Conversion des requêtes en langage naturel vers nRQL

Hasna Boumechaal [1], Sofiane Allioua [2], Zizette Boufaida [3]

[1] Université Mentouri, Constantine, Algérie boumechaal.h@gmail.com

[2] Laboratoire LIRE, Université Mentouri, Constantine, Algérie allioua.sofiane@hotmail.fr

[3] Laboratoire LIRE, Université Mentouri, Constantine, Algérie zboufaida@gmail.com

Résumé. L'interrogation des bases de connaissances telles que les ontologies est une exigence centrale du web sémantique. De plus en plus on est forcé à reconnaître l'importance de fournir un accès simple à ces dépôts de connaissances. Cependant les outils existants qui permettent aux utilisateurs d'interroger et de raisonner sur les ontologies utilisent un langage de requête avec une syntaxe complexe et difficile à maîtriser par les experts du domaine. Dans cet article, nous proposons une nouvelle approche pour la conversion des requêtes en langage naturel vers nRQL (new Racerpro Query Language) qui est le langage des requêtes du moteur d'inférence RACER, utilisant les restrictions sémantiques imposées par l'ontologie. La génération des requêtes nRQL s'appuie sur un algorithme basé sur les relations sémantiques entres tous les termes de la requête. La requête ainsi générée est alors envoyée au raisonneur pour l'interrogation de la base de connaissances.

Mots clés: recherche d'information, web sémantique, ontologie, langage naturel, langage nRQL.

1 Introduction

Dans les systèmes de recherche d'information classique, le besoin de l'utilisateur en information est considéré uniquement à travers des mots clés apparaissant dans la requête. La sélection des documents repose sur un appariement entre les termes de la requête et ceux des documents. Ce principe pose différents types des problèmes. L'ambiguïté des termes peut conduire à la sélection de documents non pertinents (problème de bruit). De plus, la prise en compte de la terminologie plutôt que de la sémantique peut conduire à la non sélection de documents pertinents (problème de silence), dans le cas où un même concept est référencé par deux termes différents dans la requête et dans les documents.

Malgré le progrès fait pour représenter et comparer les requêtes et les documents, ces systèmes semblent avoir atteint leurs limites, et une amélioration supplémentaire requiert l'utilisation des ontologies dans les systèmes de recherche d'information (SRI). Cela permet de définir d'autres types d'interrogation qui s'appuient sur les langages du Web Sémantique, où plusieurs moteurs d'inférences peuvent être intégrés

au système afin d'interroger la base de connaissance. Racer et Pellet sont des exemples de moteurs d'inférence reposant sur la logique de description. Afin d'interroger ces moteurs, plusieurs langages d'interrogation ont été définis [1] [2] [3] [4]. Ces langages fournissent des mécanismes permettant d'exprimer des requêtes complexes. La requête est alors exécutée sur la connaissance représentée dans l'ontologie et les instances qui satisfont la requête sont donc retournées.

Cependant l'accès aux ontologies n'est pas une tache simple. Cela engendre deux obstacles majeurs :

(i) L'ambiguïté de la langue naturelle ; il est nécessaire de gérer les liens entre les concepts des ontologies et les termes du langage naturel. Ceci génere des nombreuses difficultés se présentant à tous les niveaux de l'analyse linguistique des requêtes (les problèmes peuvent être d'ordre morphologique, syntaxique, et sémantique). A titre d'exemple, en français, les formes ambiguës sont estimées à environ 25 % du lexique, voire plus pour les mots les plus courants [5]. En sémantique, il n'existe aucune classification universelle.

(ii) La formalisation des requêtes en langage d'interrogation du moteur d'inférence ; même après l'analyse linguistique des requêtes en langage naturel, beaucoup de défis restent à relever dans la conversion des requêtes en langages formels, car l'interprétation sémantique des requêtes est liée d'une part à la représentation des connaissances et d'autre part à la puissance du langage d'interrogation utilisé.

Dans ce qui suit, nous proposons l'architecture d'un système de conversion des requêtes en langage naturel vers nRQL. Ce système permet à l'utilisateur d'interroger une base de connaissances sans avoir une connaissance préalable sur la structure de l'ontologie. Cela confère une souplesse à l'environnement de recherche, car les utilisateurs n'ont pas besoin de répéter la formulation des requêtes en fonction des données recherchées. La recherche ne se limite pas à trouver des ressources référencées par des mots clés, mais tente d'identifier la sémantique des mots d'une requête et d'étendre les possibilités de recherche par rapport à cette sémantique.

L'article est organisé comme suit: la section 2 présente un état de l'art des travaux existants dans le domaine. Dans la section 3, nous décrivons l'architecture globale du système. Nous commençons par détailler les différents composants puis nous donnons un aperçu sur l'algorithme de génération des requêtes nRQL, et le processus de fonctionnement de notre système. Dans la section 4, nous illustrons une étude de cas traité par ce système. La section 5 apporte quelques perspectives à la suite d'une conclusion.

2 Travaux existants

L'interrogation des bases de données par des requêtes en langue naturelle (NLDB), a été parmi les travaux de recherche les plus importants dans les années 70 et 80[6]. Cependant, les d'interfaces entre les langues naturelles et les ontologies n'a été posée

sérieusement que dans ces dernière années. Ainsi, dans le cadre du web sémantique un certain nombre de systèmes fournissant l'accès aux ontologies a été crée.

Le système AquaLog [7] prend une requête de la langue naturelle comme une question d'entrée et retourne des réponses tirées de la sémantique des données compatibles sur une ontologie. L'architecture de AquaLog repose sur un modèle en cascade, dans lequel la requête est traduite en une représentation intermédiaire sous forme des triplets linguistiques. Ces dernniers sont reformulés pour qu'ils soits compatibles avec l'ontologie, en utilisant plusieurs paramètres pour calculer la similarité entre les termes de la requête et les concepts de l'ontologie tel que la classification sémantique de WordNet.

La nouvelle version de AquaLog est PowerAqua [8] qui est un système de question-réponse pour l'interrogation des ontologies hétérogènes et distribuées sur le web, contrairement à AquaLog qui ne peut être utilisé que pour une seule ontologie. PowerAqua prend en entrée des requêtes exprimées en langage naturel et les traduit en un ensemble de requêtes formelles (en utilisant le langage RDQL ou SPARQL). Il retourne enfin des réponses pertinentes tirées des ressources distribuées sur le web sémantique.

Querix [9] est un autre système de question-réponse basé sur les ontologies, qui traduit les requêtes en langage naturel vers le langage SPARQL. Pour extraire l'arbre syntaxique de la requête, Querix détermine la séquence des catégories grammaticales des mots de la requête : Nom (N), Verbe (V), Préposition (P), Q-terme (Q), Conjonction (C), puis il génère le squelette de la requête. La requête "Quelles sont les tailles de population des villes qui se trouvent en Californie? " à par exemple le squelette Q-V-N-P-N-Q-V-P-N. Querix fait ensuite correspondre le squelette de la requête avec les triplets de l'ontologie. Enfin, le générateur des requêtes formelles produit une liste de requêtes SPARQL. Querix ne cherche pas à résoudre les ambiguïtés de langage naturel. En cas d'ambiguïté, il demande la clarification grâce un dialogue avec l'utilisateur.

SemSearch [10] est un système basé concept qui tend à réaliser une interface similaire à celle de Google. Il accepte les mots-clés en entrée et produit des résultats qui sont étroitement liées aux mots-clés de l'utilisateur par des relations sémantiques. Dans SemSearch, l'utilisateur doit toujours spécifier le mot-clé objet demandé, ce qui permet de définir le type de résultat attendu. L'un des principaux problèmes de SemSearch est que dans de nombreuses situations il peut y avoir un grand nombre d'entités dans l'ontologie correspondantes à un mot clé (terme général) de la requête. Le moteur de recherche a besoin de combiner toutes les correspondances sémantiques des mots-clés ensemble, et de construire une sous requête pour chacune de ces combinaisons. Par exemple, pour une requête contenant les mots clés k_1, k_2, ..., k_n, si on suppose que le nombre des entités correspondantes au mot-clé k_i est n_i. Il y aura $n_1 * n_2 *...* n_n$ combinaisons possibles.

ONLI [11] est un système d'interaction de l'ontologie en langage naturel, basé sur les restrictions sémantiques. Il prend en entrée une demande libre en langue naturelle, la traduit en nRQL, puis génère les réponses pertinentes. Le système a été évalué sur l'ontologie de FungalWeb. Mais, la correspondance sémantique dans ce système est basée sur les prédicats des triplets linguistiques, bien que que dans la plupart des cas ils soient vides, ce qui est ignoré par ce système. Enfin, pour faciliter l'interaction avec l'ontologie un outil d'interrogation appelé OntoIQ (Ontoligent Interactive Query

Tool) [12] à été developpé, qui transforme les modèles des requêtes automatiquement dans la syntaxe du langage nRQL. Un modèle de requête peut être un concept, un rôle, ou une conjonction de concepts et de rôles. Mais, cet outil nécessite la reconnaissance du domaine de l'ontologie, et plusieurs modèles de requêtes ne sont pas pris en compte.

PANTO [13] est une interface portable en langue naturelle pour l'accès aux ontologies, qui accepte les requêtes en langage naturel et les convertit en requêtes SPARQL. Elle prend en compte les phrases nominales, parce que la requête en langage naturel peut être généralement considérée comme la combinaison de plusieurs paires de phrases nominales. Les modifications complexes dans les requêtes en langage naturel telles que la négation, le superlatif, et la comparaison sont prises en compte par ce système. Les ontologies utilisées pour l'évaluation sont de petite taille.

Enfin, QuestIO [14] est un système pour l'accès aux informations structurées d'une base de connaissances. Il se caractérise par la simplicité de l'interface de recherche comme dans Google, et par une recherche conceptuelle basée sur la conversion automatique des requêtes écrites en langage naturel vers des requêtes formelles (SPARQL ou autres). En utilisant les techniques robustes de traitement automatique du langage naturel, et des méthodes pour la désambiguïsation fondées sur les ontologies, ce système est capable d'accepter les requêtes mal formées ou des fragments courts. Malheureusement, le processus de conversion des requêtes reste très difficile.

Bien que, les outils existants aient été principalement conçus pour améliorer la performance des technologies traditionnelles de recherche d'information, nous constatons que leurs performances sont fortement influencées par celles des techniques de traitement automatique de la langue naturelle utilisées. Dans [7] [8] [11] [13], ils ont supposés que toutes les requêtes peuvent être écrites sous forme des triplets linguistiques <terme, relation, terme>, or le cas où la relation n'est pas citée dans la requêtes est très fréquents. De plus, la plupart des systèmes existants sont principalement orientés vers les requêtes qui contiennent jusqu'à deux triplets, et ils sont basés sur la recherche des liens entre les termes de la requête et les entités de l'ontologie plutôt que la relation sémantique entre ces termes.

Notre idée dans ce travail est de construire un système qui prend en entrée une ontologie et une requête en langage naturel et renvoie la requête nRQL en sortie. La conversion est faite par l'utilisation des restrictions sémantiques imposées par l'ontologie. La génération des requêtes nRQL s'appuie sur un algorithme basé sur les relations sémantiques entres tous les termes de la requête.

3 Architecture du système

L'architecture du système proposé supporte trois phases de traitements (**Fig.1**). La première est une phase du traitement lexical de l'ontologie, la deuxième est l'analyse linguistique des requêtes, et la dernière est celle de la génération des requêtes nRQL.

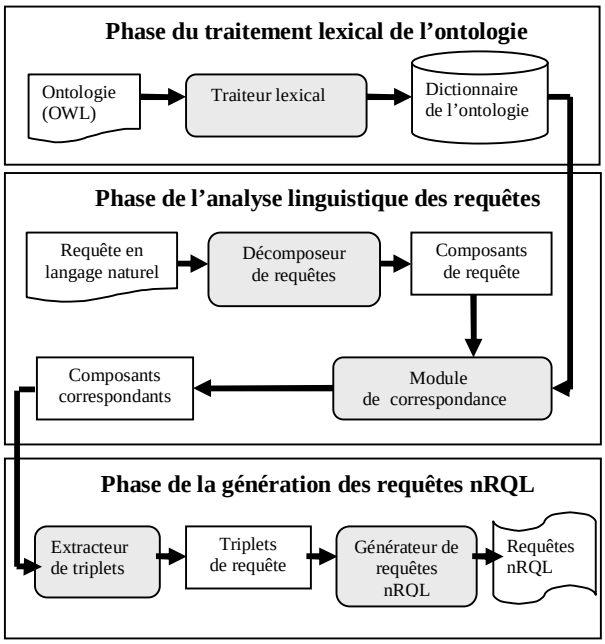

Fig. 1. Architecture du système.

3.1 Composants du système

L'architecture du système proposé est constituée des composants suivants :

1. Un traiteur lexical admet en entrée l'ontologie du domaine qui encapsulent plusieurs types d'information (taxonomie entre des concepts de même type modélisé par la relation is_a, thésaurus des concepts de type différent liés par les relations object_property et data_type_property, et des données formelles spécifiées par des axiomes). Il renvoie en sortie le dictionnaire de l'ontologie qui contient les éléments suivants*:*
   -les entités de l'ontologie ; c'est la partie la plus importante du dictionnaire de l'ontologie. Elles y compris les classes (concepts), les propriétés (relations), et les instances (individus), sont extraites et stockées pour un accès rapide et une correspondance facile avec les termes des requêtes.
   - les types des entités ; les entités peuvent être des classes, des instances, des propriétés d'objet, des propriétés de type de données.
   -les synonymes ; en vue de combler le fossé entre le vocabulaire de l'utilisateur et le vocabulaire de l'ontologie nous définissons pour chaque entité un ensemble des synonymes extraits à partir des différents dictionnaires linguistiques.

2. Un décomposeur de requêtes permet de segmenter la requête entrée par l'utilisateur pour identifier ses différents composants.
3. Un module de correspondance qui permet essentiellement de faire correspondre chaque composant de la requête avec les entités de l'ontologie en utilisant le dictionnaire de l'ontologie. Autrement dit, chaque composant sera représenté par son correspondant dans l'ontologie (si le composant appartient déjà à l'ensemble des entités de l'ontologie nous n'avons pas besoin de faire la correspondance). Il permet également de supprimer les mots vides. Le résultat généré est une séquence de concepts, de instances, et de rôles qui appartiennent à la requête.
4. Un extracteur de triplets pour traduire la séquence précédente vers un ensemble de triplets sous forme <argument, rôle, argument> ou les arguments peuvent être des classes, des instances , des littéraux, ou vides. Le rôle peut être une propriété d'objet (Object_property), une propriété de type de donnée (data_type_property), une relation de subsomption (is_a), ou vide. Donc, ce composant permet d'identifier tous les triplets possibles dans une requête.
5. Un générateur de requêtes nRQL qui convertit les triplets extraits vers le langage nRQL. L'algorithme présenté dans la section 3.2 se charge de cette conversion.

3.2 L'algorithme de génération des requêtes nRQL

L'algorithme comporte trois étapes, trouver d'abord les liaisons sémantiques entre les différents composants de chaque triplet extrait. Il faudra ensuite établir les liaisons entre les différents triplets de la requête. On doit donc pouvoir extraire une description sémantique du contenu de la requête qui permettra de générer la requête en langage nRQL:

1. *Établir les liaisons entre les différents composant d'un triplet* : Le cas le plus simple est lorsque la requête contient un seul terme, donc deux composants du triplet sont vides. Si le composant qui existe est une classe, on doit rechercher les individus appartenant à cette classe. Si le composant qui existe est une propriété, on doit rechercher les individus reliés par cette propriété. Les autres cas sont résumés dans la description suivante :
   Cas 1 **:** le cas le plus fréquent est que le rôle du triplet est vide, ici on a quelques possibilités :

   - Les deux arguments sont des classes ; le problème devient celui de rechercher une relation susceptible de relier les deux classes. Cette relation peut être une relation de subsomption ou une propriété d'objet. Dans le premier cas, on restreint l'interrogation sur la classe la plus spécifique**.** Par exemple si on a le triplet < animal, _ , carnivore > on restreint l'interrogation sur la classe carnivore qui est une sous classe de la classe animal. Dans le deuxième cas, on remplace le rôle du triplet vide par la propriété de l'objet trouvée.

- Un des arguments est une classe et l'autre une instance ; il faut d'abord vérifier si l'instance appartient à cette classe. Le cas échéant, on restreint l'interrogation sur l'instance. Dans le cas contraire, on doit rechercher s'il existe une propriété d'objet reliant les deux arguments, et on remplace le rôle du triplet vide par la propriété de l'objet trouvée.

- Les deux arguments sont des instances. On doit rechercher s'il existe une propriété de l'objet reliant les deux instances, et on remplace le rôle du triplet vide par la propriété de l'objet trouvée. Par exemple le triplet < poils, _ , jaune > deviendra < poils, couleur , jaune >.

- Un des arguments est une instance et l'autre un littéral **;** on doit rechercher s'il existe une propriété de type de donnée dont le domaine est le type de l'instance et le co_domaine est le type du littéral. On remplace le rôle du triplet vide par la propriété de type de donnée trouvée.

Cas 2 **:** le rôle n'est pas vide, ici on a deux possibilités :

- L'un des argument est vide. On doit vérifier si l'argument qui existe est le domaine ou le co_domaine du rôle. On remplace alors l'argument vide par une variable qui sera utilisée par la suite dans la génération de la requête nRQL.

- Les deux arguments existent. On doit vérifier si ce triplet est valable dans l'ontologie c'est-à-dire si ce rôle relie les deux arguments.

Dans le cas où il n'existe aucune relation sémantique entre les composants d'un triplet, ce dernier est supprimé de la liste des triplets de la requête. Après avoir trouvé la relation sémantique entre les composants de chaque triplet, il faut le mettre dans une forme intermédiaire. Par exemple, la classe C deviendra C(x) et le triplet $<C_1 , R , C_2>$ deviendra $R(C_1(x), C_2(y))$, sachant que R est le rôle dont le domaine C_1 et le co_domaine C_2.

2. *Établir les liaisons entre les triplets de la requête* : Cela est fait par la conjonction de toutes les formes intermédiaires des triplets de la requête. Il est nécessaire de supprimer les liaisons redondantes en choisissant celles les plus spécifiques.
3. *Génération de la requête nRQL* : Il reste enfin à traduire la forme intermédiaire de la requête générée au cours de la phase précédente en langage d'interrogation nRQL. Fondamentalement, une requête nRQL se compose d'une tête et d'un corps. Par exemple, la requête (retrieve (?x) (and (?x |animal|) (?x |viande| |mange|))) a la tête (?x) et le corps (and (?x |animal|) (?x |viande| |mange|)). Elle retourne tous les individus animaux qui mangent de la viande. La syntaxe et la sémantique de langage nRQL sont

décrites en détail dans [15] [16]. Dans notre travail, la requête nRQL est déterminée par quelques règles de génération. Par exemple :

R1: If C(x) then replace: C(x) with (?x |C|)

R2: If R(C_1(x), C_2(y)) then
replace: R(C_1(x), C_2(y)) with (?x |C_1| ?y |C_2| |R|)

R3: If(C(x) and R(C_1(y), C_2(z))) then
replace: (C(x) and R(C_1(y), C_2(z)))with (and (?x
|C|) (?y |C_1| ?z |C_2| |R|))

3.3 Fonctionnement du système

Le fonctionnement du système est résumé dans les étapes suivantes :

1. L'utilisateur entre la requête de recherche et le nom de l'ontologie.

2. Le décomposeur de requêtes se charge d'identifier tous les composants existants dans la requête.

3. Le module de correspondance utilise le dictionnaire de l'ontologie pour déterminer chaque composant par rapport aux entités de l'ontologie ainsi que son type (instance, classe, propriété). Il supprime également les mots vides.

4. L'extracteur de triplets doit identifier tous les triplets possibles dans la requête.

5. Enfin, le générateur de requêtes nRQL fait les liaisons entre les composants de chaque triplet et la conjonction entre tous les triplets puis traduit la requête en langage nRQL.

4 Etude de cas

Pour valider les phases précédentes, nous avons utilisé l'ontologie Animal qui formalise quelques informations du domaine des animaux.

1. L'utilisateur entre la requête : « animal mammifère couvert de poils gris » avec le nom de l'ontologie Animal.
2. Le décomposeur de requêtes décompose la requête on six composants : animal / mammifère / couvert / de / poils / gris
3. Après la correspondance avec les entités de l'ontologie Animal on a : animal $\rightarrow$ classe$_1$

mammifère → classe$_2$
couvert → synonyme : recouvert_de → propriété d'objet$_1$
de → mot vide$_1$
poils → instance$_1$
gris → instance$_2$

4. Après l'extraction de tous les triplets possibles de la requête on obtient :

Tableau 1. Les triplets de la requête.

Triplet	Type du triplet
(animal , _ , mammifère)	(class,vide, class)
(poils , _ , gris)	(instance,vide,instance)
(mammifère , _, gris)	(class,vide,instance)
(mammifère , _, poils)	(class, vide, instance)
(animal ,_ ,gris)	(class, vide, instance)
(animal , _, poils)	(class, vide, instance)
(animal, recouvert_de,_)	(class,object_property,vide)
(mammifère, recouvert_de, _)	(class, object_property, vide)
(_ , recouvert_de, poils)	(vide, object_property, instance)
(_, recouvert_de, gris)	(vide, object_property, instance)
(mammifère , recouvert_de, poils)	(class, object_property instance)
(mammifère , recouvert_de, gris)	(class, object_property, instance)
(mammifère ,recouvert_de,animal)	(class, object_property, class)
(animal , recouvert_de,gris)	(class, object_property, instance)
(animal , recouvert_de, poils)	(class, object_property, instance)
(gris , recouvert_de, poils)	(instance,object_property, instance)

5. La génération de la requête nRQL :

- Liaison entre les composants de chaque triplet et la mise dans la forme intermédiaire:

-Le triplet <animal , _ ,mammifère> est de type rôle vide et les arguments sont des classes, sachant que dans l'ontologie Animal, mammifère est subsumé par animal donc : *mammifère (x).*
-Le triplet <mammifère , recouvert_de, poils> est de type propriété d'objet dont le domaine est mammifère et le co_domaine est la classe de poils donc : *recouvert_de (mammifère (x), poils).*
-Le triplet <poils , _ , gris> est de type rôle vide et les arguments sont des instances reliées par la propriété d'objet couleur donc *: couleur(poils, gris).*
-Le triplet <mammifère , recouvert_de, _> est de type propriété d'objet avec un domaine connu (mammifère) donc : *recouvert_de (mammifère (x), y).*
-Le triplet <_ , recouvert_de, poils> est de type propriété d'objet avec un co_domaine connu (poils) donc : *recouvert_de (z, poils).*

-Le triplet < mammifère , _, poils > est de type rôle vide et les arguments sont une classe et une instance reliées par la propriété d'objet recouvert_de donc : *recouvert_de (mammifère (x), poils).*
Les autres triplets sont supprimés parce qu'il n'existe aucune relation sémantique entre leurs composants.

- Liaison entre les triplets de la requête par des conjonctions et suppression des redondances:

Carnivore (x) and recouvert_de (carnivore(x), poils) and couleur (poils, gris) and recouvert_de (carnivore(x), y) and recouvert_de (z, poils) and recouvert_de (carnivore(x), poils)
On voit que *recouvert_de (carnivore(x), poils)* est plus restreint que *recouvert_de (carnivore(x), y)* et que *recouvert_de (z, poils)* alors on supprime *recouvert_de (carnivore(x), y)* et *recouvert_de (z, poils).* La forme intermediare obtenue est la suivante :
recouvert_de (carnivore(x), poils) and couleur (poils, gris)

- Traduction en nRQL en appliquant les règles de génération :

(Retrieve (?x) (and (?x |carnivore| |poils| |recouvert_de|) (|poils| |gris| |couleur|)

5 Conclusion et perspectives

Dans cet article, nous avons présenté l'architecture d'un système de conversion des requêtes en langage naturel vers nRQL. Les requêtes dans notre système sont formulées en langage naturel. Nous avons mis toutes les fonctionnalités nécessaires pour les traiter telles que : la représentation des termes de la requête sous forme de concepts et de rôles de l'ontologie, et la recherche du lien sémantique entre tous les composants de la requête, ainsi que la génération de la requête nRQL approprié à la requête initiale de l'utilisateur afin d'augmenter la performance du système de recherche. L'améliorations qui peurrait être apportée au système est la combinaison de plusieurs ontologies de domaine dans la conversion des requêtes (par exemple les ontologies Animal et Zoologie).

Références

1. Haarslev, V., MÖller, R., Wessel, M.: Querying the Semantic Web with Racer + nRQL. The KI-04 Workshop on Applications of Description Logics (2004)
2. Karvounarakis,G., Alexaki, S., Christophides,V., Plexousakis, D., Scholl M.: Rql: a declarative query language for rdf. The 11th International World Wide Web Conference. 592--603 (2002)
3. SPARQL Query Language for RDF, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

4. Soumission W3C RDQL, http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
5. Dister A.: Réflexion sur l'homographie et la désambiguïsation des formes les plus fréquentes. Actes des Journées Internationales d'Analyse Statistique des données Textuelles. (2000)
6. Androutsopoulos, I., Ritchie, G.D., Thanisch P.: Natural Language Interfaces to Databases- An Introduction. Journal of Natural Language Engineering. 29-81 (1995)
7. Lopez, V., Pasin, M., Motta, E.: Aqualog: An ontology-portable question answering system for the semantic web. European Semantic Web Conference (ESWC). 546--562 (2005)
8. Lopez, V., Motta, E., Uren, V.: Poweraqua: Fishing the semantic web. European Semantic Web Conference (ESWC). 393--410 (2006)
9. Kaufmann, E., Bernstein, A., Zumstein, R..: Querix: A natural language interface to query ontologies based on clarification dialogs. 5th International Semantic Web Conference (ISWC 2006). 980--981(2006).
10. Lei, Y., Uren, V., Motta E.: Semsearch: a search engine for the semantic web. Managing Knowledge in a World of Networks.. 238--245 (2006)
11. Kosseim, L., SibliniI, R., Baker, C., Bergler, S.: Using Selectional Restrictions to Query an OWL Ontology. International Conference on Formal Ontology in Information Systems (FOIS 2006). 9--11 (2006)
12. Baker, C., Su, X., Butler, G., Haarslev, V.: Ontoligent Interactive Query Tool. CSWWS. 155--169 (2006)
13. Wang, C., Xiong, M., Zhou , Q., Yu, Y.: PANTO: A Portable Natural Language Interface to Ontologies. 4th European Semantic Web Conference ESWC. 473--487 (2007)
14. Tablan, V., Damljanovic, D., Bontcheva K.: A Natural Language Query Interface to Structured Information. the 5h European Semantic Web Conference (ESWC 2008). 361--375 (2008)
15. Wessel, M., MÖller, R. :A High Performance Semantic Web Query Answering Engine. International Workshop on Description Logics. 147 (2005)
16. RacerPro User's Guide Version 1.9 Racer Systems GmbH & Co. KG, http://www.racer-systems.com