Expression de requêtes SPARQL à partir de patrons: prise en compte des relations

Camille Pradel, Ollivier Haemmerlé, Nathalie Jane Hernandez

Expression de requêtes SPARQL à partir de patrons: prise en compte des relations

Camille Pradel, Ollivier Haemmerlé, Nathalie Hernandez

IRIT, Université de Toulouse le Mirail, Département de Mathématiques-Informatique,
5 allées Antonio Machado, F-31058 Toulouse Cedex 9

Résumé :

Notre objectif est de masquer la difficulté d'exprimer une requête dans le langage de graphes SPARQL. Nous proposons un mécanisme permettant d'exprimer des requêtes dans un langage pivot très simple, constitué essentiellement de mots-clés et de relations entre ces mots-clés. Notre système associe les mots-clés et les éléments de l'ontologie (concepts, relations, instances) correspondants. Il sélectionne alors des patrons de requêtes pré-écrits, puis les instancie à partir des mots-clés de la requête initiale. Plusieurs requêtes sont alors présentées à l'utilisateur sous forme de phrases descriptives en langue naturelle. L'utilisateur sélectionne alors la requête qui l'intéresse. La requête SPARQL est alors générée.

Mots-clés : SPARQL, langages de requêtes, mots-clés, relations, patrons de requêtes.

1 Introduction

L'objectif de notre travail est de simplifier l'écriture de requêtes exprimées dans des langages de type graphique (graphes conceptuels, SPARQL). Deux types d'approches ont été proposées dans cette optique. La première vise à aider l'utilisateur à formuler sa requête dans un langage destiné à l'interrogation d'annotations. Cette approche n'est pas forcément bien adaptée aux utilisateurs finals : pour écrire une requête, l'utilisateur doit en effet connaître la syntaxe du langage et la représentation des données gérées par le système (le schéma et les données à interroger). Cette aide peut reposer sur des interfaces graphiques, comme dans Athanasis *et al.* (2004) pour des requêtes RQL, Russell & Smart (2008) pour des requêtes SPARQL ou CoGui (2009) pour des graphes conceptuels quelconques qui peuvent ensuite être utilisés comme graphes requêtes en entrée de systèmes d'interrogation comme Genest & Chein (2005). Même si ces interfaces graphiques sont utiles aux utilisateurs finals, ces derniers ont besoin de temps pour s'habituer à l'interface et doivent, quoi qu'il arrive, comprendre la logique de formulation d'une requête sous forme de graphe. Les travaux présentés dans Elbassuoni *et al.* (2010) visent quant à eux à étendre le langage SPARQL ainsi que le mécanisme de projection des requêtes pour permettre l'ajout de mots-clés et de jokers – caractères génériques – lorsque l'utilisateur ne connaît pas le schéma des données à interroger. Cette approche nécessite néanmoins la maitrise de SPARQL.

D'autres travaux visent à générer automatiquement ou semi-automatiquement des requêtes à partir de requêtes utilisateurs exprimées sous forme de mots-clés. Notre travail se positionne clairement dans cette famille d'approches. L'utilisateur exprime son besoin en information de manière intuitive, sans avoir à connaître le langage de requêtes ou le formalisme de représentation de connaissances utilisés par le système. Des approches ont été proposées pour générer des requêtres formelles exprimées dans différents langages comme SeREQL Lei *et al.* (2006) ou SPARQL Zhou *et al.* (2007); Tran *et al.* (2009).

Dans ces systèmes, la génération de la requête passe par les quatre étapes suivantes :

- appariement des mots-clés aux entités sémantiques de la base de connaissances ;
- construction de graphes requêtes établissant des liens entre les entités appariées ;
- classement des requêtes construites ;
- affichage des requêtes proposées sous une forme intelligible et choix de l'utilisateur.

Les approches existantes se focalisent sur trois problèmes : l'optimisation de la première étape en utilisant des ressources externes comme WordNet ou Wikipedia Lei *et al.* (2006); Wang *et al.* (2008), l'optimisation de l'exploration de la base de connaissances en vue de construire les graphes requêtes Zhou *et al.* (2007); Tran *et al.* (2009), et l'efficacité du classement des requêtes proposées Wang *et al.* (2008).

Dans Comparot *et al.* (2010), nous avons présenté un moyen de construire des requêtes exprimées en graphes conceptuels à partir de requêtes utilisateurs composées de mots-clés. Le mécanisme proposé visait à masquer la complexité de la construction de requêtes en termes de graphes, qui n'est pas du tout naturelle pour les utilisateurs finals. Notre travail se fonde sur deux observations. La première est que les utilisateurs souhaitent des langages de requêtes simples, pour l'essentiel fondés sur l'utilisation de mots-clés, langages qu'ils ont l'habitude d'utiliser sur les moteurs de recherche courants. La seconde est que, dans le cadre d'applications réelles, les requêtes soumises par les utilisateurs sont généralement des variations autour de quelques familles de requêtes typiques. Ces observations nous ont conduits à proposer un mécanisme de transformation d'une requête utilisateur exprimée en termes de mots-clés en une requête graphe conceptuel construite en adaptant des patrons de requêtes pré-définis en cohérence avec ces mots-clés. C'était là la différence fondamentale avec les approches développées parallèlement à la nôtre.

L'utilisation de patrons évite la phase d'exploration de l'ontologie en vue de lier les concepts identifiés à partir des mots-clés, puisque les relations potentiellement pertinentes apparaissent dans les patrons. Le processus bénéficie donc des familles de requêtes pertinentes pré-établies pour lesquelles nous savons qu'un réel besoin en information existe. Un des principaux verrous réside dans la sélection du patron correspondant le mieux aux besoins de l'utilisateur.

Une limitation de notre approche était le fait que nous ne prenions pas en compte les propriétés. En d'autres termes, les mots-clés de la requête utilisateur pouvaient être rapprochés de concepts – l'équivalent des classes en RDF – mais pas de relations – l'équivalent des propriétés RDF. Une autre limitation de ce travail était liée au fait qu'il n'était pas possible de qualifier un mot-clé par son "type". Par exemple, quand l'utilisateur proposait dans sa requête le mot-clé "Eastwood", on ne pouvait pas préciser si

Eastwood devait être considéré comme un acteur ou comme un réalisateur. Cet article étend le travail de Comparot *et al.* (2010) dans trois directions. Tout d'abord, nous prenons en compte les propriétés dans la requête finale : un élément de la requête initiale peut être rapproché d'une classe aussi bien que d'une propriété. La deuxième extension est la proposition d'un langage de requêtes plus riche qu'une simple collection de mots-clés : l'utilisateur peut désormais qualifier un mot-clé par un autre mot-clé, ainsi qu'exprimer la relation existant entre deux mots-clés. Ce langage de requêtes, toujours très simple et intuitif, peut être utilisé directement par l'utilisateur final, même si notre objectif à terme est de passer de requêtes exprimées dans un langage proche de la langue naturelle à des requêtes SPARQL. Pour cette raison, nous appelons notre langage de requête "langage pivot". Cet article ne se préoccupe néanmoins pas du passage d'une requête exprimée en langue naturelle vers une requête exprimée dans le langage pivot.

La section 2 présente une vue d'ensemble de notre système, ainsi que le type d'ontologie que nous utilisons et le formalisme de nos patrons de requêtes. Les sections suivantes détaillent le processus suivi par notre système, en passant par l'expression des requêtes en section 3, l'appariement des mots-clés aux entités de notre base de connaissances en section 4, la sélection des patrons de requêtes en section 5, le classement des requêtes par ordre de pertinence en section 6 et la génération des requêtes SPARQL en section 7. Finalement, la section 8 présente rapidement l'implémentation que nous avons développée ainsi que les premiers résultats expérimentaux.

2 Vue d'ensemble de notre système

2.1 Description du processus

Le processus suivi par notre système se déroule de la manière suivante : à partir d'une requête exprimée dans notre langage pivot et d'une ontologie OWL, la requête est appariée à des éléments de notre base de connaissances (concepts, propriétés, instances). Nous associons alors les patrons de requêtes à ces éléments. Les différentes associations sont proposées à l'utilisateur dans un langage compréhensible (des phrases exprimées en langue naturelle). La phrase sélectionnée nous permet alors de construire la requête SPARQL.

2.2 La connaissance terminologique : l'ontologie

Afin de permettre une interprétation correcte des requêtes utilisateur, notre système se fonde sur l'exploitation d'une ontologie OWL contenant les connaissances terminologiques d'un domaine spécifié et d'une base de faits associée, définissant notamment les instances des concepts de l'ontologie. Notre système, dans sa version actuelle, exploite la liste des classes (ou concepts), propriétés (ou relations), types de littéraux (ou types de données) et instances existantes, ainsi que les relations taxinomiques entre classes et propriétés. La figure 1 représente un sous-ensemble de l'ontologie utilisée pour les expérimentations présentées dans la partie 8.

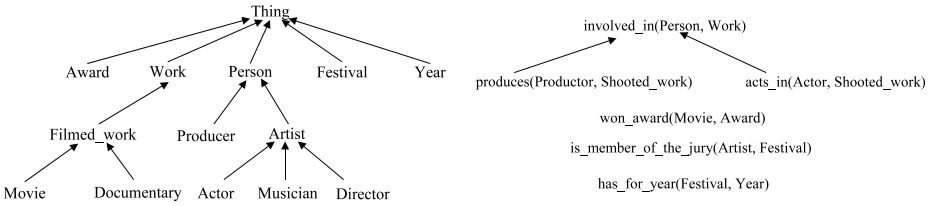

FIGURE 1 – Sous-ensemble de l'ontologie utilisée représenté sous forme de graphe orienté : la partie gauche représente une hiérarchie de concepts ; la partie droite représente les propriétés ainsi que leurs domaines et co-domaines.

De plus, pour chaque classe, propriété et instance, la définition d'étiquettes (ou labels) est essentielle au bon fonctionnement du système. En effet, de la qualité et de l'exhaustivité de ces étiquettes dépend la réussite d'une étape importante du processus d'interprétation, la phase d'appariement des mots-clés à des éléments de la base de connaissances, présentée en 4. Dans l'exemple précédent, la classe $Award$ porte les étiquettes "award" et "prize", et la relation $acts\_in$ les étiquettes "acts in" et "is actor in".

2.3 Les patrons de requêtes

Un patron est composé d'un graphe RDF qui est le prototype d'une famille de requêtes typiques. Ce patron est caractérisé par un sous-ensemble des sommets de ce graphe – aussi bien des concepts que des propriétés ou des types de données – appelés "sommets qualifiants", qui peuvent être modifiés pendant la construction de la requête finale. Le patron est également décrit par une phrase en langue naturelle dans laquelle une sous-chaîne distincte est associée à chacun des sommets qualifiants.

Dans l'état actuel des choses, les patrons sont conçus par des experts qui connaissent le domaine d'application et la forme habituelle des graphes d'annotation. Les graphes RDF des patrons sont construits manuellement, les sommets qualifiants étant sélectionnés par le concepteur du patron qui fournit également la phrase descriptive.

Définition 1
Un patron p est un quadruplet $\{\mathcal{G}_p, \mathcal{C}_p, \mathcal{R}_p, \mathcal{S}_p\}$ tel que :

- *$\mathcal{G}_p$ est le graphe RDF décrivant le patron ;*
- *$\mathcal{C}_p = \{c_1, c_2, \ldots c_n\}$ est l'ensemble de n concepts distincts du graphe $\mathcal{G}_p$, appelés concepts qualifiants du patron ;*
- *$\mathcal{R}_p = \{r_1, r_2, \ldots r_m\}$ est l'ensemble de m propriétés distinctes de $\mathcal{G}_p$, appelés propriétés qualifiantes du patron ;*
- *$\mathcal{S}_p = \{s, (wc_1, wc_2, ...wc_n, wr_1, wr_2, ...wr_m)\}$ est une description du sens du patron en langue naturelle (la phrase s) dans laquelle $n+m$ sous-chaînes distinctes correspondent aux concepts et propriétés qualifiants. Les wc_i sont les expressions correspondant aux concepts qualifiants du patron (wc_i correspond au concept c_i), les wr_i étant les expressions correspondant aux propriétés qualifiantes du patron (wr_i correspond à la propriété r_i).*

Exemple 1
Dans cet article, nous utilisons tout au long de nos exemples deux patrons p_1 et p_2. Fig. 2 et Fig. 3 présentent respectivement les graphes RDF associés à p_1 et p_2.

$\mathcal{C}_{p_1} = \{c_{11}, c_{12}, c_{13}, c_{14}\}$. $\mathcal{R}_{p_1} = \{r_{11}, r_{12}\}$. $\mathcal{S}_{p_1}$ est la phrase "A movie$_{wc_{11}}$ awarded in$_{wr_{11}}$ a festival$_{wc_{12}}$ in a year$_{wc_{13}}$ when an artist$_{wc_{14}}$ is member of the jury$_{wr_{12}}$".

$\mathcal{C}_{p_2} = \{c_{21}, c_{22}, c_{23}, c_{24}\}$. $\mathcal{R}_{p_2} = \{r_{21}\}$. $\mathcal{S}_{p_2}$ est la phrase "An artist$_{w_{21}}$ awarded$_{wr_{21}}$ for a film$_{wc_{22}}$ during a festival$_{wc_{23}}$ in a year$_{wc_{24}}$".

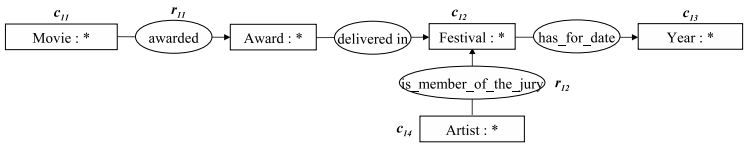

FIGURE 2 – Le graphe RDF $\mathcal{G}_{p_1}$ composant le patron p_1

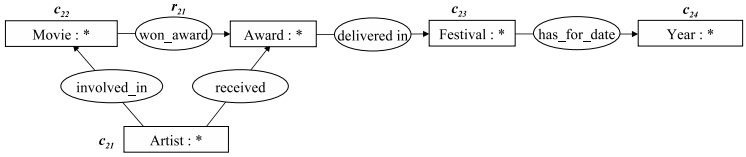

FIGURE 3 – Le graphe RDF $\mathcal{G}_{p_2}$ composant le patron p_2

3 Étape 1 : Expression des requêtes utilisateur

Le langage pivot que nous proposons est une extension d'un langage de mots-clés. L'utilisateur peut exprimer sa requête à l'aide de simples mots-clés, sans savoir s'ils seront associés à des concepts, des propriétés ou des instances. Il peut également "qualifier" un mot-clé à l'aide d'un autre mot-clé, ou encore qualifier, à l'aide d'un mot-clé, la nature de la relation existant entre deux autres mots-clés. Toutes ces possibilités peuvent être combinées afin de construire des requêtes complexes.

Une requête utilisateur exprimée dans notre langage pivot est composée d'un ensemble de sous-requêtes qui peuvent être de l'un des trois types suivants :

- un mot w, considéré comme un mot-clé au sens habituel des moteurs de recherche classiques. Par exemple la requête "homme" signifie que nous cherchons les hommes ;
- un couple de mots séparés par deux points $w_1 : w_2$, ce qui correspond au mot-clé w_1 qualifié par le mot-clé w_2. Par exemple la requête "homme : marié" signifie que nous recherchons les hommes mariés ;
- un triplet de mots $w_1 : w_2 = w_3$, qui correspond à un mot-clé w_1 lié à la valeur exprimée par w_3 par la relation w_2. Par exemple la requête "homme : épouse=Carlita" signifie que nous cherchons les hommes dont l'épouse est Carlita.

Définition 2
*Un **mot-clé** est une chaîne de caratères exprimant le sens d'un concept ou d'une propriété sur lesquels l'utilisateur effectue une recherche.*

Définition 3
*Un **élément de requête utilisateur** peut être :*

1. *une expression de la forme [?][$id]mot-clé. Le symbole optionnel ? signifie que l'utilisateur veut obtenir des résultats spécifiques correspondant à ce mot-clé. L'expression optionnelle $id (où id est un entier) permet d'utiliser un même mot-clé pour désigner deux entités distinctes. Dans ce cas-là, les valeurs de id doivent être différentes.*
2. *un littéral, qui doit être compris au sens habituel du terme. Il est caractérisé par son type, appelé **type de données** dans la suite.*

Définition 4
*Une **sous-requête utilisateur** appartient à l'un des trois ensembles Q_1, Q_2 ou Q_3 :*
- *quand la sous-requête $q_i = \{e_{i1}\} \in Q_1$, elle est un simple élément de requête e_{i1}, identifiant le sujet de la sous-requête, autrement dit une entité impliquée dans le résultat attendu,*
- *quand la sous-requête $q_i = \{e_{i1}, e_{i2}\} \in Q_2$, elle est composée de deux éléments e_{i1} et e_{i2}, séparés par deux points ; e_{i1} identifie le sujet de la sous-requête tandis que e_{i2} identifie quelque chose qui est lié au sujet,*
- *quand la sous-requête $q_i = \{e_{i1}, e_{i2}, e_{i3}\} \in Q_3$, elle est composée de trois éléments e_{i1}, e_{i2} (separé de e_{i1} par deux points), et e_{i3} (separé de e_{i2} par le caractère '=') ; e_{i1} identifie le sujet de la sous-requête, e_{i2} identifie une propriété du sujet ou une relation impliquant le sujet, et e_{i3} identifie la valeur de la propriété ou l'autre entité impliquée dans la relation.*

Définition 5
*Une **requête utilisateur** $q = \{q_1, q_2, ..., q_n\}$ est un ensemble de* sous-requêtes utilisateurs $q_i \in Q = Q_1 \cup Q_2 \cup Q_3$*, separées par des points virgules.*

Exemple 2

Nous indiquons maintenant le moyen d'exprimer, à l'aide de notre langage pivot, un certain nombre de requêtes. Ces requêtes ne doivent pas être vues comme des requêtes formelles qui vont donner un résultat évaluable algorithmiquement, mais comme un moyen d'exprimer de manière intuitive un besoin en information de l'utilisateur final. Notre système cherchera alors à proposer à l'utilisateur une requête formelle en SPARQL correspondant à son besoin effectif.

1. Liste des acteurs :
   `?actor`

2. Liste des acteurs mariés :
   `?actor: married`

3. Liste des acteurs agés de 40 ans :
   `?actor: age=40`

4. Les deux requêtes précédentes peuvent être combinées afin d'obtenir la liste des acteurs mariés agés de 40 ans :
   `?actor: age=40; ?actor: married`
   Par souci d'ergonomie, notre langage pivot autorise la simplification suivante :
   `?actor: age=40, married`

5. Liste des acteurs dont l'épouse est agée de 40 ans :
   `?actor: spouse=woman; woman: age=40`

6. Liste des acteurs agés de 40 ans qui se sont mariés en 2010 à Hollywood :
   `?actor: age=40, marriage; marriage: date=2010, place=Hollywood`
   Cette requête pourrait également s'exprimer de la manière suivante :
   `?actor: age=40, marriage date=2010, marriage place=Hollywood`

7. L'exemple suivant sera utilisé dans l'article pour illustrer le processus de construction d'une requête SPARQL :
   Liste des films récompensés dans des festivals présidés par Emir Kusturica :
   `festival: president= Emir Kusturica, ?awarded movie`

4 Étape 2 : appariement des éléments de requête

La deuxième étape de notre processus consiste à apparier chacun des éléments de la requête utilisateur avec des entités de la base de connaissances (concepts, propriétés et instances) ou avec des types de données (correspondant à des littéraux). Un élément de requête peut être apparié à plusieurs entités. Nous associons alors une valeur de confiance à chaque appariement, correspondant à la qualité supposée de l'appariement ainsi qu'à la vraisemblance de l'appariement reposant sur la place de l'élément apparié dans la sous-requête et le type de l'objet apparié.

Le processus d'appariement peut être vu comme la définition d'une fonction $match$ qui associe une valeur de confiance $t_{match} \in [0; 1]$ à chaque couple (e_q, e_s), e_q étant un élément de requête et e_s une entité de la base de connaissances ou un type de données. Quand $t_{match} = match(e_q, e_s) = 0$, cela signifie que l'élément de requête e_q ne peut pas être apparié à e_s ; dans les autres cas, e_q peut être apparié à e_s avec la valeur de confiance t_{match}.

Cette étape de notre processus consiste à trouver pour chaque élément de requête quels appariements sont envisageables, et à calculer une valeur de confiance pour tous les appariements possibles. Cette valeur de confiance est calculée en deux temps.

Dans un premier temps, nous attribuons une valeur de confiance en fonction du type de l'élément de requête et de la qualité de l'appariement. Comme nous l'avons indiqué précédemment, un élément de requête peut être apparié à une entité de la base de connaissances, ou à un type de données. Nous devons considérer les deux cas.

- Un appariement entre un mot-clé et un élément de la base de connaissances est considéré comme possible quand des similarités sont établies entre le mot-clé et une ou plusieurs étiquettes de l'entité de la base de connaissances. Parce que l'utilisateur n'est pas censé connaître l'ontologie, et dans le but d'assurer une certaine robustesse à notre système, ceci se passe sans considérer le rôle du mot-clé dans la requête ; par exemple, nous considérons aussi bien les appariements entre un élément de requête apparaissant comme sujet d'une sous-requête de Q_3 et une propriété de l'ontologie, que les appariements entre un élément de requête apparaissant en tant que relation dans une sous-requête de Q_3 et un concept ou une instance de la base de connaissances. Seul un type d'appariement n'est pas considéré : un élément de requête précédé d'un point d'interrogation (c.a.d. sur lequel porte la requête) ne peut pas être apparié à une instance, car ça n'aurait pas de sens. Cette étape du processus est obtenue grâce à un algorithme trivial de comparaison de chaînes de caractères appliqué entre les mots-clés de la requête et les étiquettes des entités de la base de connaissances. Cet algorithme détermine la valeur de confiance en fonction du niveau de similarité.
- Pour déterminer si un appariement est possible entre un élément de requête et des types de données, une première condition doit être vérifiée : l'élément de requête doit être le second élément d'une sous-requête de Q_2 ou le troisième élément d'une sous-requête de Q_3. Nous essayons alors d'interpréter le mot-clé comme étant une valeur de chacun des types de données, et si l'interprétation est un succès, alors l'appariement est considéré comme possible. Par exemple, les mots-clés '14-01-2011' et 'the 14^{th} of January, 2011' peuvent être appariés au type de données *date*. La valeur de confiance assignée à l'appariement varie en fonction du type de données et de la méthode d'interprétation ; elle doit être représentative de la fiabilité de cette méthode.

Le second temps du calcul de la valeur de confiance ne concerne que les appariements avec des éléments de la base de connaissances. Dans ce cas, nous cherchons à modifier les valeurs de confiance précédemment calculées afin de tenir compte d'incompatibilités potentielles entre le rôle des éléments de requêtes et le type des entités de la base de connaissances appariées. De fait, dans certains cas, le rôle joué par un élément de requête dans les sous-requêtes où il apparaît nous permet d'inférer le type attendu de l'entité de la base de connaissances. Nous utilisons pour cela les règles suivantes :

- pour un élément de requêtes apparaissant au moins une fois comme premier élément d'une sous-requête de Q_2 ou Q_3, ou comme troisième élément d'une sous-requête de Q_3, les éléments de la base de connaissances appariés devraient être des concepts ou des instances,
- pour un élément de requête apparaissant au moins une fois comme deuxième élé-

ment d'une sous-requête de Q_3, les éléments de la base de connaissances appariés devraient être des propriétés,
– dans les autres cas, le type attendu n'est pas contraint.

Pour chaque appariement généré dans la phase précédente qui ne respecte pas ces règles, nous diminuons la valeur de confiance en la multipliant par un facteur déterminé $f_{incompatible} \in]0;1[$. Cependant, comme il est envisageable que la formulation de la requête par l'utilisateur ne corresponde pas au parti pris de l'ontologie, on ne peut ignorer cet appariement qui pourrait en définitive se révéler être celui qui mène à l'interprétation voulue. C'est pourquoi nous considérons cet appariement comme moins fiable, mais toujours possible.

Exemple 3
Si l'on poursuit notre exemple, "festival" est apparié au concept "festival" avec une valeur de confiance de 1.0, et aux propriétés "is delivered in the festival" et "is member of the festival jury" avec une valeur de confiance de 0.325 ; "president" est apparié aux propriétés "has for jury president" et "is president of the jury" avec une valeur de confiance de 0.667 ; "Emir Kusturica" est apparié à l'instance "Emir Kusturica" avec une valeur de confiance de 1.0 ; et "awarded movie" est apparié au concept "movie" avec une valeur de confiance de 0.667, au concept "award" avec une valeur de confiance de 0.533, au concept "tv movie" et à la propriété "awarded in" avec une valeur de confiance de 0.5.

5 Étape 3 : Association des patrons à la requête utilisateur

Cette étape consiste à associer les patrons à la requête utilisateur. Pour cela, une première phase 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.

Le processus d'association peut donc être vu comme la définition d'une fonction map qui associe une valeur de confiance $t_{map} \in [0;1]$ à chaque couple (e_p, e_q), e_p étant un sommet qualifiant de patron et e_q un élément de requête. Quand $t_{map} = map(e_p, e_q) = 0$, cela signifie que l'élément de patron e_p ne peut pas être associé à e_q ; dans les autres cas, e_p peut être associé à e_q avec la valeur de confiance t_{map}.

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

1. le sommet qualifiant du patron est un concept c_1. Il peut y avoir une association m_e avec une valeur de confiance t_{map} de l'élément de patron vers tout élément de requête e_q verifiant la condition suivante : e_q a été associé lors de l'étape 2 avec la valeur de confiance t_{match} à un concept c_2 ou à une instance de ce concept, et :
   – soit c_2 est le même concept que c_1 ; dans ce cas, la valeur de confiance de l'association est la même que la valeur de confiance de l'appariement : $t_{map} = t_{match}$;
   – soit c_2 est un ancêtre de niveau l de c_1 ; dans ce cas, la valeur de confiance de l'association est la même que la valeur de confiance de l'appariement concerné,

diminuée en la multipliant l fois par un facteur déterminé : $f_{anc} \in [0,1]$: $t_{map} = t_{match} * (f_{anc})^l$;
- soit c_2 est un descendant de niveau l of c_1 ; dans ce cas, la valeur de confiance de l'association est la même que la valeur de confiance de l'appariement concerné, diminuée en la multipliant l fois par un facteur déterminé : $f_{desc} \in [0,1] : t_{map} = t_{match} * (f_{desc})^l$;

2. le sommet qualifiant du patron correspond à une propriété r_1. Nous pouvons déterminer les associations d'éléments avec la même méthode, considérant chaque élément de requête apparié à une propriété r_2, r_2 étant la même propriété que r_1, une de ses ancêtres ou une de ses descendantes.
3. le sommet qualifiant du patron correspond à un type de données. Il peut y avoir une association m_e avec une valeur de confiance t_{map} de l'élément du patron vers chaque élément de la requête apparié avec la valeur de confiance t_{match} au même type de données, et nous aurons $t_{map} = t_{match}$

Dans la suite, quand un élément de requête e_q est apparié à un élément de la base de connaissances e_s (concept, propriété, instance ou type de données), et grâce à cet appariement, nous pouvons inférer qu'un élément de patron e_p peut être associé à e_q, puis nous disons que e_p est associé à e_q au travers de e_s.

Nous pouvons alors générer pour chaque patron toutes les associations possibles à la requête utilisateur – appelées associations de requêtes. Une association de requête m_q consiste en un ensemble d'associations d'éléments (un ou plusieurs éléments de patrons associés à un ou plusieurs éléments de requêtes). Dans une association de requête donnée, un élément de patron peut être associé seulement une fois, i.e. il ne peut pas être impliqué dans plus d'une association d'élément, alors que les éléments de requêtes peuvent être associés plusieurs fois. Pour chaque patron, nous construisons l'ensemble M de toutes les associations possibles de requêtes de la façon suivante : au début, M contient seulement un mapping de requête, qui est un ensemble vide (il ne contient aucune association d'éléments) ; ensuite, pour chaque sommet qualifiant e_p du patron considéré,
- s'il n'y a pas d'association d'éléments impliquant e_p, nous ne faisons rien,
- s'il y a n associations d'éléments impliquant e_p, nous dupliquons l'ensemble d'origine d'associations $n+1$ fois, chaque ensemble considérant une association d'éléments différente (l'association d'éléments est ajoutée à chaque association de requête de l'ensemble copié), et le dernier ensemble n'évoluant pas (c'est à dire que les associations de cet ensemble considèrent que e_p n'est associé à aucun élément de requête). De cette façon, nous considérons toutes les possibilités d'association ou de non-association de e_p et nous nous assurons également que jamais n'est générée une association d'un patron avec une requête qui associerait un même sommet qualifiant à plusieurs éléments de requêtes.

Exemple 4
En poursuivant le déroulement de notre exemple, cette étape nous amène à découvrir plusieurs associations possibles d'éléments ; par exemple "festival" du patron p_1 peut être associé à l'élément de requête "festival" au travers du concept "festival" avec la valeur de confiance 1.0, "shooted work" peut être associé à "awarded movie" au travers

du concept "movie" (specialisation d'un concept) avec la valeur de confiance 0.486, "artist" peut être associé à "Emir Kusturica" au travers de l'instance "Emir Kusturica" (instance d'un concept) avec la valeur de confiance 0.9, "is member of the jury" peut être associé à "president" au travers de la propriété "is president of the jury" (specialisation d'une propriété) avec la valeur de confiance 0.6. Certains éléments de patron, comme "has for date", ou "year", ne peuvent pas être associés.
La génération de toutes les associations de requêtes possibles conduit à un ensemble de plusieurs associations. L'une d'entre-elles est "la bonne", c'est à dire qu'elle représente la requête telle qu'elle est conçue par l'utilisateur : elle utilise le patron p_1 (le plus proche de la requête) et associe c_{12} à "festival", r_{12} à "president", c_{14} à "Emir Kusturica", et c_{11} à "awarded movie". Mais pour l'instant cette association est perdue dans la masse de toutes les associations générées, bien trop nombreuses pour laisser à l'utilisateur la charge de les exploiter telles quelles.

6 Étape 4 : Évaluation de la pertinence de associations

À la fin de l'étape précédente, nous obtenons un ensemble d'associations ; chacune correspond à une requête différente. Si l'ensemble des patrons utilisés contient le patron adapté à la requête utilisateur soumise et si l'appariement des éléments de cette requête s'est déroulé correctement (notamment grâce à la présence des étiquettes adéquates), alors l'ensemble obtenu contient vraisemblablement la requête représentant le réel besoin en information de l'utilisateur. Dans le but d'identifier cette association, nous voulons présenter en priorité à l'utilisateur les associations qui nous paraissent les plus pertinentes, c'est pourquoi nous présentons dans cette partie le calcul d'une *note de pertinence* liée à chaque association. Cette note de pertinence est la composition de plusieurs notes partielles, présentées ci-dessous.

La *note d'association d'éléments* R_{map} permet de prendre en compte la confiance que l'on a en chaque association d'éléments impliquée dans l'association du patron à la requête utilisateur. Les valeurs de confiance des associations d'éléments t_{map} utilisées pour calculer cette note partielle sont elles-mêmes influencées par la valeur de confiance t_{match} de l'appariement concerné et par le nombre de niveaux l séparant l'élément de la base de connaissances apparié et l'élément représenté par le sommet qualifiant. La note d'association d'éléments est calculée de la façon suivante :

$R_{map}(m_q) = \frac{\sum_{m_{e1} \in m_q} t_{map}(m_e)}{|m_q|}$

où $e_q(m_e)$ est l'élement de la requête impliqué dans l'association d'élément m_e.

La *note de couverture de requête* R_{Qcov} représente la proportion d'éléments de la requête utilisateur initiale qui ont réellement été utilisés pour construire l'association du patron à la requête. Plus il y aura d'éléments ignorés dans la requête utilisateur, plus faible sera cette note partielle :

$R_{Qcov}(m_q) = \frac{|e_q \in q / \exists m_e \in m_q / e_q(m_e) = e_q|}{|q|}$

La *note de couverture du patron* R_{Pcov} représente la proportion de sommets qualifiants du patron qui ont réellement été associés à des éléments de requête pour construire l'association du patron à la requête. Cette note partielle n'est, d'après nous, pas aussi significative que la note de couverture de requête (il est tout à fait concevable qu'une

association pertinente ignore certains sommets qualifiants), mais la note de couverture du patron nous permet de favoriser une association d'un petit patron par rapport à une association d'un grand patron lorsqu'elles contiennent les mêmes associations d'éléments ; on s'attend en effet, dans ce cas, à ce que la deuxième association, qui ignore un grand nombre de sommets qualifiants, soit considérée comme moins pertinente que la première.

$R_{Pcov}(m_q) = \frac{|e_p \in p / \exists m_e \in m_q / e_p(m_e) = e_p|}{|p|}$

où $e_p(m_e)$ est le sommet qualifiant impliqué dans l'association d'élément m_e.

Pour terminer, on peut calculer la note de pertinence R d'une association m_q de la façon suivante :

$R(m_q) = f_{map} R_{map}(m_q) + f_{Qcov} R_{Qcov}(m_q) + f_{Pcov} R_{Pcov}(m_q)$

avec $f_{map} + f_{Qcov} + f_{Pcov} = 1$

Exemple 5

Par souci de concision, nous ne décrivons pas ici l'intégralité de la procédure d'évaluation de la pertinence de l'association décrite dans l'exemple précédent. Cette pertinence est évaluée à 0.85 par l'implémentation de notre système présentée en 8 et est la plus élevée parmi l'ensemble des associations générées, ce qui nous confirme le bien-fondé de notre note de pertinence.

7 Étape 5 : Génération de la phrase explicative et de la requête SPARQL

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 une phrase en langue naturelle qui explique 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 ses besoins. De l'association sélectionnée, le système déduit la requête formelle exprimée en SPARQL. Ces deux opérations, la génération des phrases explicatives d'une part et la formulation de la requête graphe d'autre part, sont rendues triviales respectivement par la présence d'un phrase explicative générique ratachée à chaque patron et par l'architecture même des patrons, exprimés sous forme de graphes.

Pour chaque association, la phrase explicative est obtenue en adaptant la phrase générique rattachée au patron concerné par l'association. Pour chaque association d'éléments contenue dans l'association de patron, nous remplaçons la sous-chaîne relative au sommet qualifiant impliqué par une chaîne de caractères obtenue de la façon suivante :

1. si le sommet qualifiant impliqué fait référence à un concept (c'est qu'il est associé à un élément de la requête soit au travers d'un concept, soit au travers d'une instance),
   – si il est associé au travers d'un concept (cela peut être le même concept, un cocept parent ou un concept descendant), la chaîne utilisée pour le remplacement

est un label de ce concept apparié (de préférence le label qui a permis l'appariement entre le mot-clé et le concept), précédé d'un article indéfini (comme "a" en anglais) pour exprimer le fait que l'on fait référence à n'importe quelle instance de ce concept,
   - si il est associé au travers d'une instance (cela peut être une instance de ce même concept, d'un concept parent ou d'un concept descendant), la chaîne de remplacement est un label de cette instance appariée (de préférence le label qui a permis l'appariement entre le mot-clé et l'instance),
2. si le sommet qualifiant impliqué fait référence à une propriété (c'est qu'il est associé à un élément de la requête au travers de cette même propriété, d'une propriété descendante ou d'une propriété parente), la chaîne de remplacement est un label de cette propriété appariée (de préférence le label qui a permis l'appariement entre le mot-clé et la propriété),
3. si le sommet qualifiant impliqué fait référence à un type de données (c'est qu'il est associé à un élément de la requête pour lequel l'interprétation comme une valeur de ce type a réussi), la représentation de la valeur interprétée est utilisée comme chaîne de remplacement,
4. si l'élément de requête impliqué est précédé d'un point d'interrogation dans la requête utilisateur, la chaîne de remplacement est syntaxiquement mise en valeur dans le but d'indiquer à l'utilisateur quel est l'objet de la requête.

Pour ce qui est des sommets qualifiants qui ne sont pas associés à un élément de requête, on conserve telles quelles les sous-chaînes génériques correspondantes.

L'étape de formulation de la requête en SPARQL pour l'association sélectionnée n'est pas plus complexe. En effet, le graphe de requête est équivalent au graphe du patron concerné, à quelques détails près ; voici la liste des changements effectués :

- pour chaque association d'éléments, le sommet qualifiant concerné est remplacé par l'élément de la base de connaissances apparié ou par la valeur du littéral interprété le cas échéant,
- puis, dans le graphe obtenu, chaque sommet faisant référence à une classe ou à un type de données est remplacé par une variable, explicitement déclarée au travers d'un triplet supplémentaire comme une instance de cette classe ou une valeur de ce type.

Comme nous l'avons expliqué en 3, un point d'interrogation devant un élément de la requête utilisateur signifie que cet élément est un des objets de la requête. Nous retrouvons donc naturellement les sommets qualifiants associés à ces éléments de requêtes dans la clause $SELECT$ de notre requête SPARQL. Si il n'y a aucun point d'interrogation dans la requête, il s'agit d'une requête à réponse binaire que l'on traduit par ASK en SPARQL ; dans le cas contraire, pour chaque élément de requête précédé d'un point d'interrogation,

- si le sommet qualifiant concerné fait référence à une classe ou à un type de données, il a déjà été remplacé par une variable dans l'étape précédente, on ajoute donc cette même variable dans la clause $SELECT$,
- sinon (le sommet qualifiant fait référence à une relation) il s'agit d'une demande de spécialisation ou de généralisation de relation, c'est à dire que l'utilisateur voudrait se voir préciser la nature d'une relation entre deux entités. Dans ce cas, le

sommet qualifiant est remplacé dans le graphe requête par une variable, explicitement déclarée comme une sous- ou une sur-propriété de la relation référencée par deux triplets rendus alternatifs en SPARQL avec $UNION$; cette variable est aussi ajoutée dans la clause $SELECT$.

Nous avons ainsi rencensé l'ensemble des éléments du graphe sur lesquels portent la requête et obtenu le graphe requête définitif qui, exprimé en notation Turtle, formera le contenu de la clause $WHERE$ de notre requête. La formation de la requête SPARQL est alors immédiate.

Exemple 6
*La génération de la phrase explicative appliquée à l'association décrite dans l'exemple précédent, donnera la phrase suivante (les sous-chaînes en italique ont été remplacées, même quand elles sont identiques aux sous-chaînes génériques du patron, et l'expression en gras indique l'objet de la requête) : "**A movie** awarded in a festival in a year when Emir Kusturica is president of the jury".*

Comme l'association correspondante a obtenu à l'étape précédente la meilleure note de pertinence, cette phrase sera présentée à l'utilisateur en première position et, si elle est selectionné par ce dernier, la requête SPARQL générée sera la suivante (la définition des préfixes a été retirée par souci de clarté) :

```
SELECT ?movie1 WHERE
{
    ?movie1 :awarded ?award1.
    ?award1 :delivered_in ?festival1.
    :kusturica :is_president_of_the_jury ?festival1.
    :festival1 :has_for_year ?year1.
    ?movie1 rdf:type :Movie.
    ?award1 rdf:type :Award.
    ?festival1 rdf:type :Festival.
    ?year1 rdf:type xsd:gYear;
}
```

8 Implémentation et expérimentations

L'approche présentée dans cet article a été implémentée en Java et en utilisant l'API Jena, dans le but de mettre en valeur les améliorations apportées par rapport à la méthode introduite par Comparot *et al.* (2010) ; pour évaluer cette dernière, nous avions recueilli 160 requêtes distinctes concernant le cinéma d'un point de vue artistique 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 constatons dans un premier temps que même lorsque l'ontologie et la base de fait grandissent, la durée d'exécution moyenne de l'interprétation d'une requête est tout à fait convenable. En effet, avec en tout 68 classes, 56 relations,et 194 instances, le temps nécessaire à l'exécution de chacune des étapes présentées précédemment puis au tri des mappings n'excède jamais une seconde pour les requêtes les plus complexes, et est en moyenne de 300ms sur un processeur double cœur cadencé à 2.4GHz.

Les tests présentés dans la suite ont porté sur 40 requêtes choisies aléatoirement parmi l'ensemble de requêtes recueillies. Trois utilisateurs différents ont tenté de formuler chaque requête en utilisant le langage pivot présenté en 3. 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 suivi la même démarche expérimentale que précédemment, celle proposée par Zhou *et al.* (2007), qui consiste à calculer pour chaque requête $1/r$ où r correspond à la position de la requête idéale dans les résultats retournés par le système, puis à calculer la moyenne de ces valeurs. En réalisant ce test sur chacun des 120 exemples (3 formulations pour chacune des 40 requêtes sélectionnées), nous obtenons un score moyen de 0.81, contre 0.78 lors de la précédente évaluation. De plus, la requête idéale apparaît dans une des trois premières propositions pour 89% des cas, contre 88% lors de la précédente évaluation. Les nouveautés apportées permettent essentiellement d'améliorer l'interprétation de requêtes impliquant des relations, comme par exemple "novel : author= Fred Vargas, adaptation= ?movie ; ?movie : director= Josee Dayan" qui permet de demander la liste des films réalisés par Josée Dayan qui sont des adaptations de romans de Fred Vargas.

De plus, comme expliqué précédemment, ces résultats ont été obtenus à partir de requêtes rédigées par des utilisateurs différents, ce qui montre la flexibilité de notre système qui n'est pas sensible à la subjectivité des requêtes et à la vision de chaque utilisateur.

9 Conclusion

Dans cet article, nous avons proposé une évolution du système introduit par Comparot *et al.* (2010), afin notamment d'offrir à l'utilisateur un moyen d'exprimer des relations dans les requêtes qu'il formule. Pour cela, nous avons proposé un langage de requêtes pivot que nous avons voulu simple, intuitif et flexible, et nous avons adapté le processus d'interprétation de la requête utilisateur pour prendre en compte les relations.

Les résultats de nouvelles évaluations se sont révélés meilleurs que les précédents, sans pour autant que la différence soit frappante. Mais, même si l'amélioration reste raisonnable, nous savons désormais mieux traiter certaines requêtes qui donnaient de mauvais résultats auparavant, et nous avons de sérieuses perspectives de progrès. Nous avons en effet constaté que, pour une grande partie des requêtes SPARQL que nous aimerions voir en haut de la liste mais qui ne le sont pas, elles se font dépasser par d'autres requêtes qui, en réalité, ne respectent pas la structure de la requête utilisateur initiale. Nous devons donc travailler à une amélioration de l'évaluation de la pertinence d'une requête dans le but de tenir compte de la structure de la requête.

Nous souhaitons également travailler à l'amélioration de l'ergonomie du système. Une extension du langage pivot permettrait à l'utilisateur, grâce à des variables anonymes, de référencer et de requêter des entités à propos desquelles il n'a aucune information. Nous aimerions également rendre la génération des associations plus dynamiques en ajoutant la possibilité d'utiliser des expressions régulières dans des portions de patrons, en nous fondant sur les travaux de Alkhateeb *et al.* (2009) ; ces portions pourraient alors se voir omises ou répétées dans l'association finale. Cela permettrait

notamment de rendre les phrases explicatives de chaque requête plus simples et les patrons plus génériques (donc moins nombreux). Enfin, malgré des performances actuelles très correctes, des évolutions sur le plan algorithmique seront probablement à envisager pour le passage à l'échelle du Web ; nous voudrions réduire la complexité de la méthode en utilisant une heuristique qui permettrait d'orienter la génération des associations et d'empêcher de ce fait la génération d'associations considérées comme trop peu pertinentes.

Références

ALKHATEEB F., BAGET J.-F. & EUZENAT J. (2009). Extending sparql with regular expression patterns (for querying rdf). *J. Web Sem.*, **7**(2), 57–73.

ATHANASIS N., CHRISTOPHIDES V. & KOTZINOS D. (2004). Generating on the fly queries for the semantic web : The ics-forth graphical rql interface (grql). In S. A. MCILRAITH, D. PLEXOUSAKIS & F. VAN HARMELEN, Eds., *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, p. 486–501 : Springer.

COGUI (2009). A conceptual graph editor. Web site. http ://www.lirmm.fr/cogui/.

COMPAROT C., HAEMMERLÉ O. & HERNANDEZ N. (2010). Expression de requêtes en graphes conceptuels à partir de mots-clés et de patrons (regular paper). In *Journées Francophones d'Ingénierie des Connaissances (IC), Nîmes, 08/06/2010-11/06/2010*, p. 81–92, http ://www.cepadues.com/ : Cépaduès Editions.

ELBASSUONI S., RAMANATH M., SCHENKEL R. & WEIKUM G. (2010). Searching rdf graphs with sparql and keywords. *IEEE Data Eng. Bull.*, **33**(1), 16–24.

GENEST D. & CHEIN M. (2005). A content-search information retrieval process based on conceptual graphs. *Knowl. Inf. Syst.*, **8**(3), 292–309.

LEI Y., UREN V. S. & MOTTA E. (2006). Semsearch : A search engine for the semantic web. In S. STAAB & V. SVÁTEK, Eds., *EKAW*, volume 4248 of *Lecture Notes in Computer Science*, p. 238–245 : Springer.

RUSSELL A. & SMART P. R. (2008). Nitelight : A graphical editor for sparql queries. In C. BIZER & A. JOSHI, Eds., *International Semantic Web Conference (Posters & Demos)*, volume 401 of *CEUR Workshop Proceedings* : CEUR-WS.org.

TRAN T., WANG H., RUDOLPH S. & CIMIANO P. (2009). Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE*, p. 405–416 : IEEE.

WANG H., ZHANG K., LIU Q., TRAN T. & YU Y. (2008). Q2semantic : A lightweight keyword interface to semantic search. In S. BECHHOFER, M. HAUSWIRTH, J. HOFFMANN & M. KOUBARAKIS, Eds., *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, p. 584–598 : Springer.

ZHOU Q., WANG C., XIONG M., WANG H. & YU Y. (2007). Spark : Adapting keyword query to semantic search. In K. ABERER, K.-S. CHOI, N. F. NOY, D. ALLEMANG, K.-I. LEE, L. J. B. NIXON, J. GOLBECK, P. MIKA, D. MAYNARD, R. MIZOGUCHI, G. SCHREIBER & P. CUDRÉ-MAUROUX, Eds., *ISWC/ASWC*, volume 4825 of *Lecture Notes in Computer Science*, p. 694–707 : Springer.