm.(concernMonth VALUE January) est une PE_s. Elle concerne la propriété datatype precipitationMm dont le domaine est contraint par (concernMonth VALUE January) exprimée en syntaxe Manchester (Horridge et al., 2006). Elle permet de valuer la propriété precipitationMm uniquement pour les i_s correspondant à des données météo qui concernent le mois de Janvier.

Nous définissons maintenant la notion d'expression de propriétés (PE_t) dans $\mathcal{O}_t$, en débutant par la notion de propriété élémentaire sur laquelle la définition de PE_t s'appuie.

Définition 2. *Une propriété élémentaire p_e est une propriété dans $\mathcal{O}_t$ ou son inverse.*
p_e = *op* | *dp* | op^{-1}

Définition 3. *Une expression de propriété dans $\mathcal{O}_t$ (PE_t) est une propriété élémentaire (p_e) dans $\mathcal{O}_t$, ou une expression (f) utilisant une ou plusieurs expressions de propriété dans $\mathcal{O}_t$. Une PE_t peut inclure des contraintes sur le domaine de la propriété ($PE_t.Constr(d)$), sur son co-domaine ($PE_t.Constr(r)$) ou sur un élément mis en jeu dans une contrainte de co-domaine ($PE_t.Constr(f(Constr(r)))$).*
PE_t = p_e | *f*(PE_t) | *f*(PE_t, PE_t) | $PE_t.Constr(d)$ | $PE_t.Constr(r)$ | $PE_t.Constr(f(Constr(r)))$

Par récursivité, une PE_t peut être une fonction de n PE_t. Par exemple, PE_{t1} = $f(PE_{t2}$, $f(PE_{t3}$, $PE_{t4}))$ = $f(PE_{t2}$,PE_{t3},$PE_{t4})$. Nous donnons ci-dessous des explications sur les différents éléments intervenant dans la définition d'une PE_t.

La fonction f est spécifiée par le concepteur via une interface. Lorsque f est unaire, il doit indiquer s'il s'agit d'une opération d'agrégation ou de transformation et, pour chacune d'elles, il doit préciser sa nature. Pour une opération d'agrégation, il doit choisir entre le comptage d'un nombre d'éléments, une somme, le calcul d'un minimum, d'un maximum, d'une moyenne, etc. Pour une opération de transformation de valeurs, le choix s'effectue entre un calcul mathématique, le calcul de la longueur d'une chaîne de caractères, un changement de majuscules en minuscules ou l'inverse, le calcul de la valeur absolue d'un nombre, etc. Lorsque f est n-aire, le concepteur doit indiquer si la fonction correspond à une opération ensembliste (union ou différence) ou de transformation (calcul mathématique, concaténation,

etc.). En cas de calcul mathématique, une interface aide à saisir la formule de calcul.

$Constr$ représente n'importe quelle contrainte exprimable sous la forme d'un motif de graphe (Graph Pattern en anglais) correspondant à un ensemble de motifs de triplets (et éventuellement un filtre). $Constr(f(Constr(r)))$ est une contrainte portant sur l'agrégation (f) d'une variable mise en jeu dans une contrainte de co-domaine définie au préalable. Le concepteur doit la préciser comme indiqué ci-dessus. Par exemple, le co-domaine r doit être lié à des éléments x (contrainte de co-domaine) tels que la somme de ces x doit être supérieure à 10.

La valuation d'une PE_t se définit comme suit : val(PE_t) = val(p_e) | f(val(PE_t)) | f(val(PE_t), val(PE_t)) | val($PE_t.Constr(d)$) | val($PE_t.Constr(r)$) | val($PE_t.Constr(f(Constr(r)))$)

Les exemples de propriétés cibles donnés dans la suite de ce papier correspondent à des propriétés de DBpedia. Ces exemples sont inspirés de l'application (Alec et al., 2016a,b) dans laquelle nous avons utilisé le modèle d'acquisition et la génération des requêtes SPARQL. Cette application s'intéresse à la recommandation de produits dans divers domaines tels que les destinations de vacances, les films ou encore les musiques.

Exemple 4. Avg(janPrecipitationMm, janRainMm) est une PE_t dont la valeur est la moyenne de toutes les valeurs de l'union des propriétés janPrecipitationMm et janRainMm. Pour s'assurer que cette moyenne ne prendra pas en compte les valeurs négatives erronées fréquentes dans DBpedia, on peut introduire une contrainte sur le co-domaine qui permettra de rejeter les valeurs négatives des p_e, par exemple `FILTER(r >=0)`.

Exemple 5. $artist$ est une propriété de DBpedia dont le domaine est MusicalWork et le co-domaine est Agent. De ce fait, $artist^{-1}$ est une p_e de DBpedia (donc par extension une PE_t) qui, associée aux contraintes de co-domaine `r rdf:type dbo:Album` et `r dbo:genre ?gen`, correspond à des noms d'albums de chanteurs (ces albums ayant un genre). L'ajout de la contrainte `COUNT(?gen)>=3` portant sur la variable `?gen` mise en jeu dans les contraintes citées précédemment impose qu'il y ait au moins 3 genres différents.

En résumé, une PE_t peut être vue comme une propriété de $\mathcal{O}_t$ dont la définition peut combiner différentes fonctions f et/ou des contraintes. De plus, elle peut ne pas être explicitement représentée dans $\mathcal{O}_t$ mais être le résultat d'une construction. Le concepteur définit des correspondances entre les propriétés dans $\mathcal{O}_s$ et dans $\mathcal{O}_t$ en se basant sur le modèle que nous venons de présenter. Ce modèle tel que nous l'avons décrit, n'est toutefois pas suffisant car, d'une part il ne prend pas en compte les connaissances représentées dans $\mathcal{O}_s$, comme par exemple, la restriction de cardinalité de certaines PE_s. D'autre part, il ne tient pas compte du problème de l'absence possible de valeurs des propriétés dans $\mathcal{O}_t$. Nous proposons des solutions à ces deux types de problèmes dans les sections suivantes. Un modèle de spécification de chemins d'accès aux propriétés permet de pallier le problème des valeurs manquantes. Il fait l'objet de la section 4.2. Des mécanismes de traitement des valeurs de propriétés collectées, pouvant être multiples, sont présentés en section 4.3.

4.2 Modèle de spécification de chemins d'accès à des propriétés

La section précédente a présenté les correspondances entre propriétés. Nous nous intéressons maintenant à la façon d'accéder à la valeur (ou aux valeurs) des PE_t. L'accès à une PE_t, non explicitement représentée dans $\mathcal{O}_t$, correspond à l'accès à l'ensemble des éléments permettant de la définir. Nous présentons successivement les différents types d'accès possibles aux valeurs, puis la notion de chemin d'accès et enfin les algorithmes développés pour parcourir les chemins d'accès à une PE_t dans le LOD.

4.2.1 Types d'accès associés à une PE_t

Le modèle de correspondance requiert un accès aux valeurs de tous les éléments définissant une PE_t. Lorsque les éléments entrant dans la définition d'une PE_t sont des propriétés valuées de i_t, l'accès est direct et ne pose aucun problème. En revanche, lorsque ce n'est pas le cas, le LOD comportant de nombreuses propriétés sans valeurs, ceci peut poser problème pour l'application exploitant les données recherchées. Certaines applications ont besoin de disposer de valeurs, même si celles-ci ne sont qu'approximatives. Nous avons travaillé sous cette hypothèse. Ainsi, lorsqu'une propriété d'un individu i_t n'est pas valuée, nous proposons de rechercher la valeur de cette même propriété pour un (ou d') autre(s) individu(s) que i_t, en considérant ces valeurs comme des approximations de la valeur manquante. Par exemple, si on ne trouve pas la température moyenne en Alaska, on peut rechercher la température moyenne de Juneau, sa capitale. La valeur obtenue pour Juneau sera une approximation de la valeur cherchée pour Alaska. La propriété caractérise ici un autre individu que i_t. L'accès à cette propriété pour i_t n'est donc pas direct puisqu'il faut d'abord atteindre (par un certain chemin) un autre individu que i_t pour obtenir une valeur pour la propriété. Ceci nous a conduit à définir deux types d'accès à une PE_t, un accès direct et un accès composé. Nous donnons ci-dessous la définition des différents types d'accès à une PE_t, accompagnée d'un exemple. Cette définition est basée sur la notion de chemins qui fait l'objet de la section 4.2.2.

Définition 4. *L'accès à une PE_t est dit **composé** si on y accède par un chemin de propriétés d'ordre n (n≥1), **direct** sinon.*

Exemple 6. Soit PE_t = Avg(janPrecipitationMm) correspondant à la moyenne des valeurs de janPrecipitationMm. Son accès est direct pour Abu_Dhabi et composé pour Sri_Lanka (cf. figure 5).

4.2.2 Chemins d'accès à une PE_t

Un chemin de propriétés est défini par rapport à un chemin élémentaire de propriétés, qu'il nous faut donc d'abord préciser.

Définition 5. *Un chemin élémentaire ($chemin_n^{elem}$) de propriétés d'ordre n ($n \geq 1$) est une composition de propriétés objets ($p^1.p^2...p^n$). Pour un individu cible i_t donné, un chemin élémentaire de propriétés permet d'accéder à un ensemble d'individus cibles I_t dans $\mathcal{O}_t$.*

L'accès par un chemin élémentaire à un ensemble d'individus cibles s'explique par la présence éventuelle de propriétés multi-valuées.

Exemple 7. Dans la figure 5, country.capital est un chemin élémentaire de propriétés d'ordre 2 ($p^1.p^2$). Appliqué sur l'individu Cephalonia dans DBpedia, il permet d'accéder à un ensemble

I_t ne contenant que l'individu Athens. capital est un chemin élémentaire de propriétés d'ordre 1 (p^1). Appliqué sur l'individu Sri_Lanka dans DBpedia, il permet d'accéder à l'ensemble I_t = {Colombo ; Sri_Jayawardenepura_Kotte}, car la propriété capital est multivaluée pour Sri_Lanka.

Définition 6. *Un chemin de propriétés d'ordre n ($n \geq 1$) est un chemin élémentaire ou un ensemble de chemins élémentaires, éventuellement muni d'une contrainte sur I_t et/ou sur i_t.*
$chemin_n = chemin_n^{elem} \mid Ens(chemin_n^{elem}) \mid chemin_n.Constr(i_t) \mid chemin_n.Constr(I_t)$

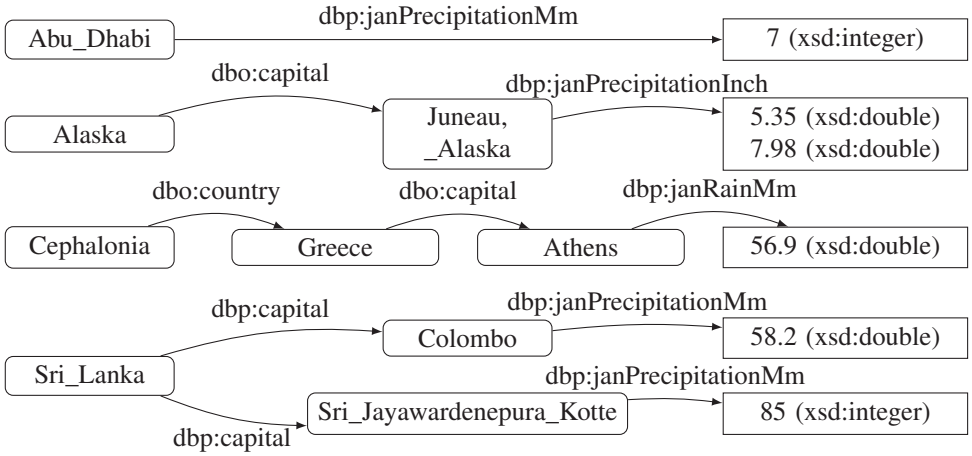

FIG. 5 – *Exemples de chemins d'accès à des précipitations dans DBpedia*

Comme vu dans la définition 6, un chemin peut être un ensemble de chemins élémentaires. Cela sera souvent le cas lorsque le concepteur veut faire appel dans le chemin à une propriété redondante. Par exemple, $part$, $isPartOf^{-1}$ et p sont trois façons d'exprimer la sous-partie dans DBpedia.

Exemple 8. Le tableau 1 représente un chemin de propriétés d'ordre 2 composé d'un ensemble de cinq chemins élémentaires (chaque ligne correspond à un chemin élémentaire). Ce chemin sera utilisé pour accéder à une PE_t permettant l'obtention de données météorologiques sur Dubai. Les valeurs météorologiques qui peuvent être ainsi obtenues sont celles de villes, auxquelles on accède par les propriétés listées en colonne 2. Ces villes sont des sous-parties géographiques de régions obtenues par les propriétés de la colonne 1, correspondant à des régions dont Dubai est soit la plus grande ville soit à l'ouest ou au nord. Les données météorologiques ainsi accessibles sont supposées être de bonnes approximations des données météorologiques de Dubai.

D'après la définition 6, des contraintes sur i_t et/ou I_t sont possibles. En somme, une contrainte de chemin peut porter sur l'individu initial, ou sur l'ensemble d'individus après l'application du chemin sur l'individu initial.

Exemple 9. Supposons qu'on veuille connaître le pays d'origine d'un film. La PE_t correspondra à l'union des propriétés correspondantes, par exemple $country$ ou encore $nationality$

Propriétés régions (p^1)	Propriétés sous-parties (p^2)
$dbp{:}west^{-1}$	$dbo{:}part$
$dbp{:}north^{-1}$	$dbo{:}part$
$dbp{:}north^{-1}$	$dbo{:}isPartOf^{-1}$
$dbp{:}largestCity^{-1}$	$dbo{:}isPartOf^{-1}$
$dbo{:}largestCity^{-1}$	$dbo{:}isPartOf^{-1}$

TAB. 1 – *Chemin appliqué sur Dubai pour une PE_t visant à obtenir des données météorologiques*

dans DBpedia. En cas de valeur manquante, nous supposerons que la nationalité du réalisateur est une bonne approximation de cette propriété. Ainsi, Windy_City_Heat est un film dont on ignore le pays d'origine dans DBpedia. Mais, sachant que Windy_City_Heat.director.nationality = Americans, on pourra en déduire que Windy_City_Heat est un film américain.

Contrainte sur i_t : Maintenant, si le concepteur estime qu'avec l'émergence de la mondialisation, beaucoup de réalisateurs réalisent de nos jours des films ailleurs que dans leur pays d'origine, il peut ajouter une contrainte exprimant que cette approximation n'est valable que si le film n'est pas récent (par exemple, avant 2000). Cette contrainte porte donc sur le film (i_t), et peut être exprimée par le motif de graphe suivant :

```
i_t dbp:released ?year. FILTER (?year <= 2000).
```

Contrainte sur I_t : D'un autre côté, le concepteur peut considérer que les approximations trouvées seront assez bonnes en général sauf pour les films de certains réalisateurs connus comme James Cameron (canadien) ou Ridley Scott (britannique) qui ont fait beaucoup de films américains. La contrainte porte donc ici sur le réalisateur (I_t) et peut être exprimée par le motif de graphe suivant :

```
I_t rdfs:label ?name. FILTER (LANGMATCHES(LANG(?name), "en") && !regex(?name,
"Ridley Scott") && !regex(?name, "James Cameron"))
```

4.2.3 Algorithmes de parcours des chemins d'accès

Le concepteur définit tous les chemins d'accès (un maximum) pour chaque PE_t impliquée dans des correspondances avec des propriétés de $\mathcal{O}_s$, en les ordonnant par pertinence.

L'algorithme 1 implémente cette idée. Le principe est de parcourir les chemins menant à une PE_t par ordre de priorité jusqu'à ce qu'on obtienne une valuation de PE_t. L'algorithme appelle la fonction userAlgo spécifiant les chemins par ordre de priorité. Un exemple d'implémentation de la partie switch est présentée. Dans l'expression des chemins de ce switch, `getVal()` est une fonction renvoyant les valeurs des propriétés considérées. Par exemple, l'expression du cas 2 renvoie les valeurs de la PE_t considérée pour l'ensemble des individus correspondant à la capitale de i_t. L'expression $subparts$ des cas 4 et 8 correspond aux propriétés DBpedia isPartOf^{-1}, part et p. En conséquence, le chemin considéré dans le cas 4 correspond à un ensemble de trois chemins élémentaires. Enfin, notons que $?prop$, dans ces chemins, signifie n'importe quelle propriété DBpedia.

Un même ensemble de chemins peut être utilisé pour plusieurs PE_t. Le paramètre `maxNbAttempts`, fixé par le concepteur, indique le nombre maximal de chemins de l'ensemble à explorer pour une PE_t particulière. Par exemple, la partie switch présentée ici est

réutilisable pour la recherche de valeurs de la latitude, de la longitude, des températures, des précipitations par mois, etc. mais pas forcément avec la même valeur de `maxNbAttempts`.

Algorithme 1 : Algorithme pour extraire des données de $\mathcal{O}_t$

Entrée : i_t l'individu cible
Sortie : info à stocker dans $\mathcal{O}_s$
Param : maxNbAttempts le nombre maximum de chemins à explorer

```
1 info ← null ;
2 attempt ← 1 ;
3 while info = null and attempt ≤ maxNbAttempts do
4 |   info ← userAlgo(i_t, attempt) ;
5 |   attempt ← attempt + 1 ;
6 end
7 return info ;
```

Fonction userAlgo

Entrée : i_t l'individu cible,
attempt le numéro de tentative
Sortie : info à stocker dans $\mathcal{O}_s$

```
1 info ← null ;
2 switch attempt do
3 |   case i          /* i=1...maxNbAttempts */        /* ordre n */
4 |   |   info ← i_t.p^1.p^2...p^n.PE_t.getVal() ; break
5 endsw
6 return info ;
```

4.3 Mécanismes de traitement des valeurs de propriétés collectées

Des traitements sur les valeurs des propriétés auxquelles les mécanismes précédents ont permis d'accéder peuvent être nécessaires. Dans une première partie, nous décrivons les choix que le concepteur peut faire, puis nous présentons de façon intuitive et sur un exemple les différents mécanismes d'agrégation utiles. Dans un dernier temps, nous définissons les mécanismes d'agrégation proposés.

4.3.1 Choix des mécanismes de traitement

Le concepteur doit spécifier la façon de traiter les données collectées via l'application des modèles de correspondance et d'accès précédemment décrits. Toutefois, s'il n'y a aucune restriction de cardinalité sur les valeurs d'une PE_s, aucun traitement spécifique n'est a priori nécessaire.

Partie switch : Exemple avec des données sur des destinations

```
1 case 1 info ← i_t.PE_t.getVal() ; break;                    /* accès direct */
2 case 2 info ← i_t.capital.PE_t.getVal() ; break;            /* ordre 1 */
3 case 3 info ← i_t.largestCity.PE_t.getVal() ; break;        /* ordre 1 */
4 case 4 info ← i_t.subparts.PE_t.getVal() ; break;           /* ordre 1 */
5 case 5 info ← i_t.country^-1.PE_t.getVal() ; break;         /* ordre 1 */
6 case 6 info ← i_t.?prop.PE_t.getVal() ∪ i_t.?prop^-.PE_t.getVal() ; break;
                                                              /* ordre 1 */
7 case 7 info ← i_t.country.capital.PE_t.getVal() ; break;    /* ordre 2 */
8 case 8 info ← i_t.?prop^-1.subparts.PE_t.getVal() ; break;  /* ordre 2 */
```

Dans le cas contraire, deux cas sont possibles. Tout d'abord, le concepteur peut souhaiter **réduire le nombre de valeurs retournées**, en ne gardant que les n premières valeurs différentes (n étant défini par le concepteur). Ceci peut être lié à la présence d'une restriction de cardinalité ou bien à la fonctionnalité de la propriété (dans ce dernier cas, $n = 1$).

Le concepteur peut aussi opter pour un **mécanisme d'agrégation**. Dans ce cas, il choisit la façon d'agréger les données collectées. Les mécanismes d'agrégation sont nécessaires pour gérer, en particulier, le cas des propriétés sources fonctionnelles. Ils sont toutefois également applicables dans d'autres situations, même si la propriété source n'est pas fonctionnelle, si le concepteur le décide. Nous ne considérons ici que le cas où l'accès est composé. En effet, lorsque l'accès est direct, l'application d'une fonction d'agrégation (cf. f dans Définition 3) à l'ensemble des valeurs de la PE_t suffit. Divers mécanismes d'agrégation ont été conçus. Nous démontrons leur intérêt avant de les définir formellement.

4.3.2 Intérêt des mécanismes d'agrégation

Quand l'accès est composé, le modèle défini section 4.2 nous conduit à prendre en compte un ensemble d'individus cibles I_t. Même si la PE_t est munie d'une fonction d'agrégation, une valeur sera obtenue pour chaque individu de I_t. Plusieurs valeurs seront donc retournées. Des mécanismes d'agrégation complémentaires sont nécessaires. Nous illustrons ces mécanismes sur des exemples dans ce qui suit.

Supposons qu'il soit possible d'accéder aux chansons d'un artiste via ses albums. Dans la figure 6, chaque chanson est caractérisée par sa durée. La durée est unique car la PE_t exprimant la durée intègre le calcul de la moyenne des durées en cas de valeurs multiples. Une même chanson peut être dans plusieurs albums différents (comme les chansons 3 et 4).

Sur cet exemple, si on recherche la durée moyenne des chansons figurant dans un album, il faut s'intéresser à toutes les chansons des albums, une chanson pouvant être considérée plusieurs fois si elle est présente sur plusieurs albums. On obtient une durée moyenne de 3min40. Ce type d'agrégation sera qualifié de simple. On agrège toutes les valeurs collectées, quel que soit l'objet caractérisé et quel que soit le chemin d'accès emprunté pour les obtenir.

En revanche, la recherche de la durée moyenne d'une chanson de l'artiste nécessite le calcul de la moyenne de la durée des 7 chansons, chaque chanson n'étant considérée qu'une seule fois. On obtient une durée de 3min30. Pour cela, l'agrégation se fait sur des valeurs

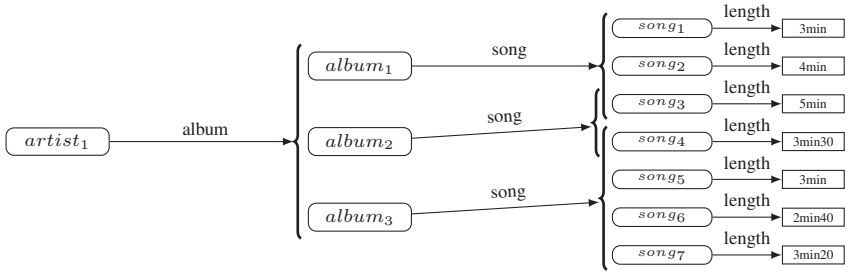

FIG. 6 – *Exemple d'accès composé pour obtenir les chansons d'un artiste*

associées aux chansons, indépendamment des albums. Les chansons représentent les individus cibles. Le fait de ne pas considérer les albums revient à ne pas considérer les chemins d'accès aux chansons. Ce type d'agrégation sera appelé agrégation sur les I_t (les chansons).

Enfin, si on recherche la durée moyenne d'un album, il faut faire la somme des durées des chansons pour chaque album. L'$album_1$ dure 12min, l'$album_2$ dure 8min30 et l'$album_3$ dure 12min30. La durée moyenne d'un album sera donc de 11 min. Ces deux agrégations successives, l'une sur $album$ et la seconde sur $artist$, nécessitent la mise en œuvre d'un mécanisme que nous appellerons agrégation propagée.

4.3.3 Spécification des mécanismes d'agrégation

Nous présentons les trois mécanismes d'agrégation cités ci-dessus et les illustrons sur l'exemple du cas 8 pour l'individu Dubai. Les chemins correspondant sont ceux présentés précédemment dans le tableau 1. Le tableau 2 indique les individus auxquels les différentes propriétés du tableau 1 permettent d'accéder. Notons que certaines lignes sont redondantes. Il s'agit de chemins élémentaires différents conduisant aux mêmes individus. Rappelons par ailleurs que les mécanismes décrits ici sont mis en œuvre après l'application des fonctions d'agrégation liées à la PE_t (cf. fonction f dans la définition 3). Ainsi, tout individu de I_t n'a qu'une valeur de PE_t. Dans cette section, nous adopterons les notations suivantes :

- Les expressions explicitant les mécanismes d'agrégation seront notées en respectant la syntaxe de l'algèbre relationnelle. Entre autres, nous utiliserons l'expression $G_1, G_2, \ldots, G_m\ \mathcal{F}_{f_1(A_1),f_2(A_2),\ldots,f_k(A_k)}\ (R)$, où chaque A_j est un attribut de la relation R sur lequel une fonction d'agrégation f_j s'applique. Les attributs précédant $\mathcal{F}$ sont des attributs de groupement (facultatifs). Pour plus de commodité, nous permettrons le renommage ($f_j(A_j)\ as\ renommage$) dans l'opération d'agrégation.
- Pour un chemin $p^1.p^2...p^j$, on nommera I_t^j les individus atteints via ce chemin à partir de i_t. Selon cette notation, I_t^n correspond à l'ensemble des individus dont les propriétés sont considérées comme de bonnes approximations des propriétés du i_t que l'on cherche à valuer (I_t^n correspond ainsi à I_t). Dans notre exemple, $n = 2$.
- L'ensemble des éléments mis en jeu dans un accès composé (individus et valeurs de la propriété recherchée) sera noté $R(I_t^1, I_t^2, ..., I_t^n, PE_t)$, PE_t étant une propriété de I_t^n. Dans notre exemple, la relation R correspond au tableau 2.

Régions liées à Dubai (I_t^1)	Villes (I_t^2)	Moy. températures août (PE_t)
Emirate_of_Sharjah	Sharjah	34.7
Emirate_of_Abu_Dhabi	Abu_Dhabi	36.2
United_Arab_Emirates	Abu_Dhabi	36.2
Emirate_of_Abu_Dhabi	Abu_Dhabi	36.2
United_Arab_Emirates	Abu_Dhabi	36.2

TAB. 2 – *Chemin du cas 8 appliqué sur Dubai pour une PE_t calculant la moyenne de la température en Août*

Régions liées à Dubai (I_t^1)	Villes (I_t^2)	PE_t
Emirate_of_Sharjah	Sharjah	34.7
Emirate_of_Abu_Dhabi	Abu_Dhabi	36.2
United_Arab_Emirates	Abu_Dhabi	36.2

TAB. 3 – *Tableau 2 sans doublons*

Agrégation simple L'agrégation simple porte sur l'ensemble des valeurs de PE_t trouvées en considérant tous les chemins pour y accéder (cf. tableau 1) mais uniquement les tuples de R différents (cf. tableau 3). Rappelons que l'algèbre relationnelle ne considère pas les doublons. En conséquence, la relation R considérée dans la formule ci-dessous intègre déjà cette contrainte. Dans notre exemple, si on prend la moyenne comme opération d'agrégation, la valuation finale considérée sera la moyenne des 3 valeurs, soit 35.7.

$$\mathcal{F}_{AGR(PE_t)\ as\ res}\ (R)$$

Agrégation sur I_t L'agrégation sur I_t porte sur les valeurs associées aux individus de l'ensemble I_t^n (sans doublons), cf. tableau 4. Les chemins qui ont permis d'accéder à ces individus sont ignorés. Dans notre exemple, l'agrégation porte sur (I_t^2, PE_t). Si on prend la moyenne comme opération d'agrégation, la valuation finale considérée sera la moyenne des 2 valeurs, i.e. 35.45.

$$\mathcal{F}_{AGR(PE_t)\ as\ res}\ (\Pi_{I_t^n, PE_t}(R))$$

Villes (I_t^2)	PE_t
Sharjah	34.7
Abu_Dhabi	36.2

TAB. 4 – *Représentation de $\Pi_{I_t^n, PE_t}(R)$ sur notre exemple*

Agrégation propagée L'agrégation propagée consiste à appliquer une suite d'opérations d'agrégation (éventuellement différentes) en effectuant différents groupements, d'abord par $I_t^1, I_t 2..., I_t^{n-1}$ puis par $I_t^1, I_t^2, ..., I_t^{n-2}$, et ainsi de suite jusqu'à I_t^1. Ensuite, on agrège l'ensemble de valeurs obtenues. Ce mécanisme donne de l'importance aux différents individus

composant les chemins. Dans l'exemple, son application consiste d'abord à faire un regroupement par région (I_t^1) comme le montre le tableau 5. On agrège ainsi les températures des différentes sous-parties géographiques (ici des villes) de chaque région. Dans un second temps, on agrège les températures obtenues pour les différentes régions en faisant la moyenne des trois valeurs trouvées, soit 35.7. Remarquons qu'on obtient ici la même valeur qu'avec une agrégation simple car, dans cet exemple, il n'y a qu'un seul I_t^2 associé à chaque I_t^1.

$$\mathcal{F}_{AGR_1(val_1)\ as\ res}\ (I_t^1 \mathcal{F}_{AGR_2(val_2)\ as\ val_1}\ (I_t^1, I_t^2\ \mathcal{F}_{AGR_3(val_3)\ as\ val_2}\ (...($$
$$I_t^1, I_t^2, ..., I_t^{n-1}\ \mathcal{F}_{AGR_n(PE_t)\ as\ val_{n-1}}\ (R)$$
$$)\qquad\qquad)\qquad\qquad)...)$$

Régions liées à Dubai ($I_t^1 = I_t^{n-1}$)	PE_t ($val_1 = val_{n-1}$)
Emirate_of_Sharjah	34.7
Emirate_of_Abu_Dhabi	36.2
United_Arab_Emirate	36.2

TAB. 5 – *Représentation de* $I_t^1, I_t^2, ..., I_t^{n-1}\ \mathcal{F}_{AGR_n(val)\ as\ val_{n-1}}\ (R)$ *sur notre exemple*

5 Génération automatique des requêtes SPARQL basée sur le modèle d'acquisition

```
CONSTRUCT { i_s  p_s  ?val_0 }
WHERE{
    i_t  traduction(traitement(chemin_n.PE_t))  ?val_0
}
```

Le modèle décrit dans la section précédente sert de support pour générer automatiquement des requêtes SPARQL de type `CONSTRUCT` permettant d'ajouter des assertions de propriétés dans $\mathcal{O}_s$. On notera p_s la propriété de l'individu i_s considérée dans PE_s dont on cherche la valeur ($?val_0$). Le triplet à ajouter dans $\mathcal{O}_s$ sera donc de la forme $\{i_s\ p_s\ ?val_0\}$. Une clause `WHERE` dans la requête de type `CONSTRUCT` indiquera ensuite comment obtenir la (ou les) valeur(s) de $?val_0$. Cette clause contiendra un motif de graphe à apparier sur le jeu de données cible, de la forme $\{i_t\ \ traduction(traitement(chemin_n.PE_t))\ \ ?val_0\}$ et tel que :

- $(i_s, i_t) \in Ind_{s,t}$,

- PE_t est l'expression de propriété cible mise en correspondance avec PE_s via le modèle de correspondance défini dans la section 4.1,

- $chemin_n$ est un chemin de propriétés d'ordre n, éventuellement vide, définissant un éventuel accès composé à PE_t,

- `traitement` correspondant aux traitements éventuellement appliqués in fine (cf. section 4.3),

- et enfin `traduction` traduit, si besoin, les valeurs obtenues pour que celles-ci soient compatibles avec O_s (cf. section 5.2.4).

Nous décrivons dans ce qui suit comment générer le contenu du bloc `WHERE` de cette requête pour une PE_s donnée, en adoptant une approche à base de patrons. Tous les patrons proposés génèrent des parties d'expressions SPARQL devant être introduites dans le bloc `WHERE` de la requête. Dans une première partie, nous présentons le processus général de génération automatique du bloc `WHERE` de la requête de type `CONSTRUCT`, puis, dans une seconde partie, les patrons construits.

5.1 Processus de génération de requêtes SPARQL 1.1

Nous décrivons ci-dessous le processus de génération du bloc `WHERE` de la requête `CONSTRUCT` qui ajoutera les assertions de propriétés souhaitées dans $\mathcal{O}_s$. Le bloc `WHERE` est généré à partir du modèle d'acquisition présenté en section 4 et à partir de patrons.

Nous proposons un ensemble de patrons classés en quatre catégories, ces dernières portant sur des éléments pouvant être considérés indépendamment les uns des autres. En effet, des expressions différentes en SPARQL 1.1 permettent de tenir compte du format de la PE_t considérée, de l'existence d'un chemin pour accéder aux valeurs de cette propriété, des traitements finaux à exécuter ou de la traduction des valeurs acquises. Nous proposons donc 4 catégories de patrons, une pour chaque type de problème : Format (de la PE_t), Chemin, Traitement, Traduction. Un patron correspond à un sous-problème d'une catégorie.

La construction du bloc `WHERE` de la requête à générer s'effectue en deux phases qui peuvent être réalisées en parallèle, chaque phase générant une partie de la requête. Une phase dite externe génère la partie de requête, que nous appellerons $partie\ externe$, liée à la traduction et aux traitements des valeurs. Une autre phase, dite interne, génère la partie de requête, que nous appellerons $partie\ interne$, liée au format de PE_t et au chemin d'accès. La partie interne est ensuite imbriquée dans la partie externe afin d'obtenir la requête SPARQL finale dont l'exécution délivrera les valeurs de propriétés recherchées.
La phase externe a deux étapes. La première étape génère la partie SPARQL portant sur la traduction (par application des patrons de la catégorie Traduction). La deuxième étape y insère la partie portant sur le Traitement (par application des patrons de la catégorie Traitement).
La phase interne a 3 étapes. La première étape génère la partie SPARQL portant sur le format de la PE_t (par application des patrons de la catégorie Format). La deuxième étape effectue un pré-traitement avant l'insertion d'un éventuel chemin (recherche de l'emplacement dans la requête pour sa déclaration et changement de noms de variables). L'étape 3 insère le chemin (par application des patrons de la catégorie Chemin).
L'ordre d'application des patrons au sein d'une catégorie sera décrit dans la section 5.2.

Au départ, le bloc `WHERE` contient uniquement l'expression "$i_t\ \ traduction(traitement(chemin_n.PE_t))\ \ ?val_0$". Dans les patrons de traduction, elle devient "$i_t\ \ traitement(chemin_n.PE_t))\ \ ?val$". Puisque la phase de traduction a été prise en compte, le terme $traduction$ n'est plus mentionné. De la même façon, dans les patrons dits de traitement, elle devient "$i_t\ \ chemin_n.PE_t\ \ ?val$", le terme $traitement$ n'ayant plus besoin d'être mentionné. L'expression, "$i_t\ \ chemin_n.PE_t\ \ ?val$", est traitée par la phase interne. Les patrons dits de Format liés à PE_t permettent de trouver le motif de graphe correspondant à "$i_t\ \ PE_t\ \ ?val$" pour la requête en cours de construction. Les patrons liés aux

chemins interviennent ensuite pour intégrer la prise en compte éventuelle d'un chemin afin d'obtenir un motif de graphe de la forme "$i_t \ \ chemin_n.PE_t \ \ ?val$".

Notons que lors de l'application de chaque patron, les noms de variables sont modifiés si nécessaire. La variable dont le contenu est renvoyé par la requête reste toutefois $?val_0$.

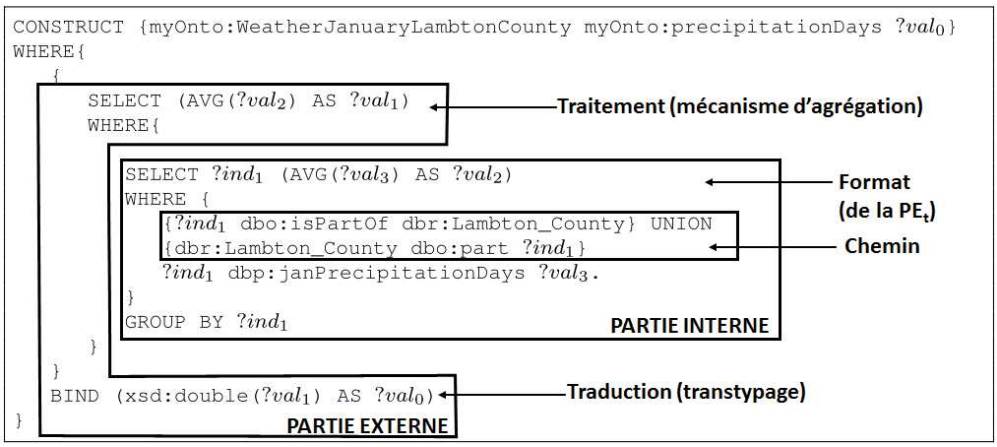

FIG. 7 – *Les parties interne et externe d'une requête*

La figure 7 reprend l'exemple 2 de la page 4 en indiquant les deux parties, interne et externe, de la requête, la partie interne étant imbriquée dans la partie externe. Les parties de la requête générées par les différentes catégories de patrons sont également précisées.

5.2 Présentation des différents patrons

Nous présentons ici les patrons construits par catégorie. Dans ces patrons, la partie **Cond** représente la condition d'application du patron, i.e. le contexte, et la partie **Action**, la solution mise en œuvre.

5.2.1 Patrons liés au format d'une PE_t

7 patrons permettent de tenir compte de l'ensemble des formats d'une expression de propriété PE_t (cf. Définition 3). Ces patrons s'appliquent lorsqu'on cherche à générer la partie de requête correspondant à "$i_t \ \ PE_t \ \ ?val_j$" pour tenir compte de la spécificité de l'expression de propriété dont on cherche les valeurs. Le processus d'application des patrons de cette catégorie est guidé par la définition de la PE_t. Ainsi, si une PE_t s'exprime en fonction d'une ou plusieurs autres PE_t, le patron associé est exprimé en fonction d'une ou plusieurs expressions "$i_t \ \ PE_t \ \ ?val$". L'imbrication des patrons s'effectuera de la même manière que l'imbrication de la PE_t, en commençant par la partie la plus externe. Par exemple si $PE_t = f(PE_t)$, on appliquera d'abord le patron associé à la fonction f. Puis, l'expression "$i_t \ \ PE_t \ \ ?val$" qu'il contient sera récursivement remplacée par application du

patron correspondant à la PE_t sur laquelle porte f. Dans le cas d'une PE_t contrainte, on appliquera d'abord le patron de contrainte.

Patron$_{elem}$
Cond : PE_t = p_e
Action : Remplacer p_e par le nom de la propriété élémentaire.
À noter que si p_e vaut op^{-1}, alors "$i_t \;\; p_e \;\; ?val_j$" sera remplacé par "$?val_j \;\; op \;\; i_t$".

```
i_t  p_e  ?val_j .
```

Patron$_{transfo_1}$
Cond : PE_t = $f(PE_t)$ avec f une fonction de transformation.
Action : Remplacer f par l'expression mathématique souhaitée ou la fonction SPARQL 1.1 correspondant à la fonction de transformation à appliquer (STRLEN, LCASE, ABS, etc).

```
{i_t  PE_t1  ?val_j+1}
BIND (f(?val_j+1) AS ?val_j)
```

Patron$_{agr}$
Cond : PE_t = $f(PE_t)$ avec f une fonction d'agrégation.
Action : Remplacer f par l'opérateur SPARQL 1.1 correspondant à la fonction d'agrégation à appliquer : COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT et SAMPLE.

```
SELECT (f(?val_j+1) AS ?val_j)
WHERE {
   i_t  PE_t  ?val_j+1
}
```

Patron$_{ens}$
Cond : PE_t = $f(PE_t, PE_t)$ avec f une opération ensembliste.
Action : Remplacer f par l'opérateur SPARQL 1.1 UNION ou MINUS selon le cas.

```
{i_t  PE_t1  ?val_j}
f
{i_t  PE_t2  ?val_j}
```

Patron$_{transfo_2}$
Cond : PE_t = $f(PE_t, PE_t)$ avec f une opération de transformation.
Action : Remplacer f par l'expression mathématique ou la fonction SPARQL 1.1 souhaitée (CONCAT, CONTAINS, etc).

```
{i_t  PE_t1  ?val_j+1_a}
{i_t  PE_t2  ?val_j+1_b}
BIND (f(?val_j+1_a,?val_j+1_b) AS ?val_j)
```

Patron$_{constr}$
Cond : PE_t = $PE_t.Constr(d)$ | $PE_t.Constr(r)$.
La contrainte $Constr$ peut porter sur le domaine de la PE_t, c'est-à-dire i_t, ou bien sur le co-domaine r, c'est-à-dire $?val_j$.
Action : Remplacer $Constr$ par son expression en motif de graphe. Si c'est une contrainte de co-domaine, remplacer r par le nom de la variable associée, c'est-à-dire $?val_j$.

```
{i_t  PE_t  ?val_j}
Constr
```

Patron$_{having}$
Cond : PE_t=$PE_t.Constr(f(Constr(r)))$
Action : Instancier l'expression du `HAVING` en fonction de la contrainte formulée. Entre autres, remplacer f par l'opérateur SPARQL 1.1 correspondant : COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT, SAMPLE. Si la variable de contrainte

```
SELECT ?val_j
WHERE {
   i_t  PE_t  ?val_j
}
GROUP BY ?val_j
HAVING (f(?contrainte) operateur valeur)
```

est dans une requête imbriquée, on ajoute l'appel à cette variable dans chaque `SELECT` imbriquant la variable ainsi qu'un groupement sur cette variable.

5.2.2 Patrons liés aux chemins d'accès

On distingue trois patrons liés aux chemins d'accès selon qu'il s'agit de déclarer un chemin élémentaire, un ensemble de chemins élémentaires ou des contraintes associées. Ces patrons s'appliquent lorsqu'on cherche à générer la partie de requête correspondant à "i_t $chemin_n$ $?ind_n$".

L'application de ces patrons fait suite à une étape de pré-traitement qui consiste à sélectionner l'emplacement où cette expression du chemin doit être déclarée et à effectuer les changements de noms de variables appropriés. En effet, on pourrait remplacer, dans la requête exprimant le format de la PE_t, toute référence à i_t par le chemin considéré. Cependant, ce chemin serait répété de nombreuses fois dans la requête si la PE_t fait appel à de nombreuses propriétés élémentaires. Grâce à l'étape de pré-traitement, le chemin ne sera cité qu'une seule fois.

L'emplacement du chemin dépend de la forme de la partie interne de la requête en cours de construction.

- Si celle-ci débute par une clause `SELECT`, on insère l'expression "i_t $chemin_n$ $?ind_n$" au sein de la clause `WHERE` liée à la clause `SELECT`. Ce sera le cas lorsque le format de la PE_t est $f(PE_t)$ ou $PE_t.Constr(f(Constr(r))$, avec f une fonction d'agrégation, et que la PE_t est éventuellement munie de contraintes de domaine et/ou de co-domaine. De plus, on ajoute dans cette clause `SELECT` les variables correspondant aux individus mis en jeu dans le chemin, c'est-à-dire [$?ind_1$, $?ind_2$, ..., $?ind_{n-1}$,] $?ind_n$ et un groupement sur ces variables. Les variables mises entre crochets sont à ignorer s'il n'y a pas de mécanisme de traitement, ou s'il s'agit d'une restriction de nombre de valeurs, ou d'une agrégation sur les I_t.
- Si la partie interne de la requête en cours de construction ne débute pas par une clause `SELECT`, on insère "i_t $chemin_n$ $?ind_n$" au début de celle-ci.

Dans les deux cas, s'il existe des clauses `SELECT` imbriquées, on ajoutera $?ind_n$ dans ces clauses et on groupera sur cette variable.

Le second pré-traitement consiste à remplacer les i_t de la partie interne en cours de construction par la variable $?ind_n$ correspondant aux I_t^n.

Les patrons de Chemin s'imbriquent en suivant la définition de $chemin_n$. Pour instancier les patrons, on partira du patron de contrainte $Patron_{chem_{constr}}$ qui s'exprime en fonction de $chemin_n$ et qui pourra imbriquer tout type de patron de Chemin. Le patron d'ensemble $Patron_{chem_{ens}}$ qui s'exprime en fonction de chemin élémentaire $p^1.p^2...p^n$ pourra imbriquer des patrons de chemins élémentaires $Patron_{chem_{elem}}$.

$\boldsymbol{Patron_{chem_{elem}}}$
Cond : $chemin_n = p^1.p^2...p^n$

$i_t\ \ p^1\ \ ?ind_1\ .\ \ ?ind_1\ \ p^2\ \ ?ind_2\ .\ \ \ldots\ \ ?ind_{n-1}\ \ p^n\ \ ?ind_n\ .$

Action : Remplacer les propriétés p^j par leur nom. À noter que si p^j vaut op^{-1}, alors "$x\ \ p^j\ \ ?ind_j$" sera remplacé par "$?ind_j\ \ op\ \ x$".

$\boldsymbol{Patron_{chem_{ens}}}$
Cond : $chemin_n = Ens(p^1.p^2...p^n, p^1.p^2...p^n, ..., p^1.p^2...p^n)$

$\{i_t\ \ p^1.p^2...p^n\ \ ?ind_n\}$ UNION $\{i_t\ \ p^1.p^2...p^n\ \ ?ind_n\}$ UNION ... UNION $\{i_t\ \ p^1.p^2...p^n\ \ ?ind_n\}$

Action : Adapter le nombre d'opérateurs UNION au nombre de chemins élémentaires de l'ensemble.

$\boldsymbol{Patron_{chem_{constr}}}$
Cond : $chemin_n = chemin_n.Constr(i_t) \mid chemin_n.Constr(I_t)$

$i_t\ \ chemin_n\ \ ?ind_n$
$Constr$

Action : Remplacer $Constr$ par son expression. S'il s'agit d'une contrainte sur I_t, on remplacera I_t par le nom de la variable qui lui est associée ($?ind_n$). Suivant la forme du chemin, "$i_t\ \ chemin_n\ \ ?ind_n$" sera remplacé par l'application du patron de Chemin correspondant.

Exemple 10. Soit PE_t = $MOYENNE$(janRainMm.$FILTER(r\ >=\ 0)$), la moyenne des valeurs positives de janRainMm. Dans la première étape de la phase interne, les patrons $Patron_{agr}$, $Patron_{Const}$ et $Patron_{elem}$ sont appliqués (dans cet ordre). On obtient :

```
1  SELECT (AVG(?val1) AS ?val0)
2  WHERE {
3
4     it dbp:janRainMm ?val1 .
5     FILTER(?val1 >=0)
6  }
```

Dans un second temps, on positionne le chemin (d'ordre 2 dans cet exemple) au sein de la clause WHERE (ligne 3). De plus, on ajoute les variables représentant les individus mis en jeu dans le chemin (lignes 1 et 7). Enfin, on fait le changement de variable nécessaire (ligne 4).

```
1  SELECT [?ind1] ?ind2 (AVG(?val1) AS ?val0)
2  WHERE {
3     it chemin2 ?ind2 .
4     ?ind2 dbp:janRainMm ?val1 .
5     FILTER(?val1 >=0)
6  }
7  GROUP BY [?ind1] ?ind2
```

Dans un troisième temps, on applique les patrons liés aux chemins. Dans le cas présent, supposons que le chemin à considérer soit country.capital (exemple du cas 7). Le patron $Patron_{chem_{elem}}$ s'applique, instanciant le chemin (ligne 3 modifiée).

```
1 SELECT [?ind1] ?ind2 (AVG(?val1) AS ?val0)
2 WHERE {
3    it dbo:country ?ind1. ?ind1 dbo:capital ?ind2.
4    ?ind2 dbp:janRainMm ?val1.
5    FILTER(?val1 >=0)
6 }
7 GROUP BY [?ind1] ?ind2
```

Exemple 11. Reprenons maintenant l'exemple 9 donné page 12 où $PE_t = UNION(country, nationality)$. Nous appliquons tout d'abord les patrons $Patron_{ens}$ et $Patron_{elem}$ de la catégorie Format, ce qui permet d'obtenir :

```
1
2
3
4 { it dbp:country ?val0 } UNION {it dbp:nationality ?val0 }
```

L'étape de pré-traitement positionne le chemin (d'ordre 1) au début (ligne 1) et fait le changement de variable nécessaire (ligne 4 modifiée).

```
1 it chemin1 ?ind1.
2
3
4 { ?ind1 dbp:country ?val0 } UNION { ?ind1 dbp:nationality ?val0 }
```

Dans la troisième étape, nous considérons ensuite le chemin $director.Constr(i_t$ `dbp:released ?year. ?`ind_1 `rdfs:label ?name. FILTER(?year<=2000 && !regex(?name, "Ridley Scott") && !regex(?name, "James Cameron"))`). Ce chemin sera construit via les patrons $Patron_{chem_{constr}}$ puis $Patron_{chem_{elem}}$ (chemin élémentaire en ligne 1 et contrainte lignes 2-3). Notons qu'ici, par simplification, nous avons considéré les deux contraintes sur i_t et I_t en une seule et même contrainte.

```
1 it dbo:director ?ind1.
2 it dbp:released ?year. ?ind1 rdfs:label ?name. FILTER(?year<=2000 &&
3 !regex(?name, "Ridley Scott") && !regex(?name, "James Cameron"))
4 { ?ind1 dbp:country ?val0 } UNION { ?ind1 dbp:nationality ?val0 }
```

5.2.3 Patrons liés aux traitements

On distingue 3 patrons liés au traitement des valeurs finales collectées, un portant sur la réduction du nombre de valeurs retournées et les deux suivants concernant les mécanismes d'agrégation décrits section 4.3.3. Les patrons liés aux traitements d'agrégation contiennent la contrainte `FinalConstr(`val`)`. Cette expression exprime une éventuelle contrainte donnée par le concepteur sur la valeur finale après application du mécanisme d'agrégation.

$\boldsymbol{Patron_{trait_{reduc}}}$
Cond : `Traitement =`
Réduction du nombre de valeurs.

```
SELECT DISTINCT ?valj
WHERE {
   it cheminn.PEt ?valj
}
LIMIT n
```

***Patron*$_{trait_{agr}}$**
Cond : `Traitement` =
1) Agrégation simple si "$i_t\ chemin_n.PE_t\ ?val_{j+1}$" commence par une clause `SELECT` contenant toutes les variables représentant les I_t^j $(1 \leq j \leq n)$.
2) Agrégation sur les I_t si "$i_t\ chemin_n.PE_t\ ?val_{j+1}$" commence par une clause `SELECT` ne contenant que la variable représentant les I_t^n.

```
{
   SELECT (AGR(?val_{j+1}) AS ?val_j)
   WHERE{
      i_t chemin_n.PE_t ?val_{j+1}.
   }
}
FinalConstr(?val_j)
```

Action : Remplacer `AGR` par l'agrégat à appliquer avec les opérateurs SPARQL 1.1. Remplacer `FinalConstr(`$?val_j$`)` par la contrainte portant sur la valeur finale agrégée obtenue.

***Patron*$_{trait_{agrProp}}$**
Cond : `Traitement` = Agrégation propagée.

```
{
   SELECT (AGR_1(?val_{j+1}) AS ?val_j)
   WHERE{
      SELECT ?ind_1 (AGR_2(?val_{j+2}) AS ?val_{j+1})
      WHERE{
         SELECT ?ind_1 ?ind_2 (AGR_3(?val_{j+3}) AS ?val_{j+2})
            WHERE{
            ...
               SELECT ?ind_1 ?ind_2 ... ?ind_{n-1} (AGR_n(?val_{j+n}) AS ?val_{j+n-1})
               WHERE{
                  i_t chemin_n.PE_t ?val_{j+n}.
               }
               GROUP BY ?ind_1 ?ind_2 ... ?ind_{n-1}
            ...
            }
            GROUP BY ?ind_1 ?ind_2
      }
      GROUP BY ?ind_1
   }
}
FinalConstr(?val_j)
```

Action : Remplacer tous les AGR_k par les agrégats à appliquer avec les opérateurs SPARQL 1.1. Remplacer `FinalConstr(`$?val_j$`)` par la contrainte portant sur la valeur finale agrégée obtenue.

5.2.4 Patrons liés à la traduction des valeurs

Cette dernière catégorie de patrons permet d'effectuer les traductions de valeurs. En effet, si p_s est une propriété datatype dont on connaît le co-domaine, un transtypage peut être nécessaire pour que les valeurs collectées soient du type voulu. Si p_s est une propriété objet, les données acquises doivent être transformées en individus de O_s. Dans les deux cas, cela se fait automatiquement.

***Patron*$_{transty}$**
Cond : p_s est une propriété datatype dont le co-domaine correspond à un type bien précis.

```
{i_t Traitement(chemin_n.PE_t) ?val_{j+1}}
BIND (transtypage(?val_{j+1}) AS ?val_j)
```

Action : Remplacer `transtypage` par le type du co-domaine.

***Patron*$_{indiv}$**
Cond : p_s est une propriété objet.

```
{i_t Traitement(chemin_n.PE_t) ?val_{j+1}}
BIND (xsd:String(?val_{j+1}) AS ?stringVal)
BIND (replace(?stringVal, prefix(O_s), "") AS ?string)
FILTER (!(?string=""))
BIND (iri(concat(prefix(O_t), encode_for_uri(?string))) AS ?val_j)
```

Action : Remplacer $\mathcal{O}_s$ et $\mathcal{O}_t$ par le préfixe de l'ontologie considérée.
Le premier `BIND` transforme en chaîne de caractères les valeurs obtenues qui peuvent être initialement de types variés (string, IRI, etc). Le second est utile en cas d'IRI. Il permet de ne garder que le fragment dénominatif de l'entité considérée. Enfin le dernier `BIND` permet de créer une IRI pour O_s reprenant la partie dénominative de l'entité.

6 Déroulement de la génération de requêtes

Dans cette section, nous déroulons la génération des requêtes sur les deux exemples donnés en section 2 tel que le fait le système, qui supporte ce processus, que nous avons implémenté. Ces exemples illustrent l'apport de notre travail par rapport à l'état de l'art. En effet, à notre connaissance, aucun travail ne porte sur le traitement d'expressions complexes telles que nous les avons définies. (Correndo et al., 2010; Makris et al., 2012) ne considèrent pas les agrégats introduits dans la version 1.1 de SPARQL. Makris et al. (2012) estiment que le travail d'agrégation peut être vu comme un post-traitement d'une requête. Cela est vrai pour des cas simples, mais s'il faut appliquer plusieurs agrégats, chacun portant sur des résultats de sous-requêtes, et si les résultats de ces agrégations font ensuite l'objet de transformations, ce n'est plus vrai. Le premier exemple en est une illustration.

Soit $PE_t = Max(Max(\text{populationDensity}), Max(\text{populationTotal})/Min(\text{areaTotal}))$, correspondant à la densité maximale de la population au km^2 du Canada, citée dans l'exemple 1 donné page 3. L'accès aux valeurs de l'ensemble des éléments composant cette PE_t dans l'ontologie cible est direct. En revanche, une seule valeur doit être retournée, la densité maximale.

La valeur recherchée ne peut pas être obtenue par un post-traitement d'une unique requête SPARQL retournant l'ensemble des valeurs nécessaires au calcul. Il faudrait 3 requêtes indépendantes retournant toutes les valeurs des 3 propriétés mises en jeu, suivies d'un certain nombre d'étapes manuelles pour réaliser le post-traitement :

- agréger chacune des valeurs des 3 propriétés,
- calculer le résultat de la division demandée,

– prendre le max entre les valeurs.

La deuxième étape est de plus basée sur le résultat de la première et la 3ème étape sur le résultat de la seconde. Un tel post-traitement est complexe à mettre en œuvre car il est pratiquement entièrement à la charge du concepteur. Générer une requête prenant automatiquement en compte les agrégats décharge totalement de l'écriture de requêtes SPARQL et semble beaucoup plus approprié. C'est l'objet de notre approche illustrée sur cet exemple par la figure 8.

Dans la figure 8, nous montrons l'évolution de la construction du bloc `WHERE` de la requête finale traitant l'expression "`traduction(`$i_t\ PE_t\ ?val_0$`)`" (sans mécanisme de traitement, ni chemin). PE_s, associée à PE_t, a pour co-domaine `xsd:double`, d'où la présence d'une traduction. La partie externe consiste alors à appliquer les patrons de la catégorie Traduction (cf. ①). La partie interne consiste à appliquer ceux de la catégorie Format (cf. ② à ⑥). Les patrons des catégories Chemin et Traitement ne sont pas utiles ici. L'évolution de la construction de la requête est détaillée ci-après :

① correspond à l'application de $Patron_{transty}.\ transtypage$ a été instancié par `xsd:double`.

② représente le début de l'étape d'application des patrons de Format. La PE_t est de la forme Max($i_t\ PE_{t'}\ ?val_2$). On instancie $Patron_{agr}$ avec l'agrégat `MAX`.

③ représente l'application de $Patron_{ens}$ instancié avec `UNION`. L'expression "$i_t\ PE_{t'}\ ?val_2$" est donc transformée en une union de deux PE_t nommées PE_{t_1} et PE_{t_2}. On a donc représenté Max($i_t\ PE_{t_1}\ ?val_2$, $i_t\ PE_{t_2}\ ?val_2$).

④ représente l'application de $Patron_{agr}$ pour la PE_{t_1} instancié avec `MAX`, et de $Patron_{tranfo2}$ pour la PE_{t_2} instancié avec la division de deux PE_t. On a donc représenté Max(Max($i_t\ PE_{t_{1_a}}\ ?val_3$), $i_t\ PE_{t_{2_a}}\ ?val_{3_a}$ / $i_t\ PE_{t_{2_b}}\ ?val_{3_b}$).

⑤ représente l'application de $Patron_{elem}$ pour la $PE_{t_{1_a}}$ instancié sur la propriété populationDensity, ainsi que de $Patron_{agr}$ instancié avec `MAX` pour la $PE_{t_{2_a}}$ et `MIN` pour la $PE_{t_{2_b}}$. On a donc représenté Max(Max(populationDensity), Max($i_t\ PE_{t_{2_{a'}}}\ ?val_4$) / Min($i_t\ PE_{t_{2_{b'}}}\ ?val_4$)).

⑥ représente l'application de $Patron_{elem}$ pour les deux PE_t restantes instancié sur les propriétés populationTotal et areaTotal. On a donc représenté la PE_t demandée : Max(Max(populationDensity), Max(populationTotal) / Min(areaTotal)).

Nous considérons maintenant l'exemple 2 page 4. Dans ce cas, on détaille les deux phases parallèles (cf. figure 6). La colonne de droite montre la phase externe (Traduction et Traitement) tandis que celle de gauche montre la phase interne (Format et Chemin). L'application des patrons de Format n'est pas détaillée ici car elle est basée sur le même principe que l'exemple précédent. On imbrique le résultat de la phase de la colonne de gauche (partie interne) dans celui de la colonne de droite (partie externe). L'évolution de la construction de la requête est détaillée ci-après :

① représente la requête suite à l'application des patrons de la catégorie Format ($Patron_{agr}$ et $Patron_{elem}$), en suivant le même processus que celui décrit dans la figure 8.

② représente le pré-traitement d'insertion du chemin : ajout de l'expression "`Lambton_County` $chemin_1\ ?ind_1$" dans la clause `WHERE` et ajout de la variable $?ind_1$ représentant les I_t(=I_t^1) dans la clause `SELECT` (et donc dans une clause `GROUP BY` ajoutée).

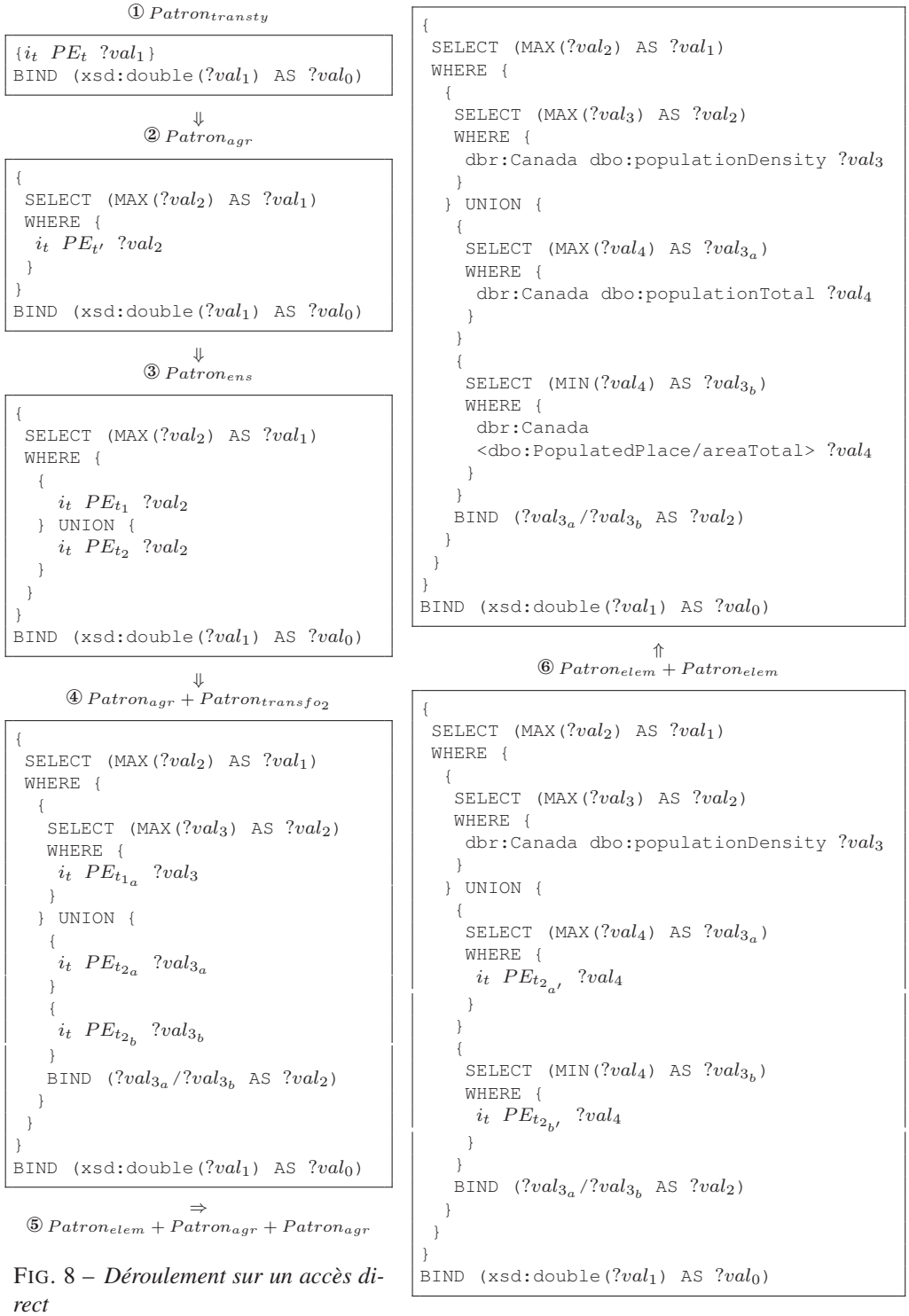

FIG. 8 – *Déroulement sur un accès direct*

③ représente l'application de $Patron_{chem_{ens}}$, transformant l'expression du chemin en un ensemble de deux chemins élémentaires.

④ représente l'application double de $Patron_{chem_{elem}}$, instancié avec les deux expressions de chemins élémentaires ($isPartOf^{-1}$ et $part$). On obtient ainsi la partie interne permettant de collecter les valeurs de la PE_t considérée en passant par le chemin donné.

❶ représente l'application de $Patron_{transty}$ instancié avec `xsd:double` car il s'agit du co-domaine de la PE_s considérée.

❷ représente l'application de $Patron_{trait_{agr}}$. On obtient ainsi la partie externe qui correspond aux traitements finaux (transtypage et agrégation sur les I_t).

❸ et ⑤ permettent de générer la requête finale en imbriquant la partie interne dans la partie externe.

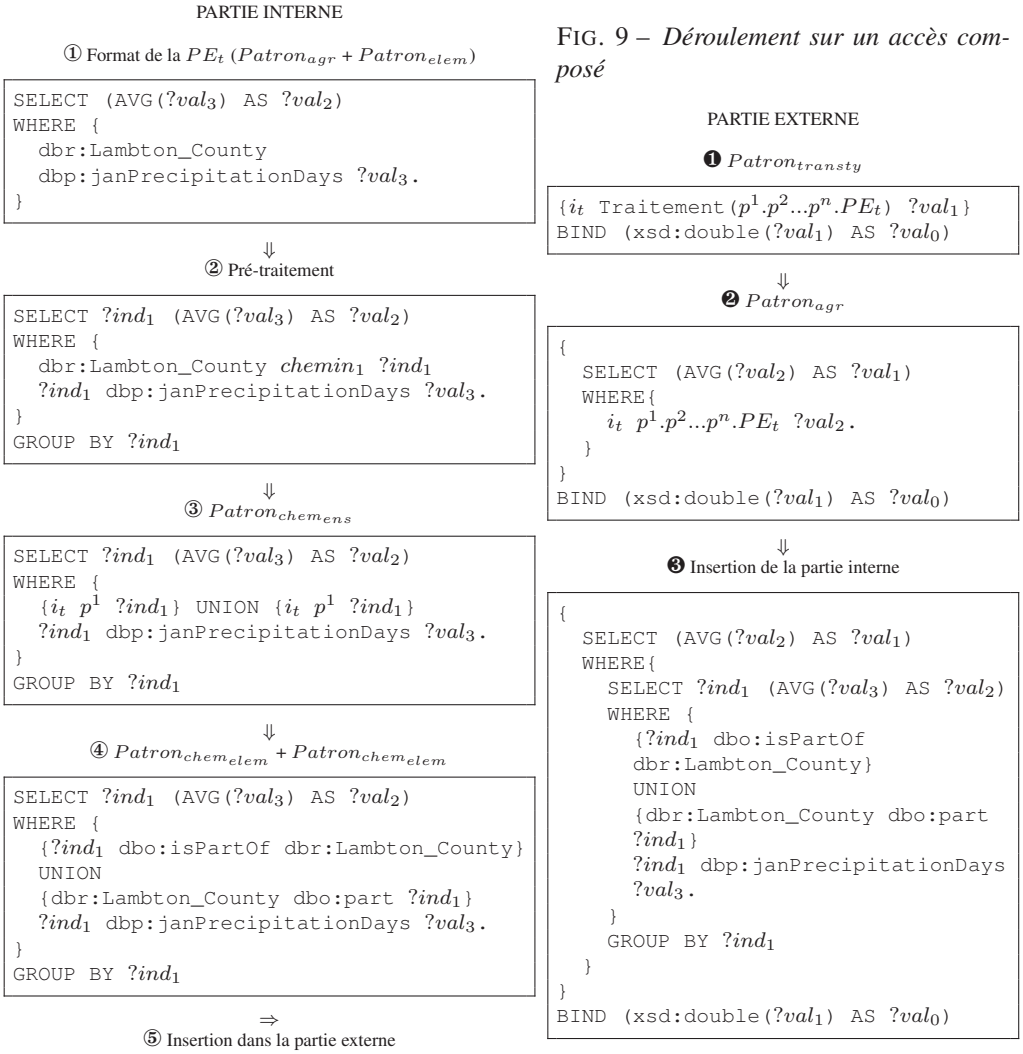

FIG. 9 – *Déroulement sur un accès composé*

7 Conclusion

Nous avons proposé un modèle pour acquérir des données issues de jeux de données du LOD. Grâce à ce modèle, une génération totalement automatique de requêtes en SPARQL 1.1 est possible et a été implémentée. En effet, dans certains cas où les correspondances entre la source et la cible sont complexes, la requête SPARQL permettant de collecter des informations sur la cible pour les ajouter dans la source peut être difficile à écrire pour un humain. Il est impératif de générer cette dernière automatiquement pour éviter les erreurs humaines que pourrait faire un concepteur. Ce processus a été mis en place dans le cadre d'un projet d'annotation sémantique de documents où l'on cherche à trouver des définitions de concepts d'une ontologie via l'application d'une technique d'apprentissage automatique. Pour que cette technique soit performante, nous devons par ailleurs disposer de données les plus complètes possibles, quitte à ce que celles-ci ne soient que des approximations de valeurs. Cela est possible grâce au modèle d'accès explorant les chemins alternatifs. De plus, des contraintes de l'ontologie (telles que les restrictions de cardinalité de certaines propriétés ou encore le type du co-domaine) doivent être respectées si l'on souhaite effectuer un quelconque raisonnement sur celle-ci. Cette nécessité est remplie par les mécanismes de traitement proposés (mécanismes d'agrégation, transtypage, etc). Le processus de génération de requêtes a été appliqué avec succès sur des requêtes comportant entre-autres des successions d'agrégation, que les travaux du domaine ne traitaient pas jusqu'alors. Ceci a permis de valider le modèle d'acquisition des données proposé. En toute généralité, le modèle permet de trouver davantage de correspondances entre des jeux de données RDF correspondant à des ontologies différentes. Il a été appliqué dans le cadre d'une approche de peuplement d'ontologie (Alec et al., 2016a,b) mais peut être utilisé pour d'autres usages, par exemple, pour faciliter la ré-écriture de requêtes et donc permettre l'interrogation de données correspondant à des ontologies différentes.

Un certain nombre de perspectives sont possibles. Tout d'abord, seul un jeu de données du LOD est exploité. Une perspective serait d'adapter notre approche pour exploiter plusieurs jeux de données, certaines données pouvant se trouver dans un jeu et d'autres dans d'autres. Par ailleurs, ces jeux de données ayant la particularité d'être interconnectés les uns aux autres via la déclaration de liens $sameAs$, il pourrait également être intéressant d'étudier leur exploitation conjointe, l'accès à une donnée pouvant nécessiter la définition de chemins composés de propriétés de jeux de données différents. En effet, avec SPARQL 1.1, il est devenu possible d'interroger des points d'accès SPARQL distants (avec le mot-clef `SERVICE`) et de les combiner. Ce dernier problème est complexe car il touche aux problèmes de liage des données du LOD qui représentent aujourd'hui un défi au cœur des recherches de la communauté du Web Sémantique, de par la nature hétérogène, distribuée et la qualité variable des données.

Une autre perspective serait de développer un langage basé sur le modèle présenté, se situant au dessus de SPARQL, et de concevoir une interface basée sur ce langage pour faciliter l'extraction des données du LOD.

Enfin, nous pourrons étudier comment certaines parties du processus proposé (détection de correspondances complexes, spécification des chemins d'accès) pourraient être davantage automatisées.

Références

Alec, C., C. Reynaud-Delaître, et B. Safar (2016a). An Ontology-driven Approach for Semantic Annotation of Documents with Specific Concepts. In *Extended Semantic Web Conference, ESWC*.

Alec, C., C. Reynaud-Delaître, et B. Safar (2016b). Une approche combinée pour l'enrichissement d'ontologie à partir de textes et de données du LOD. In *Extraction et Gestion des Connaissances, EGC*, pp. 171–182. Hermann-Editions.

Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, et Z. Ives (2008). DBpedia : A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference (ISWC)*, Volume 4825 of *Lecture Notes in Computer Science*, pp. 722–735. Springer.

Aussenac-Gilles, N., J. Charlet, et C. Reynaud-Delaître (2014). Ingénierie des connaissances. In P. Marquis, O. Papini, et H. Prade (Eds.), *Panorama de l'Intelligence Artificielle - Ses bases méthodologiques - ses développements - Représentation des connaissances et formalisation des raisonnements*, Volume 1, pp. chapitre 20. Cepadues.

Correndo, G., M. Salvadores, I. Millard, H. G. N., et Shadbolt (2010). SPARQL Query Rewriting for Implementing Data Integration over Linked Data. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, New York, NY, USA, pp. 4 :1–4 :11. ACM.

Euzenat, J. et P. Shvaiko (2013). *Ontology Matching*. Heidelberg : Springer Verlag.

Gillet, P., C. Trojahn, O. Haemmerlé, et C. Pradel (2013). Complex Correspondences for Query Patterns Rewriting (regular paper). In *Ontology Matching at ISWC 2013, Sydney, Australia, 21/10/2013*. CEUR Workshop Proceedings.

Harris, S., A. Seaborne, et E. Prud'hommeaux (2013). SPARQL 1.1 Query Language (2013). In w3c recommendation (2013).

Horridge, M., N. Drummond, J. Goodwin, A. Rector, R. Stevens, et H. Wang (2006). The Manchester OWL Syntax. In *OWLED2006 Second Workshop on OWL Experiences and Directions*, Volume 216, Athens, Georgia, USA,.

Kaufmann, E. et A. Bernstein (2010). Evaluating the Usability of Natural Language Query Languages and Interfaces to Semantic Web Knowledge Bases. *Web Semant. 8*(4), 377–393.

Makris, K., N. Bikakis, N. Gioldasis, et S. Christodoulakis (2012). SPARQL-RW: Transparent Query Access over Mapped RDF Data Sources. In *Proceedings of the 15th International Conference on Extending Database Technology, EDBT 2012*, EDBT '12, pp. 610–613. ACM.

Makris, K., N. Gioldasis, N. Bikakis, et S. Christodoulakis (2010). Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources. In *Proceedings of the 2010 International Conference on On the Move to Meaningful Internet Systems: Part II*, OTM'10, Berlin, Heidelberg, pp. 1108–1117. Springer-Verlag.

Pereira Nunes, B., A. Mera, M. A. Casanova, B. Fetahu, L. A. P. P. Leme, et S. Dietze (2013). Complex Matching of RDF Datatype Properties. In H. Decker, L. Lhotska, S. Link, J. Basl, et A. M. Tjoa (Eds.), *Database and Expert Systems Applications, DEXA 2013, Prague, Czech Republic, August 26-29*, Volume 8055 of *LNCS*, pp. 195–208. Springer Berlin Heidelberg.

Scharffe, F. (2009). *Correspondence Patterns Representation*. Thèse de doctorat, Université d'Innsbrück.

Scharffe, F., O. Zamazal, et D. Fensel (2014). Ontology Alignment Design Patterns. *Knowl. Inf. Syst. 40*(1), 1–28.

Shekarpour, S., S. Auer, A.-C. Ngomo, D. Gerbe, S. Hellmann, et C. Stadler (2011). Keyword-Driven SPARQL Query Generation Leveraging Background Knowledge. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '11, Washington, DC, USA, pp. 203–210. IEEE Computer Society.

Unger, C., L. Bühmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, et P. Cimiano (2012). Template-based Question Answering over RDF Data. In M.-S. Hacid, Z. W. Ras, D. A. Zighed, et Y. Kodratoff (Eds.), *Proceedings of the 21st International Conference on World Wide Web, WWW 2012*, WWW '12, pp. 639–648. ACM.

Zaveri, A., A. Maurino, et L.-B. Equille (2014). Web Data Quality: Current State and New Challenges. *Int. J. Semant. Web Inf. Syst. 10*, 1–6.

Summary

Nowadays, LOD (Linked Open Data) represents a promising source for many applications of the Semantic Web. However, appropriate acquisition techniques have to be developed. The work presented in this paper addresses this problem. The goal is to populate an OWL ontology with property assertions taking into account the existence of multiple, equivalent and multi-valued properties in the LOD, as well as properties without values whose valuation might be calculated elsewhere. Since the correspondences to establish are complex, we propose a model to specify them. We also define a model to specify alternative paths to access properties in case of missing values. Finally, we offer mechanisms to treat the acquired values. Secondly, we show how these models are used to automatically generate SPARQL queries and thus, facilitate interrogation. Two running examples are given.