REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MENTOURI DE CONSTANTINE
DEPARTEMENT D'INFORMATIQUE

Ecole Doctorale en Informatique de l'Est

Pôle de CONSTANTINE

Option : Intelligence Artificielle

Mémoire Présenté en vue de l'obtention du diplôme de

magister en informatique

Thème :

Conversion des requêtes en langage naturel vers nRQL

Présenté par : M^elle^ HASNA BOUMECHAAL

Dirigé par : Pr. ZIZETTE BOUFAÏDA

Soutenu-le : …./…. /2010 devant le jury composé de:

Présidente :

Dr. FAÏZA BELALA Maître de conférence, Université Mentouri de Constantine

Rapporteur :

Pr. ZIZETTE BOUFAÏDA Professeur, Université Mentouri de Constantine

Examinateurs :

Dr. FOUZIA BENCHIKHA Maître de conférence, Université de Skikda

Dr. RAMDANE MAAMRI Maître de conférence, Université Mentouri de Constantine

Remerciements

Je tiens à remercier en premier lieu dieu le tout puissant qui m'a donné le courage et la patience et qui a éclairé mon chemin pour achever ce travail.

Plus particulièrement je remercie madame Zizette Boufaida Professeur à l'université de Constantine pour la confiance qu'elle m'a accordé, pour son encadrement continu, ses remarques constructives, ses orientations, et ses conseils.

Je tiens à adresser mes vifs remerciements aux membres du jury pour s'intéresser au sujet et avoir accepté de lire ce mémoire.

Je tiens à remercier très sincèrement l'ensemble des membres de l'équipe SI & BC et en particulier Sofiane Allioua pour son aide.

Je tiens à remercier profondément mes parents qui m'ont toujours encouragé et poussé à réussir, pour tout ce qu'ils m'ont donné, leur aide éternel, et leur amour.

A tous ceux qui de prés ou loin ont contribué à la réalisation de ce travail.

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 cette thèse, 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 ; nous utilisons les restrictions sémantiques imposées par l'ontologie pour représenter les termes de la requête sous forme des rôles et des concepts de l'ontologie. Ensuite, la traduction vers nRQL s'appuie sur un algorithme basé sur les relations sémantiques entre 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.

Abstract:

The interrogation of knowledge bases such as ontologies is a central requirement of the Semantic Web. Increasingly, we are forced to recognize the importance of providing simple query access to such knowledge repositories. However, the existing tools that allow users to query and reason over ontologies use query languages with a complex syntax which are reportedly difficult for domain experts to master. In this thesis, we propose a new approach for converting natural language queries to nRQL (new Racerpro Query Language) which is the query language of the inference engine RACER; we used semantic restrictions imposed by the ontology to map terms in the query to concepts and roles in the ontology. Then, the translation into nRQL is done through an algorithm based on the semantic relationships between all terms mapped in the query. The generated query is then sent to the reasoner for querying the knowledge bases.

Keywords: Information retrieval, semantic web, ontology, natural language, nRQL language.

ملخص:

استجواب قواعد المعرفة مثل الانطولوجيا هو شرط أساسي في الويب الدلالي. على نحو متزايد نحن مضطرون إلى الاعتراف بأهمية توفير سهولة الوصول إلى مستودعات المعرفة هذه. ومع ذلك ، الأدوات الموجودة التي تسمح للمستخدمين باستجواب الانطولوجيا تستخدم لغة الاستعلام ذات جمل معقدة ويصعب السيطرة عليها من قبل خبراء المجال. في هذه المذكرة، إننا نقترح إتباع طريقة جديدة لتحويل الاستفسارات باللغة الطبيعية إلى *nRQL (new Racerpro Query Language)* التي هي لغة الاستعلام الخاصة بمحرك الاستدلال *RACER* . لأجل ذلك فإننا نستعمل القيود الدلالية التي تفرضها الانطولوجيا لتمثيل مصطلحات الاستعلام في شكل مفاهيم وأدوار في الانطولوجيا. بعد ذلك، الترجمة إلى *nRQL* مبنية على خوارزم يقوم على العلاقات الدلالية بين جميع مصطلحات الاستعلام. النتيجة يتم إرسالها إلى محرك الاستدلال لاستجواب قواعد المعرفة.

الكلمات الرئيسية:

بحث المعلومات، الويب الدلالي، الانطولوجيا، اللغة الطبيعية، لغة *nRQL*.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

1. Contexte de la recherche

Les gens ont compris depuis bien longtemps l'importance de l'archivage et de la recherche d'information. Avec l'avènement de l'informatique, il est devenu possible de stocker des grandes quantités d'informations et trouver l'information utile de ces collections est devenu une nécessité.

La recherche d'information (RI) permet à un utilisateur de formuler un besoin d'information, à l'aide d'une requête, pour obtenir une réponse issue d'un ensemble de documents. L'idée est apparue dès la naissance des premiers ordinateurs. Elle s'est développée dans les années 1960 avec la possibilité de stocker et d'analyser des masses importantes de données. Depuis, avec la gigantesque augmentation des connaissances produites et conservées numériquement, elle est devenue un secteur stratégique pour beaucoup d'applications, par l'intermédiaire de l'internet.

Dans l'histoire de la recherche d'information, les systèmes sont basés sur plusieurs modèles visant à réaliser l'appariement entre la représentation des documents et celle des requêtes. Il y a eu d'abord l'utilisation du modèle booléen basé sur l'algèbre de Boole et reposant sur une représentation booléenne des requêtes. Dans ce modèle, les documents restitués à l'utilisateur sont ceux contenant exactement les termes de la requête. Ensuite, est apparu le modèle vectoriel qui repose sur les fondements mathématiques des espaces vectoriels. Dans ce modèle, les documents et les requêtes sont représentés sous forme de vecteurs dans l'espace des termes, issus de l'indexation. Plusieurs mesures (Produit scalaire, Mesure de Jaccard, …) ont été utilisés pour calculer la similarité entre les éléments correspondant aux calculs de la distance entre les deux vecteurs. Le modèle probabiliste qui a suivi repose sur la probabilité de la pertinence d'un document connaissant la requête. Le modèle de langage mesure la probabilité de générer une requête à partir du modèle de langage du document. Donc, chaque modèle doit déterminer la relation entre un document et

une requête à partir de leurs représentations, et les documents sont restitués à l'utilisateur par ordre de pertinence supposée décroissante.

Malgré le progrès fait pour représenter et comparer la requête et les documents, ces modèles semblent avoir atteint leurs limites, et une amélioration supplémentaire requiert l'utilisation d'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. 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 tâche 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).

(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 projet, nous proposons l'architecture d'un système de conversion des requête en langage naturel vers nRQL (new Racerpro Query Language) qui est le langage des requêtes du moteur d'inférence RACER. Cette approche permettra à

l'utilisateur d'interroger une base de connaissances sans avoir une connaissance préalable sur la structure de l'ontologie. 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.

2. Plan du mémoire

Pour présenter ce travail, nous avons organisé le mémoire comme suit:

Le premier chapitre, introduit le domaine de la recherche d'information en citant les principes sur lesquels il est fondé. Nous étudions ensuite, le traitement automatique des langues naturelles (TALN) et son influence à ce domaine. Finalement, nous citons les principaux modèles existants qui visent à réaliser l'appariement entre la représentation des documents et celle des requêtes.

Le second chapitre, introduit le web sémantique en précisant son apport par rapport à la recherche d'information traductionnelle. Il comporte le rôle important des ontologies pour réduire les problèmes de représentation de connaissances. Ensuite, nous étudions les langages de représentations et d'interrogations associées au web sémantique en précisant les limites et les avantages de chacun. Enfin, nous synthétisons les différents systèmes d'interrogation d'ontologies avant de situer notre approche.

Le troisième chapitre, présente notre contribution, à savoir le développement d'un système de conversion des requêtes en langage naturel vers nRQL basé sur les restrictions sémantiques imposées par l'ontologie. Ce chapitre présente également l'architecture du système, et les interactions entre les différents composants de cette architecture, le fonctionnement du système, et une comparaison avec les systèmes existants.

Le quatrième chapitre, présente le processus de développement d'une ontologie sur le domaine des animaux, l'ontologie que nous avons développée pour valider notre proposition. Ensuite, nous présentons un échantillon des requêtes traitées par le système afin de découvrir les étapes suivies lors de la conversion des requêtes de langage naturel vers le langage nRQL, ainsi que l'expérimentation mise en place pour évaluer notre proposition.

Enfin, la conclusion générale résume notre contribution et décrit les perspectives.

Chapitre I

Recherche d'information traditionnelle et langage naturel

« S'il y avait des synonymes parfaits, il y aurait deux langues dans une même langue»

César Chesneau

1. Introduction

Le traitement automatique des langues naturelles (TALN) et la recherche d'information (RI) sont deux disciplines dont l'interaction, déjà identifiée depuis longtemps, s'est renforcée ces dernières années.

Les moteurs de recherche disponibles ont donné une nouvelle jeunesse à la recherche d'information, tout en montrant les limites de ses techniques classiques. La recherche d'information, dans la mesure où elle travaille aussi sur des textes, s'apparente au le traitement automatique des langues naturelles. Les méthodes classiques en recherche d'information effectuent néanmoins des traitements linguistiques assez limités.

Ainsi, on peut espérer qu'une meilleure prise en compte de la nature linguistique de documents et de requêtes, comme TALN cherche à l'obtenir, apporte une amélioration des résultats de la recherche d'information.

Dans le présent chapitre, nous donnons un aperçu sur le principe de la recherche d'information. Ensuite, nous nous intéressons au traitement automatique des langues

naturelles et son influence sur ce domaine. Finalement, nous citons les principaux modèles existants pour l'interprétation des requêtes utilisateurs.

2. Principe général de la recherche d'information

La recherche d'information automatisée est une discipline qui met en jeu le stockage et la représentation de l'information d'une part, l'analyse et la satisfaction d'un besoin d'information d'autre part.

Selon **[GRAV 00],** la recherche d'information est le processus consistant à chercher une réponse appropriée au besoin de l'être humain dans la masse de données disponible. Pour satisfaire ce besoin, il ne suffit pas d'apporter à l'utilisateur un document traitant du sujet demandé ; il faut également que le type du document (en termes de longueur, de complexité, de niveau de langage, de portées géographique et temporelle) soit approprié.

La recherche d'information (RI), dans le cas général, comporte deux acteurs principaux :

a) Un être humain, qui a un besoin d'information à un moment donné.

b) Un stock de données, fixé au préalable.

La recherche d'information assistée par ordinateur, ajoute un nouvel acteur qui est le système de recherche d'information (SRI). Celui-ci fait l'interface entre les deux acteurs précédemment cités.

L'utilisateur exprime son besoin à la machine au moyen d'une requête, et le système a pour mission de retrouver dans la masse d'information disponible les documents répondant à la requête. Etant donnée la quantité importante de données potentiellement retrouvée, les systèmes modernes effectuent un classement des documents de manière à placer en tête de liste ceux qui sont jugés les plus pertinents. La figure (**fig.1**) illustre ce modèle de processus de façon très générale. Une phase préalable d'analyse des documents, indépendante du besoin, est nécessaire. Elle correspond en pratique au processus d'indexation, elle utilise un modèle de documents. La formalisation de la requête, ainsi que la recherche et le classement des

documents (à l'aide d'une fonction d'appariement) représentent la base de la phase d'interrogation. Le résultat est, dans la plupart des cas, un ensemble ordonné de références à des documents de la collection (celui jugé le plus pertinent étant bien sur placé en première position). La description théorique de toutes ces étapes est appelée modèle de recherche d'information.

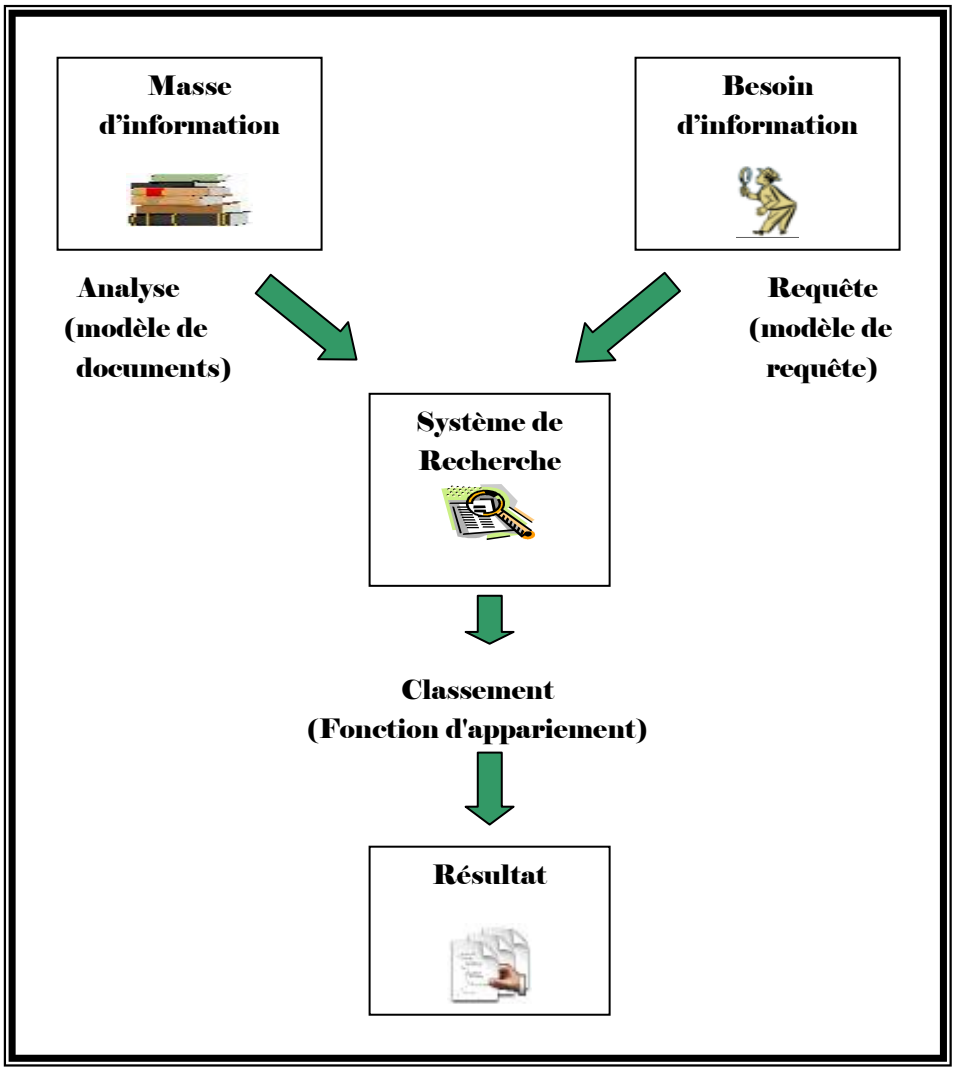

Fig.1 : Principe général de la recherche d'information.

3. Traitement automatique des langues naturelles

Le traitement automatique des langues naturelles (TALN) est une discipline à la frontière de la linguistique, l'informatique et l'intelligence artificielle, qui concerne

l'application de programmes et techniques informatiques à tous les aspects du langage humain. Parmi les applications les plus connues, on peut citer :

- la traduction automatique (historiquement la première application, dès les années 1950).
- la correction orthographique.
- le résumé automatique de texte.
- la génération automatique de textes.
- la synthèse de la parole.
- la reconnaissance vocale.
- la classification et la catégorisation de documents.
- la reconnaissance de l'écriture manuscrite.
- les agents conversationnels.
- la recherche d'information et la fouille de textes.

On voit que le TALN ou le traitement automatique de la langue (TAL) recouvre de nombreuses applications dans les domaines de l'accès à l'information. Notamment, la recherche d'information en fait grand usage : extension de la requête par les synonymes ou les mots apparentés, traitement des groupes nominaux, lemmatisation…etc. Il est ainsi possible de transformer un besoin d'information exprimée en langage naturel vers une requête formelle analysable par un moteur de recherche.

D'autre part, le domaine du langage naturel et de son traitement automatique se trouve au cœur de la problématique de la recherche d'information. Il semble évident que les progrès des futurs moteurs de recherche passeront par une meilleure compréhension de la langue, soit en ce qui concerne les documents ou bien la requête posée. L'état actuel de la recherche est loin de cette compréhension, et de nombreuses difficultés se présentent à tous les niveaux de l'analyse linguistique. Les problèmes peuvent ainsi être d'ordre morphologique, syntaxique, ou sémantique :

3.1. Le niveau morphologique

L'analyse morphologique signifie la manière dont les mots sont construits et quels sont leurs rôles dans la phrase. En pratique, dans le cadre du traitement automatique de la langue, elle consiste à :

- la tokenisation : segmenter le texte en unités élémentaires.
- le tagging : déterminer les différentes caractéristiques de ces unités, et en particulier les catégories grammaticales (nom, verbe, adjectif).
- la lemmatisation : identification de la forme canonique dans le dictionnaire.

Le problème qui se pose est que plusieurs catégories peuvent très souvent convenir à la même forme. Une forme rencontrée dans un texte sera considérée comme ambiguë si elle correspond à plusieurs entrées du dictionnaire. A titre d'exemple, en Français les formes ambiguës sont estimées à environ 25 % du lexique, voire plus pour les mots les plus courants **[DIST 00]**.

L'étiquetage automatique des mots par leurs catégories est aussi un problème bien connu et assez bien maîtrisé pour l'écrit des langues les plus étudiées. Il consiste à gérer les ambiguïtés décrites et à attribuer (au besoin avec un taux de probabilité) une seule catégorie à chaque mot du texte.

Dans la recherche d'information, la prise en compte de proximités morphologiques entre mots aide à obtenir de meilleurs résultats **[ZWEI 01]**. Les techniques employées pour parvenir à ce but sont diverses, mais ont toutes pour principe commun l'utilisation du contexte de chaque mot, c'est à dire l'élimination des combinaisons impossibles ou improbables.

3.2. Le niveau syntaxique

L'analyse syntaxique signifie la manière dont les mots se combinent pour former des syntagmes (syntagmes nominaux, verbaux, adjectivaux, prépositionnels, adverbaux), puis des propositions et enfin des phrases correctes.

Elle est souvent décomposée en :

- le chunking : identification des frontières majeures de constituants (groupe nominal, verbal, etc.).

- le parsing : construction d'un arbre représentant la structure de la phrase complète.

Les ambiguïtés concernant la syntaxe d'un énoncé peuvent avoir de nombreuses causes. Celles-ci ont toutes pour résultat une déduction erronée de l'agencement des mots dans la phrase.

3.3. Le niveau sémantique

C'est l'identification du sens des mots et de la structure logique d'une phrase ; identification des arguments de chaque prédicat et de leur rôle sémantique (agent, but, lieu, etc.).

Un premier niveau de modélisation sémantique consiste à constituer des classes de mots (catégories sémantiques). Ces classes regroupent des mots dont le sens est proche, ou au minimum des mots qui possèdent certaines propriétés sémantiques communes (des classes générales) **[JACQ 00]**. En sémantique aucune classification universelle n'existe (la constitution d'une classification universelle risque même d'être théoriquement impossible). La classification que l'on peut utiliser est la classification générale de WordNet qui est très utilisée dans le domaine du traitement automatique des langues naturelles et la recherche d'information. La signification d'un mot dans WordNet est décidée par ses jeux de synonymes **[KAMP 02]**.

Les applications des analyses sémantiques sont diverses et plus ou moins ambitieuses. Un apport sémantique peut servir à réduire les ambiguïtés syntaxiques, à mieux cibler des concepts (par exemple en recherche d'information), mais sa finalité globale est de représenter formellement l'information véhiculée par un énoncé et éventuellement d'en inférer de nouvelles connaissances ou une réponse à la requête.

Malgré que, le traitement automatique des langues naturelles recouvre les trois niveaux de l'analyse linguistique (morphologique, syntaxique, et sémantique), cependant dans les systèmes de la recherche d'information actuels seules l'analyse morphologique et l'analyse syntaxique sont utilisées. Et quand il s'agit de l'interprétation des sens des requêtes et l'extraction des informations utiles pour les utilisateurs, les capacités des moteurs de recherche actuels sont encore très limitées.

4. Modèles d'interprétation des requêtes

Une requête est la représentation la plus fidèle possible d'un besoin d'information donc leur interprétation est stricte. En recherche d'information, l'utilisateur n'a généralement pas une idée précise du type d'information qui le satisferait le plus (taille, forme et même contenu). Sa requête n'est donc plus le reflet fidèle du résultat souhaité ; elle est plutôt considérée comme une aide apportée au système pour lui permettre de répondre efficacement au besoin d'information. C'est pourquoi il n'existe pas de réponse parfaite à une requête, et il est toujours possible d'améliorer la requête de l'utilisateur pour obtenir des résultats plus proches de son attente. De plus, des requêtes très différentes peuvent conduire à des résultats de pertinence comparable.

[BRAN 03] a noté que le succès d'utilisation des techniques du traitement automatique des langues naturelles dans la recherche d'information dépend de la longueur des requêtes ; plus le nombre des termes dans la requête augmente plus le degré de pertinence des résultats est élevé.

Cependant, d'après **[TANN 06]** un résultat qualitativement satisfaisant ne dépend, que de la composition de la requête ; la clé d'un bon résultat est partagée entre la requête et le système qui la traite. Le moteur de recherche a donc la liberté d'interpréter la requête de façon beaucoup plus souple.

On retrouve les mêmes traitements sur les requêtes que sur les documents. Cependant, en raison de leurs petites tailles, les requêtes peuvent être analysées par des procédures plus lentes et complexes que celles traitant les documents.

Il existe un grand nombre de modèles de recherche d'information ; leur principale différence réside dans la façon dont les documents disponibles et la requête de l'utilisateur sont représentés et mis en correspondance.

4.1. Le modèle booléen

Dans ce modèle, un document est représenté comme une conjonction logique de termes (non pondérés), par exemple :

$d = t_1 \wedge t_2 \wedge \ldots \wedge t_n$

Une requête est une expression logique quelconque de termes. On peut utiliser les opérateurs et ($\wedge$), ou ($\vee$) et non ($\neg$). Par exemple:

$q = (t_1 \wedge t_2) \vee (t_3 \wedge \neg t_4)$

Pour qu'un document corresponde à une requête, il faut que l'implication suivante soit valide:

$d \Rightarrow q.$

Cette évaluation peut être aussi définie de la façon suivante: Un document est représenté comme un ensemble de termes, et une requête comme une expression logique de termes. La correspondance R(d, q) entre une requête et un document est déterminée de la façon suivante:

$R(d, t_i) = 1$ si $t_i \in d$; 0 sinon.

$R(d, t_1 \wedge t_2) = 1$ si $R(d, t_1) = 1$ et $R(d, t_2) = 1$; 0 sinon.

$R(d, t_1 \vee t_2) = 1$ si $R(d, t_1) = 1$ ou $R(d, t_2) = 1$; 0 sinon.

$R(d, \neg t_1) = 1$ si $R(d, t_1) = 0$; 0 sinon.

On peut remarquer trois problèmes dans ce modèle :

1. La correspondance entre un document et une requête est soit 1, soit 0. En conséquence, le système détermine un ensemble de documents non

ordonnés comme réponse à une requête. Il n'est pas possible de dire que tel document est mieux qu'un autre. Cela crée beaucoup de problèmes aux usagers, car ils doivent encore fouiller dans cet ensemble de documents non ordonnés pour trouver des documents qui les intéressent. C'est difficile dans le cas où beaucoup de documents répondent au critère de la requête.

2. Tous les termes dans un document ou dans une requête étant pondérés de la même façon simple (0 ou 1), il est difficile d'exprimer qu'un terme est plus important qu'un autre dans leur représentation. Ainsi, un document qui décrit en détail "informatique", mais mentionne un peu "commerce" se trouve être représenté par {informatique, commerce} dans laquelle les deux termes deviennent aussi important l'un que l'autre. Cela ne correspond pas à ce qu'on souhaite avoir.
3. Le langage d'interrogation est une expression quelconque de la logique de propositions (un terme étant une proposition). Cela offre une très grande flexibilité aux usagers d'exprimer leurs besoins. Cependant, un problème en pratique est que les usagers manipulent très mal les opérateurs logiques, surtout dans beaucoup de cas, les mots "et" et "ou" ne correspondent pas tout à fait aux opérateurs logique $\wedge$ et $\vee$.

Il faut remarquer que le modèle booléen standard n'est utilisé que dans très peu de systèmes de nos jours. Si on utilise un modèle booléen, c'est plutôt une extension de ce modèle qu'on utilise. Les extensions proposées essaient justement de corriger ces lacunes. Par exemple, le modèle booléen basé sur des ensembles flous, qui est une extension du modèle booléen standard, vise à tenir compte de la pondération des termes dans les documents.

4.2. Le modèle vectoriel

C'est un autre modèle souvent utilisé. Dans ce modèle, un document, ainsi qu'une requête, est représenté comme un vecteur de poids. Chaque poids dans le vecteur désigne l'importance d'un terme correspondant dans ce document ou dans la requête. Pour qu'un vecteur prenne une signification, il faut d'abord

définir un espace vectoriel. L'espace vectoriel est définit par l'ensemble de termes que le système a rencontré durant l'indexation. Soit l'espace vectoriel suivant:

$$<t_1, t_2, t_3, \ldots, t_n>$$

Un document et une requête peut être représenté comme suit:

$$d = < a_1, a_2, a_3, \ldots, a_n>$$

$$q = < b_1, b_2, b_3, \ldots, b_n>$$

Où a_i et b_i correspondent aux poids du terme t_i dans le document et dans la requête.

Étant donné ces deux vecteurs, leur degré de correspondance est déterminé par leur similarité.

Il y a plusieurs façons de calculer la similarité entre deux vecteurs. En voici quelques unes:

$$Sim0(d, q) = \Sigma_i (a_i * b_i) \text{ (produit interne)}$$

$$Sim1(d, q) = \Sigma_i (a_i * b_i) / [\Sigma_i (a_i)^2 * [\Sigma_i (b_i)^2]^{1/2} \text{ (cosinus)}$$

$$Sim2(d, q) = 2 \Sigma_i (a_i * b_i) / [[\Sigma_i (a_i)^2 + [\Sigma_i (b_i)^2]$$

$$Sim3(d, q) = \Sigma_i (a_i * b_i) / [[\Sigma_i (a_i)^2 + [\Sigma_i (b_i)^2 - \Sigma_i (a_i * b_i)]$$

A l'exception de la première formule, toutes les autres sont normalisées, c'est-à-dire qu'elles donnent une valeur dans [0, 1].

On n'a pas besoin de comparer chaque document individuellement avec la requête comme exprimé dans ces formules. On peut très bien le faire pour l'ensemble de documents dans la base de documents. L'idée est d'utiliser un fichier inversé. Un fichier inversé donne, pour chaque terme, l'ensemble de documents qui le contiennent et le poids. Ainsi, pour chaque terme de la requête, on peut connaître, par une seule consultation de ce fichier, l'ensemble de documents concernés. On fera cela pour tous les termes de la requête (qui sont

peu nombreux), et cela restreint naturellement l'évaluation à un sous-ensemble de documents dans la base. Le reste du travail consiste à combiner ces ensembles de documents et de calculer le poids.

Par exemple, si on décrit l'évaluation de Sim0 :

1. Initialiser la similarité de chaque document à 0;
2. Pour chaque terme t_i de la requête dont le poids est b_i,
    - trouver l'ensemble de documents correspondants dans le fichier inversé;
    - Pour chaque document dans cet ensemble: additionner le produit (b_i * poids du document dans cet ensemble) à la similarité du document.
3. Une fois que tous les termes de la requête sont utilisés, on obtient la similarité des documents pour la requête.

4.3. le modèle probabiliste

Il complète le modèle vectoriel en calculant la pertinence de chaque index pour un document en fonction de documents répondant à des requêtes sur une base documentaire comparable. Ici aussi, un pourcentage quantifie la pertinence des réponses.

Ce modèle suppose que des jugements de pertinence sont fournis, notamment, suite à une interaction avec l'utilisateur. La pertinence est alors définie sur la base de la distribution des termes de la requête dans les documents pertinents et les documents non pertinents.

La formule générale pour le calcul des poids des termes :

$$wh = \log \frac{ph * (1 - qh)}{qh * (1 - ph)}$$

Où *ph* est la probabilité que le terme *th* soit présent dans un document pertinent, et *qh* est la probabilité que le terme *th* soit présent dans un document *non* pertinent.

D'autres variantes du modèle probabiliste ont été proposées. Par exemple, le modèle Okapi BM25 dans lequel le calcul des poids des termes intègre des aspects relatifs à la fréquence locale des termes, leur rareté et la longueur des documents :

$$w_{i,h} = log\left(\frac{n_d - df_h + 0,5}{df_h + 0,5}\right) * \frac{(k_1+1) * tf_{i,h}}{k_1 * \left((1-b) + b * \frac{dl_i}{\overline{dl}}\right) + tf_{i,h}}$$

Où $dl = moy_{di \in D}$ (dli) et $k1$, b sont des paramètres qui dépendent de la collection ainsi que du type des requêtes.

4.4. Les modèles de langues

L'utilisation des modèles de langue (ML) en recherche d'information a été introduite en 1998 par Ponte et Croft **[PONTE 98]**. Les résultats obtenus par ces modèles ont montré des performances équivalentes voire supérieures aux modèles classiques.

Pourtant, ils se basent sur l'hypothèse d'indépendance des termes (unigramme) ou des multi termes (bigramme, trigramme). Dans les ML, comme généralement en RI, cette hypothèse est plus une facilité mathématique qu'une réalité, une partie de la sémantique des documents s'exprimant à travers les relations qu'entretiennent les mots, en particulier les relations syntaxiques.

Le principe de base des modèles de langue en RI est d'ordonner chaque document D de la collection C suivant sa capacité à générer la requête Q. Ainsi, il s'agit d'estimer la probabilité de génération P(Q|D) comme suit :

$$P(Q \mid D) = \prod_{t \in Q} P(t \mid \theta_D)^{c(t;Q)}$$

Où c(t ;Q) est la fréquence du terme t dans la requête Q, et θ_D est le modèle du document.

Suivant le cadre de travail proposé dans **[LAFF 01]**, la similarité entre un document et une requête peut aussi être exprimée par la mesure de divergence de Kullback-Leibler. La KL-divergence exprime la distance entre les distributions mises en relation. Ainsi, une fonction d'ordonnancement peut être définie comme suit :

$$Score(Q,D) = -KL(\theta_Q \parallel \theta_D) = \sum_t P(t \mid \theta_Q) \log \frac{P(t \mid \theta_D)}{P(t \mid \theta_Q)} \propto \sum_t P(t \mid \theta_Q) \log P(t \mid \theta_D)$$

Où θQ est le modèle de Q, généralement estimé par la fréquence relative des mots-clefs dans la requête. Cette fonction de score peut aussi être vue comme une entropie croisée.

Dans l'ensemble des modèles précédents, la priorité des moteurs de recherche est de retourner en un minimum de temps, le plus de documents pertinentes sur un sujet donné, en s'appuyant sur les techniques des systèmes de recherche d'information traditionnels (SRI), qui sont basés sur le traitement syntaxique de mot clés, en négligeant le coté sémantique qui rendra les résultats plus pertinents.

5. Conclusion

La capacité à manipuler le contenu des documents web sur la base de leur sémantique permettrait à des programmes de réaliser des tâches qui sont actuellement réalisées à la main et ouvre la voie à de nouvelles possibilités d'automatisation. Le web sémantique va structurer le contenu sémantique des documents web et ainsi permettre à des programmes d'interpréter leur contenu et de raisonner dessus ; où les informations ne seraient plus stockées mais comprises par les ordinateurs afin d'apporter à l'utilisateur ce qu'il cherche vraiment. Ceci fera l'objet du prochain chapitre.

Chapitre II

Vers un web sémantique

« The Semantic Web can increase how we use and engage ourselves with computers»

Tim Berners-Lee

1. Introduction

Les pages web représentent une masse de connaissances aussi énormes que disparates. Cette masse augmente sans cesse ainsi que le nombre d'utilisateurs qui veut trouver facilement les informations qu'ils y recherchent.

Les systèmes de recherche d'information actuels se limitent à une recherche par mots clés, basée sur le contenu textuel des documents. Sachant que, en recherche d'information l'absence de la sémantique conduite à une formulation très complexe de requêtes, ce qui pose le problème d'accès aux informations pertinentes sur le Web.

Cependant, grâce aux efforts de la communauté du web sémantique (W3C), une deuxième génération est établie dont la vision initiée en 1998 par Sir Tim Berners-Lee **[LEE 01]** a pour objectif de structurer les informations disponibles sur le web. Cela permettra contrairement au web actuel qui est vu comme un web syntaxique de rendre le contenu sémantique des ressources web interprétable non seulement par l'homme, mais aussi par la machine.

Nous commençons ce chapitre par l'apport du web sémantique à la recherche d'information traditionnelle, puis nous parlerons du besoin de l'utilisation d'ontologies afin de réduire les problèmes de représentation de connaissances. Ensuite, nous étudierons les langages de représentations et d'interrogations associées à ce nouveau web en précisant les limites et les avantages de chacun, ainsi que le raisonnement sémantique. Finalement, la dernière section va être plus spécifiquement consacrée aux différents systèmes d'interrogation d'ontologies.

2. L'apport du web sémantique à la recherche d'information traditionnelle

Le web sémantique est un web intelligent qui serait non ambiguë et compréhensible par la machine, où les informations seraient mieux traitées, donc plus faciles à optimiser la recherche d'information avec la possibilité de mener des requêtes complexes. Afin d'augmenter les performances des systèmes de recherche d'information classiques, le web sémantique doit permettre d'abord de **[LAUB 02]** :

- localiser, d'identifier et de transformer des ressources de manière robuste et saine tout en renforçant l'esprit d'ouverture du web.
- Elle doit contribuer à assurer, le plus automatiquement possible, l'interopérabilité et les transformations entre les différents formalismes et les différentes ontologies.
- Elle doit faciliter la mise en oeuvre des calculs et des raisonnements complexes tout en offrant des garanties supérieures sur leur validité.
- Elle doit s'appuyer sur un certain niveau de consensus portant, par exemple, sur les langages de représentation ou sur les ontologies utilisées.
- Elle doit fournir l'accès simple aux dépôts de connaissance, ce qui permet à l'utilisateur d'interroger et de raisonner sur les ontologies utilisant une requête écrite en langage naturel.

3. Les ontologies

La variété des sources d'information sur le web (textes, images, etc.) plaide pour un traitement de l'information qui soit indépendant des formes sous lesquelles elle

est stockée. Dans ce cadre, les ontologies vont servir à l'interprétation de ces connaissances en spécifiant la sémantique de la représentation utilisée.

3.1. Définition et rôles des ontologies

Parmi les conséquences d'une absence de compréhension partagées sur le web on a: la mauvaise communication, la difficulté de l'interprétation de requêtes, l'interopérabilité limité, et le potentiel limité de la réutilisation et du partage de connaissances. Ainsi, il est nécessaire de réduire ou d'éliminer la confusion conceptuelle et terminologique et de parvenir à une compréhension commune (les ontologies).

Il est difficile de définir ce qu'est une ontologie d'une façon définitive. Le mot est en effet employé dans des contextes très différents touchant la philosophie, la linguistique ou l'intelligence artificielle.

Pour Uschold et Gruninger **[USCH 96]**, une ontologie est une structure commune pour les différents points de vue. Elle est basée sur la communication (entre personnes, entre les personnes et les systèmes, entre systèmes).

L'ontologie est l'étude de la nature de connaissances et de leur justification **[SOWA 99]**. Elle se base sur la structure de connaissances pour en produire de nouvelles. En conséquence, l'ontologie au sens philosophique a été une source d'inspiration pour l'acquisition, la représentation théorique et le partage de connaissances **[CHAN 99]**, ainsi que pour le traitement du langage naturel.

Selon **[WETL 01]**, une ontologie est indépendante des contraintes d'exécution, son but étant de spécifier la conceptualisation du monde sous-jacente à l'application. Il rejoint la définition de Gruber, pour qui une ontologie est une spécification explicite d'une conceptualisation **[GRUB 93]**.

En résumé, l'ontologie est un nouvel objet de l'intelligence artificielle, qui a récemment atteint sa maturité en devenant un outil conceptuel puissant pour la modélisation de connaissances **[GAND 02]**. L'ontologie fournit une base cohérente sur laquelle construire, et une référence partagée sur laquelle s'aligner,

sous forme d'un vocabulaire conceptuel consensuel avec lequel on peut décrire et communiquer.

Ainsi, le développement d'une ontologie dans un domaine donné doit permettre d' :

- ✓ Améliorer la communication, qui à son tour, peut donner lieu à une plus grande réutilisation, du partage de connaissances, de l'interopérabilité.
- ✓ Lever l'ambiguïté. L'un des rôles principaux de l'ontologie est la désambiguïsation, en fournissant un terrain sémantique, un consensus vocabulaire conceptuel, sur lequel on peut construire des descriptions et des actes de communication.
- ✓ Extraire une meilleure interprétation de la requête de l'utilisateur en représentant les termes écrits en langage naturel sous forme de concepts et de rôles de l'ontologie **[TODI 01]**.

3.2. Le cycle de vie de l'ontologie

Les ontologies étant destinées à être utilisées comme des composants logiciels dans des systèmes répondant à des objectifs opérationnels différents, leur développement doit s'appuyer sur les mêmes principes que ceux appliqués en génie logiciel. En particulier, les ontologies doivent être considérées comme des objets techniques évolutifs et possédants un cycle de vie qui nécessite d'être spécifié. Les activités liées aux ontologies sont d'une part des activités de gestion de projet (planification, contrôle, assurance qualité), et d'autre part des activités de développement (spécification, conceptualisation, formalisation) ; s'y ajoutent des activités transversales de support telles que l'évaluation, la gestion de la configuration **[BLAZ 98]**.

Bien qu'aucune méthodologie générale n'ait pour l'instant réussi à s'imposer, de nombreux principes et critères de construction d'ontologies ont été proposés. Mais quelque soit la méthodologie adoptée, le processus de construction d'une ontologie est une collaboration qui réunit des experts du domaine de connaissance, des ingénieurs de la connaissance. Cette collaboration ne peut être

fructueuse que si les objectifs du processus ont été clairement définis, ainsi que les besoins qui en découlent.

Une fois le but défini, le processus de construction de l'ontologie peut démarrer, en commençant par la phase de conceptualisation. Donc le processus général de représentation des connaissances peut être découpé en trois phases (**Fig.2**) :

1. ***La conceptualisation*** *:* identification des connaissances contenues dans un corpus représentatif du domaine. Ce travail doit être mené par un expert du domaine, assisté par un ingénieur de la connaissance.
2. ***L'ontologisation*** **:** formalisation, autant que possible, du modèle conceptuel obtenu à l'étape précédente. Ce travail doit être mené par l'ingénieur de la connaissance, assisté de l'expert du domaine.
3. ***L'opérationnalisation*** **:** transcription de l'ontologie dans un langage formel et opérationnel de représentation de connaissances. Ce travail doit être mené par l'ingénieur de la connaissance.

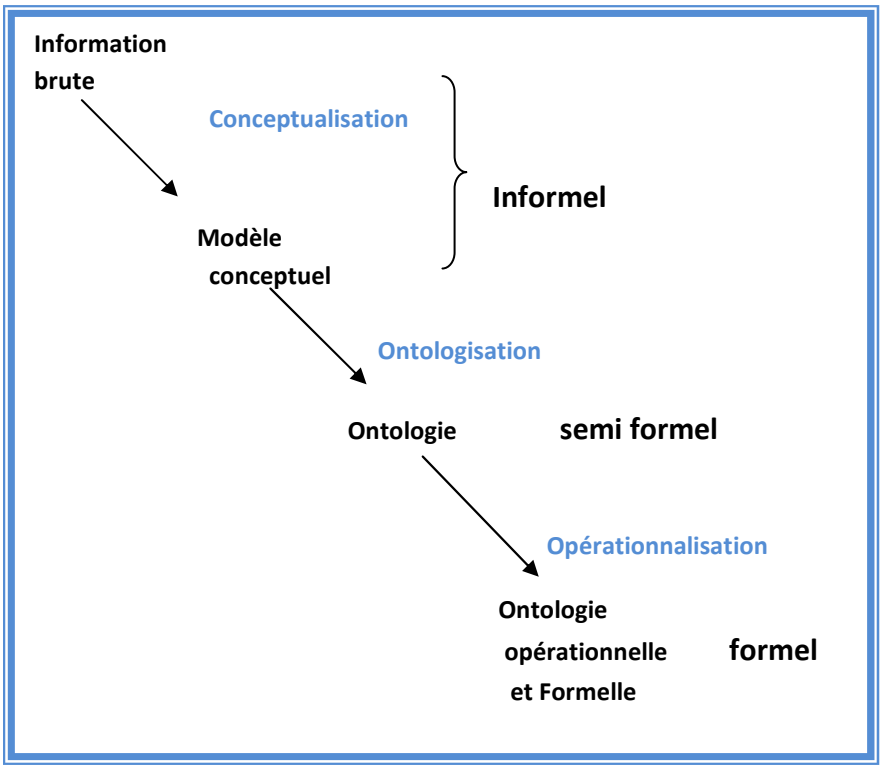

Fig. 2 : Construction d'une ontologie opérationnelle

Il est à noter que ce processus n'est pas linéaire et que de nombreux aller retours sont a priori nécessaires pour bâtir une ontologie opérationnelle adaptée aux besoins.

4. Langages de représentations et d'interrogations de connaissances

Depuis plusieurs années, le web sémantique est considéré comme la prochaine évolution possible pour l'accès à la connaissance en réseau. Toutefois, une des difficultés majeures pour le développement du web sémantique est son besoin en connaissances structurées. C'est pourquoi, une grande partie des recherches se concentre sur la formalisation des ontologies ce qui a, entre autre, donné naissance à des différents langages de niveau de complexité croissante (**fig.3**) qui sont proposés afin de mieux exploiter, combiner, interroger et raisonner sur les contenus des ressources de web.

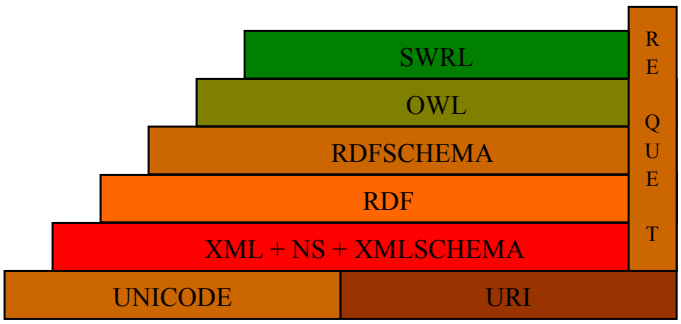

Fig.3 : Les couches du web sémantique

4.1. Documents structurés en XML et schéma XML

Le langage XML (eXtended Markup Language) permet de décrire la structure arborescente de documents à l'aide d'un système de balises. Ce dernier permet de marquer les éléments qui composent la structure et les relations entre ces éléments, pour cela le document XML est beaucoup plus facile accessibles aux machines. D'autre part, il permet de séparer le contenu au format du document ; c'est à dire la même information peut être affichée de différentes façons, sans nécessiter de multiples copies du même contenu **[ANTO 08]**.

Un schéma XML est un langage formel des contraintes, exprimées comme des règles ou un modèle de structure, qui s'appliquent à une classe de documents XML **[ERIC 02]**. Il respecte une syntaxe XML : réutilisation de la technologie XML (analyseurs, éditeurs, ...). Avec les schémas XML, il est possible de contraindre la structure arborescente d'un document mais pas la sémantique des informations contenues dans ce document.

Le schéma XML permet notamment de vérifier la validité de document XML. Chaque schéma est associé un nom symbolique (URI : Uniform Ressource Identifier). Les balises issues du schéma sont préfixées par cette URI, appelée Namespace. Les espaces de nom (Namespace) sont introduits dans le format de représentation pour permettre la modularité des schémas.

4.2. L'interrogation des documents XML

Dans les bases de données relationnelles, des parties d'une base de données peuvent être sélectionnées et extraites à l'aide des langages d'interrogations telles que SQL. De même pour les documents XML, il existe plusieurs propositions de langages d'interrogations (XQuery, XPATH, XQL, XML-QL…). Le concept central de ces langages est l'expression de chemin qui indique comment on peut atteindre un noeud dans la représentation arborescente d'un document XML.

4.2.1. XPATH

XPATH est un langage qui permet d'adresser une partie de document XML non seulement avec l'objectif d'interroger, mais aussi pour pouvoir transformer un document XML **[WATT 02]**. Il fonctionne sur l'arbre du modèle de données XML. Les concepts clés sont des expressions.

- ☞ *Exemple :*

```
<Zoo lieu=’alger’>
  <Animal nom=’tigre’>
     <Environnement=’ jungle’/>
     <Couvert =’poils’/>
  </Animal>
  <Animal nom=’poisson’>
    <Environnement=’ aquatique’/>
    <Couvert =’écailles’/>
  </Animal>
```

Requête 1 : Liste des noeuds animaux sous le noeud zoo

/zoo/animal

Requête 2 : Liste des noeuds animaux dans l’arbre XML

//animal

Requête 3 : Liste des attributs lieu des noeuds zoo

/zoo/@lieu

Cependant, ce langage possède quelques limitations. Par exemple :

- impossible d'exprimer une corrélation entre des données situées sur des chemins différents.
- pas de jointure
- il ne peut retourner que des noeuds existants.
- pas de re-formulation, combinaison de plusieurs documents.
- langage d'adressage non pas langage de requêtes (requête: sélection, corrélation, reconstruction).

4.2.2. XQuery

XQuery est un langage de requête conçu par le W3C (World Wide Web Consortium). Il permet de sélectionner les éléments d'intérêt des données XML, les réorganiser, transformer, et retourner les résultats **[WALM 07]**. XQuery a un ensemble de fonctionnalités riche qui permettent à exécuter nombreux types d'opérations sur les données et les documents XML :

- ✓ Sélection d'information en fonction des critères spécifiques.
- ✓ Filtrage de l'information indésirable.
- ✓ Recherche de l'information dans un document ou un ensemble de documents.
- ✓ Transformation et restructuration des données XML dans un autre vocabulaire ou structure XML.
- ✓ Effectuer des calculs arithmétiques sur les nombres et les dates.
- ✓ Manipulation des chaînes pour reformater le texte.

Cependant, XQuery est un langage complexe à mettre en œuvre à cause de la complexité des structures manipulées (les arbres XML), et la semi structuration de ces données (complexité du typage).

4.3. Description de ressources en RDF et RDFS

XML permet de représenter des données structurées. Mais, la sémantique des données représentées en XML n'est pas rendue accessible à la machine (XML ne permet de spécifier ni la sémantique d'une balise, ni le typage de son contenu, ni des relations normalisées entre les balises).

Le W3C a adopté le langage RDF (Resource Description Framework) comme formalisme standard de représentations. Le langage RDF permet de voir le web comme un ensemble de ressources reliées par des liens étiquetés sémantiquement. Donc le document RDF est un ensemble de triplets (statements) Ressource, Propriété, Valeur :

- Une ressource peut être une page web, une partie d'une page, un ensemble de pages, un objet ou toute entité qui peut être accédée par un identificateur (Uniform Resource Identifier).
- Une propriété est également une ressource qui a un rôle spécial. Elle permet de décrire une autre ressource.
- Une valeur peut être un littéral ou une ressource.

Un schéma RDF (RDFS) permet de définir des classes et une hiérarchie de spécialisation sur les classes. Il permet également de définir des propriétés et une hiérarchie de spécialisation sur les propriétés, et de définir des restrictions sur la valeur d'une propriété (co_domaine) et sur le type de ressource décrit par la propriété (domaine).

Voici d'autres notions définies dans RDFS :

- La notion de classe (rdfs :Class) qui peut être rapprochée de la notion de concept d'une ontologie.
- La notion de sous-classe (rdfs :subClassOf) qui est un spécialisation d'une classe déjà définie.
- La notion de type (rdf :Type) pour les instances d'une classe.
- Les notions de source (rdfs :domain) et de cible (rdfs :range) d'une propriété.
- La notion de sous propriété (rdfs:subPropertyOf) qui permet d'associer une propriété à l'une de ses super propriétés Classes.

4.4. Interrogation des méta-données RDF

Plusieurs propositions de langages d'interrogation de documents RDF existent (RQL, SquishQL, RDQL, SPARQL, ...) :

4.4.1. RDQL

RDQL (RDF Data Query Language **[SEAB 04]**) est un langage d'interrogation de données définies en RDF. Ce langage n'est pas standardisé. Sa syntaxe est très proche de SQL :

SELECT variable [, variable]*
FROM documents rdf [, documents rdf]*
WHERE modèle de triplets
AND restrictions booléennes
USING définition des raccourcis

– La clause SELECT définit la liste des variables que l'on désire obtenir. Une variable est composée de caractères alphanumériques avec le '_' et commence par un '?'. Les variables d'une requête sont séparées par un espace

(et/ou) une virgule. La valeur * signifie que l'on désire obtenir l'ensemble des variables utilisées dans la requête.

– La clause FROM définit l'emplacement des documents RDF utilisés pour la requête.

– La clause WHERE définit les triplets RDF (sujet - prédicat - objet), les éléments de ce triplet sont décrits soit par les valeurs de l'ontologie interrogée soit par des variables. Plusieurs triplets peuvent être définis dans une même clause WHERE, cette succession de triplets indiquant la conjonction de chacun des termes.

– La clause AND définit les restrictions booléennes de la requête. Une restriction booléenne est constituée de valeurs ou variables composées à l'aide d'opérateurs :

– opérations numériques : +, −, *, /

– conjonction, disjonction : || ||, &&

– comparateurs sur les numériques : <,>,<=,>=,==

– comparateurs sur les chaînes de caractères : EQ pour l'égalité, NE pour la différence.

– La clause USING permet l'utilisation de simplification de nommage à l'aide d'alias

A titre d'exemple, la requête RDQL pour rechercher un mammifère avec le nom éléphant :

```
SELECT ?name ?class
FROM <http ://www.LaboLire.dz/AnimalClassification/animal.rdf>
WHERE ( ?class, Animal:Class, ‘‘Mammal’’)
AND (?name EQ ‘‘elephant’’)
USING Animal FOR <http ://www.w3.org/2001/Animal-rdf/3.0/
```

4.4.2. SPARQL

SPARQL (SPARQL Protocol And RDF Query Language **[PRUD 08]**) est une amélioration de RDQL. Ce langage est en cours de standardisation au niveau du W3C. Il permet d'interroger un ensemble de descriptions RDF à partir d'opérations de mise en correspondance de patterns de graphes, de

conjonctions et de disjonctions de patterns et de patterns optionnels (L'opérateur OPTIONAL définit des triplets RDF facultatifs pour le résultat de la requête).

4.4.3. RQL

RQL **[KARV 02]** est un langage déclaratif qui décrit le type de données désiré et non pas comment on accède à ces données. RDF permet de décrire une ressource selon différents points de vue (chaque point de vue correspond à un RDF schéma). RQL exploite cette richesse de RDF. Dans la suite, nous donnons les requêtes de base du langage RQL :

- Class et Property ; renvoient respectivement la liste de toutes les classes et de toutes les propriétés.

- subClassOf() et subClassOf^() ; renvoient respectivement l'ensemble des sous-classes et les sous-classes directes.

-subPropertyOf() et subPropertyOf^() ; renvoient respectivement l'ensemble des sous-propriétés et les sous- propriétés directes.

-domain(), range() ; renvoient respectivement les restrictions définies pour une propriété donnée.

4.5. Limitation des descriptions RDF et RDFS

L'expressivité de RDF et RDFS est trop limitée pour représenter la sémantique portée par les cas d'utilisations (use cases) identifiés par le W3C. Les limitations constatées sont :

- rdfs:range définit le domaine de valeurs d'une propriété quelque soit la classe concernée. Par exemple, on ne peut pas exprimer que les vaches ne mangent que de l'herbe alors que d'autres sortes d'animaux mangent également de la viande.

- RDFS ne permet pas d'exprimer que deux classes sont disjointes. Par exemple, les classes des hommes et des femmes sont disjointes.

- RDFS ne permet pas de créer des classes par combinaison ensembliste d'autres classes (intersection, union, complément). Par exemple, on veut construire la classe personne comme l'union disjointe des classes des hommes et des femmes.

- RDFS ne permet pas de définir de restriction sur le nombre d'occurrences de valeurs que peut prendre une propriété. Par exemple, on ne peut pas dire qu'une personne a exactement deux parents.

- Enfin, RDFS ne permet pas de définir certaines caractéristiques des propriétés comme la transitivité (par exemple : estPlusGrandQue), l'unicité (par exemple : estLePèreDe), la propriété inverse (par exemple : mange est la propriété inverse de estMangéPar).

4.6. Description d'ontologies en OWL

Pour représenter les ontologies, le W3C a proposé un standard, connu actuellement sous le nom d'OWL (Ontology Web Language). Ce dernier est conçu pour les applications ayant besoin de traiter le contenu des informations au lieu de les présenter simplement aux humains. Le langage OWL permet une interprétation du contenu web par les machines supérieure à celle offerte par les langages XML, RDF et le schéma RDF (RDFS), en fournissant un vocabulaire supplémentaire avec une sémantique formelle **[LACO 05]**. Si RDF et RDFS apportent à l'utilisateur la capacité de décrire des classes et des propriétés, OWL intègre, en plus, des outils de comparaison des propriétés et des classes : identité, équivalence, contraire, cardinalité, symétrie, transitivité, disjonction, etc.

OWL se décompose en trois niveaux de langage, d'expressivité décroissante, ont été définis (**Fig.4**) :

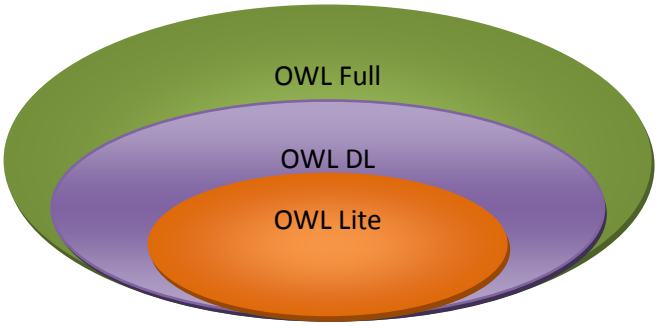

Fig.4 : Les 3 niveaux d'OWL.

· ***OWL Full****:* inclut toutes les primitives OWL qui peuvent être combinées avec toutes les primitives RDF et RDFS. Il a l'avantage de la compatibilité complète avec RDF/RDFS, mais l'inconvénient d'avoir un niveau d'expressivité qui le rend indécidable.

· ***OWL DL (Description Logic):*** définit des restrictions sur la manière dont les primitives OWL et RDF/RDFS peuvent être combinées. Il est fondé sur la logique de description SHIN(D) (d'où son nom OWL Description Logics). Malgré sa complexité, OWL DL garantit la complétude des raisonnements (calculabilité des inférences) et leur décidabilité (leur calcul se fait en une durée finie).

Il a l'inconvénient de ne pas être complètement compatible avec RDF. Un document RDF devra donc parfois être étendu sur certains aspects ou restreint sur d'autres pour être un document OWL-DL légal.

· ***OWL Lite:*** est une version limitée d'OWL DL qui en facilite son utilisation et son implémentation. Par exemple, OWL Lite n'inclut pas la disjonction entre classes et les cardinalités arbitraires.

OWL Lite est le sous langage d'OWL le plus simple. Il est destiné aux utilisateurs qui ont besoin d'une hiérarchie de concepts simple. OWL Lite est adapté, par exemple, aux migrations rapides depuis d'anciens thésaurus **[MCGU 04].**

Il existe entre ces trois sous langage une dépendance de nature hiérarchique ; toute ontologie OWL Lite valide est également une ontologie OWL DL valide, et toute ontologie OWL DL valide est également une ontologie OWL Full valide.

4.7. Interrogation d'ontologies OWL

Pour l'interrogation d'ontologies OWL**,** plusieurs langages d'interrogations sont proposés :

4.7.1. OWL-QL

OWL-QL est un langage formel, précise la relation sémantique entre la requête, la réponse, et la base de connaissances utilisée pour produire cette réponse. Candidat à la standardisation W3C en tant que :

a) Langage d'interrogation du web sémantique.

b) Protocole de dialogue requête-réponse entre agents sur le web sémantique.

OWL-QL doit pouvoir supporter des scénarii dialogues requête-réponses dans lesquels les agents serveurs de réponses utilisent des méthodes automatisées de raisonnement pour dériver des réponses aux requêtes.

Une requête OWL-QL est un objet contenant obligatoirement un pattern de requête qui spécifie une collection d'instructions OWL incluant des URI considérées comme variables.

Chaque variable apparaissant dans une requête OWL-QL peut être considérée comme une must-bind variable (réponse obligatoire), une may-bind variable (réponse optionnelle) ou une don't bind variable (réponse non renvoyée).

Par exemple, on interroge une base de connaissance contenant les informations suivantes :

· Chaque personne a exactement un père.
· Ali est une personne.
· Mohamed est une personne.
· Rachid est une personne.

· Rachid est le père de Ali.

Soit le Query pattern : {(hasFather ?p ?f)}

· Si ?f est une don't-bind variable : la réponse contient 3 réponses = {(père de Ali, père de Mohamed, père de Rachid}.

· Si ?f est une must-bind variable : la réponse contient 1 réponse = {(Rachid est le père de Ali }.

· Si ?f est une may-bind variable : la réponse contient 3 réponses = {(Rachid est le père de Ali, père de Mohamed, père de Rachid}.

4.7.2. *nRQL*

nRQL (new Racerpro query Language) est le langage d'interrogation spécifique de moteur d'inférence Racer. Selon **[WESS 05]** le langage nRQL peut être caractérisé par:

1. il permet d'interroger les ontologies sur le niveau terminologie (TBOX) et le niveau assersionnel (ABOX).

2. les variables et les individus peuvent être utilisés dans les requêtes.

3. différents types d'atomes des requêtes sont disponibles: les atomes des requêtes concept, les atomes des requêtes rôle, les atomes des requêtes contrainte, les atomes de la requête SAME-AS, et des requêtes auxiliaires.

Par exemples :

(?x |woman|) est un atome de la requête concept.

(?x ?y |has-child|) est un atome de la requête rôle.

(?x ?x(constraint (|has-father| |age|) (|has-mother| |age|) =)) est un atome de la requête contrainte (demander les personnes dont les parents ont le même âge).

Et (same-as ?x |Amira|) est un atome de la requête SAME-AS.

4. les requêtes complexes, sont construites à partir d'atomes des requêtes en utilisant les constructeurs booléens : AND, OR, NEG. Ces constructeurs

peuvent être combinés de manière arbitraire pour créer des requêtes complexes.

5. nRQL est un support spécial pour interroger la partie concrète du domaine (attributs). Cela permet d'introduire des conditions spécifiques sur les attributs dans les requêtes. Par exemple, nous pouvons utiliser la requête suivante pour rechercher tous les individus dont l'âge est supérieur à 20 ans : (retrieve (?x) (?x (> age 20))).

6. nRQL offre également des requêtes définies. Par exemple : (defquery mother (?x) (and (?x woman)(?x ?y has-child))) peut être utilisé pour définir une requête mother qui peut être ensuite utilisée dans les autres requêtes (par exemple dans la requête : (retrieve (?x) (?x mother))).

7. Un langage de requêtes puissant pour interroger les documents RDF et OWL. De plus, il est le plus puissant par rapport aux autres langages d'interrogations (offre l'utilisation de la négation d'un rôle, les conditions spécifiques sur les attributs…).

La syntaxe et la sémantique de langage nRQL sont décrites en détail dans **[RACE 05]**.

5. Raisonnement sémantique

Avant de découvrir le raisonnement sémantique, il faut d'abord connaître les logiques de description (LD) qui sont actuellement le modèle naturel de représentation et de raisonnement ontologique au sein du web sémantique (OWL DL). Elles sont issues d'un ensemble des recherches et d'études effectuées sur la logique des prédicats, les réseaux sémantiques et les langages de frames.

Les logiques de description **[NAPO 97]** sont un des formalismes qui permettant une représentation formelle des bases de connaissances terminologiques (les *ontologies*). Elles sont différentes de leurs prédécesseurs, tels que réseaux sémantiques et frames, étant donné qu'elles sont équipées d'une logique formelle basée sur des sémantiques. Les logiques de description jouent également un rôle important dans les domaines qui comprennent le traitement du langage naturel

(peut être utilisée pour communiquer le sens des phrases) et la gestion des bases de données.

Dans le formalisme des logiques de descriptions, les concepts génériques permettent de décrire les éléments complexes du domaine (une classe d'individus partageant un ensemble de propriétés communes) et les concepts individuels représentent les individus ou faits et peuvent être reliés à un ou plusieurs concepts génériques par un lien 'is-a' qui marque l'appartenance d'un individu, tandis que les rôles décrivent une relation binaire entre un concept auquel est attaché le rôle et un concept valeur (co-domaine).

La modélisation des connaissances d'un domaine avec les DL se réalise en deux niveaux. Le premier est le niveau terminologique ou **TBox**, décrit les connaissances générales d'un domaine. Alors que le second est le niveau assertionnel ou **ABox**, représente une instanciation spécifique. Donc, une TBox comprend la définition des concepts et des rôles, alors qu'une ABox contient un ensemble d'assertions sur les individus.

Une fois l'ontologie chargée, l'inférence s'effectue au niveau terminologique (TBox) ou assertionel (ABox).

Au niveau terminologique quatre principaux problèmes d'inférence se présentent [BAAR 03] :

- **Satisfiabilité** : Un concept C d'une terminologie $\mathcal{T}$ est satisfiable si et seulement existe un modèle $\mathcal{I}$ de $\mathcal{T}$ tel que $C^{\mathcal{I}} \neq \emptyset$.
- **Subsomption** : Un concept C est subsumé par un concept D pour une terminologie si et seulement si $C^{\mathcal{I}} \sqsubseteq D^{\mathcal{I}}$ pour tout modèle $\mathcal{I}$ de $\mathcal{T}$.
- **Équivalence** : Un concept C est équivalent à un concept D pour une terminologie et seulement si $C^{\mathcal{I}} \equiv D^{\mathcal{I}}$ pour chaque modèle $\mathcal{I}$ de $\mathcal{T}$.
- **Disjonction (disjointness)** : Des concepts C et D sont disjoints par rapport terminologie $\mathcal{T}$ si et seulement si $C^{\mathcal{I}} \setminus D^{\mathcal{I}} = \emptyset$; pour chaque modèle $\mathcal{I}$ de $\mathcal{T}$.

Ces quatre types de problèmes sont restreints par le moteur d'inférence à des problèmes de subsomption ou de satisfiabilité traités par des algorithmes NC (normalisation / comparaison) ou avec les tableaux sémantiques.

Les problèmes d'inférence d'une TBox peuvent être réduits aux problèmes de subsomption :

C est insatisfiable	$\Longleftrightarrow$	C est subsumé par $\bot$
C et D sont équivalents	$\Longleftrightarrow$	C est subsumé par D, et D par C
C et D sont disjoints	$\Longleftrightarrow$	$C \sqcap D$ est subsumé par $\bot$

Les problèmes d'inférence d'une TBox peuvent être réduits aux problèmes de satisfiabilité :

C est subsumé par D	$\Longleftrightarrow$	$C \sqcap \neg D$ est insatisfiable
C et D sont équivalents	$\Longleftrightarrow$	$C \sqcap \neg D$ et $\neg C \sqcap D$ sont insatisfiables
C et D sont disjoints	$\Longleftrightarrow$	$C \sqcap D$ est insatisfiable

Finalement, la subsomption peut être réduite à un problème de satisfiabilité et réciproquement.

$$C \cap \neg D \text{ est non satisfiable} \Leftrightarrow C \subseteq D$$

$$\neg C \subseteq \bot \Rightarrow C \text{ est satisfiable}$$

Le niveau factuel comprend quatre principaux problèmes d'inférence **[BAAD 03]**:

- **Cohérence** : Une ABox A est cohérente par rapport à une TBox $\mathcal{T}$ si et seulement s'il existe un modèle $\mathcal{I}$ de A et $\mathcal{T}$.
- **Vérification d'instance** : Vérifier par inférence si une assertion C(a) est vraie pour tout modèle $\mathcal{I}$ d'une ABox A et d'une TBox $\mathcal{T}$.
- **Vérification de rôle** : Vérifier par inférence si une assertion R(a, b) est vraie pour tout modèle $\mathcal{I}$ d'une ABox A et d'une TBox $\mathcal{T}$.
- **Problème de récupération** : Pour une ABox A, un concept C d'une terminologie $\mathcal{T}$, inférer les individus $a^{\mathcal{I}}1 \ldots a^{\mathcal{I}}n \in CI$ pour tout modèle $\mathcal{I}$ de $\mathcal{T}$.

Il existe de très nombreux moteurs d'inférences (Racer, Pellet, FaCT, FaCT++, Flora, Jena, Jess, Bossam...). La plupart de ces moteurs sont conçus pour raisonner sur les logiques de description, mais acceptent en entrée des fichiers OWL. Certains moteurs d'inférence ne peuvent raisonner qu'au niveau terminologique (c'est-à-dire au niveau des concepts et des propriétés tel que FaCT et FaCT++ qui se spécialisent en raisonnant sur des TBox seulement) alors que des moteurs comme Pellet et Racer

permettant le raisonnement sur la ABox et la TBox et exploitent des ontologies possédant un niveau d'expressivité en logique de description et en OWL satisfaisant.

5.1. Pellet

Est le moteur le plus récent. C'est un des projets du MINDSWAP Group, un groupe de recherche sur le web sémantique de l'université du Maryland. Il est disponible en OpenSource et offre des évolutions fréquentes (encore en développement). Pellet travaille sur des ontologies décrites en RDF ou OWL et permet les requêtes avec RDQL et SPARQL sur les deux niveaux ABox et TBox. Bien qu'il offre davantage des services pour raisonner et modifier une base de connaissances, Pellet échoue à atteindre le niveau de performance de FaCT et Racer.

5.2. Racer (Renamed Abox and Concept Expression Reasoner)

Le moteur d'inférence sans doute le plus connu, commercialisé par *Racer Systems GmbH & Co. KG*, fondé en 2004 par des chercheurs qui travaillaient à l'université de Hambourg, travaille sur les ontologies modélisées par son langage, mais accepte des ontologies décrites en RDF ou OWL, ces dernières étant traduites vers le langage utilisé par Racer. Ce moteur d'inférence possède également son propre langage de requête nRQL (new Racerpro query Language) pour interroger les ontologies sur les deux niveaux ABox et TBox.

Dans suite, nous définissons quelques services d'inférence offerts par RACER **[HAAR 01]** :

- ✓ Permet de calculer la relation de subsomption entre les concepts. Cette inférence est nécessaire pour construire une hiérarchie de concept (classification de la TBox).
- ✓ permet de vérifier si un nom de concept est contradictoire. Habituellement, un concept incompatible est la conséquence d'erreurs de modélisation. Vérification de la cohérence de touts les concepts TBox.
- ✓ Prise en charge des requêtes auxiliaires telle que : la recherche du concept ou l'instance dans une base de connaissances, la recherche du

rôle des pères et des fils, l'extraction de l'ensemble des instances en le domaine et dans la portée d'un rôle.

Cependant, les moteurs d'inférence existants souffrent de plusieurs faiblesses, en particulier :

- la capacité d'inférence limitée : tous les moteurs n'offrent que les services d'inférence standard, à savoir : les services de subsomption, de satisfiabilité et de vérification d'instance qui sont insuffisants pour répondre aux applications réelles.
- le manque des fonctionnalités de SGBC, surtout, la capacité de gestion des connaissances telles que le stockage et la maintenance des connaissances.

6. Travaux existants sur l'interrogation des ontologies

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**[ANDR 95]**. Cependant, le problème de l'interrogation des bases de connaissance par des requêtes en langue naturelle n'a été posée sérieusement que dans ces dernières années. Ainsi, dans le cadre du web sémantique, un certain nombre de systèmes fournissant l'accès aux ontologies a été créé :

Le système AquaLog **[LOPE 05]** 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 de 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. Enfin les réponses sont extraites à partir de ces triplets.

La nouvelle version de AquaLog est PowerAqua **[LOPE 06]** 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 traduisant 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 **[KAUF 06]** 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? " a 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 **[LEI 06]** est un système basé concept qui permet de 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 **[KOSS 06]** est un système d'interaction d'ontologies 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 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) **[BAKE 06]** a 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 **[WANG 07]** 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 **[TABL 08]** 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 des langues naturelles, 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 des langues naturelles utilisées. La plupart des systèmes ont supposé que toutes les requêtes peuvent être écrites sous forme de triplets linguistiques <terme, relation, terme>, or le cas où la relation n'est pas citée dans la requêtes est très fréquent parce que l'utilisateur en général saisit une séquence de mots, et il n'est pas exigé de donner la relation entre eux.

7. Conclusion

Malgré l'apport du web sémantique sur les techniques de recherche d'information traductionnelles, on ne tente pas encore de combler le fossé entre la machine et l'humain. Le moteur de recherche doit toujours recevoir une requête exprimée dans un langage formel (c'est-à-dire possédant une syntaxe et une sémantique définies à l'avance et non ambiguës, par opposition au langage naturel). L'idée est plutôt de construire un système, capable de traduire du mieux possible les requêtes en langage naturel vers un langage formel. Les recherches dans ce domaine ont été intensives pour l'interrogation des bases de connaissances.

Dans le prochain chapitre de ce mémoire, nous allons présenter l'architecture d'un système de conversion des requêtes en langue naturel vers le langage formel nRQL, en utilisant les restrictions sémantiques imposées par l'ontologie.

Chapitre III

Conception d'un système de conversion des requêtes en langage naturel vers nRQL

« L'un des problèmes de l'Intelligence Artificielle est la représentation des connaissances et la conversion de ces dernières en des formalismes capables d'être interprétés par un ordinateur»

Berard Dugourd

1. Introduction

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.

Les ontologies sont souvent riches de plusieurs milliers de concepts et ne restent alors directement appréhendables que par leur concepteur. Leur accès par des utilisateurs, même professionnels, nécessite de gérer le lien entre les concepts des ontologies et les termes du langage naturel, que ce soit pour une simple compréhension ou pour l'indexation et la construction de requêtes destinées à des tâches de recherche d'information.

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 ce chapitre, nous proposons l'architecture d'un système de conversion des requêtes en langage naturel vers nRQL (new Racerpro query Language) qui est le langage d'interrogation spécifique au raisonneur RACER. Ce système permet à l'utilisateur d'interroger une base de connaissances sans avoir une connaissance préalable sur la structure de l'ontologie. Cela confére 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.

2. Spécification des besoins

Dans les systèmes de recherche d'information classique, le besoin en information de l'utilisateur 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 de 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 silense), dans le cas où un même concept est référencé par deux termes différents dans la requête et dans les documents.

Pour pallier ces problèmes, les ontologies sont utilisées afin d'aider l'utilisateur dans la spécification de son besoin en information ce qui permet de mieux interpréter la sémantique de la requête. Le sens de la requête initiale est alors spécifié soit par l'ajout de nouveaux termes en relation avec les concepts référés par les termes initiaux, soit par la désambiguïsation des termes de la requête.

Bien que l'indexation utilisant une ontologie présente quelques inconvénients comme par exemple : la définition d'une ontologie (concepts et relations entre concepts) est un travail difficile et de longue haleine, les avantages d'une telle indexation sont multiples :

- ✓ permettre une représentation fidèle du contenu sémantique des documents.
- ✓ faciliter la recherche d'information au sein de collections hétérogènes en indexant tout type de document à partir des mêmes concepts.
- ✓ permettre une recherche intelligente, et cela en exploitant la sémantique des liens entre concepts d'indexation, par des mécanismes d'inférence (qui permettent des raisonnements élaborés).

Les requêtes dans notre système sont formulées en langage naturel, nous devons par conséquent mettre les fonctionnalités nécessaires pour les traiter. La recherche ne se limite pas à trouver des ressources référencées par des mots clés, mais identifie la sémantique des mots d'une requête afin d'accéder aux informations les plus pertinentes.

L'utilisateur doit entrer sa requête avec le nom de l'ontologie, et le système doit représenter tous les termes de la requête sous forme de concepts et de rôles de l'ontologie choisie par l'utilisateur.

Le système doit trouver les liaisons sémantiques entre tous les composants de la requête. Ainsi, il doit générer la requête en langage nRQL (new Racerpro query Language) qui est le langage d'interrogation spécifique au raisonneur RACER. La requête nRQL générée doit être correcte et conforme avec la requête en langage naturel initiale afin d'augmenter la performance du système de recherche.

3. Objectif de travail

L'objectif principal de notre travail consiste à faciliter l'interrogation de la base de connaissances par la proposition d'une approche **[BOUMa 09]** pour la conversion des requêtes en langage naturel vers nRQL, en utilisant les restrictions sémantiques imposées par l'ontologie. L'aboutissement d'un tel objectif est le fruit des sous objectifs suivants :

- Description de l'architecture globale du système de conversion des requêtes en langage naturel vers nRQL, ainsi que les interactions entre les différents composants de cette architecture.

- L'implémentation d'une ontologie de domaine destinée à la modélisation des connaissances du domaine. l'ontologie que nous avons développée pour valider notre proposition.
- Le développement d'une interface qui facilite l'interaction entre l'utilisateur (saisit la requête), et le système (affiche le résultat de conversion).
- L'implémentation d'un outil de conversion des requêtes qui prend en entrée une ontologie et une requête en langage naturel et renvoie la requête nRQL en sortie. Ainsi, cet outil permet de décomposer la requête, supprimer les mots vides, charger l'ontologie, trouver pour chaque composant ses correspondant dans l'ontologie, définir les types des composants par rapport à l'ontologie (classe, propriété, instance), extraire les triplets de la requête, trouver les relations sémantiques entre les composants de chaque triplet par rapport aux relations binaires de l'ontologie, conjonction des triplets de la requête, suppression des redondances, et enfin la génération de la requête nRQL finale.

4. L'architecture du système

L'architecture du système proposé dans ce travail supporte deux phases de traitement (**Fig.5**). La première est la phase d'analyse linguistique des requêtes qui permet d'abord de segmenter la requête entrée par l'utilisateur pour identifier ses différents composants. Puis, elle supprime les mots vides et identifie chaque composant dans l'ontologie en exploitant le dictionnaire linguistique WordNet **[MILL 95]**. Le résultat de cette phase est la séquence de concepts, d'instances, et de rôles de l'ontologie qui appartiennent à la requête.

La deuxième phase est la génération des requêtes nRQL. Cette phase permet de traduire la séquence de la phase précédente vers le langage d'interrogation nRQL qui est un langage spécifique à des interrogations via le raisonneur RACER. Cette traduction vers la requête nRQL est faite en passant par plusieurs formes intermédiaires de la requête.

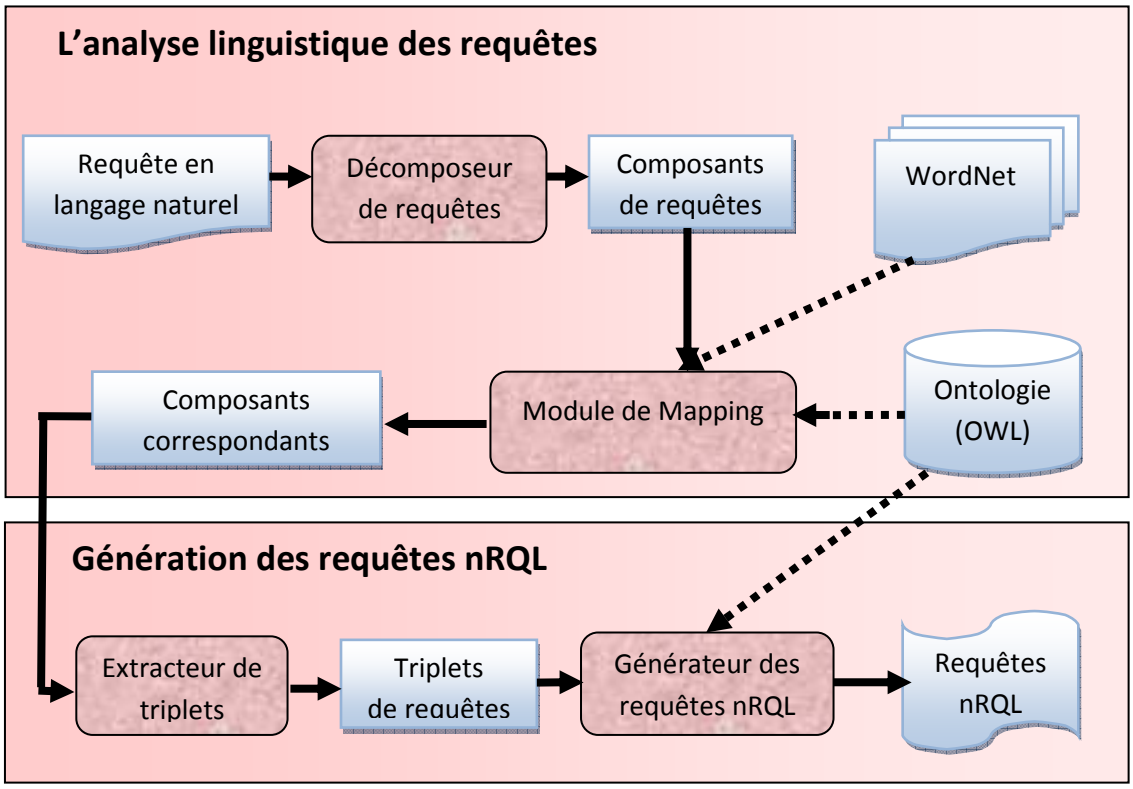

Fig.5 : Architecture du système.

4.1. Composants du système

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

4.1.1. L'ontologie de domaine

Elle encapsule 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.

Cette ontologie est utilisée dans la phase d'analyse linguistique des requêtes pour vérifier soit l'appartenance des termes initiaux de la requête à l'ontologie, soit l'appartenance de leurs synonymes dans le cas où ces termes n'appartiennent pas à l'ontologie.

Elle est aussi utilisée dans la phase de la génération des requêtes nRQL afin de vérifier s'il y a des relations sémantiques entre les

différents termes de la requête, et d'enrichir la requête par toutes les relations trouvées.

4.1.2. Le décomposeur de requêtes

Ce composant permet d'identifier les différents termes de la requête pour la représenter sous forme d'un ensemble de composants. Ainsi, le résultat généré est l'ensemble de composants de la requête qui seront envoyés au module de Mapping.

4.1.3. Le module de Mapping

En vue de combler le fossé sémantique entre la terminologie de l'utilisateur et le vocabulaire de l'ontologie, ce composant permet donc de recevoir les composants de la requête envoyés par le décomposeur de requête. Puis, il supprime les mots vides et identifie les entités de l'ontologie correspondantes à chaque terme de la requête, utilisant non seulement l'ontologie de domaine mais aussi des dictionnaires généraux comme WordNet **[FELL 98]**, afin de trouver les mots sémantiquement approprié (les synonymes).

Pour déterminer le sens d'un terme dans une requête, nous vérifions d'abord son appartenance à l'ontologie. Si ce terme appartient déjà à l'ontologie nous n'avons pas besoin de faire la correspondance. Par contre, s'il n'appartient pas à l'ontologie on doit donc rechercher ses synonymes dans WordNet, et parmi ces derniers nous sélectionnons ceux qui appartiennent à l'ontologie. Autrement dit, chaque composant sera représenté par ses correspondants dans l'ontologie.

Dans le cas où ni le terme ni au moins un parmi ses synonymes n'appartiennent à l'ontologie, ce terme sera supprimé automatiquement de la liste des composants de la requête.

L'utilisation d'un dictionnaire comme WordNet, quand une version électronique de celui-ci existe et est facilement exploitable, est le moyen le plus simple pour réaliser la désambigüisation des termes de requêtes.

Les majorités des travaux utilisant les dictionnaires pour la désambigüisation de requête, ont démontré que pour améliorer les résultats de la recherche d'information, il est nécessaire de combiner le dictionnaire avec une mesure de similarité sémantique qui permet de classifier les synonymes selon le facteur de similarité calculé. Cela permet par la suite de classifier les requêtes nRQL générées.

De nombreuses mesures **[YANG 05]** ont été définies afin de calculer la similarité sémantique entre deux concepts dans une taxonomie :

- Mesure basée sur le chemin (path based measures) entre les deux concepts à comparer telles que définies par Rada, Leacock ou Jiang en 1997**[JIAN 97]**.

- Mesure basée sur la notion de contenu d'information (Information Content ou IC) telle que celle définie par Wu et Palmer **[WU 94]** et Resnik.

- Mesure basée sur une combinaison du chemin et du contenu d'information par D. Lin en 98**[LIN 98]**.

Dans ce travail, nous avons utilisé la mesure de **LIN.** Cette mesure a l'avantage d'être simple à implémenter et d'avoir aussi de bonnes performances par rapport aux autres mesures de similarité. Selon D. Lin la formule suivante permet de calculer la similarité entre les deux concepts ci, cj :

$$Sim(ci,cj) = \frac{2 * IC(ncn(ci,cj))}{IC(ci) + IC(cj)}$$

Ou ncn est le nœud le plus proche commun (nearest common node) pour les deux nœuds ci, cj. Le contenu en information (IC) d'un concept c se calcule de la façon suivante :

$$IC(c) = -\log(p(c))$$

L'utilisation de la fonction – log permet de réduire le contenu en information d'un concept ayant une forte probabilité d'apparition dans le corpus. p(c) est la probabilité du concept c en WordNet. On peut calculer la probabilité d'un concept en fonction de sa fréquence :

$$P(c) = \frac{frec(c)}{N}$$

Sachant que N est le nombre total des nœuds, et frec(c) est calculée par la formule suivante :

$$freq(c) = \sum_{n \in words(c)} count(n)$$

Où words(c) est l'ensemble de concepts subsumés par le concept c, et count(n) le nombre d'occurrences du terme n dans le corpus.

En conséquence, le Mapping des termes de la requête aux entités de l'ontologie associe à chaque terme un ou plusieurs correspondants dans l'ontologie avec une mesure de confiance. Ainsi, le résultat généré est une ou plusieurs séquences de concepts, d'individus, et de rôles de l'ontologie correspondants aux termes initiaux de la requête.

4.1.4. WordNet

WordNet est une base de données lexicographique développée à l'université de Princeton par un groupe dirigé par George Miller **[MILL 95].** La raison principale qui motive son utilisation est qu'elle a pour objectif de représenter la langue naturelle. Une définition succincte de

WordNet est aussi donnée par **[HABE 01].** Cette ontologie linguistique générale **[GUAR 99]** est très utilisée dans le domaine de la recherche d'information notamment pour la désambiguïsation de termes**.** Elle couvre la grande majorité des noms, verbes, adjectifs et adverbes de la langue anglaise (des versions de *WordNet* pour d'autres langues existent, mais la version anglaise est cependant la plus complète à ce jour). Les mots dans WordNet sont organisés en ensembles appelés Synset. Ce dernier regroupe un ensemble de mots interchangeables, dénotant un sens ou un usage particulier.

La version de WordNet utilisée dans ce travail est **WordNet 2.1**. Cette dernière répertorie plus de 200 000 mots de classes ouvertes (pour lesquelles l'ajout d'éléments lexicaux est possible) ainsi que plus de 115 000 synsets.

La relation sémantique de base entre les mots codée dans WordNet est la *synonymie.* Les synsets sont liés par des relations telles que *spécifique/générique ou hyperonymie /hyponymie (is-a), et la relation partie-tout* ou *meronymie/holonymie (part-whole).*

4.1.5. L'extracteur de triplets

L'extracteur de triplets permet d'abord de décomposer la requête (la séquence précédente généré par le module de Mapping) en deux parties ; les arguments qui sont les classes, les instances, et les littéraux de la requête. Les relations qui sont les propriétés de la requête. Puis, l'extraction de triplets qui fait de la manière suivante :

1. Les arguments deux à deux avec une relation vide ; afin de trouver toutes les relations sémantiques qui peuvent être existé entre les arguments de la requête, et d'enrichir la requête par ces relations trouvées.
2. Les arguments deux à deux avec toutes les relations existantes ; afin de vérifier s'il existe des triplets valides dans la requête.

3. Chaque argument avec toutes les relations existantes afin de vérifier si l'argument est la domaine ou co_domaine de la relation et remplacer l'argument vide par une variable qui sera utilisé par la suite pour générer la requête nRQL.

Par exemple si on a la requête : « Bird Eat Meat Food », tel que :

Bird → classe

Eat → propriété d'objet

Meat → instance

Food → classe

Selon les types des composants on a:

Les arguments = {Bird, Meat, Food}

Les relations = {Eat}

Alors les triplets extraits sont :

a) Les arguments deux à deux avec relation vide :

< Bird, vide, Meat >, < Bird, vide, Food >, < Meat, vide, Food >

b) Les arguments deux à deux avec toutes les relations existantes :

< Bird, Eat, Meat >, < Bird, Eat, Food >, < Meat, Eat, Food >

c) Chaque argument avec toutes les relations existantes :

< Bird, Eat, vide >, < Meat, Eat, vide >, < Food, Eat, vide >

Ainsi, ce composant sert à traduire la requête vers un ensemble de triplets sous forme <argument, relation, argument> où les arguments peuvent être des classes, des instances, des littéraux, ou vides. Et la relation peut être une propriété d'objet (Object_property), une propriété de type de donnée (data_type_property), ou vide.

Nous représentons la requête sous forme des triplets afin de faciliter la recherche des relations sémantiques entre les composants de chaque triplet de la requête par rapport aux relations binaires entre les concepts de l'ontologie (une relation de subsomption ou une propriété).

4.1.6. Le générateur des requêtes nRQL

Ce composant convertit les triplets extraits vers le langage nRQL. L'algorithme présenté dans la section 4.2 se charge de cette conversion.

4.2. L'algorithme de génération des requêtes nRQL

L'algorithme comporte trois étapes **[BOUMb 09]**. On commence par trouver 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:

4.2.1 Établir les liaisons sémantiques entre les différents composants 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 d'objet, 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 la relation 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 la relation vide du triplet par la propriété d'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 la relation vide du triplet par la propriété d'objet trouvée.

- Les deux arguments sont des instances. On doit rechercher s'il existe une propriété d'objet reliant les classes des instances, et on remplace la relation vide du triplet par la propriété d'objet trouvée. Par exemple le triplet < Hair, _ , Orange> deviendra < Hair, Has_color, Orange >. Sachant que, Has_color est la propriété d'objet dont le domaine est la classe Surface et le co_domaine est la classe Color.

- 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 la classe de l'instance et le co_domaine est le type du littéral. On remplace la relation vide du triplet par la propriété de type de donnée trouvée.

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

Cas 2 : la relation n'est pas vide, ici on a deux possibilités :

- L'un des arguments est vide. On doit vérifier si l'argument qui existe est le domaine ou le co_domaine de la relation. 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 cette relation 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. Les triplets restants doivent etre dans l'ordre <domaine, relation, co_domaine>.

4.2.2 Établir les liaisons entre les triplets de la requête

Cela est fait par la conjonction de tous les triplets de la requête. Il est nécessaire de supprimer les liaisons redondantes en choisissant celles les plus spécifiques par l'utilisation de quelques règles de spécification. A titre d'exemple :

- Si la requête contient parmi ses paramètres, le nom d'une classe et le fils de cette classe alors on restreint les interrogations au fils :

If[($class_1$, Object_Property, $class_2$) **and** ($class_3$, Object_Property, $class_2$) **and** ($class_1$ **is_a** $class_3$ **)]then**

Delete ($class_3$, Object_Property, $class_2$)

If[($class_1$, data_type_property ,literal) **and** ($class_2$, data_type_property ,literal) **and (**$class_1$ **is_a** $class_2$ **)]then**

Delete ($class_2$, data_type_property, literal)

-Même chose pour le co_domaine d'une propriété :

If[($class_1$, Object_Property, $class_2$) **and** ($class_1$, Object_Property, $class_3$) **and (**$class_2$ **is_a** $class_3$ **)]then**

Delete ($class_1$, Object_Property, $class_3$)

- Si la requête contient parmi ses paramètres, le nom d'une classe et une instance de cette classe alors on restreint les interrogations à l'instance :

If[($class_1$, Object_Property, $class_2$) **and** (instance, Object_Property, $class_2$) **and** $class_1(x) == class'(instance)$**]then**

Delete ($class_1$, Object_Property, $class_2$)

If[($class_1$, data_type_property ,literal) **and** (instance, data_type_property ,literal) **and** $class_1(x) == class'(instance)$**]then**

Delete ($class_1$, data_type_property ,literal)

-Même chose pour le co_domaine d'une propriété :

If[($class_1$, Object_Property, $class_2$) **and** ($class_1$, Object_Property, instance) **and** $class_2(y) == class'(instance)$**]then**

Delete ($class_1$, Object_Property, $class_2$)

- Si la requête contient parmi ses paramètres, le nom d'une classe et une variable alors on restreint les interrogations à la classe :

If[(class$_1$, Object_Property, class$_2$) **and** (VAR, Object_Property, class$_2$)**]then**

Delete (VAR, Object_Property, class$_2$)

-Même chose pour le co_domaine d'une propriété :

If[(class$_1$, Object_Property, class$_2$)**and**(class$_1$, Object_Property, VAR)**]]then**

Delete (class$_1$, Object_Property, VAR)

- Si la requête contient parmi ses paramètres, le nom d'une instance et une variable alors on restreint les interrogations à l'instance :

If[(instance,Object_Property, class$_2$) **and** (VAR, Object_Property, class$_2$)**]then**

Delete (VAR, Object_Property, class$_2$)

-Même chose pour le co_domaine d'une propriété :

If[(class$_1$,Object_Property, instance) **and** (class$_1$,Object_Property, VAR)**]then**

Delete (class$_1$,Object_Property, VAR)

Le résultat de cette étape est la forme intermédiaire optimale de la requête après la suppression de toutes les redondances existantes.

4.2.3 Génération de la requête nRQL finale

Il reste enfin à traduire la forme intermédiaire de la requête générée au cours de l'étape précédente vers le 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. Dans la suite, nous donnons quelques requêtes en langage nRQL **[HAAR 04]**:

- Pour savoir si un concept a des individus ou non:

  (retrieve () (?x |Concept|))

  Si le concept contient des individus le résultat retourné est T (true) sinon le raisonneur retourne NIL.

- Pour avoir toutes les instances d'un concept:

(retrieve (?x) (?x |Concept|))

Si le concept est défini, les instances retournées sont celles ajoutées par le raisonneur par inférence et non par l'instanciation de l'utilisateur.

Si le concept est primitif, les individus retournés sont ceux instanciers par l'utilisateur (((?X |instance1|)) ((?X |instance2|)) ... ((|instance n|)))

- vérifier l'appartenance d'un individu à un concept:

  (retrieve () (|individus| |Concept|))

  Si l'individu appartient au concept, le résultat retourné est T (true) sinon NIL.

- les individus liés par un rôle:

  (retrieve (?x ?y) (?x ?y |role|))

  les individus liés par ce rôle sont retournés

  (((?X |indx1|) (?Y |indy1|)) ((?X |indx2|) (?Y |indy2|)) ... ((?X |indx3|) (?Y |indy3|)))

- les individus liés à un individu prédéfini:

  (retrieve (?x) (?x |individus| |role|))

  Ou

  (retrieve (?x) (|individus| ?x |role|))

- On peut réaliser des imbrications de requêtes simples avec les opérateurs logiques pour avoir des requêtes complexes (AND OR NOT):

  (retrieve (?x) (OR (?x |Concept|) (?x |Concept|)))

  (retrieve (?x ?y) (AND(?x ?y |role|)(?x |individus| |role|)))

  (retrieve (?x ?y) (AND (?x |Concept|) (?x ?y |role|)))

  (retrieve (?x ?y) (NOT(?x ?y |role|)))

  (retrieve (?x) (AND(?x ?y |role|) (NOT(?y |Concept|))))

- pour plus de complexité on peut utiliser des relations entre variables. Si on veut chercher par exemple l'individu z qui est lié à y et ce dernier est lié à x :

(retrieve (?z) (AND(?z ?y |role$_1$|) (?y ?x |role$_2$|)))

Dans notre travail, la requete nRQL est déterminée par quelques règles de génération :

VAR → **? x**

Instance → **|Instance|**

Class → **? x |Class|**

Object_Property → **| Object_Property |**

<class$_1$, object_property, class$_2$> → **(?x |class$_1$| ?y |class$_2$| | Object_Property |**

<class$_1$, object_property$_1$, class$_2$> , <class$_3$, object_property, instance> →

(and (?x |class$_1$| ?y |class$_2$| | Object_Property$_1$|) (? z |class$_3$||instance| |Object_Property$_2$|))

Fig.6 : Règles de génération

Remarque : les variables de la requête nRQL finale sont déterminées selon l'ordre d'occurrence dans les triplets.

4.3. 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 identifie tous les composants existants dans la requête.
3. Le module de Mapping utilise l'ontologie et WordNet 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 fait les liaisons sémantiques entre les composants de chaque triplet et la conjonction entre tous les triplets puis traduit la requête en langage nRQL.

5. Caractéristique du système proposé

Avec l'avènement du web sémantique la grande partie des recherches se concentre sur la structuration des informations disponibles sur le web, ce qui permet la naissance des différents langages de représentation de connaissances (XML, RDF, OWL,…). Cependant, non seulement les connaissances structurés permettent à satisfaire les besoins en information d'utilisateurs, on a plutôt besoin des systèmes de la recherche sémantique basés sur l'interprétation sémantique des requêtes ainsi que des documents, afin d'apporter à l'utilisateur ce qu'il cherche vraiment. Toutefois, les progrès de la recherche sémantique ont été retardés en raison de la complexité de ses langages de requête.

L'idée est de construire un système, capable de traduire du mieux possible les requêtes en langage naturel vers le langage nRQL. Ces requêtes nRQL sont envoyées par la suite au moteur d'inférence RACER pour avoir les ressources relatives. Cependant, l'accés aux bases de connaissances n'est pas facile, cela engendre deux obstacles majeurs :

1. L'ambiguïté de la langue naturelle ; comment représenter les termes du langage naturel de la requête sous forme des concepts et des rôles de l'ontologie.

2. L'interprétation sémantique et la formalisation correcte des requêtes en langage d'interrogation nRQL.

Pour régler le premier obstacle, nous avons utilisé la version éléctronique de *WordNet(2.1)* qui est disponible gratuitement pour le téléchargement. Sachant que WordNet est le dictionnaire le plus utilisé pour la désambiguïsation des termes de la langue naturelle. Ainsi , si le terme n'appartient pas à l'ontologie on doit rechercher ses synonymes dans WordNet et parmi ces derniers nous sélectionnons ceux qui appartiennent à l'ontologie. Puis, nous avons toutes les combinaisons possibles pour arriver enfin aux séquences des concepts et des rôles correspondant aux termes initiaux de la requête.

Pour régler le deuxiéme obstacle, nous avons représenté d'abord chaque séquence sous forme des triplets, cela permet de faciliter la comparaison des triplets de la requête avec les triplets de l'ontologie. Puis, nous avons utilisé un algorithme de génération des requêtes nRQL qui prend en entré les triplets de la requête et renvoi en sortie la requête nRQL appropriée en se basant sur les relations sémantiques entre tous les composants de la requête. Cet algorithme permet d'abord d'enrichir la requête par les nouvelles relations trouvées parce que les relations entre les ressources de la base de connaissances sont exigées pour être explicitement énoncées dans les requêtes nRQL. Ensuite, il utilise des règles de spécification pour supprimer les redondances et optimiser la requête. Enfin, il utilise des règles de génération pour générer la requête nRQL finale.

6. Calcul de la pertinence

La pertinence est un concept clé de la recherche d'information depuis les années 50. Elle représente le degré de corrélation entre une requête et la réponse apportée. Dans notre système, pour calculer le degré de la pertinence des résultats retournés nous disposons quelques facteurs :

1. Lors du Mapping des termes des requêtes vers les entités de l'ontologie dans la phase d'analyse linguistique des requêtes, plus le facteur de similarité entre le terme de la requête et le concept de l'ontologie augmente plus le degré de pertinence de résultats retournés par cette ontologie est élevé.

2. La longueur des requêtes (nombres des termes dans la requête), Plus le nombre des termes augmente plus le résultat est satisfaisant.

3. Le degré de pertinence augmente selon le niveau de spécification des termes des requêtes d'interrogation. Plus ils sont spécifiques, plus le résultat est précis.

Les résultats retournés sont groupés par ordre décroissant du degré de pertinence selon le premier facteur, puis si on a des résultats avec des degrés égaux on les réorganise selon le troisième facteur.

7. Comparaison avec les systèmes existants

7.1. Inconvénients des systèmes existants

- La plupart des systèmes existants sont principalement orientée vers les petites requêtes qui contiennent jusqu'à deux triplets. En plus, ils n'expriment que les questions (parce que la magorité des systèmes existants sont des systèmes à question-reponse).
- Ils sont basés sur la recherche des liens entre les termes de la requête et les entités de l'ontologie plutôt que sur la relation sémantique entre ces termes par rapport aux relations binaires de l'ontologie.
- Ils ont supposé que chaque requête peut être ecrite sous forme de triplets linguistiques ; pour cela ils ont intégré les différentes techniques de TALN pour générer l'arbre syntaxique de la requête. Cet arbre syntaxique facilite donc l'extraction des triplets linguistique.

7.2. Avantages du notre système

- ✓ Le système proposé permettra à l'utilisateur d'interroger une base de connaissances sans avoir une connaissance préalable sur la structure de l'ontologie ; en utilisant une requête en langage naturel.
- ✓ La désambigüisation des termes de la requête en langage naturel est faite en expoitant WordNet qui permet de trouver pour chaque terme ses correspondants dans l'ontologie.
- ✓ La génération des requêtes nRQL s'appuie sur un algorithme de génération basé sur les relations sémantiques entres tous les composants de la requête.
- ✓ Enrichissement de la requête par toutes les nouvelles relations sémantique trouvées entre ses composants.

- ✓ L'utilisation des règles de spécification permet de supprimer les liaisons redondantes en choisissant celles les plus spécifiques, ce qui augmente la pertinence.

8. Conclusion

Dans ce chapitre nous avons présenté une 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 des concepts et des 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. Certains composants de cette architecture seront implémentés dans le prochain chapitre.

Chapitre IV

Etude de cas : Application à l'ontologie de domaine « AnimalOntology »

«Ce n'est pas grand-chose d'avoir des idées, le tout est de les appliquer»

Emile Auguste

1. Introduction

La conception d'ontologies est une tâche difficile qui nécessite la mise en place de procédés élaborés afin d'extraire la connaissance d'un domaine, manipulable par les systèmes informatiques et interprétable par les êtres humains.

Plusieurs principes ont été définis pour la construction d'ontologies. Ces principes insistent sur la nécessaire clarté de la définition des éléments que l'ontologie doit contenir (rôle et portée de l'ontologie, définition des concepts, limitation des ambiguïtés) ainsi que sur la séparation des phases de conception et d'implantation dans un langage formel de l'ontologie. L'ensemble de ces principes reste cependant abstrait.

Dans cette partie, nous allons développer une petite ontologie sur le domaine des animaux appelée *AnimalOntology*. L'ontologie que nous avons utilisée pour valider notre proposition. Ensuite, nous illustrons quelques exemples des requêtes en

langage naturel converties par notre système ; afin de découvrir toutes les étapes suivies lors de la conversion des requêtes de langage naturel vers le langage nRQL.

2. Construction de l'ontologie *AnimalOntology*

Bien que la conception des ontologies soit un processus créatif il ne peut pas y avoir d'ontologies identiques faites par des personnes différentes. Les applications potentielles d'une ontologie plus le point de vue du concepteur sur le domaine traité affecteront indubitablement les choix de conception de l'ontologie.

Nous nous sommes rendu compte que les ontologies sont toujours liées à une méthodologie de conception, à un outil de construction, et un langage de représentation d'ontologies. De nombreuses méthodologies ont été définies pour cadrer le développement d'ontologies de domaine **[GUEV 06]**.

Dans ce travail, nous nous appuyons sur le processus de construction spécifié dans le travail de Hemam Mounir **[HEMA 05]** pour concevoir l'ontologie de domaine. Le cycle de vie d'ontologies constitué de six étapes: étape de spécification des besoins, étape de conceptualisation, étape de formalisation, étape de codification, étape d'évolution, et étape de vérification (**fig.7**).

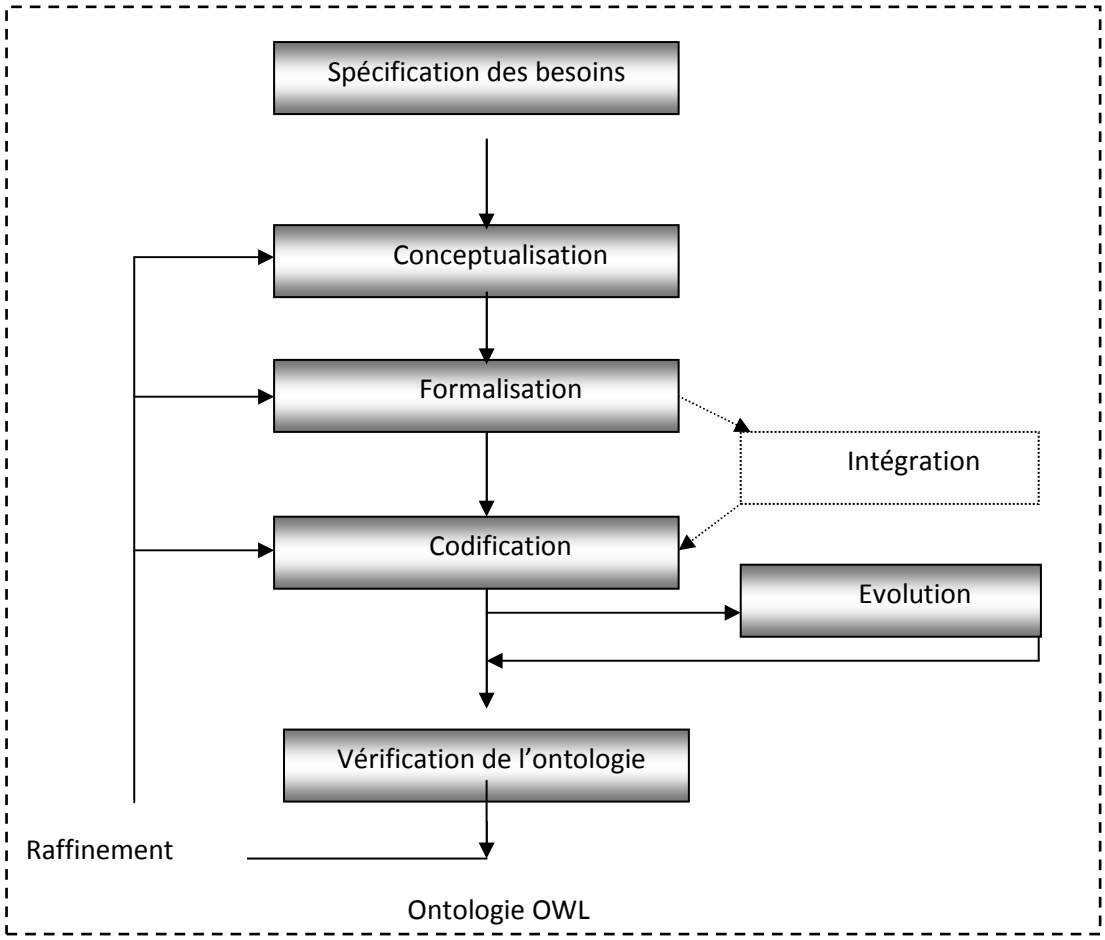

Fig.7 : Cycle de vie d'une ontologie

Ainsi, le processus de développement de l'ontologie transforme le produit initial (l'objectif et les besoins de construire l'ontologie) vers un produit final (l'ontologie documentées, et codifiées dans un langage formel).

2.1. Spécification des besoins

L'étape de spécification, permet de produire un document de spécification (**fig.8**) de la future ontologie. Ce document décrit, entre autres, l'objet et le domaine que va couvrir l'ontologie, ses utilisateurs, ses sources d'informations (les experts du domaine, les documents techniques,…etc.), et la liste des termes les plus importants pour le domaine à représenter (la porté de l'ontologie).

- **Domaine :** Description des d'animaux selon leurs caractéristiques physiques, environnements, nourritures, …etc.

- **Objectif opérationnel:** l'ontologie est utilisée comme un référentiel afin d'aider l'utilisateur dans la spécification de son besoin en information ce qui permet de mieux interpréter la sémantique des requêtes des utilisateurs.

- **Utilisateur :** les chercheurs de documents.

- **Degré de formalisme :** formel.

- **Porté (liste des termes importants) :** taxonomie des animaux (animal, vertebrate, invertebrate, mammal, fish, …etc.)

Fig.8 : Le document de spécification

2.2. Conceptualisation

Cette étape vise à structurer la connaissance du domaine en un modèle conceptuel. La représentation est à ce stade est informelle. On distingue les activités principales suivantes :

2.2.1. Construction du glossaire de termes

Le glossaire décrit tous les termes qui sont utiles et potentiellement utilisables dans le domaine.

Terme	Description	Attributs
Animal	Est un être vivant multicellulaire capable d'appréhender son environnement et de s'y déplacer volontairement.	Name ScientificName Locomotion Family
Vertebrate	Animal chordé possédant une colonne vertébrale segmentée au stade adulte.	-
Invertebrate	Tout animal dépourvu de colonne vertébrale. Les invertébrés incluent toutes les espèces qui ne sont pas des vertébrés qui, eux, ont un squelette et des vertèbres.	-
Arthropod	Embranchement d'invertébrés possédant un squelette externe et des appendices articulés, comprenant les crustacés, les insectes, les mille-pattes (myriapodes) et les araignées (arachnides).	-
Mollusc	Embranchement d'invertébrés à corps mou, généralement pourvus d'une coquille, externe ou interne.	-
Amphibian	Petits vertébrés tétrapodes à sang froid et à peau nue, généralement aquatiques.	-
Bird	Vertébrés à plumes dont les membres antérieurs sont transformés en ailes.	-
Fish	Animaux vertébrés aquatiques possédant des nageoires et respirant par des branchies.	-
Mammal	Vertébrés à sang chaud nourrissant leurs petits avec le	-

	lait de leurs mamelles et en général couverts de poils.	
Reptile	Vertébrés comprenant les serpents, les lézards, les tortues, les crocodiles et de nombreuses espèces fossiles éteintes, tels les dinosaures.	-
Description	Décrit les caractéristiques physiques des animaux	-
Body	Il décrit la structure physique du corps de l'animal, et ses organes.	-
Color	la couleur du corps.	-
Environment	Milieu dans lequel un animal fonctionne, incluant l'air, l'eau, la terre,…etc.	-
Food	décrit la nourriture de l'animal	-
……	……	……

Fig.9 : Glossaire des termes

Remarque : Lors du développement de cette ontologie, quelques concepts sont inespérés de l'ontologie *Animal* développée par Allioua Sofiane dans **[ALLI 08].**

2.2.2. Construction des hiérarchies de concepts

Chaque hiérarchie organise un groupe de concepts sous forme d'une taxonomie :

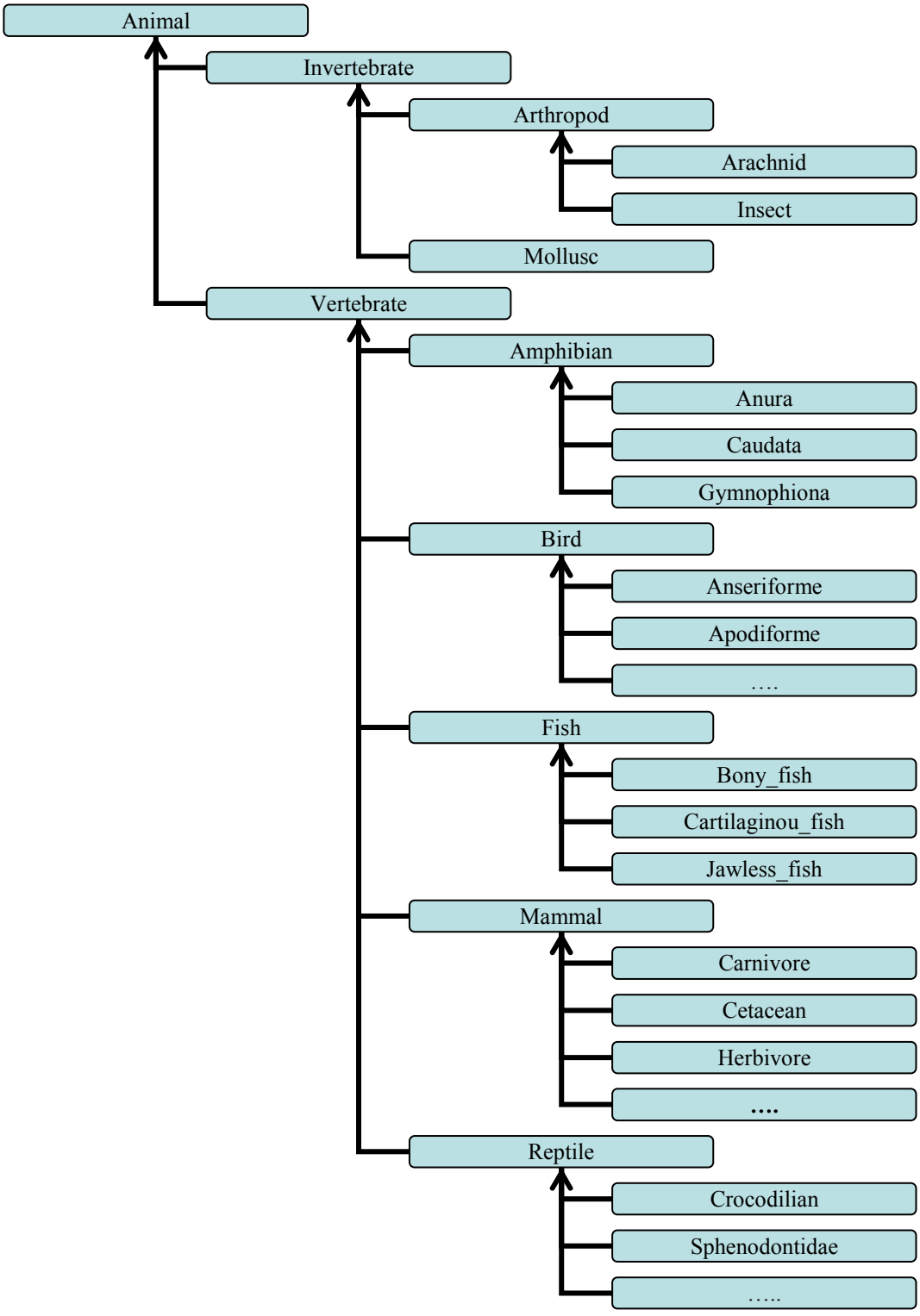

Fig.10 : Hiérarchie 1

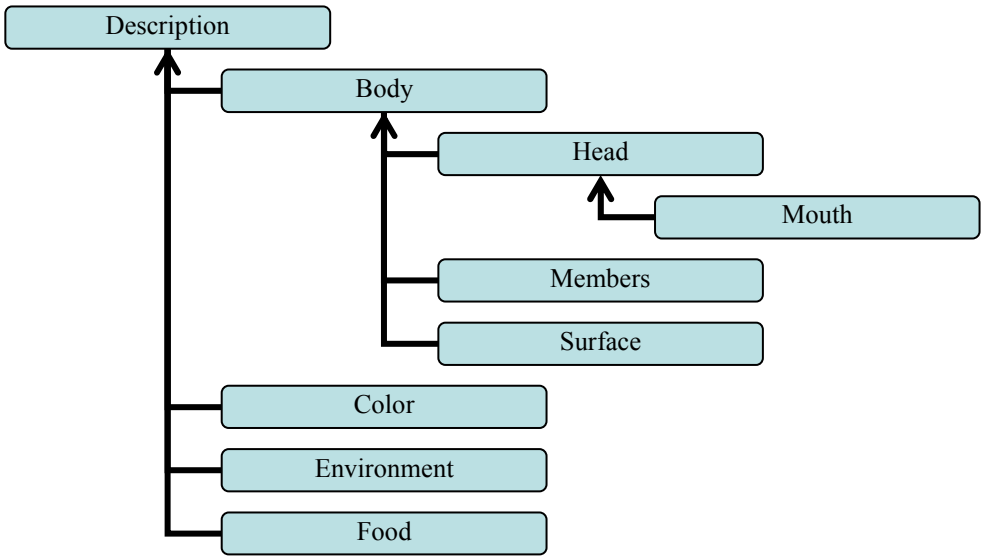

Fig.11 : Hiérarchie 2

2.2.3. Construction d'un diagramme des relations binaires

Ce diagramme permet de représenter de manière graphique les diverses relations qui existent entre les divers concepts de même ou de différentes hiérarchies.

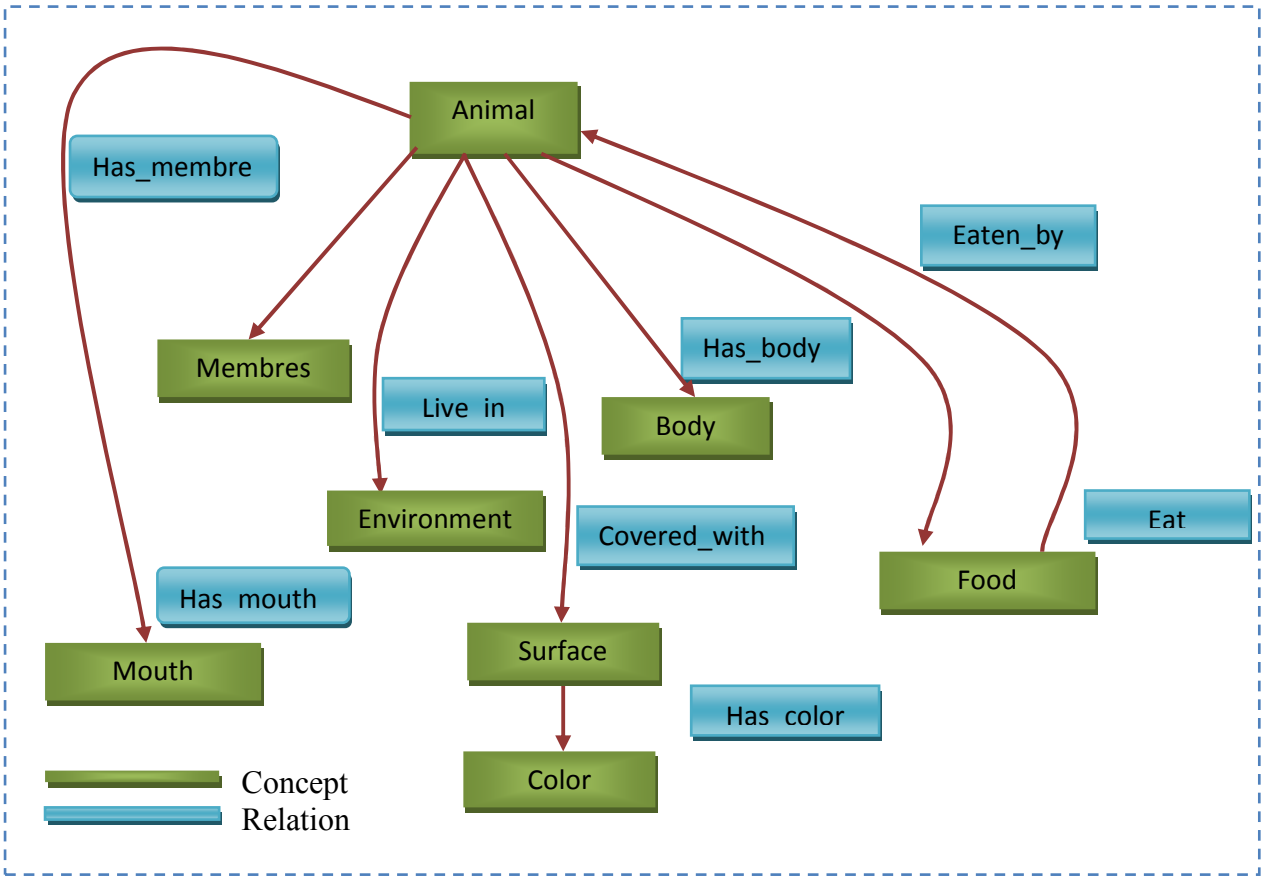

Fig.12 : Diagramme des relations binaires

2.2.4. *Construction de la table des relations binaires*

La table des relations binaires définie pour chaque relation utilisée dans le diagramme précédent, le nom de la relation, le nom des concepts sources et cibles, le nom de la relation inverse et les cardinalité source et cible.

Nom de la relation	***Concept source***	***Cardinalité source***	***Concept cible***	***Cardinalité cible***	***Relation inverse***
Covered_with	Animal	(1, 1)	Surface	(1, n)	-
Eat	Animal	(1, n)	Food	(1, n)	*Eaten_by*
Live_in	Animal	(1, 2)	Environment	(1, n)	-
Has_body	Animal	(1,1)	Body	(1, n)	-

Has_color	Surface	(1, n)	color	(1, n)	-
…….	……...	……	……...	…….	……...

Tableau.1 : Table des relations binaires

2.2.5. *Construction de la table des attributs*

Elle permet de spécifier les contraintes des différents attributs.

Nom de l'attribut	***Type***	***Cardinalité (Min/Max)***	***Valeur par défaut***	***Domaine des valeurs***
Name	Chaîne de caractères	(1,1)	-	-
Scientific_name	Chaîne de caractères	(1,n)	-	-
Locomotion	Chaîne de caractères	(1,n)	-	-
Family	Chaîne de caractères	(1,1)	-	-
……	……...	……...	……	……...

Tableau.2 : Table des attributs

2.2.6. Construction de la table des axiomes logiques

La table des axiomes définit les concepts au moyen des expressions logiques.

Nom du concept	***Description***	***Expression logique***
Invertebrate	Tous les animaux qui ne sont pas des vertébrés.	Animal ∩ ¬Vertebrate
Vertebrate	Tout Animal possède un squelette.	Animal ∩ Has_membres ∋ « Skeleton »

Bird	Vertébrés à plumes dont les membres antérieurs sont des ailes, avec un bec.	Vertebrate ∩ Covered_with ə «Feather» ∩ Has_membres ə «Wings» ∩ Has_mouth ə «Beak» ∩ LiveIn ə «Aerial»
Fish	Vertébrés aquatiques possédant des nageoires et respirant par des branchies.	Vertebrate ∩ Covered_with ə «Scale» ∩ Has_membres ə « fins» ∩ Live_in ə «Aquatic»
Mollusc	Invertébrée couvé par une carapace.	Invertebrate ∩ Covered_with ə «Carapace»
Carnivore	tout animal qui se nourrit principalement de la viande, et recouvert de poils.	Mammal ∩ Covered_with ə «Hair» ∩ Eat ə «Meat» ∩ Has_mouth ə «Tooth»
Food	décrit la nourriture de l'animal	Description ∩ ∃Eaten_by Animal

Tableau.3 : Table des axiomes logiques

2.2.7. *Construction de la table des instances*

Elle permet de décrire les instances des concepts avec leurs attributs et valeurs.

Nom de l'instance	***Attributs***	***Valeurs***
Cat_fish	Name Scientific_name Locomotion Family	Cat_fish Siluriformes swim Akysidae
Elephant	Name Scientific_name Locomotion Family	Elephant Proboscidea Walk Elephatidae
……..	……..	…….

Tableau.4 : Table des instances

2.2.8. *Construction de la table des assertions*

Les assertions affirment l'existence de relations entre des instances.

Relation	**Instance Sources**	**Instances Cibles**
Live_in	Cat_fish	Aquatic
Eat	Bear	Meat
Covered_with	Cat	Hair
……	……	……

Tableau.5 : Table des assertions

2.3. Formalisation

Elle permet de faciliter la représentation de l'ontologie dans un langage complètement formel et opérationnel (*OWL*).

2.3.1 Représentation de la partie terminologique

Cette dernière comprend des définitions de concepts et de rôles ainsi que des inclusions de concepts, en utilisant la syntaxe de la logique de descriptions de type SHIQ :

Animal *:= Invertebrate ∪ Vertebrate*

Invertebrate *:= animal ∩ ¬Vertebrate*

Vertebrate *:= Amphibian ∪ Bird ∪ Fish ∪ Mammal ∪ Reptile*

Vertebrate *:= animal ∩ ∃ Has_membres .{Skeleton}*

Amphibian *:= ¬ (Bird ∪ Fish ∪ Mammal ∪ Reptile)*

Bird *:= ¬ (Amphibian ∪ Fish ∪ Mammal ∪ Reptile)*

Fish *:= ¬ (Amphibian ∪ Bird ∪ Reptile)*

Fish *:= Vertebrate ∩ (∃ Covered_with .{ Scale}) ∩ (∃ Has_membres .{ fins}) ∩ (∃ Live_in .{Aquatic})*

Mammal *:= ¬ (Amphibian ∪ Bird ∪ Reptile)*

Mammal *:= Vertebrate ∩ ∃ Has_membres .{ Breast}*

.....

Fig.13 : Définition des concepts.

Covered_with(Animal, Surface)

Has_body (Animal, Body)

Live_in (Animal, Environment)

Has_membres (Animal, Members)

Eat (Animal, Food)

Has _color (Surface, Color)

……..

Fig.14 : Définition des rôles.

Animal ⊆ T	Amphibian ⊆ Vertebrate
Description ⊆ T	Bird ⊆ Vertebrate
Invertebrate ⊆ animal	Fish ⊆ Vertebrate
Vertebrate ⊆ animal	Mammal ⊆ Vertebrate
Arthropod ⊆ Invertebrate	Reptile ⊆ Vertebrate
Mollusc ⊆ Invertebrate	Body ⊆ Description
Color ⊆ Description	Food ⊆ Description
Environment ⊆ Description	……..

Fig.15 : Inclusions des concepts.

2.3.2 Représentation de la partie assertionnelle

La partie assertionnelle (A-box), est un ensemble d'axiomes décrivant des situations concrètes, par rapport au domaine que l'on investit. Pour chaque individu (instance définie dans la table des instances) et pour chaque rôle (relation définie dans la table des assertions).

Elephant: Herbivore
Giraffe: Herbivore
Rabbit: Herbivore
Cat : Carnivore
Dog : Carnivore
Fox : Carnivore
Chimpanzee : Primate
Gorilla : Primate
Aerial: Environment
Aquatic : Environment
(Rabbit, Plants) : Eat
(Fox, Forest) : Live_in
(Cat, Hair) : Covered_with
.....

Fig.16 : Définition des individus et assertions

2.4. Codification

Différents outils ont été proposés pour aider à la conception manuelle d'ontologies. Ces outils permettent d'éditer une ontologie, d'ajouter des concepts et des relations, etc. Ils intègrent différents langages de formalisation (RDF, OWL).

Dans ce travail nous utilisons Protégé OWL[1] qui est une interface modulaire, développée au Stanford Medical Informatics de l'Université de Stanford, permettant l'édition, la visualisation, le contrôle (vérification des contraintes) d'ontologies. En plus, cet outil permet de formuler l'ontologie dans le langage de représentation de connaissance OWL, et de vérifier l'ontologie par le raisonneur RACER (calculer la relation de subsomption entre les concepts, et vérifier la cohérence de tous les concepts).

[1] http://protege.stanford.edu

2.4.1. Définition de la hiérarchie des classes

La hiérarchie de classes (ou taxinomie) se présente dans Protégé sous la forme d'un arbre (**Fig.17**).

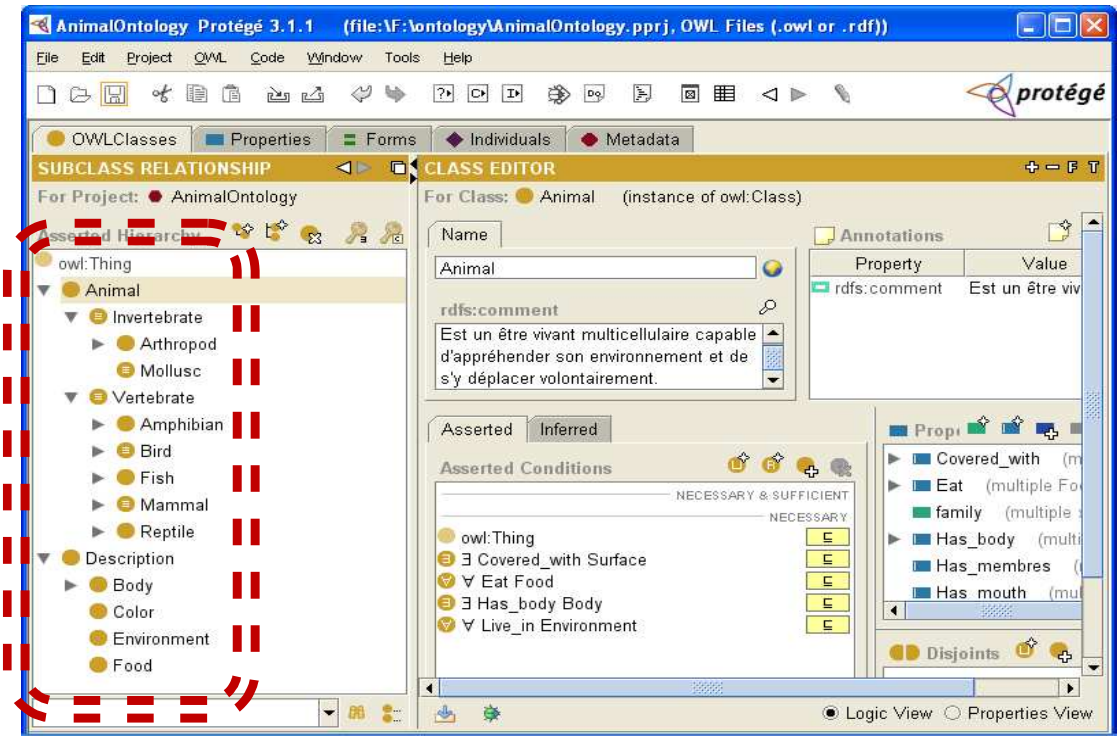

Fig.17 : Hiérarchie des classes

2.4.1. Définition des propriétés des classes

Après avoir défini les classes de l'ontologie, il convient de déterminer les propriétés (Rôles et attributs) qui associées ces classes.

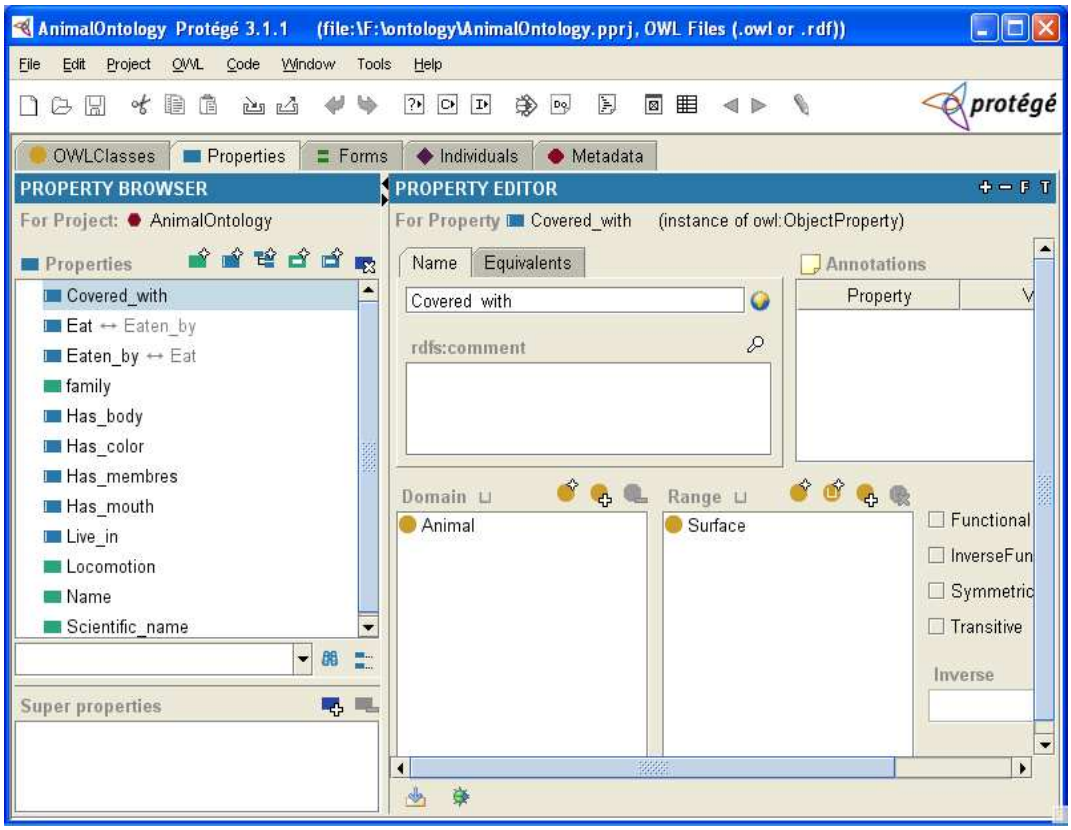

Fig.18: Définition des propriétés

2.5. Vérification

La phase d'évaluation (vérification) intervient alors pour vérifier et valider l'ontologie en question ainsi que son environnement logiciel et sa documentation. Les problèmes de cohérence, de satisfiabilité, de subsomption sont alors vérifiés grâce à la connexion au moteur d'inférence RACER.

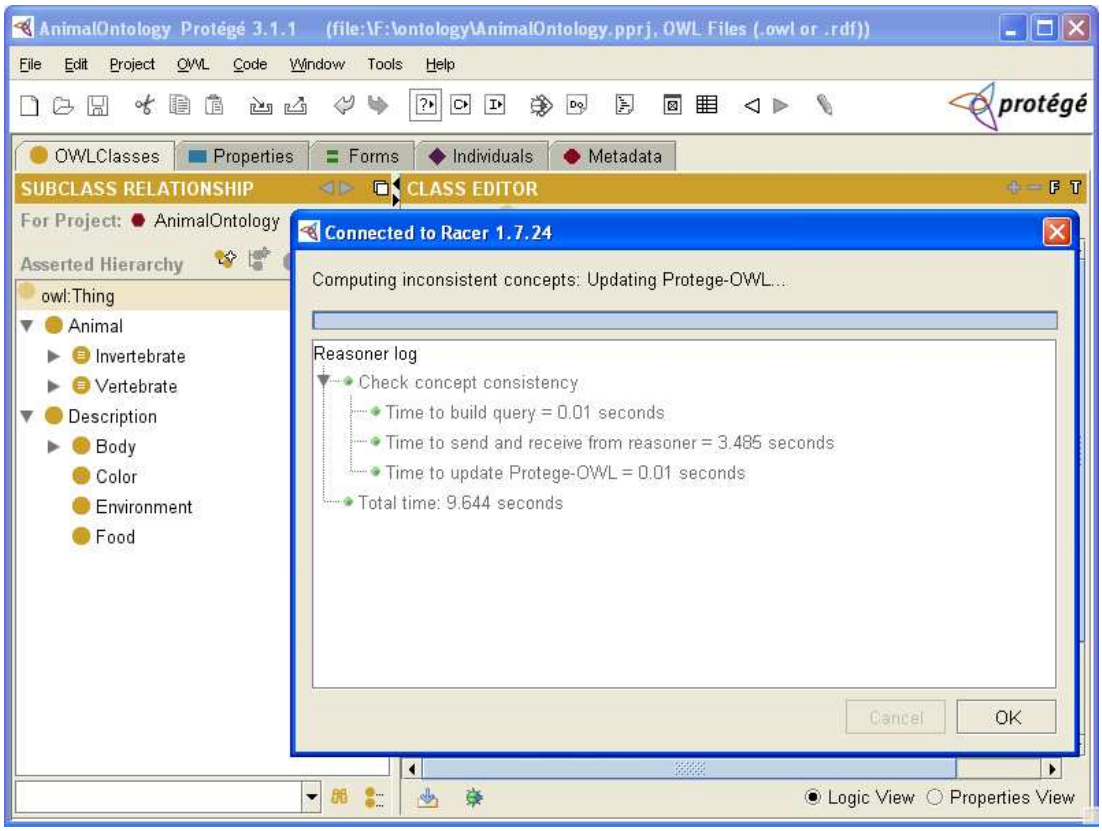

Fig.19 : Vérification de l'inconsistance des concepts

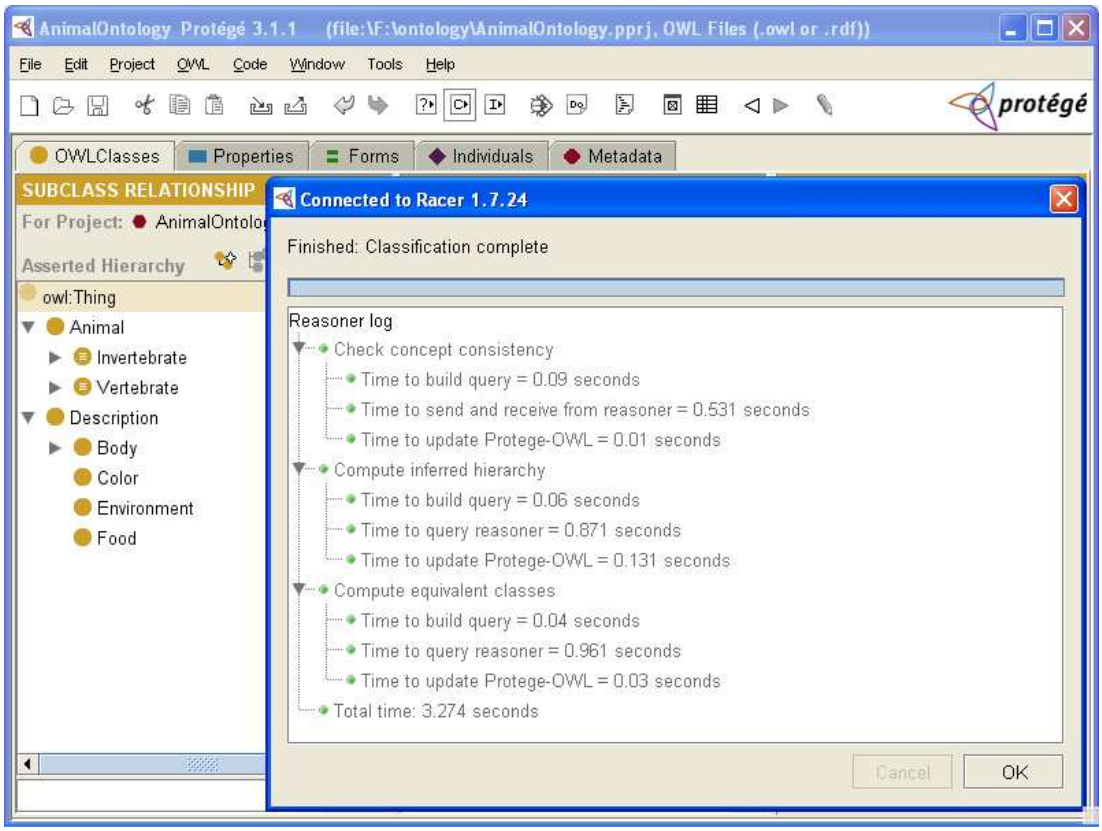

Fig.20 : Vérification de la classification

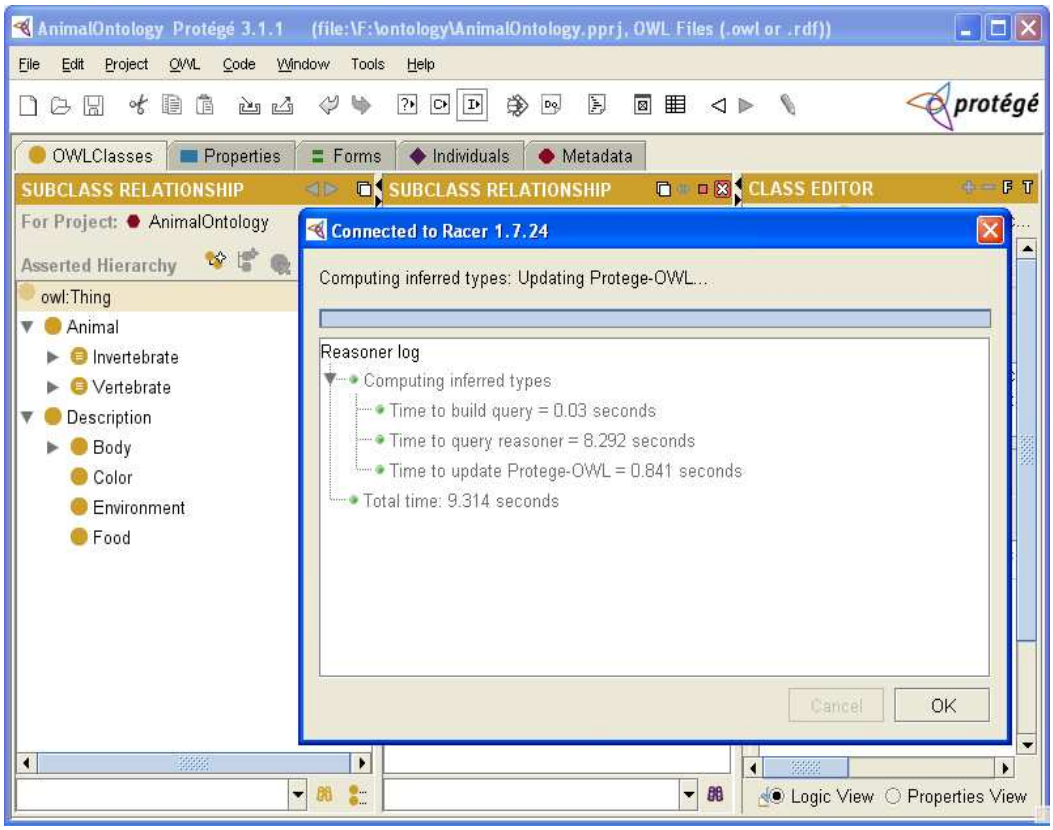

Fig.21 : Vérification des types inférés

3. Etude de cas

Une fois l'ontologie validée, elle est donc prête pour être interrogée par les requêtes d'utilisateurs. Afin de faciliter cette interrogation notre système fonctionne avec une interface (**Fig.22)** qui permet à l'utilisateur de formuler son besoin en informations à partir d'une requête en langage naturel. Puis, le système lui présente le résultat de conversion à travers cette interface.

Fig.22 : l'interface du système

Dans ce que suit, nous présenterons un échantillon des requêtes traitées par le système; afin d'illustrer les étapes prises par ce système lors de la conversion des requêtes en langage naturel vers nRQL.

☞ ***Requêtes simple avec un seul terme :***

Par exemple pour la requête = « bird », on a un seul composant de type classe **(Fig.23)**. Parce que ce composant appartient à l'ontologie, on a pas besoin ni de faire le Mapping, ni de l'extraction de triplets, ni de l'application de l'algorithme de génération des requêtes nRQL. On applique directement les règles de génération des requêtes nRQL.

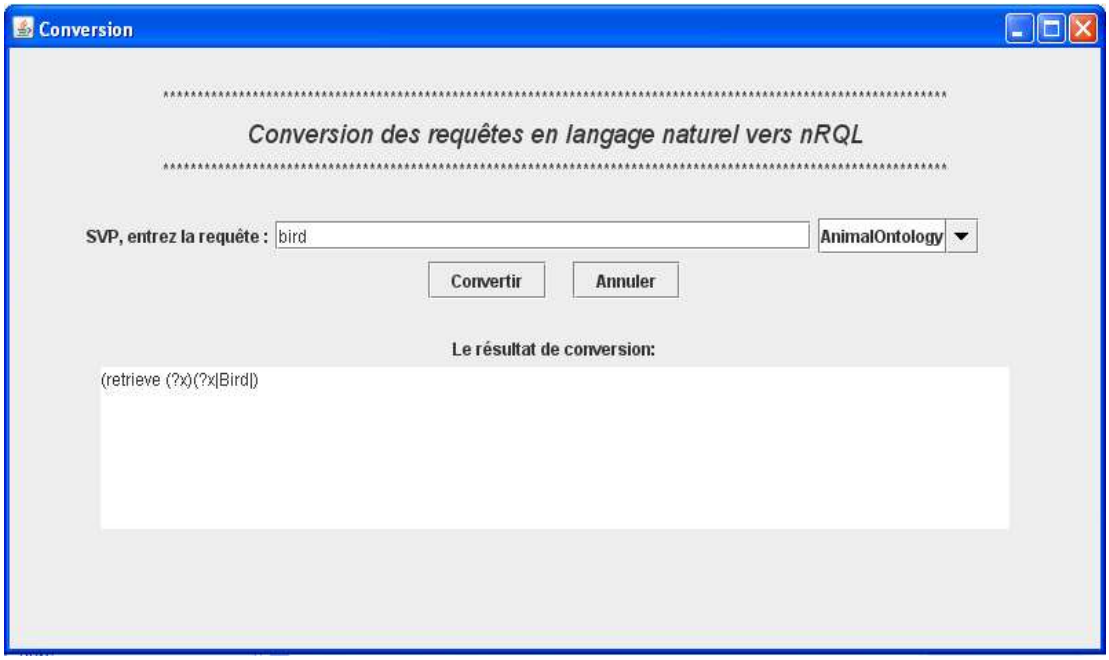

Fig.23 : requête classe

La même chose pour la requête = « live_in », on a un seul composant de type propriété d'objet. Donc, la requête nRQL généré doit rechercher les individus reliés par cette propriété **(Fig.24)**.

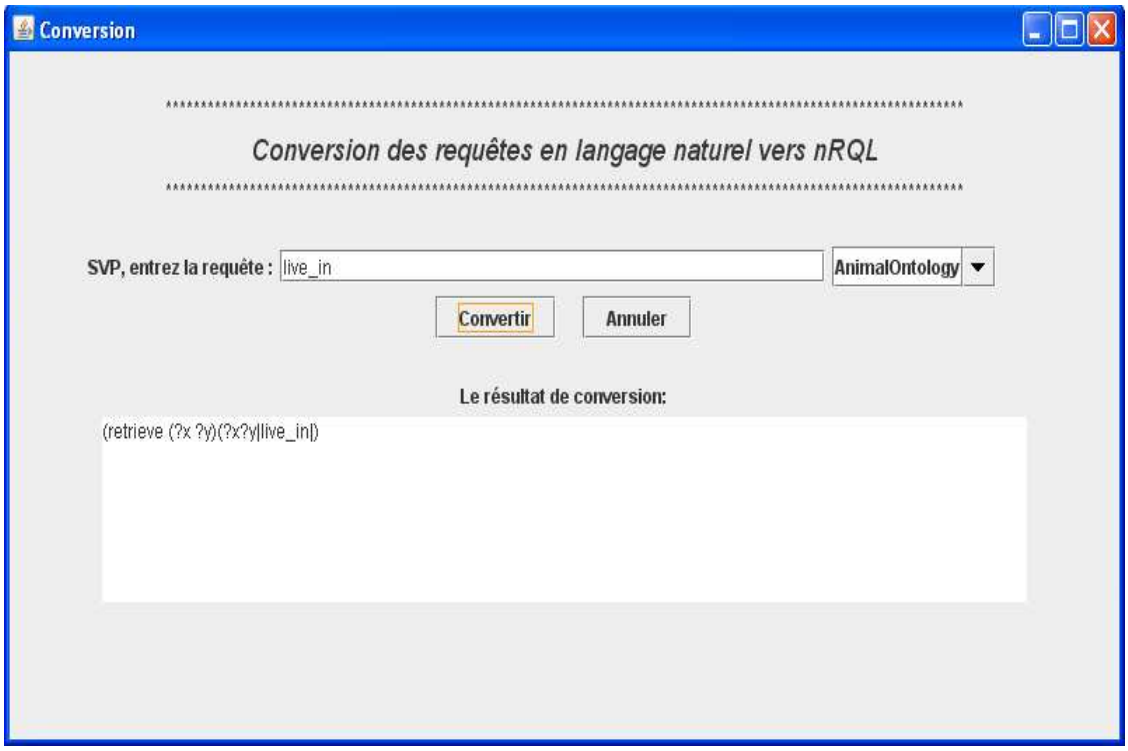

Fig.24 : requête propriété d'objet

☞ ***Requête complexe avec un terme ambigu:***

L'utilisateur saisit la requête : «aquatic animal consumes the planktons» avec le nom de l'ontologie AnimalOntology. Parce que la requête est longue, on doit parcourir toutes les étapes du système :

Fig.25 : requête initiale

1. Le décomposeur de requêtes décompose la requête on cinq composants : aquatic / animal / consumes / the / planktons
2. Le module de Mapping va supprimer les mots vides et identifier pour chaque terme de la requête ses correspondants dans l'ontologie.

Aquatic → instance$_1$ (Aquatic ∈ l'ontologie)

Animal → class$_1$ (Animal ∈ l'ontologie)

Consumes → consumes ∉ l'ontologie; on doit donc vérifier l'appartenance de ses synonyme.

The → mot vide$_1$

Planktons → instance$_2$ (Plancktons ∈ l'ontologie)

Parce que le terme consumes n'appertient pas à l'ontologie, nous devons donc vérifier l'appartenance de ses synonymes. En utilisant WordNet 2.1 nous avons trouvé 18 synonyme **(Fig.26)**, parmi ces derniers nous avons selectionné *eat* qui est le seul synonyme qui appartient à l'ontologie. Par conséquent, le terme ***consumes*** sera remplacé par son correspondant dans l'ontologie ***eat.***

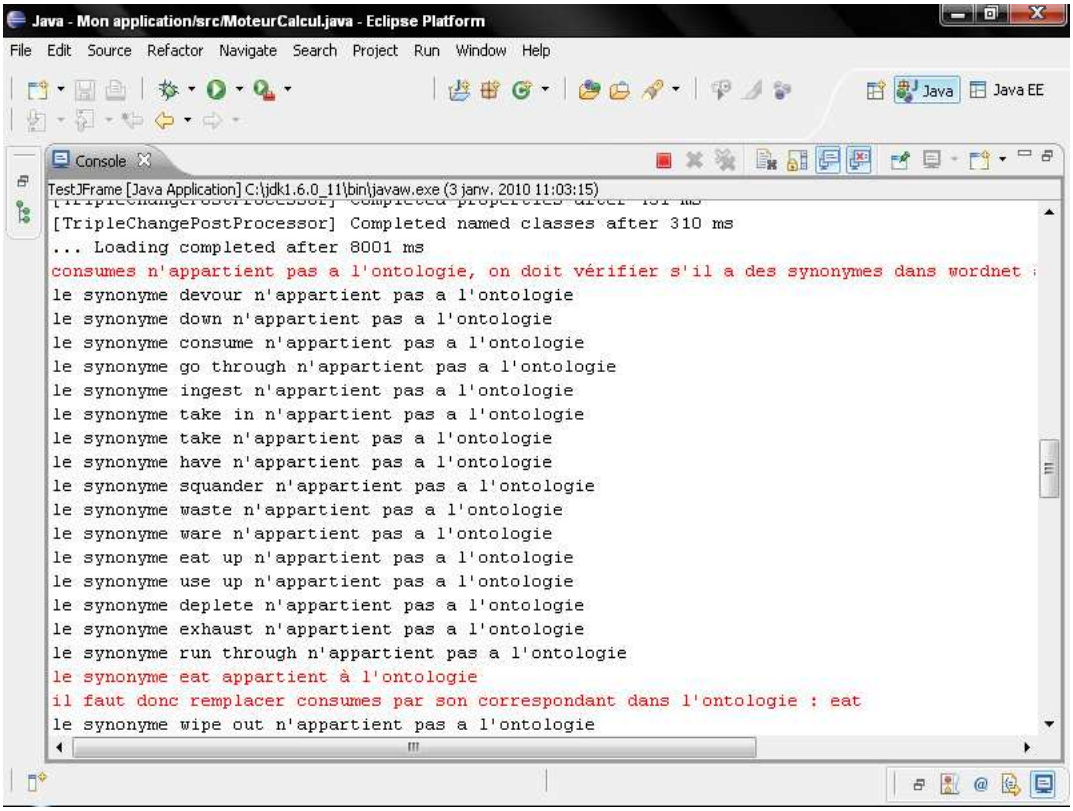

Fig.26 : les synonymes de consumes

3. Après le mapping des termes de la requête vers les concepts de l'ontologie, l'extracteur de triplets va identifier tous les triplets possible de la requête :

Triplet extrait	*Type de triplet*
(Aquatic , _ , Animal)	(instance, vide, class)
(Aquatic , _ , Planktons)	(instance, vide, instance)

(Animal , _, Planktons)	(class, vide, instance)
(Aquatic , Eat, Animal)	(instance, object_property, class)
(Aquatic , Eat, Planktons)	(instance, object_property, instance)
(Animal , Eat, Planktons)	(class, object_property, instance)
(Aquatic, Eat,_)	(instance, object_property, vide)
(Animal, Eat, _)	(class, object_property, vide)
(Planktons, Eat, _)	(instance, object_property,vide)

Tableau.6 : les triplets de la requête

4. Enfin, le générateur de requêtes nRQL traduit les triplets précédents vers le langage nRQL en appliquant les trois étapes de l'algorithme de génération des requêtes nRQL :

   a) Liaison entre les composants de chaque triplet et la mise dans la forme <domaine, propriété, co_domaine>:

   - Le triplet (Aquatic , _ , Animal) est de type relation vide et les arguments sont une classe et une instance reliées par la propriété d'objet trouvée *Live_in.* Donc, la forme valide du triplet est : <*Animal, Live_in , Aquatic*>.
   - Le triplet (Animal , _, Planktons) est de type relation vide et les arguments sont une classe et une instance reliées par la propriété d'objet trouvée *Eat.* Donc, la forme valide du triplet est : <*Animal,* Eat *, Planktons*>.
   - Le triplet (Animal , Eat, Planktons) est de type propriété d'objet dont le domaine est Animal et le co_domaine est la classe de Planktons . Donc, la forme valide du triplet est : < *Animal,* Eat *, Planktons*>.
   - Le triplet (Animal, Eat, _) est de type propriété d'objet avec un domaine connu (Animal). Donc, la forme valide du triplet est : < *Animal, Eat , VAR*>.

☞ Le triplet (Planktons, Eat, _) est de type propriété d'objet avec un co_domaine connu (Planktons). Donc, la forme valide du triplet est : < *VAR, Eat ,* Planktons >

Les autres triplets sont supprimés parce qu'il n'existe aucune relation sémantique entre leurs composants.

b) Liaison entre les triplets de la requête par des conjonctions, et suppression des redondances:

< *Animal, Live_in , Aquatic*>***and*** < *Animal,* Eat *, Planktons*> ***and*** < *Animal,* Eat *, Planktons*> ***and*** < *Animal, Eat , VAR*> ***and*** < *VAR, Eat ,* Planktons >

On voit que < *Animal,* Eat *, Planktons*> est répété et plus restreint que < *Animal, Eat , VAR*> et que < *VAR, Eat ,* Planktons >, alors on les supprime pour obtenir la forme intermédiaire optimale suivante :

< *Animal, Live_in , Aquatic*> ***and*** < *Animal,* Eat *, Planktons*>

c) Traduction en nRQL en appliquant les règles de génération :

Animal → ? x | Animal |

Live_in → | Live_in |

Aquatic → | Aquatic |

Eat → | Eat |

Planktons → | Planktons |

< Animal, Live_in , Aquatic> → (?x | Animal || Aquatic || Live_in |)

< Animal, Eat , Planktons> → (?x | Animal || Planktons || Eat |)

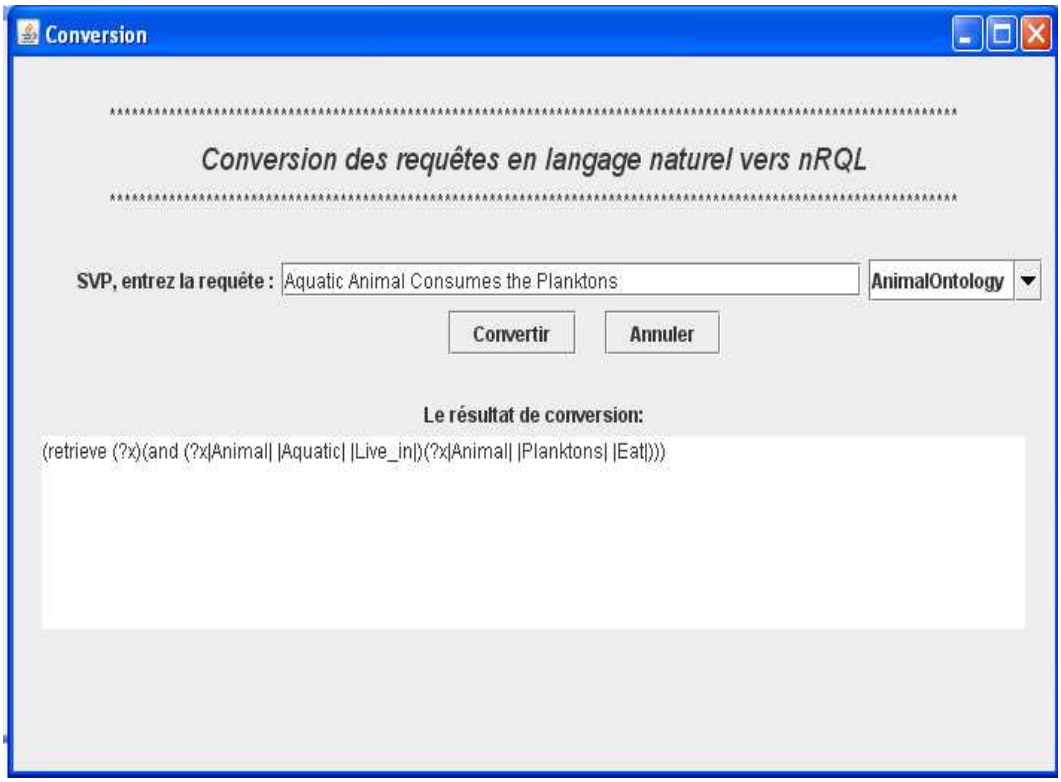

Fig.27 : Le résultat de conversion

4. Evaluation

Notre système est implémenté en JAVA, en utilisant WordNet[2] et les packages de protégé. Un utilisateur saisit sa requête en langage naturel (par exemple : *Animal with Orange Hair live in Forest*), et il obtient en revanche une requête en langage nRQL. Cette requête nRQL traduite peut ainsi être directement envoyée au raisonneur RACER pour avoir les ressources relatives. Pour évaluer notre approche, nous avons utilisé une petite ontologie sur le domaine des animaux (*AnimalOntology).* Les utilisateurs du système ont été informés du contenu et du domaine de l'ontologie, mais ils ne savaient pas nécessairement les noms des relations et la structure de l'ontologie.

Afin d'évaluer l'efficacité de notre approche, nous avons utilisé 20 requêtes différentes. Nous avons pris une métrique pour l'évaluation**: Le rappel** est défini comme le nombre des requêtes correctement traduites divisées par toutes les requêtes de l'ensemble d'évaluation (c.-à-d. 20 dans notre cas). Avec les premiers tests notre

[2] http://wordnet.princeton.edu/wordnet/

système a réalisé des résultats de conversion encourageant ; parce que nous avons obtenus un rappel de 0.9. Les travaux futurs seront axés sur le renforcement de la performance ; à savoir l'amélioration du temps de réponse et l'évaluation avec les grandes ontologies.

5. Conclusion

Dans ce chapitre, nous avons présenté une étude de cas qui consiste à construire une ontologie sur le domaine des animaux, l'ontologie que nous avons développée pour valider notre proposition à savoir le développement d'un système de conversion des requêtes en langage naturel vers nRQL. Nous avons montré comment les termes initiaux de la requête ont été représentés sous forme des termes spécifiques de l'ontologie. Ensuite, comment nous avons trouvé toutes les relations sémantiques entre ces nouveaux termes par rapport aux relations binaires de l'ontologie. Enfin, comment nous avons généré la requête nRQL approprié à la requête initiale.

Le chapitre suivant apporte quelques perspectives à la suite de la conclusion générale.

** Conclusion générale**

Les approches de la recherche d'information actuelles ne saisissent pas formellement la signification explicite d'une requête en langue naturelle ; mais fournissent une voie confortable pour l'utilisateur qui spécifie ces besoins en informations sur la base des mots-clés. La recherche sémantique promet de fournir des résultats plus précis que la traditionnelle recherche par mots-clés. Toutefois, les progrès de la recherche sémantique ont été retardés en raison de la complexité de ses langages de requête.

Nous avons abordé dans ce mémoire, la problématique de la construction des requêtes formelles et nous avons présenté une nouvelle approche pour la résoudre. Notre système vise à traduire les requêtes de langage naturel vers des requêtes nRQL ; en utilisant les restrictions sémantiques imposées par l'ontologie pour réduire l'écart entre la recherche sémantique basée sur les ontologies et les utilisateurs finaux habitué à utiliser les moteur de recherche classique basé sur les mots-clés. Notre approche peut se résumer ainsi : Une fois que l'utilisateur entre une requête en langage naturel, la première étape réalisée est d'analyse linguistique des requêtes, cette étape permet de trouver les éléments correspondants dans l'ontologie en fonction des termes initiaux de l'utilisateur, cette correspondance est faite grâce au dictionnaire WordNet. Puis, l'étape de la génération de requêtes nRQL qui permet d'abord de représenter la requête sous forme des triplets <argument, relation, argument>. Ces derniers sont traduits par la suite vers le langage nRQL en appliquant l'algorithme de génération des requêtes nRQL, en se basant sur les relations sémantiques entre tous les termes de la requête. Enfin, la requête nRQL sera remise à l'utilisateur final.

Dans l'état actuel de cette étude, nous envisageons plusieurs améliorations ou extensions possibles au système proposé telles que:

1. La combinaison de plusieurs ontologies hétérogènes et distribuées sur le web dans le processus de conversion des requêtes ; il semble évident que dans la modélisation des connaissances d'un domaine, il ne faut pas être limité par une seule ontologie de domaine.
2. Elargir la portée des requêtes par l'introduction de certains opérateurs structurés (par exemple, NOT, OR,…etc.) et en améliorant le support d'interaction humaine pour traduire les besoins en informations plus complexes.
3. L'utilisation des méthodes linguistiques et des ressources lexicales dans la phase d'analyse linguistique des requêtes afin d'améliorer l'analyse morphologique, syntaxique, et sémantique des requêtes de langage naturel.

*** Bibliographie ***

[ALLI 08] S.Allioua, "Recherche sémantique d'images Annotées". Mémoire Pour obtenir le diplôme de magister en informatique, Juin 2008. Université Mentouri de Constantine.

[ANDR 95] I. Androutsopoulos, G.D. Ritchie, P. Thanisch: "Natural Language Interfaces to Databases- An Introduction". Journal of Natural Language Engineering. 29-81 (1995).

[ANTO 08] G. Antoniou, F. van Harmelen. "A Semantic Web Primer", second edition , 2008. The MIT Press Cambridge, Massachusetts London, England.

[BAAD 03] F. Baader, W. Nutt. "Basic Description Logics". Cambridge University Press, pp. 47100, 2003.

[BAKE 06] C. Baker, X. Su, G. Butler, V. Haarslev. "Ontoligent Interactive Query Tool". CSWWS 2006. University, Montreal, Quebec, Canada.

[BLAZ 98] M. Blazquez, M. Fernandez., J. M garcia-pinar, A. gomez-perez., "Building Ontologies at the Knowledge Level using the Ontology Design Environment". in Proceedings of the Banff Workshop on Knwoledge Acquisition for Knwoledge-based Systems, 1998.

[BOUMa 09] H. Boumechaal, S. Allioua, Z. Boufaida,"A new approach to query an OWL ontology", International Conference On Applied Informatics ICAI09, Bordj Bou Arréridj – Algérie, 15-17 november 2009.

[BOUMb 09] H. Boumechaal, S. Allioua, Z. Boufaida, "Conversion des requêtes en langage naturel vers nRQL ", Conférence Internationale sur l'Informatique et ses Applications CIIA'09, Saida- Alegria, 03-04 May 2009.

[BRAN 03] T.Brants. “Natural Language Processing in Information Retrieval” CLIN 2003 - Google Inc. California – USA.

[CHAN 99] B. Chandrasekaran, J. R. Josephson and V. R. Benjamins. “What are ontologies and why do we need them? ”. IEEE Intelligent Systems. 14(1):20-26. 1999.

[DIST 00] A. Dister. “Réflexion sur l'homographie et la désambiguïsation des formes les plus fréquentes”. In Actes des Journées Internationales d'Analyse Statistique des données Textuelles (JADT 2000), 2000. Université de Liège – 1b, Quai Roosevelt, B-4000 Liège – Belgique.

[ERIC 02] Eric van der Vlist “XML Schema_Publisher”: O'Reilly First Edition June 2002 ISBN: 0-596-00252-1, 400 pages

[FELL 98] C. Fellbaum, “WordNet, An Electronic Lexical Database”, Bradford Books, May, 1998.

[GAND 02] F. Gandon. “Ontology Engineering: a Survey and a Return on”. ExperienceACACIA Team Thème 3 : Interaction homme-machine, images données, connaissances Rapport de Recherche n° 4396 - March 2002.

[GRAV 00] L. Gravano. “Characterizing Web Resources for Improved Searché”. In Proceedings of the First DELOS Network of Excellence Workshop on Information Seeking, Searching and Querying in Digital Libraries, Zurich, Switzerland, Computer Science Department Columbia University. Décembre. 2000.

[GRUB 93] Thomas R. Gruber. “Formal ontology in conceptual analysis and knowledge representation”. Chapter: Towards principles for the design of ontologies used for knowledge sharing. Kluwer Academic Publishers. 1993.

[GUAR 99]: N. Guarino, C. Masolo, and G. Vetere. “OntoSeek : content-based access to the web”. *IEEE Intelligent Systems, 1999.*

[GUEV 06] Sinuhé David Hernandez Guevara. "Méthodologies et Outils pour la Construction collaborative des ontologies″ . Mémoire pour obtenir le diplôme de Master Recherche en Informatique juin 2006.

[HAAR 01] V. Haarslev and R. MÖller "RACER System Description". University of Hamburg, Computer Science Department Vogt-KÖlln-Str. 30, 22527 Hamburg, Germany 2001.

[HAAR 04] V.Haarslev, R. MÖller, M, Wessel,.: "Querying the Semantic Web with Racer + nRQL". In: Proc. of the KI-04 Workshop on Applications of Description Logics 2004, ADL '04.

[HABE 01] : B. Habert, L. Monceaux " WordNet, la mère (le père) de tous les réseaux de mots ? ". Disponible sur : http://www.limsi.fr/Individu/habert/0001/SeminaireLMonceaux290501.ppt, rapport LIR, Mai 2001.

[HEMA 05] M. Hemam, "Un processus de développement d'ontologies dans le cadre du web sémantique". Mémoire Pour obtenir le diplôme de magister en informatique .Centre Universitaire Larbi Ben M'hidi -Oum El Bouaghi- Institut des sciences exactes, 2005.

[JACQ 00] C. Jacquemin, P.Zweigenbaumy. Chapitre quatrième : " Traitement automatique des langues pour l'accès au contenu des documents? " Le document en sciences du traitement de l'information, chapitre 4, pages 71–109. Cepadues, Toulouse, 2000 .

[JIAN 97] J. Jiang, D. Conrath. "Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy". The International Conference Research on Computational Linguistics (ROCLING X), 1997.

[KAMP 02] J.Kamps. "Visualizing WordNet Structure". Institute for logic, language and computation university of amsterdam nieuwe achtergracht 166 1018wv amsterdam the netherlands 2002.

[KARV 02] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl : "Rql: a declarative query language for rdf". The 11th International World Wide Web Conference. 592--603 (2002).

[KAUF 06] E. Kaufmann, A. Bernstein, R. Zumstein.: "Querix: A natural language interface to query ontologies based on clarification dialogs". In: 5th International Semantic Web Conference (ISWC 2006), Springer (November 2006).

[KOSS 06] L. Kosseim, R. SibliniI, C. Baker, et S.Bergler. "Using Selectional Restrictions to Query an OWL Ontology". International Conference on Formal Ontology in Information Systems (FOIS 2006) .Baltimore, Maryland (USA), November 9-11, 2006.

[LACO 05] X. Lacot. "Introduction à OWL, un langage XML d'ontologies Web". Projet à l'Ecole Nationale Supérieure des Télécommunications. Juin 2005.

[LAFF 01] J. Lafferty, C. Zhai, "Document language models, query models, and risk minimization for information retrieval". Proceedings of the ACM SIGIR '01 conference, 2001, p. 111-119.

[LAUB 02] P. Laublet, C. Reynaud, J. Charlet "Sur quelques aspects du Web sémantique". Actes des deuxièmes assises nationales du GdR I3 2002.

[LEE 01] T. Berners-Lee, J. Hendler & O. Lassila. "*The Semantic Web*". In Scientific American. 2001

[LEI 06] Y. Lei, V. Uren, E. Motta. "Semsearch: a search engine for the semantic web". In: Managing Knowledge in a World of Networks, Springer Berlin / Heidelberg (2006).

[LIN 98] D. Lin. "An information-theoretic definition of similarity".15th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, 1998,pp. 296-304.

[LINN 35] C. Linnaeus. "*Systema Naturae* published in 1735 in the Netherlands http://books.google.com/books?id=Ix0AAAAAQAAJ&pg=PA1

[LOPE 05] V.Lopez, M. Pasin, E. Motta. "Aqualog: An ontology-portable question answering system for the semantic web". In: Proceedings European Semantic Web Conference (ESWC), Crete (2005) 546–562.

[LOPE 06] V. Lopez, E. Motta, V. Uren. "Poweraqua: Fishing the semantic web". In: Proceedings European Semantic Web Conference (ESWC), Montenegro (2006).

[MCGU 04] D.L. McGuinness, F. van Harmelen. "Vue d'ensemble du langage d'ontologie Web OWL", http://www.yoyodesign.org/doc/w3c/owl-features-20040210/.

[MILL 95] G. Miller. "Wordnet: A lexical database". Communication of the ACM, 38(11):39--41, 1995.

[NAPO 97] A. Napoli : "Une introduction aux logiques de descriptions". Thème 3, Interaction homme-machine, images, données, connaissances Projet SYCO. Rapport de recherche n° 3314, Décembre 1997, 72 pages.

[PONTE 98] J. M. Ponte, W. B. Croft. "A Language Modeling Approach to Information Retrieval ". In 21st annual international ACM SIGIR conference on Research and Development in Information Retrieval, p. 275–281, Melbourne, Australia 1998.

[PRUD 08] E. Prud'hommeaux, A. Seaborne. "SPARQL Query Language for RDF". W3C recommandation, 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[RACE 05] "RacerPro User's Guide Version 1.9". Racer Systems GmbH & Co. KG http://www.racer-systems.com December , 2005

[SEAB 04] A. Seaborne: “RDQL - A Query Language for RDF” .W3C Member Submission , 2004 http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

[SOWA 99] John F. Sowa. “Knowledge representation: Logical, philosophical and computational foundations”. Brooks Cole Publishing Co., Pacific Grove, CA USA. 1999.

[TABL 08] V. Tablan, D. Damljanovic, and K. Bontcheva: “A Natural Language Query Interface to Structured Information”. In Proceedings of the 5h European Semantic Web Conference (ESWC 2008), Tenerife, Spain, June, 2008.

[TANN 06] X.Tannier. “ Extraction et recherche d'information en langage naturel dans les documents semi structurés ”. Thèse pour obtenir le grade de Docteur de l'Ecole Nationale Supérieure des Mines de Saint-Étienne l'Université Jean Monnet, septembre 2006.

[TODI 01] A. Todirasçu, F. Rousselot. “Ontologies for Information Retrieval”. TALN 2001, Tours, 2-5 juillet 2001

[USCH 96] M. Uschold, M Gruninger. “Ontologies: Principles, methods and applications”, to appear knowledge ingeniring review the University of Edinburgh 1996.

[WALM 07] P. Walmsley. “XQuery ”. Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. April 2007.

[WANG 07] C. Wang, M. Xiong, Q. Zhou , Y. Yu : “PANTO: A Portable Natural Language Interface to Ontologies”. 4th European Semantic Web Conference (ESWC2007).

[WATT 02] A. Watt. “XPath Essentials”. John Wiley & Sons, Inc.2002

[WELT 01] C. Welty, N. Guarino. "Supporting ontological analysis of taxonomic relationships". Data Knowledge Engineering. 39(1):51-74. 2001.

[WESS 05] M. Wessel and R. M¨oller. "A High Performance Semantic Web Query Answering Engine". Hamburg University of Technology (TUHH) Hamburg-Harburg, Germany 2005.

[WU 94] Z. Wu, M. Palmer (1994). "Verb Semantics and Lexical Selection". The 32nd. Annual Meeting of the Association for Computational Linguistics.

[YANG 05] D. Yang, D. M. W. Powers (2005). "Measuring semantic similarity in the taxonomy of wordnet". In ACSC '05 : Proceedings of the Twenty-eighth Australasian conference on Computer Science, Darlinghurst, Australia, Australia, pp. 315–322. Australian Computer Society, Inc.

[ZWEI 01] P. Zweigenbaum, N. Grabar , S. Darmoni. "L'apport de connaissances morphologiques pour la projection de requêtes sur une terminologie normalisée".TALN 2001, Tours, 2-5 juillet 2001.

*** Glossaire ***

Ce glossaire regroupe et donne la définition des principaux termes utilisés dans ce mémoire :

DL : Description Logic

OWL: Ontology Web Langage

nRQL : new Racerpro query Language

RACER: Renamed Abox and Concept Expression Reasoner

RDF: Resource Description Framework

RDF(S): Resource Description Framework Schema

RDQL : RDF Data Query Language

RI : Recherche d'Information

SPARQL : SPARQL Protocol And RDF Query Language

SRI : Système de Recherche d'Information

TALN : Traitement Automatique des Langues Naturelles

URI : Uniform Resource Identifier

W3C: World Wide Web Consortium

XML : eXtensible Markup Langage

XMLS : eXtensible Markup Langage Schema

*** Annexe A***

Program Conversion;

```
// les packages de protégé
import edu.stanford.smi.protege.model.DefaultKnowledgeBase;
import edu.stanford.smi.protegex.owl.ProtegeOWL;
import edu.stanford.smi.protegex.owl.jena.JenaOWLModel;
import edu.stanford.smi.protegex.owl.model.OWLClass;
import edu.stanford.smi.protegex.owl.model.OWLIndividual;
import edu.stanford.smi.protegex.owl.model.OWLModel;
import edu.stanford.smi.protegex.owl.model.OWLNamedClass;
import edu.stanford.smi.protegex.owl.model.OWLProperty;
import edu.stanford.smi.protegex.owl.model.RDFIndividual;
import edu.stanford.smi.protegex.owl.model.RDFProperty;
// les packages deWordNet
import edu.smu.tspell.wordnet.NounSynset;
import edu.smu.tspell.wordnet.Synset;
import edu.smu.tspell.wordnet.SynsetType;
import edu.smu.tspell.wordnet.WordNetDatabase;
import edu.smu.tspell.wordnet.impl.file.ReferenceSynset;
import edu.smu.tspell.wordnet.impl.file.synset.VerbReferenceSynset;
// verifier s'il existe une relation de subsomption entre deux classes
public static DefaultMutableTreeNode ConceptTree (RDFResource cls1, RDFResource cls2){
DefaultMutableTreeNode parent = new DefaultMutableTreeNode(cls1.getName());
for(Iterator        it=((RDFSClass) cls1).getNamedSubclasses(false).iterator(); it.hasNext();){
                RDFSClass subclass =(RDFSClass) it.next();
                if (subclass == cls2){
                        trouv=true; }
                parent.add(ConceptTree(subclass, cls2)); }
              return parent; }
// définir la classe d'une instance donnée
 public static RDFResource getInstType(String instName){
          try{
          OWLIndividual inst = owlModel.getOWLIndividual(instName);
          return inst.getRDFType();
          }catch(Exception e){
    System.out.println("erreur dans getInstType("+instName+")");
```

```
                    return null; } }
/*****************************Charger l'ontologie*********************************/
        try{
String uri = "file:///F:/ontology/AnimalOntology.owl"; // l'adresse de l'ontologie
owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri);
}catch(Exception exception){
System.out.println("testeOWLUtils: Erreur dans le chargement de l'ontologie");
System.out.println("testeOWLUtils: l'ontologie n'existe pas ? ...");
System.exit(1); }
  /**************Définir les types des composants par rapport aux entités de l'ontologie*********/
LinkedList<String> argument = new LinkedList(); //liste des arguments
LinkedList<String>  relation = new LinkedList(); // liste des relations
for (int i=0 ; i< list1.size() ;i++){
try{
RDFResource resource = owlModel.getRDFResourceByBrowserText(list1.get(i));
//le terme est de type instance
if(resource.toString().startsWith("SimpleInstance")){
argument.add(list1.get(i)); }
//le terme est de type classe
else if(resource.toString().startsWith("Cls"))          {
argument.add(list1.get(i)); }
//le terme est de type relation
else if(resource.toString().startsWith("Slot"))          {
relation.add(list1.get(i)); }
}catch(Exception e){
System.out.println ( list1.get(i) + " n'appartient pas a l'ontologie , on doit vérifier s'il y a des synonymes dans wordnet appartiennent à l'ontologie ");
//chercher les synonymes dans WordNet pour ce terme
System.setProperty("wordnet.database.dir", "C:/WordNet/2.1/dict/");
WordNetDatabase database = WordNetDatabase.getFileInstance();
Synset[] synsets = database.getSynsets(list1.get(i));
// sélectionner les synonymes qui appartiennent à l'ontologie
for (int c=0 ; c< synonym.size() ;c++){
try{
RDFResource resource1 = owlModel.getRDFResourceByBrowserText(synonym.get(c));
//le synonyme est de type instance
if(resource1.toString().startsWith("SimpleInstance")){
argument.add(synonym.get(c));
```

```
System.err.println ( "le synonyme " + synonym.get(c)+ " appartient à l'ontologie");
System.err.println (  "il faut donc remplacer "+ list1.get(i)+" par son correspondant dans l'ontologie :
"+synonym.get(c)); }
  //le synonyme est de type classe
else if(resource1.toString().startsWith("Cls")) {
argument.add(synonym.get(c));
System.err.println (  "le synonyme " + synonym.get(c)+ " appartient à l'ontologie");
System.err.println (  "il faut donc remplacer "+ list1.get(i)+" par son correspondant dans l'ontologie :
"+synonym.get(c)); }
//le synonyme est de type relation
else if(resource1.toString().startsWith("Slot"))          {
relation.add(synonym.get(c));
System.err.println (  "le synonyme " + synonym.get(c)+ " appartient à l'ontologie");
System.err.println (  "il faut donc remplacer "+ list1.get(i)+" par son correspondant dans l'ontologie :
"+synonym.get(c));
}catch(Exception e1){
System.out.println ( "le synonyme " + synonym.get(c) + " n'appartient pas a l'ontologie ");}}}}}

/*****Trouver les relations sémantiques entre les composants de chaque triplet de la requête ******/
for (int t=0; t< nbtriplet; t++){
/********************** la relation du triplet est vide******************/
if (triple[t][1]==" "){
RDFResource  resource1  =  owlModel.getRDFResourceByBrowserText(triple[t][0]);//le  premier
argument
RDFResource  resource2  =  owlModel.getRDFResourceByBrowserText(triple[t][2]);//le  deuxième
argument
/*****************les deux arguments sont des classes********************/
if ((resource1.toString().startsWith("Cls")) && (resource2.toString().startsWith("Cls"))){
// on doit vérifier s'il y a une relation de subsomption

// sinon on doit vérifier s'il y a une propriété d'objet}

/************un des arguments est une classe et l'autre une instance**********/
if ((resource1.toString().startsWith("Cls")) && (resource2.toString().startsWith("SimpleInstance"))){
//on doit vérifier d'abord si cette instance appartient à la classe

// sinon trouver une propriété d'objet}

else                    if              ((resource1.toString().startsWith("SimpleInstance"))              &&
(resource2.toString().startsWith("Cls"))){
//on doit vérifier d'abord si cette instance appartient à la classe
```

```
// sinon trouver une propriété d'objet}

 /*****************les deux arguments sont des instances*****************/
 if                    ((resource1.toString().startsWith("SimpleInstance"))                    &&
 (resource2.toString().startsWith("SimpleInstance"))){

 // on doit vérifier s'in existe une propriété d'objet entre les classes des instances}

 }else {
 /*************** la relation du triplet n'est pas vide*********************/
 /******************** un des arguments est vide*********************/
 if (triple[t][2]==" "){
 RDFResource argexiste =  owlModel.getRDFResourceByBrowserText(triple[t][0]);       // l'argument
 existe
 RDFResource relationexiste =  owlModel.getRDFResourceByBrowserText(triple[t][1]); // la rtelation
 existe
 // l'argument existe est une instance
  if ((argexiste.toString().startsWith("SimpleInstance"))) {
 // on doit vérifier si cette instance est un domaine ou co_domaine de la relation
 // l'argument existe est une classe

 }else if (argexiste.toString().startsWith("Cls")){

// on doit vérifier si cette classe est un domaine ou co_domaine de la relation}

 }else{
 /********************* les deux arguments existent*********************/

 // on doit verifier si le triplet est valide}}

 /*************************suppression des redondances*****************************/

 //pour les variables

 //pour les instances

//pour les classe

// pour les doublons

 /************************génération de la requête nRQL finale*****************************/
 String[] var = { "x", "y", "z", "w", "t", "v","o","p"};// les variables
                int v=0;
                int fo=Ntriple*3;
                String fi[][]= new String [fo][2];
                int f=0;
                for (int o=0; o< Ntriple; o++){
```

```
                if (TRIPLE[o][0]=="variable"){
                    fi[f][0]="variable"+ v;
                    fi[f][1]="?"+var[v];
                     v++;
                              }else {
RDFResource resource =  owlModel.getRDFResourceByBrowserText(TRIPLE[o][0]);
if (  resource.toString().startsWith("SimpleInstance")) {
                                        fi[f][0]=TRIPLE[o][0];
                                fi[f][1]="|"+TRIPLE[o][0]+"|";
                                        }else
                  if (  resource.toString().startsWith("Cls")) {
                     fi[f][0]=TRIPLE[o][0];
                              fi[f][1]="?"+ var[v]+ "|"+TRIPLE[o][0]+"|";
                                    v++;}}
                f++;
                fi[f][0]=TRIPLE[o][1];
                     fi[f][1]="|"+TRIPLE[o][1]+"|";
                     f++;
                if (TRIPLE[o][2]=="variable"){
                    fi[f][0]="variable"+ v;
                    fi[f][1]="?"+var[v];
                    v++;
                              }else {
RDFResource resource =  owlModel.getRDFResourceByBrowserText(TRIPLE[o][2]);
if (  resource.toString().startsWith("SimpleInstance")) {
                                fi[f][0]=TRIPLE[o][2];
                                fi[f][1]="|"+TRIPLE[o][2]+"|";
                                }else
                  if (  resource.toString().startsWith("Cls")) {
                              fi[f][0]=TRIPLE[o][2];
                              fi[f][1]="?"+ var[v]+ "|"+TRIPLE[o][2]+"|";
                              v++;}}
                f++;}
String generation=" ";
if ( Ntriple==1){
generation= "("+ TRIPLE[0][0] + " " + TRIPLE[0][2] + " " + TRIPLE[0][1]+ ")";
requetenRQL="("+ "retrieve (?x)"+generation+")";
}else{
```

```
for (int o=0; o< Ntriple; o++){
generation=  generation+  "("+ TRIPLE[o][0] + " " + TRIPLE[o][2] + " " + TRIPLE[o][1]+ ")"; }
requetenRQL="("+ "retrieve (?x)"+"("+ "and"  +generation+")"+ ")";
                                }}  }
                                        System.out.println (" la requête nRQL finale");
                                         System.out.println (requetenRQL);
```