Grabmann Bertrand

UNIVERSITE DE FRANCHE - COMTE
25 000 Besançon- France

Mémoire de Master 2 à distance, Recherche en Informatique.
Equipe CARTOON

femto-st SCIENCES & TECHNOLOGIES

Sous la direction de
MARIE-LAURE BETBEDER
Chercheur à Femto-ST (DISC), équipe CARTOON

Interrogation des Ontologies

Résumé

Les ontologies sont de plus en plus utilisées pour le stockage et l'interrogation de données ordonnées et complexes offrant des possibilités de raisonnement qui ne sont pas offertes par des bases de données classiques ainsi que de grandes capacités d'interconnections entre elles, capacités notamment grandement utilisées dans le web sémantique, en génie logiciel ou encore dans divers domaines constituant une forme de représentation, de formalisation de la connaissance au sujet d'un monde ou d'une certaine partie de ce monde.

L'interrogation des données avec SPARQL et l'expression des requêtes s'avère vite complexe, notamment dans la recherche des instances, propriétés et concepts représentés dans une ontologie qui peut en comprendre des centaines de milliers sinon plus.

Dans ce mémoire, nous établirons un état de l'art de l'existant en matière de création de requête assistée d'ontologie et proposerons un outil d'aide à la construction de requêtes. Nous montrerons que celui-ci permettra de se familiariser de manière rapide et instinctive à une ontologie, et ce, de manière générique.

Abstract

Ontologies are used more and more for storage et querying of organized and complex datas offering reasonning possibilities which are not given by classical databases as well as important interconnection abilities between them, abilities substancially used in semantic web, software engineering or in varied domains constituting a form of representation, of formalisation of knowledge on a world or part of it.

Datas querying with SPARQL and queries expression quickly reveals to be difficult and complex, especially in the research of instances, properties and conceptual objects depictedin an ontology which could contains hundreds of thousand or far more.

In this master's thesis we'll giving a survey concerning the domain of assisted creation of queries in ontologies et we'll propose a software helping to the construction of queries. We will show that this tool allows user to be quickly and instinctively comfortable with an ontology, and this, generically.

Mots clefs : ontologie, SPARQL, requêtes assistées, browser

Remerciements

Je tiens tout d'abord à remercier *Marie-Laure Betbeder* ainsi que l'équipe CARTOON (FEMTO) pour l'aide et le temps consacrés à l'encadrement de ce stage enrichissant en laboratoire de recherche.

Je remercie *Kitsiri Chochiang* pour sa collaboration dans l'élaboration de la partie test de ce mémoire.

Merci également à toute l'équipe pédagogique du Centre de Télé-Enseignement Universitaire de l'université de Franche-Comté qui m'a permis d'atteindre mes objectifs.

Je remercie également tout spécialement *Camelia* pour son soutien durant mes années d'études et dédicace ce mémoire de recherche à ma fille, *Linda*.

Sommaire

Table des annexes

Table des figures

Table des tableaux

Introduction

L'utilisation des ontologies est de plus en plus courante pour le stockage et l'interrogation de données complexes ordonnées et ce dans de nombreux domaines. Cette modélisation offre des possibilités de raisonnement (entre autres au niveau sémantique) qui ne sont pas couvertes par les bases de données. Les ontologies sont utilisées en intelligence artificielle, par le Web sémantique, en génie logiciel, ou encore en informatique biomédicale ou d'autres domaines comme une forme de représentation de la connaissance au sujet d'un monde ou d'une certaine partie de ce monde.

L'expression des requêtes peut s'avérer complexe d'autant plus si l'on utilise une union d'ontologies avec des agrégats et des filtres. Au sein du laboratoire CARTOON de l'université de Franche-Comté, le travail proposé est de développer un outil d'interrogation d'ontologies permettant d'exprimer des requêtes complexes, cet outil guidant la rédaction de requêtes SPARQL en fournissant une aide contextuelle en cours de saisie issue d'un raisonnement sur l'ontologie cible, et ce, pour n'importe quelle ontologie.

Nous effectuerons dans un premier temps une présentation de SPARQL, langage de requêtage et protocole au niveau sémantique et syntaxique. Puis, nous effectuerons un état des lieux nous permettant de situer quelques méthodes ayant trait à la simplification des écritures de requêtes, d'une part les méthodes textuelles se servant d'analyse syntaxique et sémantique, d'autre part les méthodes graphiques . Nous mettrons en évidence qu'un compromis est alors à effectuer entre le taux de décision de l'utilisateur et le taux d'erreur possible. Sur ces bases, nous proposerons ensuite une technique différente permettant de composer les requêtes à l'aide d'un browser contextuel des IRI présentes dans l'ontologie. Nous testerons ensuite cette technique sur une ontologie différente de celle sur laquelle est basée la conception afin d'évaluer la mise en confort d'un utilisateur devant celle-ci pour en extraire des informations.

En préambule nous définirons tout d'abord ce qu'est une ontologie et ensuite les principaux moyens utilisés pour représenter une ontologie informatique dans un système d'information.

Définitions préalables

Avant d'aborder notre étude, des définitions préalables sont nécessaires. Nous devons savoir sur quoi nous travaillons, les sources d'extraction de nos données, les notions conceptuelles et les réflexions ayant permis d'aboutir à ce type de structure , les différents moyens de coder ces notions et leurs propriétés.

A – Qu'est ce qu'une ontologie ?

Dans son article, Viinikkala [art-1] souligne la différence fondamentale entre l'Ontologie philosophique avec l'ontologie utilisée dans le domaine des systèmes d'information.
Selon son article, l'Ontologie (avec un grand O) philosophique avait pour but une catégorisation exhaustive des entités dans toutes les sphères de la réalité,le but recherché étant la vérité. L'ontologie dans les SI a par contre une visée fonctionnelle et destinée à un ou plusieurs buts, plus locale donc que l'Ontologie philosophique . Elle a pour but le partage et la réutilisabilité dans le sens d'un *engagement ontologique*, c'est à dire un accord pour utiliser un vocabulaire et créer des assertions qui respectent la théorie spécifiée par une ontologie.

Dans l'article de Quine, [art -2], on ne peut parler d'engagement ontologique qu'en *affirmant* une théorie (assertion, : 'il existe'). Ainsi dans une proposition de type type $(\exists x)F(x)$ est affirmée l'existence d'un individu x dont il est dit qu'il possède la propriété F.

Une théorie a des conditions de vérité. Ces conditions de vérité nous décrivent comment le monde doit être pour que la théorie soit vraie. Elles établissent des exigences sur le monde. Parfois, même souvent, elles exigent du monde (i.e. univers à temps t des variables) que certaines entités ou classes d'entités existent.
L'engagement ontologique d'une théorie est l'ensemble de ces entités ou classes d'entités qui doivent exister pour que la théorie soit vraie.
J. Vidal Rosset complète cette définition dans son livre « Pour introduire à la lecture de Quine (2006) » en écrivant « Le critère d'engagement ontologique ne dit pas ce qui existe. Il permet de dire ce qu'une théorie assume comme existant. »

Guarino [art -3] formalise cette notion et assimile cet engagement ontologique à *une structure intensionnelle de premier ordre* :

- L, un langage logique de premier ordre
- V, un vocabulaire constitué de prédicats et de constantes
- C=(D,W,R) une *structure relationnelle en intension* (conceptualisation)
    - D : univers du discours
    - W : un ensemble des mondes possibles
    - R un ensemble de relations conceptuelles (relations intensionnelles) sur le domaine <D,W>

    Une relation intensionnelle pn d'arité n sur <D,W> est une fonction totale de W dans l'ensemble des relations n-aires sur D.

L'engagement ontologique est un tuple K=(C,I) où I (appelée fonction d'interprétation intensionnelle) est une fonction
$V \rightarrow D \cup R$ qui établit un mapping de chacun des symboles du vocabulaire à ,soit un élément de D ,soit à une relation intensionnelle appartenant au set R.

Pour ensuite poursuivre avec l'idée de Viinikala, l'ontologie est ensuite décrite de manière formelle dans un langage défini.
Il mentionne ensuite différents domaines d'utilisation de l'ontologie informatique :

- Etablissement de schémas conceptuels dans les architectures de bases de données.
- Etablissement de modèles d'applications dedomaines dans l'ingéniérie logicielle

- Création des modèles de classes dans la POA.
- Ecriture de logiciels.
- Extraction et interrogation de sources d'informations sur internet (ex WEB sémantique).
- Utilisation dans les traductions naturelles et linguistiques, aide contextuelle et élimination des ambigüités.
- Harmonisation et interopérabilisation des données dans les multinationales.
- Génération automatique d'ontologies à partir de deux ou plusieurs ontologies

Une ontologie informatique se propose de décrire, sous forme assimilable par une machine, un domaine du savoir ce qui nécessite un choix de conceptualisation. Cette définition nous rapproche de celle donnée [art -3] par Guarino et Al en 2009 la décrivant comme étant une spécification formelle, explicite d'une conceptualisation partagée.

La principale difficulté est alors de pouvoir obtenir un niveau de finesse suffisant (par enrichissement du vocabulaire ou description détaillée de concepts pertinemment extraits d'un domaine) pour se rapprocher au mieux d'une description du monde en extension par une description en intension. Cette conceptualisation (ou structure relationnelle intensionnelle selon Guarino) est composée d'un ensemble de définitions de concepts possédant des propriétés et de relations entre ces concepts.
On doit comprendre un concept indépendamment de son contexte (monde ou état à l'instant t des instances).

Plus concrètement, un système ontologique sera formellement composé :

- D'un **ensemble de concepts** (ou classes sous le logiciel protégé par exemple) hiérarchisé par une relation de subsomption (classe/ sous-classe) . Ce travail préalable nécessite une expertise du domaine étudié à partir d'un corpus ou après analyse sémantique, d'une taxonomie ou d'un thésaurus
- D'un **ensemble de relations binaires** : elles englobent les différents liens et interactions entre les concepts de l'ontologie : sous-classe de (subsomption), partie_de, instance_de, is_a, instanciation, etc…
- D'un **ensemble d'axiomes** de l'ontologie : décrivent la structure du modèle, les assertions de l'ontologie qui serviront ensuite pour le moteur d'inférence. Ces assertions (ou contraintes) ont pour but de définir ou de préciser la signification des objets de l'ontologie, les contraintes sur les attributs et les arguments de relations (restriction des domaines),les propriétés de relations (ex : transitivité, symétrie, inverse de,…)
- Les **instances** qui constituent l'univers de base (univers du discours). (définition extensionnelle de l'ontologie)
- Les **faits** qui décrivent des situations concrètes particulières entre instances.

Cette structure permettra d'une part de se donner un vocabulaire partagé pour décrire un domaine, d'un typage des données et de signature des relations ainsi que de la capacité de raisonner (inférence)..

Les principales motivations de l'ontologie sont, pour B. Chandrasekaran [art -4] *les suivantes :*

- Réutilisation des connaissances par généralisation abstraction (Reuse) pour différentes tâches, Permettre la réutilisation du savoir sur un domaine. « Par exemple, divers constructeurs de composants électroniques peuvent utiliser un vocabulaire et une syntaxe communs pour établir un catalogue qui décrit leurs produits. Les constructeurs peuvent alors partager les catalogues et les utiliser dans des systèmes de design automatisés. Cette sorte de partage accroît considérablement le potentiel de la réutilisation de la connaissance (knowledge reuse). ».

- Partage : Partager la compréhension commune de la structure de l'information ,(shared conceptualisation), standardisation, lever les ambiguïtés sur les termes . (Compréhension des concepts issus du langage naturel par exemple). « On peut partager le language de représentation des connaissances avec d'autres personnes qui ont le même besoin de représentation de la connaissance de ce domaine et en conséquent, éliminer le besoin d'établir une autre analyse du domaine de connaissance. Les ontologies partagées peuvent ainsi donc former la base d'un langage de représentation d'un domaine de connaissance spécifique. Elles nous permettent de construire des bases de connaissances qui décrivent des situations spécifiques. »

- Accord sur la conceptualisation (l'ontological commitment de Guarino), faire en sorte que les personnes et logiciels se comprennent entre eux,« clarification de la structure du domaine de la connaissance », selon l'auteur. Etant donné un domaine, son ontologie forme le cœur de tout système de représentation de ce domaine. Toujours selon *B. Chandrasekaran,* « Le répertoire de représentation des objets, relations, états, événements et processus ne disent rien au sujet de quelles classes de ces entités existent. Le 'modélisateur' de ces domaines fabrique ces 'assertions' (commitments, ou engagements de vérité). Quand nous nous déplaçons d'une ontologie de haut niveau à des niveaux taxonomiques plus bas, des assertions spécifiques aux domaines et phénomènes apparaîssent. Pour modéliser les objets sur terre on peut faire certaines assertions. Par exemple, animaux, minéraux et plantes sont des sous-catégories d'objets ; has-life(x) et contains-carbon(x) sont des propriétés d'objets et can-eat(x,y) est une relation possible entre deux objects. Ces assertions sont spécifiques aux objets et phénomènes dans ce domaine. De plus, ces 'commitments' ne sont pas arbitraires. Pour qu'ils soient utiles ils doivent reflèter une réalité sous-jacente. »

- Expliciter ce qui est considéré comme implicite sur un domaine. « L'ontologie capture la structure conceptuelle intrinsèque du domaine . Pour construire un langage de représentation d'un domaine de connaissance, nous avons besoin d'associer des termes avec les concepts et leurs relations dans l'ontologie et définir une syntaxe pour 'encoder' ces connaissances en terme de concepts et de relations ».

- Raisonner sur les données, notamment par inférence (domaine de l'AI).

- Analyser le savoir sur un domaine : c'est la base de tout système d'information cohérent . « Tout système d'information se base sur de la connaissance. Tout logiciel qui réalise quelque chose d'utile ne peut être écrit sans conception dirigée (assertion informatique ou 'commitment') à un modèle relevant du monde réel, d'entités, de propriétés et de relations dans ce monde. La structure des données et des procédures engagent implicitement ou explicitement à une structure de domaine ontologique . »
  L'auteur poursuit sur le domaine de la conception logicielle orientée objet disant « qu 'elle dépend d'une ontologie de domaine appropriée. Les objets, leurs attributs et leur procédures reflètent plus ou moins des aspects du domaine qui sont importants pour l'application. Les systèmes d'objets représentant une analyse complète et utile sur un domaine peuvent être réutilisés pour un programme applicatif différent. »

Quatre grandes catégories d'ontologies *furent distinguées par Van Heijst en 1997* [art -5] *:*

Plutôt que par leur but, Van Heijst distingua les principaux types d'ontologies par le degrè d'abstraction de leur contenu.
Voic les différents degrès, en partant du degrè le plus abstrait :

Les ontologies de représentation :

* Elles sont au niveau le plus élevé structurant les connaissances de haut niveau avec des catégories dont l'organisation dépend de réflexions philosophiques .
* Elles contiennent des objets et non des structures .
* Elles ne s'instancient pas, elles se spécialisent (elles donnent les objets les plus généraux du domaine, les autres objets en seront des spécialisations et non des instances).
* Y figure la spécification des catégories de concepts de très haut niveau (objet, événement, état, processus…).
« Elles expliquent la conceptualisation sous-jacente au formalisme de la représentation des connaissances »
* « Elles sont voulues pour être neutres en respect avec les entités du monde . » Les ontologies de domaine et les ontologies génériques reprennent les primitives définies dans cette ontologie deplus haut niveau. Un exemple de ce type d'ontologie est la « frame ontologie » utilisée dans Ontolingua.

Les ontologies génériques (Core ontologie) :

* « Elles sont similaires aux ontologies de domaine mais les concepts qu'elle définit sont considérés pour être génériques concernant beaucoup de champs. »

* « Généralement, elles définissent des concepts comme le temps, les évènements, les processus, les actions, les composants… »
* « Les concepts dans les ontologies de domaine sont souvent définis comme des spécialisations des concepts issus des ontologies génériques. » Les ontologies génériques fournissent des concepts structurant du domaine et les relations entre ces
concepts. Exemple, en médecine : concepts de diagnostic, signe, structure anatomique ; relations comme celles liées à la localisation d'une pathologie sur une structure anatomique.
* « La frontière entre l'ontologie générique et l'ontologie de domaine est imprécise mais la distinction est intuitivement compréhensible et est utile pour la construction de librairies. »

Les ontologies du domaine :

* Elle fournit les concepts du domaines tels qu'ils sont manipulés par des professionnels du domaine ; Spécification sémantiquement riche d'un domaine. Comme le dit l'auteur « elles expriment les conceptualisations qui sont spécifiques à des domaines particuliers. »
* Définitions de concepts et de relations entre ces concepts pour construire des ontologies spécialisées à partir d'ontologies plus générales (Ontologies génériques) décrivant un domaine spécifique. « Elles expriment des conceptualisations qui sont spécifiques à un domaine. »
* La différence entre le domaine de la connaissance et une ontologie de domaine réside « en ce que d'un côté le domaine de la connaissance décrit des situations factuelles dans un certain domaine, de l'autre côté, l'ontologie de domaine pose des contraintes sur la structure et le contenu du domaine de la connaissance. »

Les ontologies d'applications :

* « Elles contiennent toutes les définitions nécessaires à modéliser un savoir particulier requis pour une application particulière. »
* « Ordinairement, elles sont un mélange de concepts pris dansle domaine des ontologies génériques et des ontologies du domaine . » Toujours selon Van Heijst, « les ontologies d'application peuvent contenir des extensions de méthodes et de tâches spécifiques ».
* Vu leur spécialisation, elles ne sont pas réutilisables pour la construction d'autres ontologies.

L'utilisation effective d'une ontologie requiert un langage de représentation de celle-ci et des modules de raisonnement (ou inférence) de diverses efficacités. Ce qui nous amène à discuter de quelques formats de données utilisés.

B – Formats de représentation d'une ontologie

On distingue trois catégories de formats de langages descriptifs d'une ontologie :

- Basés sur les graphes : RDF et RDF Schéma, Topics Maps (standard supporté par l'ISO) [art-6, art -7]
- Basés sur la logique : KIF [web-1] (logique du premier ordre), KL –One [art-8], OIL, DAML+OIL[web -2], OWL (Logique descriptive, standard supporté parle W3C)
- Basés sur l'objet : UML + OCL

Pour un historique sommaire de l'évolution du partage des connaissances sur le web, en 1998 fût créé **XML** rendant possible leur structuration et leur manipulation (idéal pour la création des taxonomies). Dans la même lancée en 1999 fût créé **RDF**, un modèle de graphe (de triplets ,modèle sur lequel nous reviendrons plus longuement ci-dessous) permettant de décrire de manière formelle les ressources du net ainsi que leurs méta données.(dont l'une des syntaxes la plus utilisée est celle basée sur XML : RDF/XML.

RDF restant assez basique, il est alors étendu par **RDF-Schéma** [web -3] (ou **RDFS**) avec son vocabulaire de termes et de relations entre ces termes. (par exemple : Resource, Class, Property, type, SubClassOff, range, domain, etc…) . Il est reconnu comme un langage d'ontologie définissant :

- Des classes et des propriétés
- Des sous-classes, des sous-propriétés,

- Des domaines de définitions et des domaines images des propriétés.

DAML+OIL et **OWL** (qui dérive de DAML+OIL) étendent encore ces langages avec des primitives plus riches. RDF et RDFS ne permettent pas de raisonner ou de mener des raisonnements automatisés sur leur modèle de connaissance. **OWL** intègre, en plus des outils de RDF/RDFS des outils de comparaison des propriétés et des classes : identité, équivalence, contraire, cardinalité, symétrie, transitivité, disjonction, des classes énumérées… OWL permet aussi que l'information puisse être reçue à partir de sources distribuées notamment par la possibilité de mettre en relation des ontologies et d'importer des informations provenant explicitement d'autres ontologies. [web -4] (fig 1).

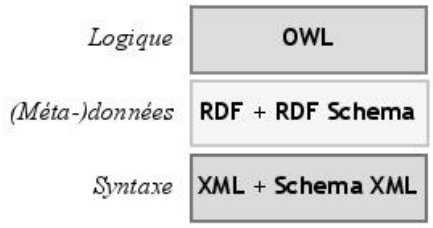


Figure 1 Les trois couches de base du Web Sémantique [web –39]

Après avoir fait le lien entre ces différents formats de description d'ontologies ou leur évolution, revenons un peu plus longuement sur le **modèle RDF/XML**.

1-RDF/XML

RDF ou Resource Description Framework est un formalisme graphique pour décrire les méta données. Un document de type RDF est un ensemble de triplets (sujet, prédicat, objet).

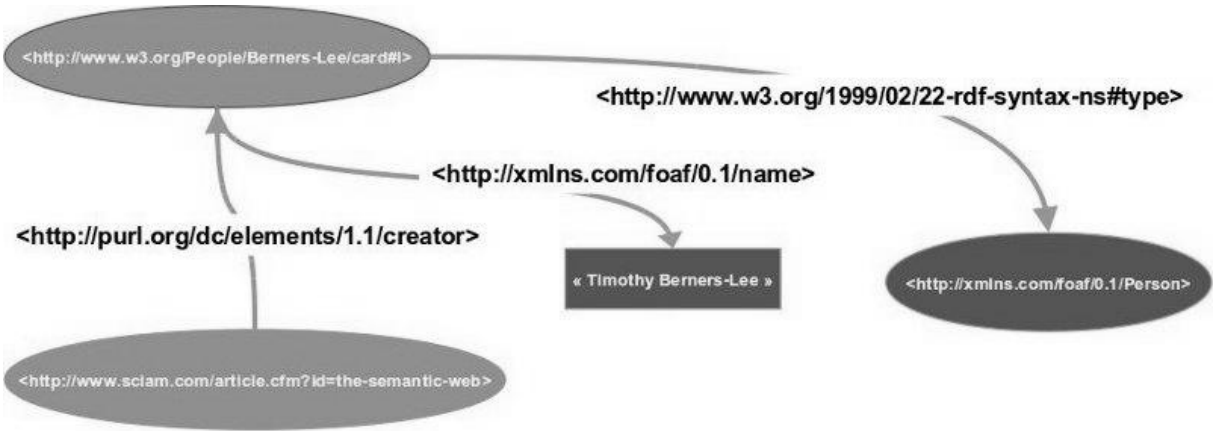


Figure 2 Exemple de Schéma RDF [web –40]

Sujet	Prédicat	Objet
<http://www.w3.org/People/Berners-Lee/card#i>	<http://xmlns.com/foaf/0.1/name>	"Tim Berners-Lee"

Tableau 1 Triplet résultat sujet ,prédicat ,objet

L'une des syntaxes de RDF est RDF/XML (celle que nous approfondirons ici) [web -5]. (on relève aussi la Notation3 N3 [web -6], Turtle [web -7], JSON-LD [web -8], etc…)
Le sujet et l'objet sont des ressources liées par le prédicat. Objet=predicat(sujet) en termes fonctionnels.

Le sujet est soit un IRI (identificateur de ressource internationalisé) ou soit un nœud anonyme.

Le prédicat est un IRI.
L'objet est soit un IRI, un nœud anonyme ou un littéral.

Illustrons concrètement le langage sur des exemples afin d'en expliquer les balises principales [web -9].

Soit l'exemple suivant :

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:si="http://www.w3schools.com/rdf/">
<rdf:Description
  rdf:about="http://www.w3schools.com">
  <si:title>W3Schools.com</si:title>
  <si:author>Jan Egil Refsnes</si:author>
</rdf:Description>
</rdf:RDF>
```

La balise <rdf:RDF> est l'élément racine de notre document RDF. Elle définira notre document xml comme étant un document RDF et contiendra les espaces de nommage (namespace). Les namespaces, définis par l'élément « *xmlns :* « permettent, dans la suite du document, de simplifier, de réduire les écritures et de clarifier ainsi la lecture du fichier.

La balise **<rdf:Description>** (déclaration) permet de dire que l'on veut décrire un **sujet** dont l'on donne l'identifiant unique.
Dans notre exemple, on veut décrire le site http://www.w3schools.com par son identifiant unique (ici une URL, c'aurait pu être une URI).

On ajoute ensuite des **propriétés** (ou prédicats).
Ici par exemple <si:title> est le prédicat concernant le sujet http://www.w3schools.com est ayant comme valeur le littéral « W3Schools.com ».

On aurait pu se référer, au lieu d'un littéral, à un autre
On aurait alors écrit
< si:title rdf:resource="http://www.examplebidon.com/titreclassiques/#titre_du_jour"/>

Avec la balise **<rdf:resource>** ,un sujet dans un document RDF peut aussi être référencé comme l'objet d'une propriété (ou prédicat) dans une autre déclaration RDF.
Ici le sujet : http://www.w3schools.com
Le prédicat : <si:title>
L'objet : rdf:resource=http://www.examplebidon.com/titreclassiques/#titre_du_jour

L'écriture peut être différente :

<si:title>W3Schools.com</si:title>

Les éléments de propriétés peuvent être décrits comme des attributs :

si:title="W3Schools.com"

ou encore comme des ressources :

<si:title rdf:resource=" http://www.w3schools.com/rdf/W3Schools.com " />

Nœuds anonymes :
Dans certains cas (les conteneurs par exemple), il est parfois nécessaire de se référer à un nœud anonyme tels qu'en position de sujet et d'objet de plusieurs triplets rdf. Dans ce cas, on donne un identificateur de nœud anonyme pour l'identifier dans le document. Dans le cas d'un nœud anonyme objet, on remplace le nœud

rdf :resource= »ref uri » par rdf:nodeID="*identificateur de nœud anonyme*" ou dans le cas d'un nœud anonyme sujet, on remplace le rdf :about= »xx » par rdf:nodeID="*identificateur de nœud anonyme*".

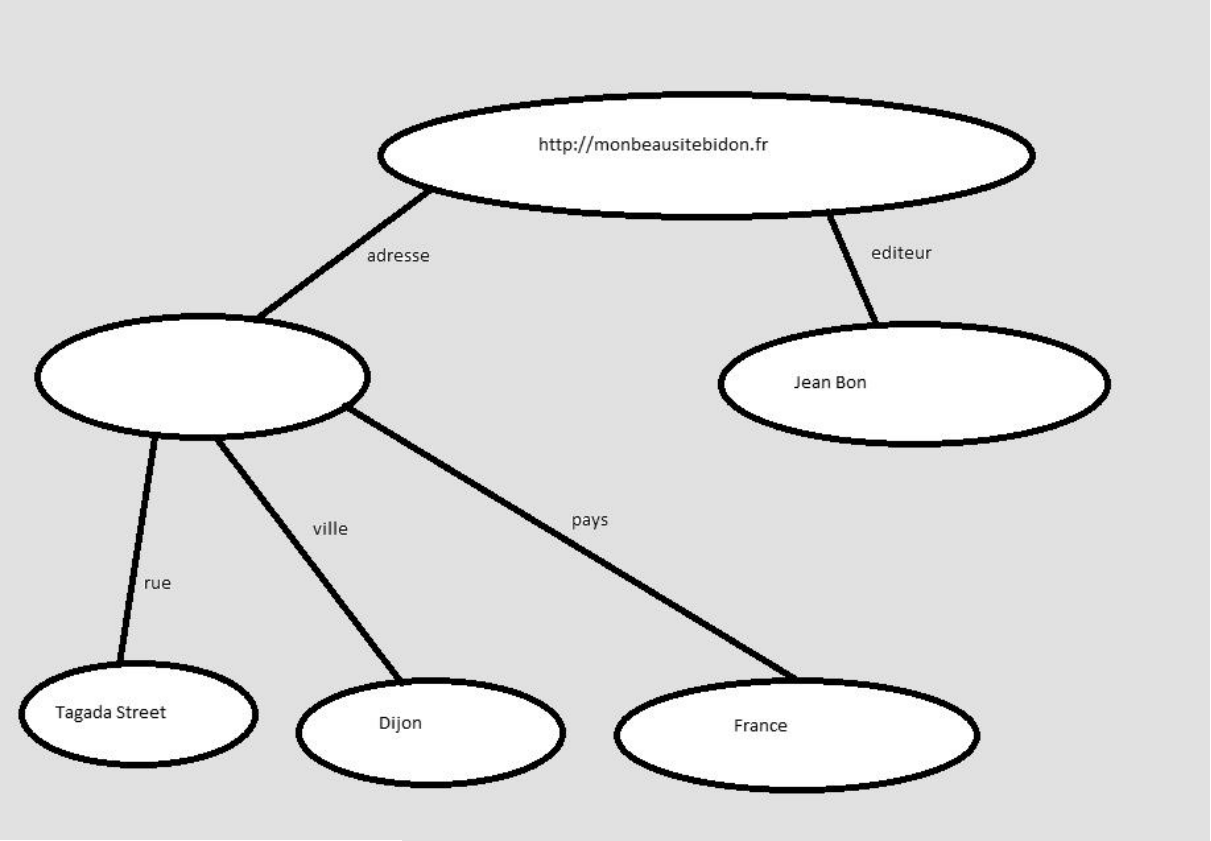


Figure 3 Illustration d'un nœud anonyme

La figure 4 illustre l'utilité des noeuds anonymes, ici dans le cas d'une adresse.

Code rdf/xml traduisant le graphe de la figure 4

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:dc="http://mesactions.fr/">

   <rdf:Description rdf:about="http://monbeausitebidon.fr"
         dc:editeur="Jean Bon">
         <dc:adresse rdf:nodeID="ano1" />  ←déclaration du noeud anonyme (faisant office ici d'objet)
   </rdf:Description>

   <rdf:Description rdf:nodeID="ano1" ← utilisation du noeud anonyme (faisant office ici de sujet)
      dc:rue="Tagada street"
      dc:ville="Dijon">
      <dc:pays rdf:resource="http://liste_de_pays.fr/France" />
   </rdf:Description>


</rdf:RDF>
```

A noter que W3 propose un validateur en ligne de code RDF/XMl :
http://www.w3.org/RDF/Validator/

Dans notre cas on obtient :

Your RDF document validated successfully.

Tableau 2 Triplets du modèle de données

Number	Subject	Predicate	Object
1	http://monbeausitebidon.fr	http://mesactions.fr/editeur	"Jean Bon"
2	http://monbeausitebidon.fr	http://mesactions.fr/adresse	genid:Uano1
3	genid:Uano1	http://mesactions.fr/rue	"Tagada street"
4	genid:Uano1	http://mesactions.fr/ville	"Dijon"
5	genid:Uano1	http://mesactions.fr/pays	http://liste_de_pays.fr/France

Les nœuds anonymes se voient ici donner un identifiant du type genid :U + marque de l'identificateur.

-RDF n'est pas typé mais permet d'associer un type à un littéral, avec **rdf :datatype**
Exemple
<cd :age rdf :datatype=http://www.w3.org/2001/XMLSchema#integer>
55
</cd:age>

- **Les conteneurs: bag,alt seq**

Un Bag (ressource identifiée par un type rdf :Bag) représente un groupe de ressources ou de littéraux (pouvant inclure des doublons) où l'ordre des membres du groupe n'a pas d'importance.

Un Alt représente un groupe de ressources ou littéraux qui sont des alternatives.

Une Seq représente un groupe de ressources ou littéraux, pouvant contenir plusieurs fois les mêmes valeurs, dans un ordre particulier.

Exemple de bag :

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:s="http://www.w3schools.com/etudiants/">

<rdf:Description rdf:about="http://exemple.fr/essais/">
   <s:etudiants>
      <rdf:Bag>
         <rdf:li rdf:resource="http://exemple.fr/etudiant/Jean"/>
         <rdf:li rdf:resource="http://exemple.fr/ etudiant /Marc"/>
         <rdf:li rdf:resource="http://exemple.fr/ etudiant /Luc"/>
      </rdf:Bag>
   </s:etudiants >
</rdf:Description>

</rdf:RDF>
```

Le graphe associé :

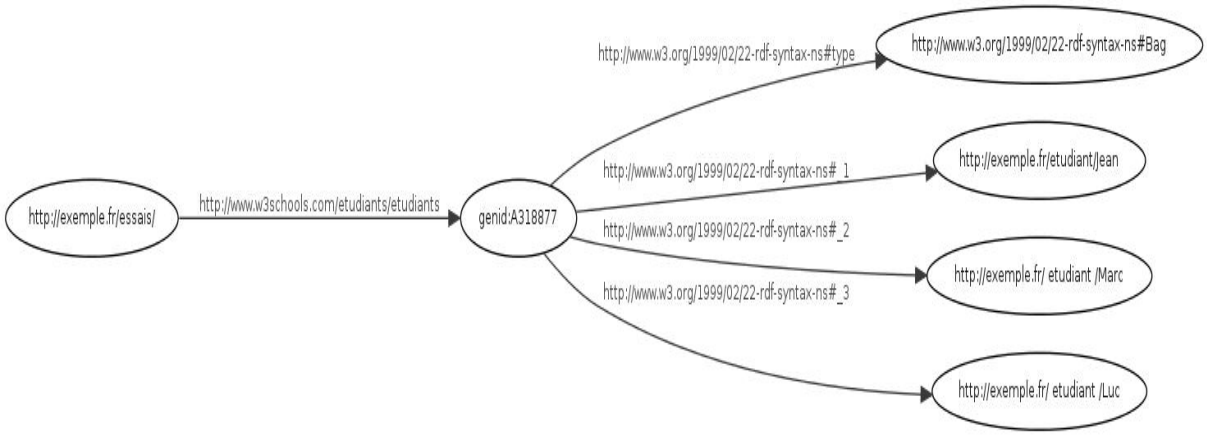


Figure 4 Illustration du conteneur de type « bag »

- **Réification :**

La réification permet de considérer un triplet comme un nœud. C'est le cas ou une déclaration est reconstituée comme la ressource d'une autre déclaration.

Exemple :

Jean déclare que Henri est créateur du site www.goshopping.com

Code associé :

```
<?xml version="1.0"?>
 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdfsyntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:s="http://www.schemas.org/schema/">
 <rdf:Description>
  <rdf:subject rdf:resource="http://www.goshopping.com"/>
  <rdf:predicate rdf:resource="http://purl.org/dc/elements/1.1/creator"/>
  <rdf:object>Henri</rdf:object>
 <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntaxns#Statement"/>
 <s:attributedTo>Jean</s:attributedTo>
 </rdf:Description>
 </rdf:RDF>
```

2-RDFS/XML

RDFS [web -10] permet de définir des vocabulaires RDF avec des structures :

- De Type : *rdfs :Class*
- De Subsomption : *rdfs :subClassOf*
- De Propriétés : *rdfs :propertyOf*
- De relations entre les propriétés : *rdfs :subPropertyOf*
- De contraintes sur les propriétés : *rdfs :domain* et *rdfs :range*
- De primitives d'annotations et d'informations : *rdfs :seeAlso, rdfs :isDefinedBy,rdfs :comment,rdfs :label.*
- Le préfixe pour RDF Schema est rdfs et son URI de référence : http://www.w3.org/2000/01/rdf-schema#

C'est un premier langage d'ontologie.

Principales caractéristiques de RDFS :

- **Classes et sous-classes :**

- Les variables d'un monde donné peuvent être regroupées en classes. La propriété **rdf :type** permet de dire qu'une ressource est instance d'une classe.
- Une classe peut spécialiser plusieurs classes (exemple : Sirène spécialise Femme et Poisson)
- L'ensemble des instances d'une classe caractérise son extension.

Une classe est elle même une ressource et les ressources qui sont des classes forment une classe appelée : **rdfs :Class**.

- On utilise **rdfs :subClassOf** pour déclarer qu'une classe est sous-classe d'une autre classe.

⇨ *cela permet déjà une forme de raisonnement simple* à savoir que si B est sous-classe de A alors toutes les instances de B sont des instances de A.

Dans le code suivant, on établit la taxonomie simple suivante :
Classe Animal
    Classe Poisson
    Classe Amphibien
    Classe Reptile
    Classe Oiseau
    Classe Mammifère
Classe Végétal

Le code rdf/rdfs/xml sera le suivant :

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

   <rdfs:Class rdf:about="http://www.moncode.org/rdfs/ont.rdfs#Animal"/>

   <rdfs:Class rdf:about="http://www.moncode.org/rdfs/ont.rdfs#Amphibien">
      <rdfs:subClassOf rdf:resource="http://www.moncode.org/rdfs/ont.rdfs#Animal"/>
   </rdfs:Class>

   <rdfs:Class rdf:about="http://www.moncode.org/rdfs/ont.rdfs#Mammifère">
      <rdfs:subClassOf rdf:resource="http://www.moncode.org/rdfs/ont.rdfs#Animal"/>
   </rdfs:Class>

   <rdfs:Class rdf:about="http://www.moncode.org/rdfs/ont.rdfs#Oiseau">
      <rdfs:subClassOf rdf:resource="http://www.moncode.org/rdfs/ont.rdfs#Animal"/>
   </rdfs:Class>

   <rdfs:Class rdf:about="http://www.moncode.org/rdfs/ont.rdfs#Poisson">
      <rdfs:subClassOf rdf:resource="http://www.moncode.org/rdfs/ont.rdfs#Animal"/>
   </rdfs:Class>

   <rdfs:Class rdf:about="http://www.moncode.org/rdfs/ont.rdfs#Reptile">
      <rdfs:subClassOf rdf:resource="http://www.moncode.org/rdfs/ont.rdfs#Animal"/>
   </rdfs:Class>

   <rdfs:Class rdf:about="http://www.moncode.org/rdfs/ont.rdfs#Végétal"/>
</rdf:RDF>
```

- Propriétés et sous-propriétés (ou spécialisations) :

Une propriété est de type rdf :Property.
RDFS décrit une propriété en terme de classes de ressources auquelle elle s'applique.
RDFS précise cette notion de propriété en donnant la possibilité de donner un type (littéral ou classe) à un sujet ou un objet (au sens des triplets précédemment vu).

- rdfs :domain définit le type des sujets lié à une propriété
- rdfs :range définit le type de données des valeurs de la propriétés (objet)

- rdfs :subPropertyOf définit la relation de sous-propriété entre deux propriétés. (spécialisation de propriétés).
- Une propriété peut avoir plusieurs domaines et plusieurs co-domaines.

Exemple de spécialisation de propriété :

Class Pizza
Garniture_fromage subClassOf Garniture

```
<rdf:Property rdf:ID=”a_garniture”>
  <rdfs:domain rdf :resource=”#Pizza”>
  <rdfs:range rdf :resource=”#Garniture”>
</rdf:Property>

<rdf:Property rdf:ID=”a_garniture_fromage”>
  <rdfs:subPropertyOf rdf:resource=”# a_garniture”>
     <rdfs:domain rdf :resource=”#Pizza”>
  <rdfs:range rdf :resource=”#Garniture_fromage”>
  </rdfs:subPropertyOf>
</rdf:Property>
```

a_garniture_fromage(pizza,garniture) => a_garniture (pizza,garniture)

On constate une inférence.

- Inférence : [web -11]

RDFS permet d’inférer des triplets à partir des triplets créés, des relations de subsomption, de sous-propriétés.

- Si x est de classe A et que A est inclus dans une classe B (A sous classe de B) alors A est de classe B.
- Si p1(x,y) avec p1 sous-propriété de p2 alors p2(x,y) est également vraie.

(exemple : a_garniture_fromage(pizza,garniture) => a_garniture (pizza,garniture))

- Transitivité des sous-classes et des sous-propriétés.

A sous classe de B et B sous classe de C alors A sous classe de C
P sous propriété de P1 et P1 sous propriété de P2 alors P sous propriétés de P2

- inférences sur C et les domaines et co-domaines :
  - si p propriété de domaine C et p(x,y) alors x est de type C
  - si p propriété de codomaine C et p(x,y) alors y est de type C
  - si on a défini plusieurs domaines pour p alors si p(x,y), x est instance de tous les domaines de p (idem pour les co-domaines)

3-OWL

OWL vs RDFS [web -12] : **Mais que manque-t-il à RDFS ?**

Owl est une « sur-couche » de rdfs, ajoutant d’autres outils pour saisir et affiner la sémantique des objets étudiés dans un domaine du savoir.

- Au niveau du vocabulaire : OWL a un vocabulaire plus riche et plus expressif. Avec OWL il est possible de spécifier les cardinalités des relations et des propriétés de type de données [web -13]. (les propriétés d'objet permettent de relier des instances à d'autres instances les propriétés de type de donnée permettent de relier des individus à des valeurs de données)

Exemple :

```
<owl :Class rdf :ID= ”datedeNaissance”/>
  <owl:DatatypeProperty rdf:ID =”anneedeNaissance”>
    <rdfs:domain rdf:resource=”#datedeNaissance”/>
    <rdfs :range rdf :resource= ”&xsd;positiveInteger”/>
  </owl:DatatypeProperty>
```

</owl :Class>

- Il est possible d'utiliser des opérateurs logiques dans les définitions (par exemple l'union de la classe pour le codomaine d'une relation).
- Relations entre classes (disjointWith : des ressources appartenant à une classe ne peuvent appartenir à l'autre,

ComplementOf : les ressources appartenant à une classe sont toutes celles qui n'appartiennent pas à l'autre, equivalentClass : indique que les deux classes ont le même set de ressources…)

- Egalité : SameAs : indique que deux instances réfèrent au même concept (ex : Queen of England et Elizabeth II)
- Des propriétés plus riches : **Symétrie** (p(A,B) vrai alors p(B,A) vrai, **Transitivité** (P(A,C) et P(C,B) vrai alors P(A,B) vrai, **InverseOf** : une propriété P est inverse d'une propriété P2. Exemple :

Pierre a_pour_professeur Henri, Henri est_professeur_de Pierre. a_pour_professeur est propriété inverse de est_professeur_de

- Restriction de propriété de classe

OWL ajoute des vocabulaires pour la description des propriétés et des classes, des relations entre classes (exemple disjointness), des cardinalités, des caractéristiques de propriétés (symmetry), et des classes énumérées.

OWL se décline en trois sous-langages [web -14] de plus en plus expressifs :

- OWL Lite:utilisant la logque de description SHIF(D) prise en charge de taxonomies et de contraintes simples sur le langage RDF (ex : disjointness des classes, des propriétés, …)

- OWL DL: (Ontology Web Language Description Logics) utilisant la logique de description DL SHOIN(D), expressivité riche tout en conservant la complétude informatique et la décidabilité (inclut toutes les fonctionnalités du langage OWL mais avec certaines restrictions pour respecter la complétude informatique : par exemple, une classe ne peut être une propriété ou une instance, une propriété ne peut être une classe ou une instance). Elle vérifie la hiérarchie des concepts et détecte les inconsistances.

- OWL Full : expressivité maximale et liberté syntaxique. Permet l'augmentation du vocabulaire prédéfini (Méta-classes OWL ou RDF) . Plus grande liberté que OWL DL mais avec un support de raisonnement moins prévisible.(pas d'algorithme efficace pour le raisonnement automatique).

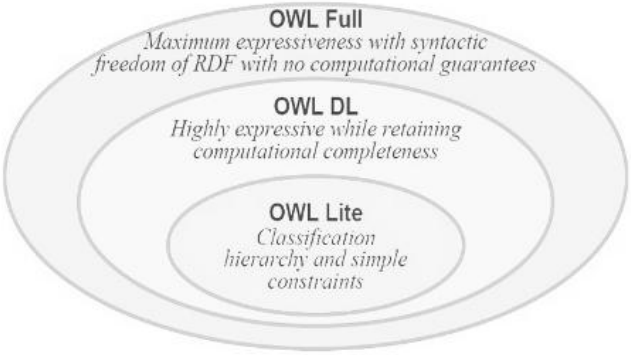


Figure 5 Trois versions de OWL

Ces trois versions (figure 5) varient quand à la liberté dans l'utilisation de RDF/RDFS, de la plus contrainte (OWL lite) à la plus libre (OWL full).

C'est particulièrement la version OWL DL qui nous intéressera et qui est utilisée par Protégé.
Elle utilise les logiques de description SHOIN(D) [web -15].
Les logiques de description sont une famille de languages formels de représentation de connaissances.
Elles ont une base [art -9] commune enrichies de différentes extensions.

Cette base commune est appelée AL (attributive language) et est défini à partir des éléments syntaxiques suivants :

- A : concept atomique
- $\perp$:concept universel BOTTOM
- T : concept universel TOP
- $\neg$A : négation d'un concept atomique
- C $\cap$ D : conjonction de concepts
- $\forall$r .C : quantificateur universel
- $\exists$r : quantificateur existentiel non typé

Le S de SHOIN désigne en fait ALC (Attributive language with complement) qui est la logique laplus importante, base de toutes les logiques de description »expressifs ».
ALC= AL $\cup$ {$\neg$C} avec C concept primitif ou défini. C'est AL avec en complément tous les concepts autorisés et pas juste les concepts atomiques.

Le H de SHOIN désigne le complément des rôles de hiérarchie (subproperties – rdfs :subPropertyOf)
Le O de SHOIN est un complément permettant d'utiliser les classes énumérées et les restrictions de valeurs des objets.
Le I de SHOIN permet l'expression des propriétés inverses
Le N de SHOIN permet l'expression des restrictions de Cardinalités.

(SHIF pour OWL Lite ne permet par exemple pas l'expression des restrictions de Cardinalités ou les classes énumérées).

Outils pour concevoir des ontologies OWL :
Le format owl devenant assez vite illisible pour l'œil humain (surtout devant des ontologies complexes) ou descriptible, il est assez vite requis l'aide d'éditeur (GUI). Ces éditeurs sont bien souvent de raisonneurs permattant de tester la cohérence de l'ontologie crée. Citons, parmi lesplus connus :

- Oiled
- OntoEdit
- Ontolingua
- OntoSaurus
- Protégé
- WebODE
- WebOnto

Par la suite nous examinerons plus en profondeur la manipulation du logiciel « protege ».

Outils pour la consultation d'ontologies [web -16][art -10]**:**

De nombreuses propositions ont été données pour interroger le schéma à base RDF/RDFS, citons :
RDQL (2004) [web -17], ICS-FORTH RQL [web -18], SeRQL (2003) [art -11], SPARQL [web -19]

Dans ce qui suit, nous examinerons le langage/protocole SPARQL qui est la recommandation du W3C en cours pour interroger les ontologies.

Partie 1

SPARQL, Présentation

L'étude et l'élaboration d'un outil d'aide à la création de requête SPARQL nécessite une présentation de ce langage. Ce chapitre aura donc pour but d'établir tout d'abord un historique de ce langage puis une présentation de la manipulation de ce langage et de ses mots clefs afin d'élaborer une requête et enfin une partie plus technique ayant trait à la sémantique de ce langage et à sa façon de fonctionner pour retrouver les données demandées par l'utilisateur. Nous aboutirons enfin à la problématique liée à la structuration de ce langage et de la recherche de ses objets (les I.R.I) .

A - Présentation globale et historique.

Bastian Quilitz, en introduction de son article **[art-12]**, souligne le contexte actuel dans lequel beaucoup d'applications web requièrent l'intégration de données issues de sources de données réparties et autonomes. Il était, encore depuis peu, difficile d'accéder aux données ou de les interroger du fait qu'il n'y avait pas encore de langage de requête standard ou d'interface dans ce sens. Avec **SPARQL**, la situation a changé et il est maintenant possible d'interroger des schémas de type RDF, RDFS, OWL à travers ce langage.

Selon le W3C, SPARQL est un langage mais aussi un protocole créé par un groupe de travail du W3C (Consortium *World Wide Web*) pour l'accès au WEB sémantique. La version 1.0, en 2008, devint la recommandation officielle du W3C, puis en 2013, la version 1.1 qui permet en plus l'enregistrement et la fusion de données de sources différentes, des sous-requêtes, un opérateur de négation, des méthodes d'agrégation des résultats.**[web-20][web-21]**

Figure 6 Logo officiel de SPARQL

Dans son article **[art-13]**, Jorge Perez & al (alors que SPARQL n'était encore qu'un potentiel candidat à la recommandation du W3C) nous apporte cette description de SPARQL : C'est un langage de requête basé sur du

« graph matching » (ou utilisant du pattern matching sur de la donnée de type Graphe de triplet rdf) . Étant donné un graphe source D, une requête est un pattern qui est comparé sur le graphe D. La valeur obtenue de cette comparaison (matching) est ensuite traitée pour donner la réponse. La source D peut être constituée de plusieurs sources de graphes différents. Toujours selon Perez, une requête SPARQL est composée de trois étapes :

- L'étape de comparaison (pattern matching) qui inclut plusieurs méthodes de pattern matching sur les graphes comme les parties optionnelles, les unions de patterns, les imbrications, les filtres (restrictions), les valeurs des matchings potentiels ainsi que la possibilité de choisir les sources de données à analyser.
- L'étape des modificateurs des solutions qui, une fois que les données issues du matching sont prêtes, permettent de modifier ces valeurs en leur appliquant des opérateurs classiques comme les projections, 'distinct', les agrégations (order, group by), les limit et offset.
- L'étape du formatage de sortie des données : renvoi de booléens en fonction du retour de la requête (ASK), sélection de valeurs de variables qui coïncident avec le pattern (SELECT), construction d'un graphe par de nouveaux triples à partir de ces valeurs (CONSTRUCT) et descriptions au sujet de requêtes de ressources(DESCRIBE).

B – Exemple de création d'une requête SPARQL

(inspiré de : [web-22], [web-23], [web-24],[web-25], [web-26])

1- Mise en place

Nous utilisons dans cette section les fichiers human_2007_09_11.rdfs et human_2007_09_11.rdf du tutoriel du logiciel Corese [web –38].(voir annexes 1 et 2 pour le détail de ces fichiers).
human_2007_09_11.rdfs implémente le modèle des classes et propriétés tandis que human_2007_09_11.rdf contient des instances de ces classes et propriétés.(le style de requête se passe à l'identique pour des fichiers owl).
Cette ontologie simple décrit des liens de base de l'humain en tant que sous classe de la classe 'animal', réparti en deux sous-classes principales 'homme', 'femme' puis des classes de fonctions ('professeur', chercheur') viennent compléter l'exemple.

Les classes suivantes sont donc requêtées :

Figure 7 Classes utilisées dans l'ontologie étudiée

avec les propriétés suivantes :

Figure 8 Propriétés d'objets

Créons une première requête, toujours avec notre outils 'protégé', onglet : SPARQL query (après avoir réalisé l'importation des deux fichiers).

« Qui a pour épouse Catherine ? »

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
PREFIX inst: <http://www.inria.fr/2007/09/11/humans.rdfs-instances#>
SELECT ?subject ?object
        WHERE { ?subject humans:hasSpouse inst:Catherine}
```

Les déclarations **de préfixes** permettent d'abréger les écritures dans la requête.
Sans celles-ci on aurait :

```
SELECT ?subject ?object
        WHERE { ?subject <http://www.inria.fr/2007/09/11/humans.rdfs#hasSpouse>
<http://www.inria.fr/2007/09/11/humans.rdfs-instances#Catherine>}
```

La requête fera le lien avec le fichier concerné
Pour le préfixe dans la requête :
PREFIX humans: http://www.inria.fr/2007/09/11/humans.rdfs#
On le rapproche du fichier human_2007_09_11.rdfs grâce à la ligne suivante : xml:base ="&humans;"
Et pour le préfixe PREFIX inst: http://www.inria.fr/2007/09/11/humans.rdfs-instances#
Dans le fichier human_2007_09_11.rdf dans la ligne : xml:base ="&humans;-instances" >
Dans un fichier owl, c'est aussi avec la ligne xml :base que l'on pourra trouver le préfixe associant la requête au fichier concerné.

Remarquons comme la requête contenue dans le WHERE est elle même un graphe de triplet. C'est ici un pattern de base.
(nous verrons plus loin et plus précisément le fonctionnement d'un pattern matching issu d'une requête).Les triplets pattern (ou motifs de triplets) sont similaires aux triplets mais chaque partie de ces triplets peut être remplacée.

2- Règles de base du langage

Les termes utilisés dans les motifs de triplets peuvent être :

-des IRI (Internationalized Resource identifier)
-des littéraux
-des variables, préfixées par ? (ou $)
-des nœuds blancs : ils jouent le rôle de variables non distinguées. Il s'écrivent sous forme de label (ex : _ :b12) ou [].

Un triplet RDF est défini comme suit (sujet, propriété, objet)=$(U \cup B) \times U \times (U \cup B \cup L)$
Avec : U : URI, B : nœud blanc, L : Littéral

Un graphe RDF est un ensemble de triplets

Un pattern de triplet, du type que l'on retrouve dans les clauses WHERE se constitue sous la forme :
$(U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$ avec V : ensemble des variables .

Les variables SPARQL commencent avec un point d'interrogation « ? » . Elles prennent les valeurs des nœuds du graphe du dataset RDF (ici humans.rdfs-instances) parcouru pour voir si leurs valeurs coïncident ou non (matching).

La clause SELECT retourne une table des variables et des valeurs qui satisfont la requête.(tuples)
Ici en l'occurrence :

```
SELECT ?subject ?object
        WHERE { ?subject <http://www.inria.fr/2007/09/11/humans.rdfs#hasSpouse>
<http://www.inria.fr/2007/09/11/humans.rdfs-instances#Catherine>}
```

subject	object
Karl	

Figure 9 Requête et résultat retourné

Les clauses CONSTRUCT ou DESCRIBE renvoient un graphe RDF utilisant la valuation des variables présentes dans la requête.
La clause CONSTRUCT est donc utile pour la transformation de triplets.

Par exemple , une requête qui transforme les triplets de type 'Marthe a_pour_fils Paul' en : 'Paul a_pour_parent Marthe' se décrit comme suit :

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
PREFIX inst: <http://www.inria.fr/2007/09/11/humans.rdfs-instances#>
CONSTRUCT { ?x humans:hasParent ?y . }
        WHERE
        { ?y humans:hasChild ?x  . }
```

La clause ASK retourne vraie si un jeu de réponses est trouvé, sinon faux.

```
ASK WHERE
        { ?y humans:hasChild ?x  . }
```

3- Requêtes basées sur des Patterns de triplets multiples

a- Utilisation pour retrouver plusieurs propriétés sur une même ressource.

Si on cherche, par exemple, les sujets dont l'épouse s'appelle Catherine et optionnellement leur pointure, on a :

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
PREFIX inst: <http://www.inria.fr/2007/09/11/humans.rdfs-instances#>
```

```
SELECT ?subject ?size
        WHERE {
        ?subject humans:hasSpouse inst:Catherine .
        OPTIONAL{ ?subject humans:shoesize ?size . }
}
```

On trouve une valuation de ?subject, « Karl ». La requête cherche ensuite si une propriété shoesize est disponible.

Sans le mot clef '**OPTIONAL**', si la valuation du triplet « ?subject humans:shoesize ?size » ne 'matchait' pas à une portion de notre Graphe Dataset d'origine, alors nous n'aurions aucun tuple en retour.
On peut donc définir des parties optionnelles dans cette clause.
Les variables dans cette clause n'ont pas forcément de valuation.
Le mot clef OPTIONAL est en fait dans le processus interne de SPARQL, comme nous le verrons plus loin, un 'left outer join' .

Les triplets peuvent être **factorisés** (façon 'turtle') :

Rerenons l'exemple de requête précédent (« les sujets dont l'épouse s'appelle Catherine et optionnellement leur pointure »)

```
SELECT ?subject ?size
        WHERE {
        ?subject humans:hasSpouse ?b .
        ?subject humans:shoesize ?size .
}
```

Peut se factoriser en :

```
SELECT ?subject ?size
        WHERE {
        ?subject
                humans:hasSpouse ?b ;
                humans:shoesize ?size .
}
```

SELECT * est le raccourci qui sert à afficher toutes les variables présentes dans la clause WHERE.

b- Utilisation pour *'traverser' un graphe*.

Toujours en reprenant notre exemple :

```
SELECT ?subject ?b ?size
        WHERE {
        ?subject humans:hasSpouse ?b .
        ?b humans:shoesize ?size .
}
```

On atteint le premier nœud est_epoux_de
A partir de ce noeud, on étudie la propriété 'taille_de_chaussure'.

Une requête SPARQL peut aussi spécifier le DATASET de référence à utiliser avec la **clause FROM** et FROM NAMED.
Si une requête fournit cette clause alors celle-ci est utilisée à la place de tout autre DATASET qu'il aurait eu à utiliser si la clause FROM ne figurait pas dans la requête.

Exemple (extrait de W3.org) qui permet d'afficher l'adresse de la page web de toutes les personnes que connaît Berners -Lee.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX card: <http://www.w3.org/People/Berners-Lee/card#>
SELECT ?homepage
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
   card:i foaf:knows ?known .
   ?known foaf:homepage ?homepage .
}
```

4 - Le mot clef 'a'

Soit la requête suivante permettant de lister les objets distincts et leur type.

```
SELECT DISTINCT ?subject ?b
        WHERE {
        ?subject a ?b.
}
```

'a' est un raccourci signifiant http://www.w3.org/1999/02/22-rdf-syntax-ns#type> et permettant d'avoir le type d'un élément.
DISTINCT permet d'éliminer les tuples doublon.

5- Les Filtres

Ils s'appliquent à un motif de graphe et ajoutent à la condition de la requête.
Il permet de restreindre les solutions du groupe dans lequel il apparaît. Sa position dans le groupe n'a pas d'importance. Pour donner un exemple de syntaxe , recherchons les sujets qui sont des hommes , qui ont un nom et dont le nom est la chaîne de caractères « Jack ».

```
SELECT ?subject ?name
        WHERE {
        ?subject a humans:Man.
        ?subject humans:name ?name.
        FILTER regex(?name, "Jack").

}
```

Autre exemple , pour illustrer des filtres de formats différents , rechercher les sujets qui sont des hommes, qui ontune prorpriété 'pointure' et dont cette pointure et plus grande que 7.

```
SELECT ?subject ?name
        WHERE {
        ?subject a humans:Man.
        ?subject humans:shoesize ?sz.
        FILTER (xsd:integer(?sz) > 7).

}
```

La taille de chaussures shoesize est d'un format String, on la convertit donc avec xsd:integer.

6- Non existence d'un motif

Dans la version 1.0, la non existence d'un motif de graphe est testée en définissant le motif comme optionnel et en vérifiant qu'il n'a pas été trouvé avec la fonction BOUND.

Exemple :
Les hommes qui n'ont pas d'enfants :

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
select ?x where {
 ?x a humans:Man
 OPTIONAL{ ?x humans:hasChild ?y }
 FILTER ( !bound(?y) )
 }
```

Dans la version 1.1, il y a la clause NOT EXIST :

- on cherche les tuples correspondant à la requête
- on élimine les tuples qui ne correspondent pas au filtre

Toujours sur notre exemple : « Les hommes qui n'ont pas d'enfants :»

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
select ?x where {
 ?x a humans:Man
        FILTER NOT EXISTS{ ?x humans:hasChild ?y }
 }
```

7- L'union

Située dans la requête, elle permet de satisfaire un motif de graphe ou un autre.
Prenons l'exemple suivant, de rechercher les sujets qui sont des hommes OU des femmes.

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
select ?x ?y {
        { ?x a humans:Man}
 UNION
        { ?x a humans:Woman}
 }
```

8- Ordre des solutions

Nous avions vu précédemment la clause DISTINCT qui permet d'éliminer les tuples résultats en doublon.
Nous avons aussi les opérateurs OFFSET et LIMIT qui permettent de prendre les enregistrements à partir du OFFSETième et avec un nombre défini par LIMIT de lignes de tuples.

Exemple ici de lister les sujets qui sont des personnes et donner au maximum 3 solution à partir dela 3ème.

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
select ?x {
        { ?x a humans:Person}

 }
OFFSET 2
LIMIT 3
```

ORDER BY permet d'ordonner les solutions par ordre croissant (alphabétiqueou numérique)

Exemple, sortir, dans l'ordre alphabétique les sujets qui sont du type 'Person' .

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
select ?x {
        { ?x a humans:Person}

 }
ORDER BY  ?x
```

Depuis 2013, SPARQL 1.1 propose de nouvelles possibilités.
Nous avions précédemment vu la clause FILTER NOT EXISTS, nous avons également les propriétés suivantes :

9- Des projections :

Création et assignation de nouvelles valeurs dans la clause SELECT avec le mot clef AS :

Exemple :

Lister dans une variable ‘double_size’ le double de la taille des tshirts des sujets qui sont des personnes.

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
PREFIX inst: <http://www.inria.fr/2007/09/11/humans.rdfs-instances#>

SELECT (xsd :int(?size) *2 AS ?double_size)
WHERE{
        ?b a humans:Person.
        ?b humans:shirtsize ?size.
}
```

10- L’opérateur MINUS

On évalue le MINUS et on le soustrait des résultats :

Exemple :

Sélectionner les sujets qui sont des personnes et qui ne sont pas des chercheurs.

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
PREFIX inst: <http://www.inria.fr/2007/09/11/humans.rdfs-instances#>
SELECT ?b
WHERE{
        ?b a humans:Person.
        MINUS {
                ?b a humans:Researcher.
        }
}
```

11- Les opérateurs d’agrégation

SUM, MIN,MAX, AVG,COUNT,SAMPLE(valeur aléatoire dans set de tuples), GROUP_BY,HAVING

Prenons pour illustrer cela la requête suivante consistant à afficher chaque personne distincte ainsi que la somme de leurs pointures pour peu que cette somme soit inférieure à 10.

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
PREFIX inst: <http://www.inria.fr/2007/09/11/humans.rdfs-instances#>
SELECT (SUM(?shoesize) AS $sum_size))
```

```
WHERE{
        ?b a humans:Person.
        ?b humans:shoesize ?sz.
}
GROUP BY ?b
HAVING ($sum_size <10)
```

12 - Les chemins de propriétés

C'est ce qui relie deux noeuds dans un graphe. A partir d'expressions régulières reliant les propriétés, ce système offre entre autres possibilité de connaître les amis des amis des amis…aussi loin qu'on le veut (opérateur *), de trouver toutes les catégories et sous-catégories d'un terme.

Syntax Form	Matches
uri	A URI or a prefixed name. A path of length one.
^elt	Inverse path (object to subject).
(elt)	A group path *elt*, brackets control precedence.
elt1 / elt2	A sequence path of *elt1*, followed by *elt2*
elt1 ^ elt2	Shorthand for *elt1 / ^elt2*, that is *elt1* followed by the inverse of *elt2*.
elt1 | elt2	A alternative path of *elt1*, or *elt2* (all possibilities are tried).
*elt**	A path of zero or more occurrences of *elt*.
elt+	A path of one or more occurrences of *elt*.
elt?	A path of zero or one *elt*.
elt{n,m}	A path between n and m occurrences of *elt*.
elt{n}	Exactly *n* occurrences of *elt*. A fixed length path.
elt{n,}	*n* or more occurrences of *elt*.
elt{,n}	Between 0 and *n* occurrences of *elt*.

Tableau 3 Chemins de propriétés
[web –27]

Exemple de séquence :

Lister les « Amis des amis de John » :

```
SELECT  ?b
WHERE{
        ?b humans:hasFriend/humans:hasFriend inst:John.

}
```

et aussi les Amis des amis des amis….des amis de John :

```
SELECT  ?b
WHERE{
        ?b humans:hasFriend* inst:John.

}
```

13 - Les requêtes imbriquées

Similaire au fonctionnement des requêtes imbriquées dans SQL,

Exemple :

Noms des amis des amis de John :

```
SELECT distinct ?fr ?nom
	WHERE {
	?fr humans:hasFriend inst:John.{
	SELECT ?nom
		WHERE{
			?nom humans:hasFriend ?fr.
		}

	}}
```

14- Mises à jour du graphe

Avec la 1.1 il est désormais possible de mettre à jour également le Dataset graph

Opérations sur le graphe lui même (graph managment).

- Création et suppression de nœuds : CREATE et DROP
- Ajout ou suppression de triplets : LOAD & CLEAR

Opérations sur les données

- Insertion de données sans variables : INSERT DATA
- Suppression de données sans variable : DELETE DATA
- Insertion de Triplets dans le graphe : INSERT
- Suppression de triplets dans le graphe : DELETE

15 - Requêtes distribuées : mot clef SERVICE

Aspect intéressant et grand pas en avant dans l'optique du partage et de la réutilisation de données, la commande 'SERVICE' sert à l'interrogation de points d'accès distants (un ou plusieurs).

Exemple consistant à trouver, sur l'accés distant DBPedia les 50 premières personnes et leur nom.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?person ?name WHERE {
	SERVICE <http://dbpedia.org/sparql> {
		?p a foaf:Person ;
		foaf:givenName ?name
	}
}
LIMIT 50
```

Ici on fait appel à la base Sémantique de DBPEDIA.

Le mot clef BINDINGS pourra attribuerles contraintes sur les différents points d'accès.

Exemple trouver toutes les personnes issues des ontologies distantes Dbpedia et semanticweb qui ont pour nom ‘John’.

```
SELECT ?person WHERE {
        SERVICE <http://dbpedia.org/sparql> {
                ?p a foaf:Person ;
                foaf:givenName ?name
        }
        SERVICE <http://data.semanticweb.org/sparql> {
                ?p a foaf:Person ;
                foaf:givenName ?name
        }
}
```

BINDINGS ?name { ("john") }
Une liste de points d’accés importants est donnée sur http://www.w3.org/wiki/SparqlEndpoints.

Le web sémantique étant très mobile, vivant et en perpétuel mouvement, beaucoup de liens de points d’accès peuvent être créés, déplacés ou supprimés.
Un SPARQL endpoint *est une interface web qui permet d'interroger de l'information numérique structurée en RDF (*Resource Description Framework*).*

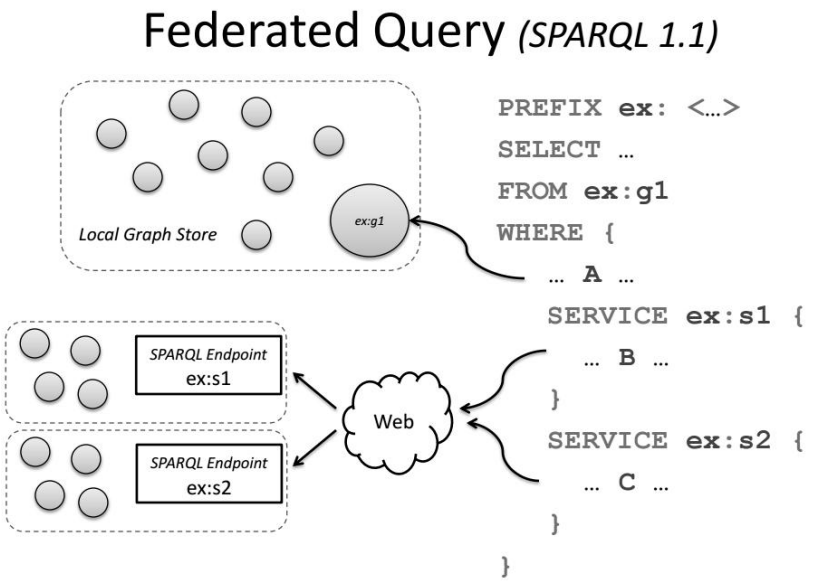


Figure 10 Exemple de requête distribuée sur deux sources (deux points d’accès) distantes
[web -28] .

Exemples de requêtes distribuées :

Parcours des personnes de notre fichiers humans qui ont pour nom « John » et recherche dans la base dbpedia dont les noms contiennent le motif « John »
On a ici une distribution d’une part locale, d’une autre part distante.

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
PREFIX inst: <http://www.inria.fr/2007/09/11/humans.rdfs-instances#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT * WHERE {
        {?pers a humans:Person.
        ?pers humans:name ?nom.
        FILTER(regex(?pers,"john","i")).}
```

```
        SERVICE <http://dbpedia.org/sparql> {
                ?perso a foaf:Person.
                ?perso foaf:givenName ?name.

        }
        FILTER(regex(?name,?nom,"i")).

}
```

Puis, parcours des noms des personnes de notre fichier humans et recherche de ceux ci sur dbpedia :

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
PREFIX inst: <http://www.inria.fr/2007/09/11/humans.rdfs-instances#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT * WHERE {
        {?pers a humans:Person.
        ?pers humans:name ?nom.
        }

        SERVICE <http://dbpedia.org/sparql> {
                ?perso a foaf:Person.
                ?perso foaf:givenName ?name.

        }
        FILTER(regex(?name,?nom,"i")).

}
```

C- Sémantique du langage SPARQL

Olaf Hartig établit en 2008 la formalisation algébrique et la sémantique de SPARQL expliquant ainsi son fonctionnement. [art-14]
Il décrit les 3 étapes d'une requête :

- Transformer une requête SPARQL en un arbre de syntaxe abstraite.
- Conversion de cet arbre en une requête abstraite (expression algébrique)
- Evaluation de l'expression algébrique sur un Dataset RDF.

Etape 1 : Chaque symbole dans la requête abstraite est associé avec un opérateur algébrique du même nom.

Le parser SPARQL crée donc tout d'abord un arbre de syntaxe abstrait à l'aide dela grammaire du langage.
Extrait de W3.org : [web-29] :

[13]	*WhereClause*	::=	'WHERE'? GroupGraphPattern

[20]	*GroupGraphPattern*	::=	'{' TriplesBlock? ((GraphPatternNotTriples | Filter) '.'? TriplesBlock?)* '}'
[21]	*TriplesBlock*	::=	TriplesSameSubject ('.' TriplesBlock?)?
[22]	*GraphPatternNotTriples*	::=	OptionalGraphPattern | GroupOrUnionGraphPattern | GraphGraphPattern

[23]	*OptionalGraphPattern*	::=	'OPTIONAL' GroupGraphPattern

[24]	*GraphGraphPattern*	::=	'GRAPH' VarOrIRIref GroupGraphPattern

[25]	*GroupOrUnionGraphPattern*	::=	GroupGraphPattern ('UNION' GroupGraphPattern)*

[26]	*Filter*	::=	'FILTER' Constraint

Tableau 4 Extrait de grammaire du langage SPARQL

Etape 2 : Conversion de cet arbre en une requête abstraite (expression algébrique).

Une requête abstraite SPARQL est un tuple (E,DS,R) avec :
- E : une expression algébrique SPARQL
- DS : un dataset RDF {G,(<u1>,G1),…, ,(<un>,Gn)}
  Comme le précisait Atanas Kiryakov dans son article [art-15], G et chaque Gi sont des Graphes et chaque <ui> est un IRI (n internationalized URI). La paire (<ui>,Gi) est appelé un graphe nommé ou <ui> est le nom du graphe Gi. G est appelé le graphe par défaut. Il contient tous les triples qui appartiennent au DataSET mais pas aux graphes specifiques nommés.
- R est une formule de requête

Olaf Hartig explique que convertir un arbre en expression algébrique requiert 3 étapes :
- Développement des abréviations pour les URI et les pattern de triplets.
- Traduction récursive des pattern de requête
- Traduction des modificateurs de solutions.

* L'étape d'expansion des abréviations, sur un exemple :

On part de ?pers a humans:Person ;
humans:name ?nom.

On expanse les adresses :
?pers <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.inria.fr/2007/09/11/humans.rdfs/Person>;
<http://www.inria.fr/2007/09/11/humans.rdfs/name> ?nom

Expansion écriture turtle :
?pers <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.inria.fr/2007/09/11/humans.rdfs/Person>.
?pers <http://www.inria.fr/2007/09/11/humans.rdfs/name> ?nom

* L'étape de traduction récursive des patterns de requête est effectuée parl'algorithme suivant proposé par Marcelo Arenas [art-16] :
Avec TB : TripleBLocks

Algorithm 1 Transformation $\mathcal{T}$ of SPARQL pattern syntax into algebraic syntax

```
1: // Input: a SPARQL graph pattern GroupGP
2: // Output: an algebraic expression E = T(GroupGP)
3: E ← ∅; FS ← ∅
4: for each syntactic form f in GroupGP do
5:    if f is TB then E ← (E AND T(TB))
6:    if f is OPTIONAL GroupGP_1 then E ← (E OPT T(GroupGP_1))
7:    if f is GroupGP_1 UNION ··· UNION GroupGP_n then
8:       if n > 1 then E' ← (T(GroupGP_1) UNION ··· UNION T(GroupGP_n))
9:       else E' ← T(GroupGP_1)
10:      E ← (E AND E')
11:   if f is GRAPH VarOrIRIref GroupGP_1 then
12:      E ← (E AND (VarOrIRIref GRAPH T(GroupGP_1)))
13:   if f is FILTER constraint then FS ← (FS ∧ constraint)
14: end for
15: if FS ≠ ∅ then E ← (E FILTER FS)
```

Figure 11 Algorithme de transformation

Arenas développe son algorithme sur l'exemple suivant :

```
{
?x :age ?y
FILTER (?y > 30)
?x :knows ?z .
?z :home_country ?c
FILTER (?c = "Chile")
OPTIONAL { ?z :phone ?p }
}
```

En suivant la grammaire de SPARQL, le pattern de requête ci-dessus est constitué d'un simple GroupGraphPattern qui contient dans l'ordre : *TriplesBlock Filter TriplesBlock Filter OptionalGraphPattern*

OptionalGraphPattern contient quand à elle un GroupGraphPattern composé d'un simple *TriplesBlock*.

La fonction de translation dans l'algorithme ci-dessus commence avec E={} et FS={}. En appliquant l'algorithme sur le pattern nous obtenons :

$$E = \Big(\big(\big(\{\} \text{ AND } \mathcal{T}(\texttt{TB}_1) \big) \text{ AND } \mathcal{T}(\texttt{TB}_2) \big) \text{ OPT } \mathcal{T}(\texttt{GroupGP}_1) \Big)$$
$$FS = \big((?Y > 30) \wedge ?C = \text{Chile} \big),$$

Avec TB1 = ?x :age ?y
TB2 = ?x :knows ?z . ?z :home_country ?c et
GroupGP1 = { ?z :phone ?p }

Les translations $\mathcal{T}(\texttt{TB}_1)$ et $\mathcal{T}(\texttt{TB}_2)$ correspondent à {?X, :age, ?Y } et à {(?X, :knows, ?Z), (?Z, :home_country, ?C)} respectivement.

Avec $\mathcal{T}(\texttt{GroupGP}_1)$ en entrée de l'algorithme on obtient :
E'=({} AND {(?Z,:phone,?P}).

Au final,le pattern aura la syntaxe algébrique suivante :

```
[( (    ({} AND {(?X,:age,?Y)})
        AND {(?X,:knows,?Z),(?Z,:home country,?C)})
        OPT ({} AND {(?Z,:phone,?P)}))
        FILTER ((?Y > 30) ∧ ?C = Chile )].
```

Etape 3 : Evaluation de l'expression algébrique sur un Dataset RDF.

L'article de Marcelo Arenas définit d'abord la *notion de mapping*,fonction partielle μ : V→T
Avec T=I union B union L (ensembles des IRI, Blank nodes et Littéraux RDF) et V un set infini de variables disjointes des ensembles précédents.(variables d'interrogation)

Soit un triplet donné t, et un mapping μ tel var(t) contenu dans dom(μ),μ(t) est le triple obtenu en remplacant les variables dans t en accord avec μ.
Soit un graph pattern basique P, alors μ(P)= Union des triplets t appartenant à P des μ(t).
Autrement dit, μ(P) est le set des triplets obtenu en remplacant les variables dans les triplets en accord avec μ .

Soient Ω1 et Ω2 des sets de mappings. Marcelo Arenas définit ainsi les opérations d'union et de différence entre ceux-ci :

Ω1 ⋈⋉ Ω2 = {μ1 ∪ μ2 | μ1 ∈ Ω1,μ2 ∈ Ω2 and μ1,μ2 sont des mapping compatibles}: JOIN
Ω1 ∪ Ω2 = {μ | μ ∈ Ω1 or μ ∈ Ω2} : UNION

Ω1 \Ω2 = {μ appartient Ω1 |pour tout μ′ appartient à Ω2, μ and μ' ne sont pas compatibles} : DIFFERENCE

Autre définition : Soit D un Dataset RDF sur T, t un pattern de triplet et P1, P2 des graph patterns. Alors l'évaluation d'un graph pattern sur D, noté $[[.]]_D$ est défini récursivement comme suit :

(1) [[t]]g = {μ : | dom(μ) = var(t) and μ(t) appartient à D} ou var(t) est le set des variables utilisées dans le triplet t

(2) $[\![(P_1 \text{ AND } P_2)]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$.
(3) $[\![(P_1 \text{ OPT } P_2)]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$.
(4) $[\![(P_1 \text{ UNION } P_2)]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D$.

Dans [art-13] son article Jorge Perez donne quelques exemples :

Exemple 1 :

On considère le RDF DataSet suivant :

D = { (B1, name, paul), (B1, phone, 777-3426),
(B2, name, john), (B2, email, john@acd.edu),
(B3, name, george), (B3, webPage, www.george.edu),
(B4, name, ringo), (B4, email, ringo@acd.edu),
(B4, webPage, www.starr.edu), (B4, phone, 888-4537), }

Ce qui suit des patterns de requête et leur évaluation (mapping) sur D conformément à la sémantique récursive établie ci-dessus :

(1) P1 = ((?A, email, ?E) OPT (?A, webPage, ?W))

$[[P1]]_D$ =

	?A	?E	?W
μ1	B2	john@acd.edu	
μ2	B4	ringo@acd.edu	www.starr.edu

Tableau 5 tableau de mapping [[P1]]D

(2) P2 = (((?A, name, ?N) OPT (?A, email, ?E)) OPT (?A, webPage, ?W))

$[[P2]]_D$ =

	?A	?N	?E	?W
μ1	B1	Paul		
μ2	B2	John	john@acd.edu	
μ3	B3	George		www.george.edu
μ4	B4	Ringo	ringo@acd.edu	www.starr.edu

Tableau 6 Tableau de mapping [[P2]]D

L'auteur définit ensuite l'équivalence de deux patterns de requête P1 et P2,si $[[P1]]_D = [[P2]]_D$ tout Dataset RDF D.
Il démontre ensuite les propriétés suivantes : (P1,P2,P3 patterns de requête et R, graphe de Filtre)

(1) AND et UNION sont associatifs et commutatifs
(2) (P1 AND (P2 UNION P3)) ≡ ((P1 AND P2) UNION (P1 AND P3)).
(3) (P1 OPT (P2 UNION P3)) ≡ ((P1 OPT P2) UNION (P1 OPT P3)).
(4) ((P1 UNION P2) OPT P3) ≡ ((P1 OPT P3) UNION (P2 OPT P3)).
(5) ((P1 UNION P2) FILTER R) ≡ ((P1 FILTER R) UNION (P2 FILTER R)).

D- Problématique

Outre la structuration de type SQL des requêtes pouvant mener, dans le cas d'union de requêtes complexes intégrant un ou plusieurs filtres, l'utilisateur d'une ontologie aura tôt fait de réaliser la difficulté d'obtenir les identificateurs de ressources (I.R.I.) du fait de leur complexité, de leur taille et parfois d'une sémantique souffrant du manque de consensus commun que nous vons abordé dans la partie préalable. Le but des ontologies, notamment au point de vue du web sémantique et aussi de

pouvoir disposer d'une information aisément consultable ou , à un niveau supérieur, élaborable (pour un programmeur ayant à présenter une partie des données à partir de requêtes ciblées). Se pose alors les questions « comment réaliser le plus aisément possible mes requêtes ? , Comment se repérer dans une ontologie dont la structure ne nous est pas familière et qui peut atteindre des nombres d'objets colossaux (millions) ? »

Partie 2

SPARQL, Tour d'horizon des outils et méthodes d'aide à la construction de requêtes.

Avec l'émergence des ontologies et des structures de données sémantiques s'est affiché le besoin lié de l'extraction des données stockées , extraction devant tirer profit des capacités de raisonnement offertes par exemple par les formats RDFS/OWL. Ces extractions se révélant assez peu instinctives pour un utilisateur non initié, des outils d'aide à la construction de requête ont donc émergés dans les années 2000 et suivantes. Ce chapitre consistera en un état de l'art de ce qui est proposé des années 2000 à 2015.

A - Analyse sémantique d'une requête en langage naturel.

Dans son article « *Représentation sémantique de questions pour interroger le Web sémantique* » [art-17], Romain Beaumont et son équipe mettent en évidence le fait que l'interrogation des bases de connaissances de web sémantique avec SPARQL est souvent « non maîtrisé des utilisateurs non experts, cela requiérant de connaître de le schéma de la base, et c'est pourquoi les systèmes d'interrogation en langage naturel se développent actuellement.

Le but souligné est de donner à un utilisateur la possibilité de profiter des outils offerts par le web sémantique en masquant ses principales difficultés ce qui amènera à la conception de méthodes ou d'interfaces, d'aides contextuelles faciles à utiliser.
Se pose alors le problème de construction automatique de requêtes, devant intégrer des problèmes de distance lexicale entre les mots dela question et les relations de la base de connaissance ».

Est alors proposé dans cet article une « méthode d'analyse des questions qui opère par transformation de graphes fondée sur des contraintes très générales sur la structure des requêtes ».

L'article décrit en trois points ce que nécessite une requête SPARQL :
- Identifier les triplets et relations de la base mentionnés dans la question
- Les associer en triplets
- Construire la requête elle-même avec sa structure, des opérateurs.

Dans leur article "Paraphrase-Driven Learning for Open Question Answering" [art-18], Fader et al. proposèrent une technique par appariement basée sur des techniques d'apprentissage pour répondre à des questions simples, du style :

Who wrote the Winnie the Pooh books?

Who is the author of winnie the pooh?
What was the name of the authur of winnie the pooh?
Who wrote the series of books for Winnie the poo?
Who wrote the children's storybook 'Winnie the Pooh'?
Who is poohs creator?

La méthode alors proposée se base sur les bases de connaissances Dbpedia tout en s'appuyant sur WordNet pour ce qui est de l'identification des relations et la recherche de leurs variations.

Le but suivi consiste en la transformation d'une question en un ensemble de représentations sémantiques sous forme de graphes G^S_Q (Graphe sémantique de la question) permettant d'interroger la base de connaissance. Chaque graphe représente une manière d'interprêter la question, les nœuds étant des entités, les arêtes étant des relations. Chaque entité et relation sont associées à un score de pertinence permettant de calculer le score de chaque graphe.

Le plan de la méthode est le suivant (figure meth_spqr-1 ci dessous) :

1- **Appariement de termes à la base de connaissances**
Consiste à associer les termes de la question à des entités, relations et types de la base de connaissance. (figure 1.a) par détermination de syntagmes.
   a. Extraction et identification des entités :
   Les auteurs se servent dans ce but de DBPedia sportilight (développé par Daiber et al en 2013) et de QALD3 pour tester l'efficacité del'extraction (efficacité de 0.86 pour notre cas).

   b. Identification des types :
   Les types identifiés sont ceux de DBpedia et Yago.
   On associe des syntagmes aux labels des types. Les scores sont « déterminés par la longueur en nombre de mots du type. (…) Les scores des types identifiés sont déterminés par la longueur en nombre de
   mots du type : plus le type est long plus il est spécifique, par exemple on préférera « StatesOfTheUnitedStates » à « States. »

   c. Extraction de relations :
   A pour but de détecter des mentions de relations dans la question qui pourront être reliées à des relations sémantiques. Différents outils pour ce but : Reverb(Fader et al. 2011), RelEx(Fundel et al. 2007) .
   Le premier relève des relations mais dont le pourcentage qualitatif n'est que faiblement élevé. Le deuxième proposant la réécriture de graphes a la désavantage d'une perte d'informations trop forte.
   L'auteur choisit une méthode qui prend en entrée toutes les mentions de relations disponibles. Les mots pleins étant potentiellement des mentions de relations (adjectif, nom ou verbe).

   d. Identification de relations et traitement des variantes.
   Consiste à relier une mention potentielle de relation à une relation en ontologie.
   On considère les cas de variations suivants :
      1- La catégorie syntaxique du label et de la mention de relation sont différentes . (une mention de relation étant définie comme une définition en langage naturel d'une instance de relation).
      2- Les mentions de relation présentent une distance sémantique importante avec le label de la relation dans l'ontologie.
      3- Les labels et mentions de relations peuvent contenir plusieurs mots ayant chacun leurs propres variations.

   Deux solutions envisagées par les auteurs pour gérer ces cas de variations :
      1- Utilisation d'un corpus de paraphrases pour identifier les relations équivalentes .
      2- Utilisation d'une base lexicale (dans ce cas, WordNet en particulier dans le cas où la mention de relation est formée d'un seul mot). Wordnet est une ressource permettant d'associer les mots à des concepts et des relations sémantiques entre concepts.

Les relations de wordnet utilisées sont en particulier les formes dérivées des mots, les synonymes et les hyperonymes.

2- Formation d'une représentation sémantique de la question

a. Représentation sémantique :
Représentation en graphe, sous forme d'un ensemble de triplets (e1,R,e2) avec e1 domaine, R relation et e2 codomaine (e1 et e2 étant des entités décrites par 4 champs : mention, type, valeur et score, R une relation possèdant un label dans la base de connaissance, ainsi qu'un URI, un domaine et un codomaine)
Plusieurs graphes sémantiques seront réalisés à l'issue de l'analyse des questions.

b. Formation des triplets :
Deux étapes :
- Formation de graphes syntaxiques en graphes syntaxico-sémantiques
- Formation des graphes sémantiques formés de triplets sémantiques à partir des graphes syntaxico-sémantiques.

c. Identification des triplets incompatibles : Evaluation de tous les triplets disponibles en fonctions des contraintes sémantiques de la base. (filtrage par score de compatibilité en fonction de domaine ou co-domaines non compatibles par exemple).

d. Pondération d'un graphe sémantique
A chaque graphe sémantique est affecté un score, la formule étant somme des scores des triplets dans le graphe divisés par le nombre de triplets.
Les graphes sont ensuite convertis en requêtes, ensuite exécutées dans l'ordre décroissant des scores obtenus.

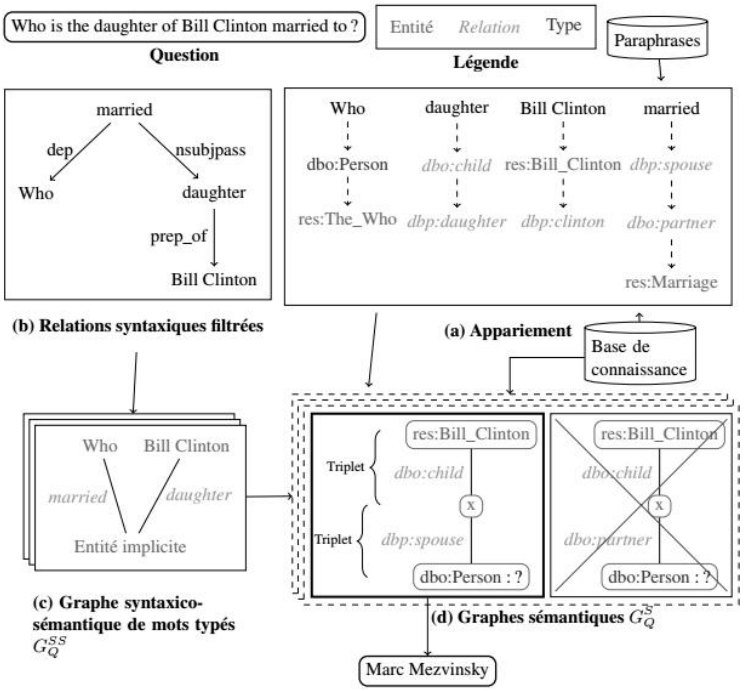


Figure 12 Principe de la représentation sémantique d'une question

La méthode a cependant des résultats encore assez limités même si proches (mais inférieurs à la méthode CASIA proposée par He et al en 2013).

Elle offre toutefois un aspect analytique intéressant quand à l'analyse et au traitement de questions en langage naturel pour aboutir à des requêtes SPARQL(sans doute aussi dû à un nombre assez important d'incohérences ou d'inconsistances au niveau du remplissage des bases de connaissances). L'exemple suivant,cité par l'auteur (au niveau de l'identification des triplets incompatibles) est assez évocateur :

‘*Par exemple la réponse à la question Who developed Minecraft ? est dans DBpedia sous la forme du triplet res:Minecraft dbo:developer res:4J_Studios. Le domaine de dbo:developer est dbo:UnitOfWork et le co-domaine est dbo:Agent. res:Minecraft a pour type dbo:VideoGame qui n’est pas compatible avec dbo:UnitOfWork’*

B- Outils graphiques pour la construction sémantique de requêtes

1-NITELIGHT

Dans leur article [art-19] *‘NITELIGHT: A Graphical Tool for Semantic Query Construction’* Alistair Russell & al. exposent une méthode graphique pour la construction contextuelle de requêtes à partir de conventions graphiques qui nous servira d’illustration à une possible mise en œuvre de ce style d’aide contextuelle.

Commençant par un bref rappel de la syntaxe des requêtes SPARQL,la méthode propose une construction à partir des triplets typiques des constructions RDF.

Notations graphiques (vSPARQL)

Les nœuds correspondront aux sujets et aux objets des triplets RDF, les liens correspondront aux prédicats.
- Les patterns de recherche simples de triplets auront les conventions de représentation suivantes :

Les nœuds sont représentés graphiquement par des objets géométriques dont la couleur et la forme exprimeront le type. (variables liées ou non liées, valeurs littérales ou URI). Les nœuds sont aussi associés à un label qui indique l’URI, la valeur littérale ou la variable de requête représentée par le nœud. Les liens sont représentés comme de simples lignes et associés à un label qui indique le prédicat représenté ou le nom de la variable requêtée. Les flèches indiquent quel nœud représente le sujet et quel nœud représente l’objet du triplet représenté(cf *figure 14*) .

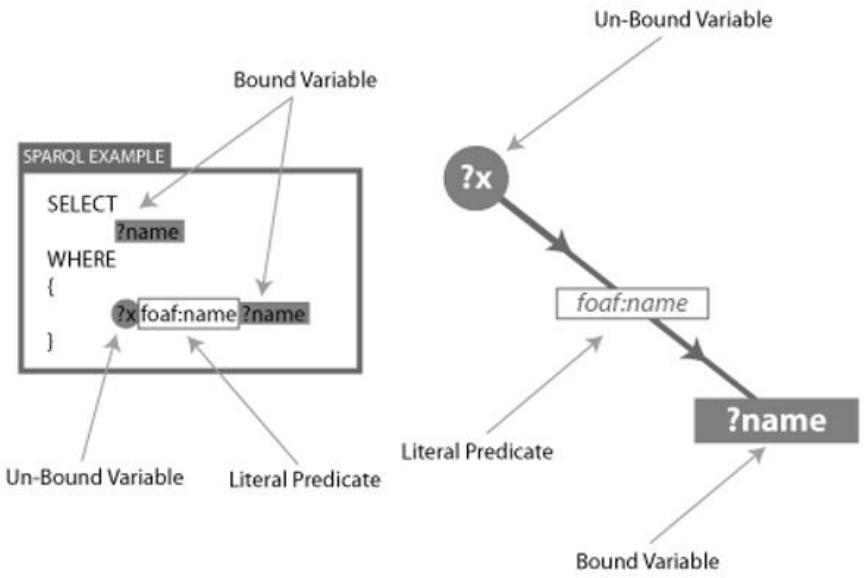


Figure 13 Convention graphique vSPARQL

- Les patterns de recherches multiples de triplets sont représentés en ajoutant d'autres nodes et liens. S'il y a une variable partagée par plusieurs patterns de triplets, celle-ci sera représentée par un nœud commun avec des liens multiples. (voir figure ci-dessous).

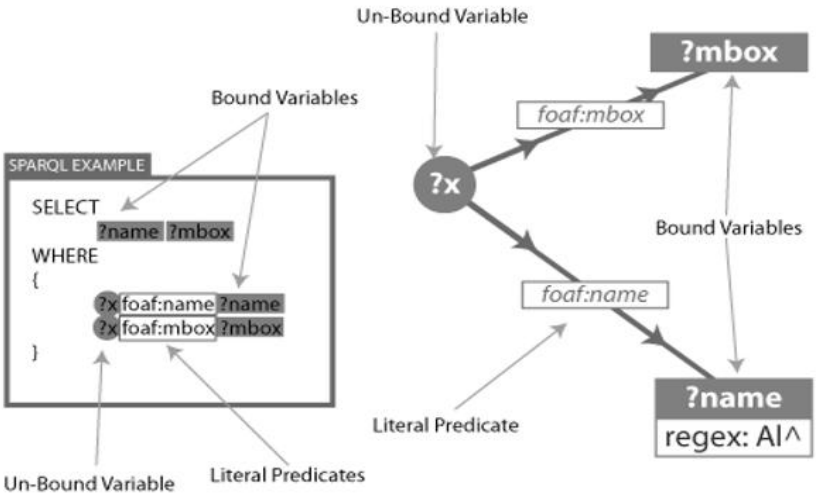


Figure 14 Illustration du pattern de recherches multiples de triplets

- Ordre des variables et des triplets.

Par convention, et pour respecter l'ordre parfois important dans les requêtes des variables et des triplets de pattern, une valeur numérique est affichée sur le côté haut-gauche de chaque nœud et des étiquettes de liens . Ainsi les nœuds qui sont dupliqués à travers le graphe partagerons le même indice d'ordre.

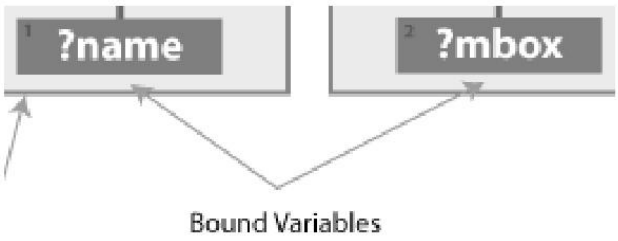


Figure 15 notation de l'ordre des variables et triplets

- Pattern graphique :

Pour rappel, dans SPARQL, « un pattern (ou motif) graphique consiste en un ou plusieurs pattern de triplets qui sont comparés à l'intégralité du graphe RDF . Les patterns graphiques influent sur les liaisons des variables car chaque variable a une portée locale conforme au pattern graphique dans laquelle elle est contenue. Cela signifie qu'une même variable peut-être liée à différentes valeurs dans différents patterns graphiques. Utiliser un pattern graphique signifie que les triples contenus dans un pattern graphique sont comparés au graphe RDF entier etne sont affectés par aucun pattern graphique précédent.

Le suppport graphique pour la représentation des motifs graphiques sont réalisés en organisant les collections triplets nœud-lien-nœud en groupes » (voir figure ci-dessous).

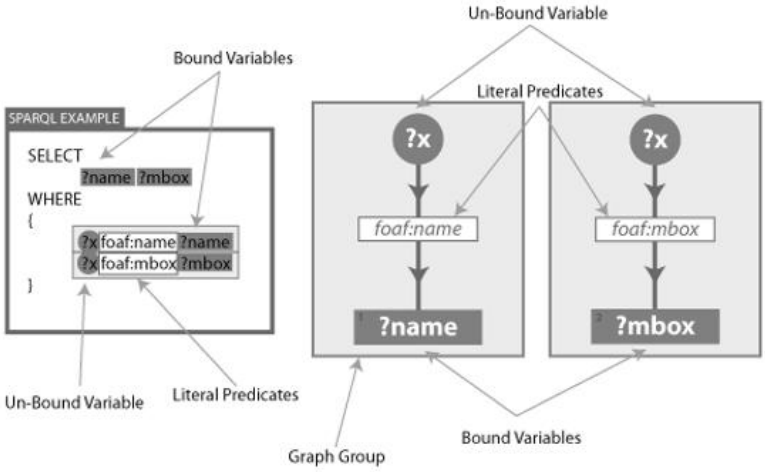


Figure 16 Représentation des motifs graphiques

- Pattern graphiques optionnels :

Ils sont représentés comme dans la figure suivante :

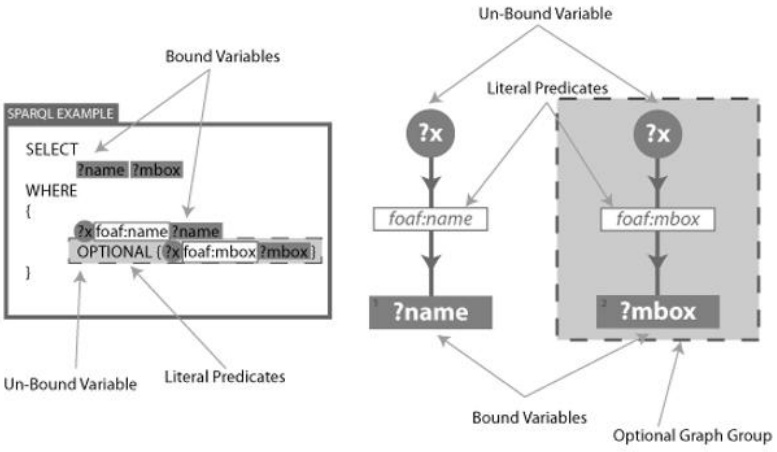


Figure 17 Pattern graphiques optionnels

- L'union de pattern de recherche est réalisée en ajoutant une étiquette liant les deux motifs graphiques.

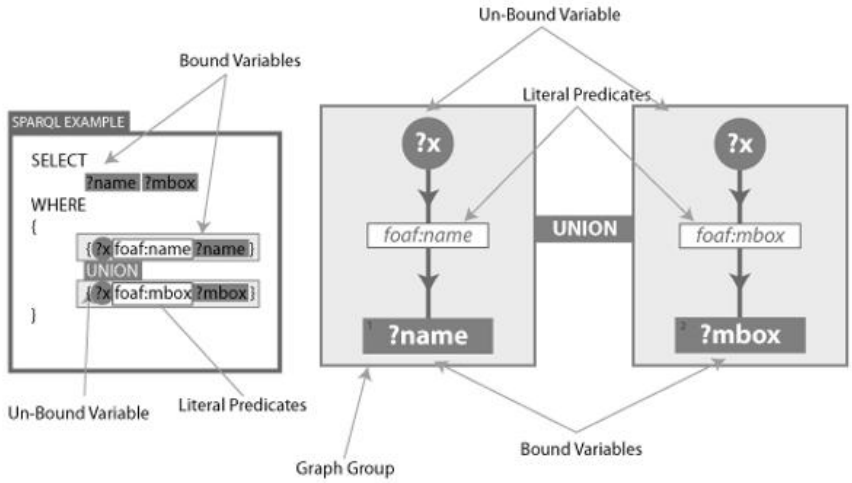


Figure 18 Union de patterns de recherche

- Classement selon une variable

Les résultats de la requête peuvent être classés dans l'ordre de variables liées ou non. Cela est représenté par un indicateur sur la droite du nœud représentant la variable concernée (ascendant ou descendant). (figure ci-dessus)

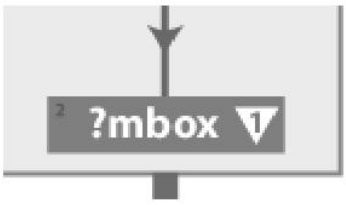


Figure 19 Indicateur ascendant ou descendant

- Les filtres de variables
Utilisés pour restreindre les ensembles de résultats retournés par une requête utilisant des expressions numériques ou régulières. Ils sont représentés par une petite boite juste sous le nœud représentant la variable comme suivant :

Figure 20 Filtre de variable

Prototype global de l'éditeur GUI :

Développé sous interface java (module JENA [web-30] pour SPARQL et SWT pour le kit graphique)

Vue globale de l'interface graphique :

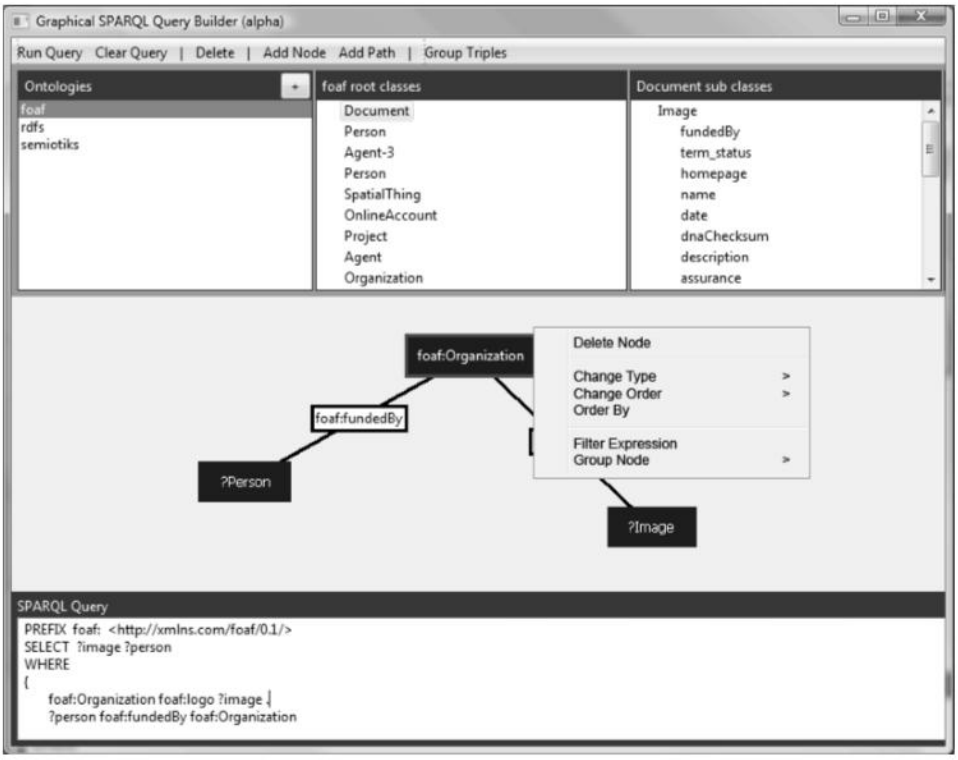


Figure 21 Vue globale de l'interface graphique

5 composants principaux de l'interface :

- Le panneau de composition de requête

Il permet la construction graphique des requêtes et leur affinage avec les normes graphiques vSPARQL vues précédemment . Les nœuds et liens sont des objets sélectionnables qui peuvent être édités avec, soit la barre d'outils rapides, soit un menu contextuel, ce qui permet aussi de définir des filtres, définir des tris ou des regroupements de données.

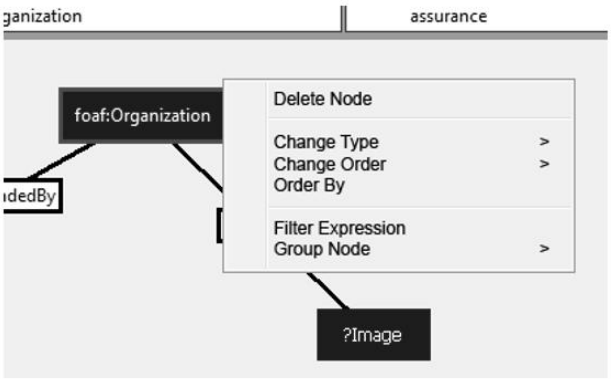


Figure 22 Panneau de composition de requête

- Le navigateur d'ontologies

Point important pour fournir aux utilisateurs un point de départ à leurs requêtes. La colonne de gauche est une liste des ontologies présentemment chargées. De nouvelles ontologies pouvant y être ajoutées avec le bouton +.

La seconde colonne affiche l'énumération des classes racine de l'ontologie choisie. Les colonnes suivantes sont ensuites les sous-classes de celles-choisies dans les colonnes précédentes. Ce procédé peut être répété et des colonnes ajoutées à droite au fur-et-à-mesure de l'exploration des sous-classes.

Les classes choisies peuvent être 'drag & droppées' dans la fenêtre de requête afin de définir celle-ci.

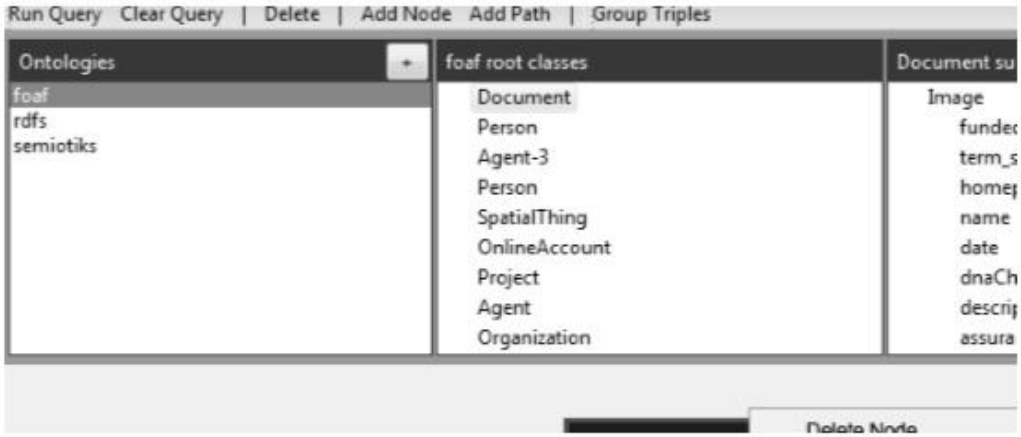


Figure 23 Navigateur d'ontologie

- L'aperçu de syntaxe SPARQL

Celui-ci traduit en langage SPARQL la requête visualisée dans la fenêtre graphique.

- Le panneau d'affichage des résultats

Les résultats peuvent provenir de tous les SPARQL 'endpoints'. Ils sont affichés sous forme de tableau.
Il n'y a pas encore de bidirectionnalité entre cette fenêtre et la fenêtre de requête graphique.

- La barre d'outils rapide

2- Query VOWL

Query Vowl [art-20], développé par Florian Haag et son équipe est une interface graphique pour la création de requêtes SPARQL basé sur la technique de visualisation d'ontologies VOWL[art-21] et définissant, à partir de cette technique un mapping vers l'écriture des requêtes en SPARQL.

a- VOWL

Une première étape à la constitution d'une requête d'ontologie est de connaître au moins un minimum de la structure de cette ontologie. La visualisation est un moyen d'y parvenir.
Nous commençons donc par une présentation de cette technique de visualisation d'ontologies VOWL développée par Steffen Lohmann qui permet une symbolisation graphique de la plupart des éléments du langage OWL. Celle-ci s'intègre également dans le logiciel 'Protégé' sous forme de plugin. L'article souligne que pour un public de plus en plus large et forcément pas toujours expert en ontologies, la claire visualisation de la structure de l'ontologie requêtée est essentielle. VOWL vise donc à une représentation aussi claire qu'intuitive . Une version est également déclinée sur le web, 'WEBVOWL' [web –31]

VOWL, particulièrement sa deuxième version, cible « la représentation visuelle des classes, propriétés et de types de données . Cette information est connu sous le nom de Tbox (Terminological Box) en logique de description et est distinguée des individus et des valeurs de données qui constituent les Abox (Assertionnal Box).

La TBox est la notion la plus importante pour comprendre la conceptualisation décrite dans une ontologie et, de ce fait, considérée comme un ‘citoyen d’honneur’ dans la plupart des visualisations d’ontologies. »
VOWL représente donc en priorité les TBox et, optionnellement ensuite, intègre les informations d’instances (Abox) dans la visualisation, donnant alors une préférence pour l’affichage détaillé sur des instances ou individus dans une autre partie de l’interface de l’utilisateur. Cela vient aussi du fait que l’affichage, ne serait-ce que de quelques instances, rendrait difficile le placement d’informations additionnelles telles les valeurs de propriétés de ces individus. N’oublions pas que le but est de donner à l’utilisateur la vision la plus claire de l’ontologie utilisée afin qu’il puisse lui même s’en servir pour créer ses propres requêtes par exemple.

La figure ci dessous montre la visualisation VOWL (sous plugin de Protégé) d’une petite ontologie (FOAF) :

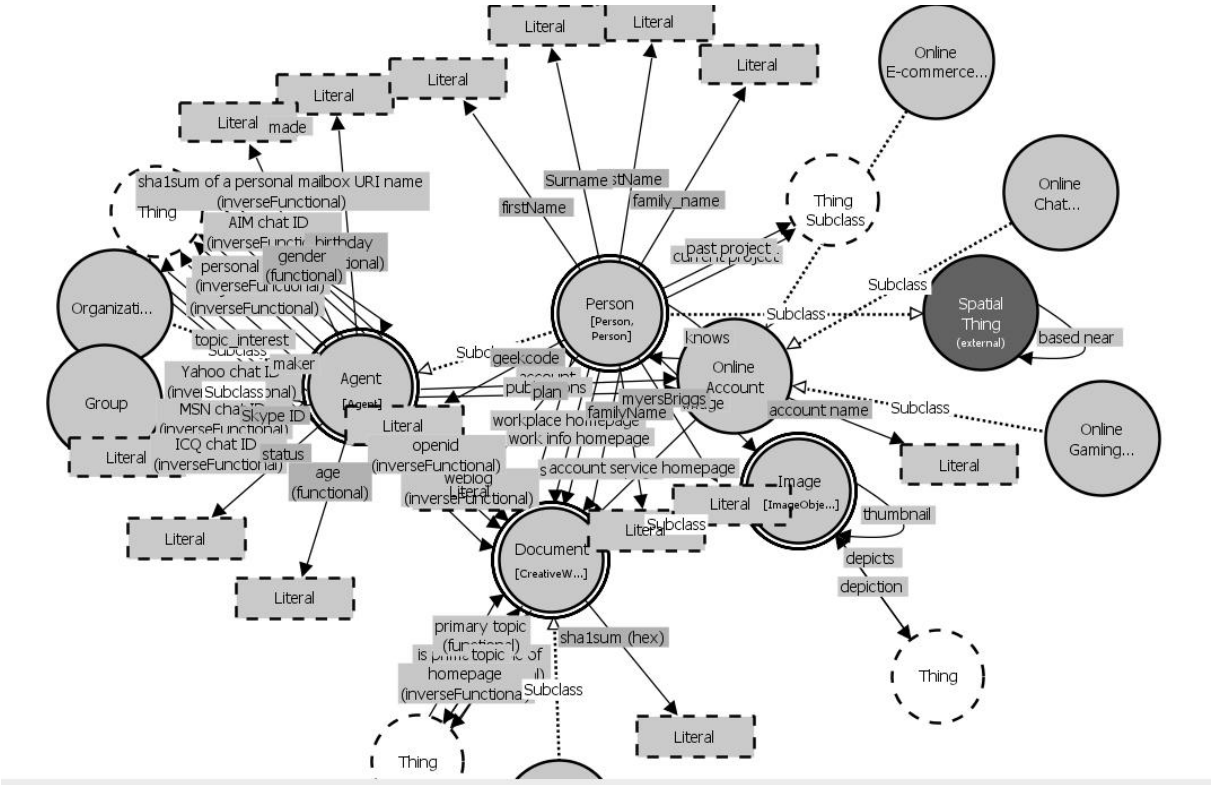


Figure 24 Visualisation VOWL (sous plugin de Protégé) d’une petite ontologie (FOAF)

Les briques graphiques de base de cette visualisation sont un ensemble défini de primitives graphiques et de conventions de couleurs qui exprimeront les attributs spécifiques aux éléments OWL (Propriétés d’objet, propriété de données, classes différentes, caractéristiques de propriétés…) et de leurs combinaisons.
VOWL définit également, par soucis de clarté, de multiplier certaines classes (comme Thing) ou d’en grouper d’autres (classes équivalentes).

Les primitives graphiques sont résumées dans l’article par le tableau suivant :

Table 1
Graphical primitives used in VOWL

Primitive	Application	Primitive	Application
◯	classes	▭	datatypes, property labels
⌒	properties	┅	special classes/properties
▷ ▶	property directions	text number symbol	labels, cardinalities

Tableau 7 Primitives graphiques utilisées dans VOWL

Les classes sont dépeintes par des cercles qui sont connectés par des lignes représentant les propriétés avec leur domaine et leur image. Les labels des propriétés et des types de données sont affichés dans des rectangles.

Le nombre des individus contenus dans une classe peut être, dans certaines implémentations de VOWL, visualisés par la taille du cercle.

Une exception est faite pour la représentation des classes qui représentent owl :Thing (classe mère de toutes les classes des ontologies, chaque classe créée par l'utilisateur est implicitement une sous-classe de owl :Thing). Celle-ci est toujours représentée dans un cercle discontinu de plus petite taille. « La plupart des lignes connectées ont une flèche qui pointe vers la classe ou le type de données qui est défini comme l'image de la propriété représentée par la ligne. S'il n'y a pas de domaine et/ou d'axiome d'image défini pour la propriété, owl :Thing est utilisé comme domaine ou image . »

Pour les propriétés inverses, des flêches sont définies aux deux extrémités de la ligne représentant la relation et les sens sont « mis en lumière » lorsque l'on passe dessus avec le curseur. Les axiomes de sous-propriétés sont également affichés lors du passage de la souris afin d'éviter un enchevêtrement de lignes qui nuirait à la clarté du schéma.

Concernant la représentation des labels de propriétés (figure ci-dessous) , les rectangles n'ont pas de bordure afin de les distinguer de ceux qui représentent les types de données (en lignes discontinues). Le texte du label est, par défaut, celui donné par le tag rdfs :label. S'il n'y a pas de label choisi, la dernière partie de l'IRI de la relation est choisie.

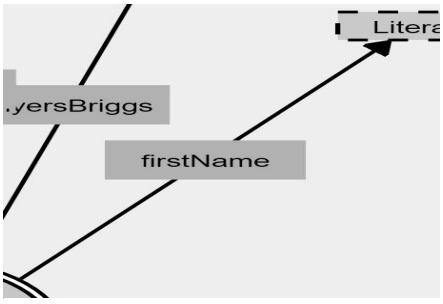


Figure 25 Représentation d'un label de propriété

Le **thème de couleurs** est défini comme dans le tableau ci-dessous :

Excerpt of the VOWL color scheme

Name	Color	Application
General		classes, object properties, disjointness
External		external classes and properties
Deprecated		deprecated classes and properties
Datatype		datatypes, literals
Datatype property		datatype properties
Highlighting		circles, rectangles, lines, borders, arrows

tableau 2.2: Signification des couleurs employées

Il s'agit de couleurs recommandées qui peuvent être modifiées par l'utilisateur dans les paramètres de VOWL. Cependant, les couleurs par défaut sont choisies aussi dansun but précis . Les couleurs 'externes' définies comme étant une version plus sombre que les autres couleurs et les couleurs destinées au type de données qui sont des couleurs 'pastel'. Les couleurs ne sont pas essentielles la visualisation par exemple sur une impression noir et blanc mais apportent un plus dans la compréhension du schéma. (dont l'efficacité est toutefois contestée dans le banc de test de l'article effectué par des utilisateurs lambda).

VOWL utilise ensuite la notion d'éléments visuels pour décrire les structures des langages OWL et RDF, RDFS. Ces éléments visuels sont dépeints dans le tableau ci-dessous :

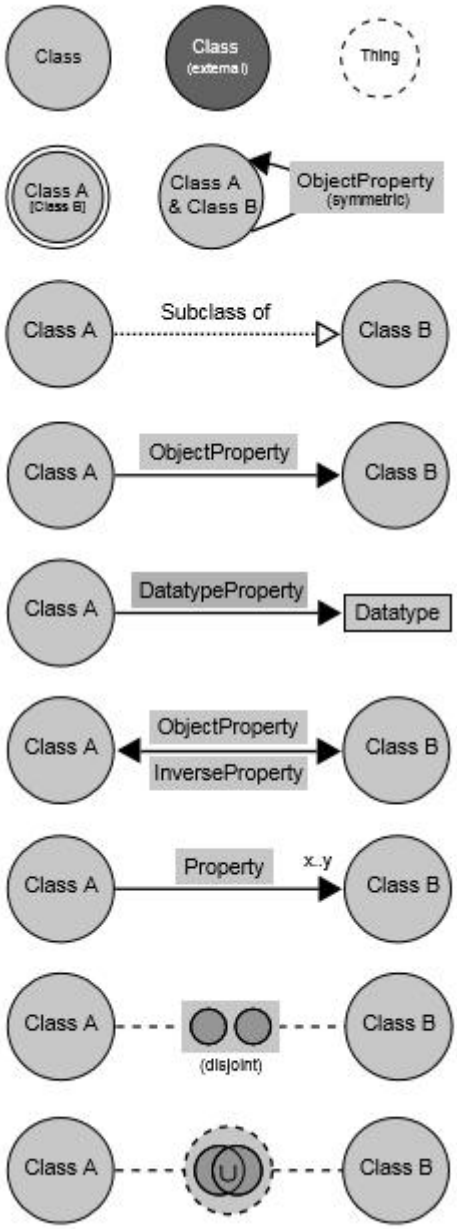


Tableau 8 Eléments visuels de description des structures de langages sous VOWL

Les classes externes dont l'IRI différent de l'ontologie étudiée sont représentées avec une couleur plus sombre et avec un tag 'external'. Les opérateurs de disjonction permettent de faciliter la compréhension de l'effet sous-jacent pour des utilisateurs qui ne sont pas familiers avec les symboles mathématiques usuels. Certains des symboles sont directement inspirés de ceux employés dans les représentations UML comme la généralisation ou la spécialisation, notations jugées intuitives par les participants au banc d'essai de la visualisation VOWL. Des spécifications supplémentaires sont consultables sur http://vowl.visualdataweb.org/v2/#notation.

Dans sa globalité, le graphe représentant l'intégralité de l'ontologie est constitué de telle manière que les nœuds les plus connectés soient placés au centre de la visualisation, le nombre de connections d'une classe étant souvent considéré dans une ontologie comme témoin de son importance. Comme décrit ci-dessus, certains éléments sont dupliqués et d'autres fusionnés permettant ainsi de 'soulager' la prééminence de certains concepts (comme owl :Thing) et permettant d'éviter de longs traits et croisements à travers le graphe qui nuiraient à la visualisation.
La plupart des ontologies sont constituées de très nombreuses classes et propriétés. Pour visualiser cela, VOWL permet de zoomer ou de dézoomer sur la partie de l'ontologie qui nous intéresse permettant ainsi une navigation visuelle instinctive de la globalité de l'ontologie.

b- QueryVOWL

L'examen des spécifications graphiques de VOWL nous permet maintenant d'aborder le fonctionnement de QueryVOWL.
Ce logiciel utilise les éléments graphiques de VOWL et établit les mappings vers SPARQL à partir de ceux-ci.
Le but est qu'aucune notion de SPARQL ne soit requise pour écrire une requête surune ontologie.

Les éléments de VOWL sont donc redéfinis en leur rajoutant des fragments de requête SPARQL.

Un prototype est disponible sur le web à l'adresse suivante :
http://vowl.visualdataweb.org/queryvowl/queryvowl.html
Ce prototype est basé sur les standarts HTML,JavaScript, CSS et SVG (graphisme vectoriel). Elle est basée sur le point d'entrée de l'ontologie dbpedia.
Chaque fragment de code SPARQL ajouté l'est dans le contexte des éléments graphiques déjà présents.

Ci-dessous un exemple illustrant l'interface graphique :

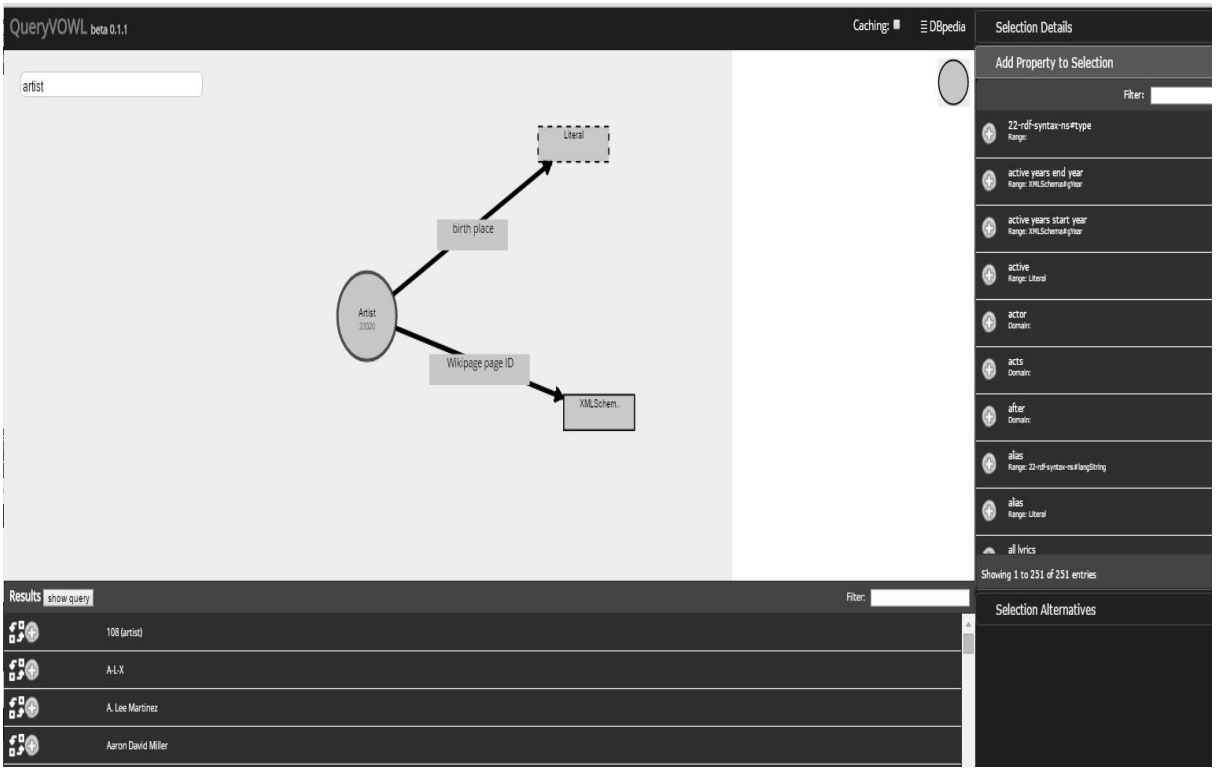


Figure 26 Interface graphique de QueryVOWL

L'élément sélectionné dans la clause SELECT sera celui sur lequel l'utilisateur sera graphiquement placé. Ainsi par exemple :

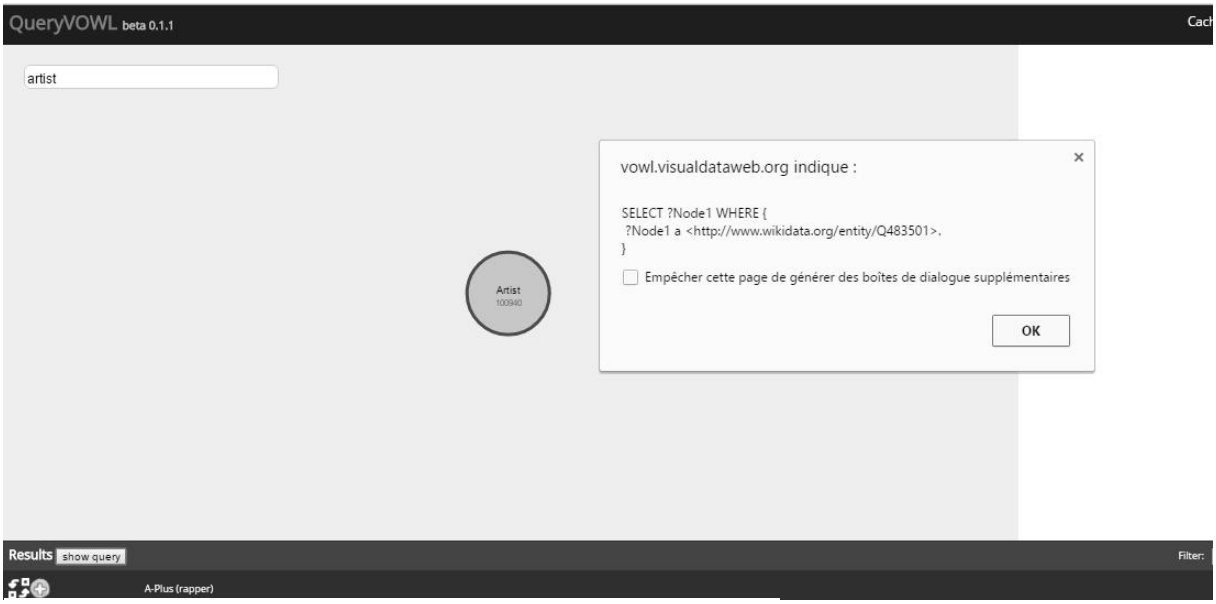


Figure 27 Création d'une clause SELECT par sélection d'un élément graphique

En se plaçant sur le nœud 1, le SELECT aura trait au ?Node1. Ce pourrait aussi bien être aussi sur le nœud d'un littéral ou d'un label de propriété.
La requête est alors automatiquement générée à partir du graphique et envoyé à un 'endpoint' SPARQL choisi (ici dbpedia est imposé par le prototype). (cf *figure 32*)

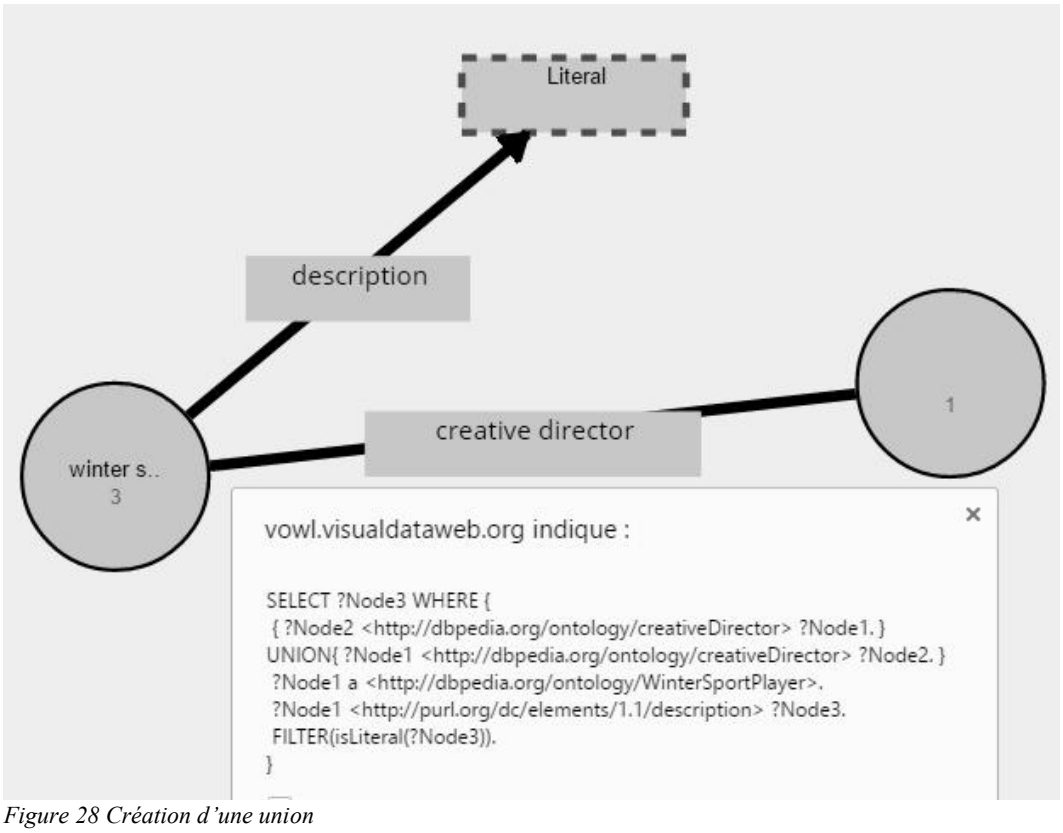


Figure 28 Création d'une union

L'interface générale est constituée de trois panneaux : le panneau principal de création graphique au milieu, une colonne de côté à droite recensant les propriétés, contextuelle à la sélection effectuée et une liste de résultats.

- *Le panneau principal* :

Il contient l'outil de création par drag&drop et utilise la norme graphique W3 SVG (comme pour WebVOWL vu plus haut). Il contient aussi des fonctionnalités et des icônes pour insérer directement des éléments graphiques sur les nœuds et une boite de recherche munie d'un système d'auto complétion basée sur AJAX/JavaScript. Cette boite de recherche voir figure ci-dessous) permet de trouver des entités spécifiques, des classes ou des propriétés dans le dataset qui constitue l'ontologie étudiée et de les insérer directement dans le graphe QueryVOWL. Des IRI's peuvent être également directement déposés par copié-collé.

Figure 29 Boite de recherche

- La barre latérale de droite (cf figure ci-dessous)

Celle-ci est divisée en trois parties. Toutes les informations y figurant sont retrouvées par des requêtes SPARQL mises à jour et effectuées lors de chaque changement dans la fenêtre graphique.
La première partie (SELECTION DETAILS) donne les détails sur l'élément sélectionné et inclut unhyperlien cliquable vers l'IRI concernée et des informations de données littérales si l'élément sélectionné est une instance.

La seconde liste (Add property to selection) liste les propriétés qui peuvent être ajoutées à la classe sélectionnées pour compléter la requête et détaille leur domaine et leur range (domaine image) .

La troisième liste (Selection alternatives) suggère des éléments qui peuvent remplacer l'élément présentemment sélectionné.Pour les classes et les instances, d'autres classes ou instances sont listées susceptibles d'être des remplacements adéquats. Pour les propriétés, d'autres propriétés sont listées. Ces suggestions sont

retrouvées sur labase du schéma RDF utilisé. Ces fonctions peuvent aussi être retrouvées directement au niveau des éléments graphiques.

Figure 30 Exemple de barre latérale

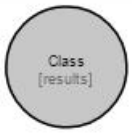
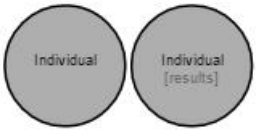
- *La liste des résultats*

Celle-ci contient toutes les instances correspondantes au résultat de la requête avec des icônes permettant le remplacement de l'élément selectionné dans le panneau principal ou la mise sur graphique de l'instance choisie.

Pour ce qui est de la recherche des données à partir du 'endpoint', un cache est proposé afin de pallier aux temps de réponse parfois un peu long de recherche des données correspondantes aux requêtes et requêtes sous-jacentes des panneaux latéraux.

Le prototype ne propose pour le moment pas toutes les fonctionnalités proposées (comme les disjonctions de classes par exemple).

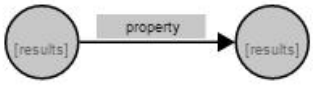
Concernant le fonctionnement du mapping, les deux tableaux suivants montrent comment le code est généré par rapport à chaque type d'élément graphique [web –32]

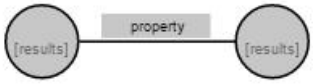
Graphical representation	Description	SPARQL mapping
	A placeholder for an arbitrary individual. The individual may be restricted by connections to other nodes. Each result for the complete query assigns a particular individual to a placeholder. Circle with the same visual appearance as a VOWL OWL class. The circle contains a results preview.	A new variable is created for each individual: `?x`
	A placeholder for an individual of a given type. An individual can be restricted to belong to a given class. In that case, the class name is indicated above the results preview.	The variable representing the individual is included in a triple that restricts the type: `?x a example:Class .`
	A specific individual is represented by a circle whose visual style matches that of a VOWL RDFS class. Such a specific individual node can either be interpreted as a constant value in the graph pattern, or as a set of values restricted to exactly one value. In order to reflect the latter interpretation, a results preview can be optionally displayed below the name of the individual.	The IRI of the individual appears in the query: `example:Individual`

	A Union node acts as a placeholder for any one individual or literal eligible for the connected nodes.
	The VOWL Union notation is used to represent a Union node.
		A new variable is generated for each Union node. Restrictions valid for any connected nodes (restrictions defined both in the connected nodes and the first level of connections to other nodes) are applied to the new variable and combined to a UNION:
		`?x a example:Class1 .`
		`?y a example:Class2 .`
		`{`
		`  ?s a example:Class1 .`
		`} UNION {`
		`  ?s a example:Class2 .`
		`}`

	An Intersection node acts as a placeholder for individuals to which all restrictions of all connected nodes apply.
	The VOWL Intersection notation is used to represent an Intersection node.
		A new variable is generated for each Intersection node. Restrictions valid for any connected nodes (restrictions defined both in the connected nodes and the first level of connections to other nodes) are applied to the new variable:
		`?x a example:Class1 .`
		`?y a example:Class2 .`
		`?s a example:Class1 ;`
		`  a example:Class2 .`

	A rectangle with a dashed border, like the VOWL RDFS literal, and the text "Literal", or the name of a datatype, represents an unrestricted literal value.	Just like an unrestricted individual, an unrestricted literal value is represented by a new variable: `?l`
	A restricted literal value is displayed as a rectangle with the same visual style as a VOWL RDFS datatype node. The restriction is displayed as a text consisting of a comparison operator and a value supplied for the comparison.	The restriction is added as a FILTER expression: `FILTER(?l = "Value") .`

Graphical representation	Description	SPARQL mapping
	The connection of two individuals, or an individual and a literal, by a property can be enforced with a property edge. The property edge looks like a VOWL ObjectProperty. For unrestricted properties (i.e., properties that may have any IRI as long as they connect the indicated individuals in the specified direction), the label box on the property edge is empty.	A new variable is generated for the property and used as a predicate in the triple pattern between the connected nodes: `?x ?p ?y .`
	The connection of two individuals, or an individual and a literal, by a property can be enforced with a property edge. The property edge looks like a VOWL ObjectProperty.	The property is supplied as a predicate in a triple pattern between the connected nodes: `?x example:property ?y .`

	Undirected property edges can be used to express a connection of two individuals, or an individual and a literal, by a property without specifying its direction.
	The undirected property edge looks like other property edges but lacks an arrowhead. Both unrestricted and restricted properties can be undirected.
		The two triples expressing the two directions are combined to a UNION:
		`{ ?x example:property ?y . } UNION { ?y example:property ?x . }`
		For restricted properties, the SPARQL property path syntax may be used alternatively:
		`?x example:property|^example:property ?y .`

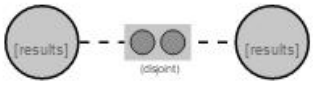
	To enforce that two individuals or literals are different, they can be connected with a disjoint edge. The disjoint edge looks like the VOWL DisjointWith notation.	A FILTER expression is used to enforce that the two connected nodes are different: `FILTER(?x != ?y) .`

Tableau 9 Mapping graphisme/code requête VOWL

3- QaRS

Qars est une autre aide graphique à la construction de requêtes. Au niveau des réponses, il a de plus l'avantage d'être muni d'un système 'query-and-relax' permettant, lors de l'échec d'une requête (dans le cas où il n'y a pas de triplets correspondants), de proposer une requête alternative 'étendue' et permettre aussi de voir les parties de requêtes qui peuvent amener l'erreur (procédé MFS, Minimal Failing Subqueries que nous examinerons plus précisément par la suite). [art-22] [art-23] [art-24]

L'article propose à ce sujet une ontologie simple (cf *figure 36*)

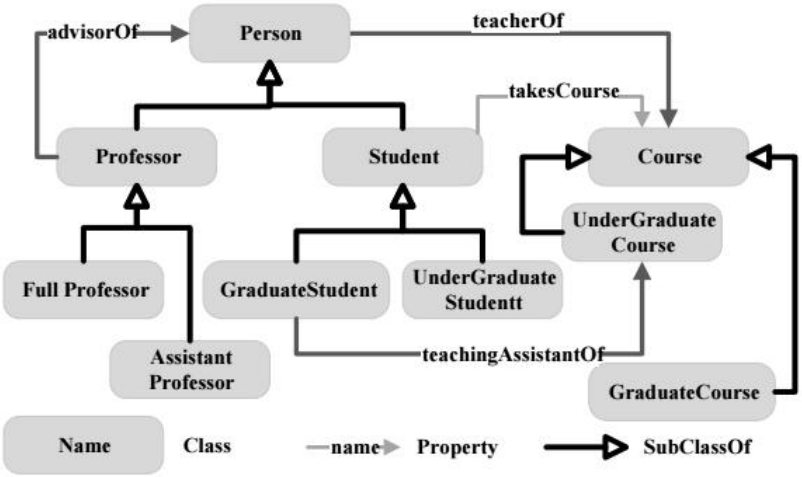


Figure 31 Ontologie simple servant de base à l'étude de QARS

Et la requête suivante qui recherche le ?x ?y ?z tels que :

x soit l'assistant-enseignant de y
et que x soit un assistant professeur
et que y soit du type 'undergraduate course'
et que z soit enseignant de y
et que z soit advisor de x

qui se traduit par :

```
SELECT ?X ?Y ?Z WHERE {
        ?Z ub:teacherOf ?Y.
        ?Y rdf:type ub:UnderGraduateCourse.
        ?X ub:teachingAssistantOf ?Y.
        ?Z ub:advisorOf ?X.
        ?X rdf :type ub:AssistantProfessor.
}
```

La requête ne retournera aucun résultat. L'erreur résulte d'une connaissance incomplète de l'utilisateur qui pourrait être automatiquement (ou manuellement) rectifiée.
Ici, l'utilisateur dans les lignes
?X ub:teachingAssistantOf ?Y.
et
?X rdf :type ub:AssistantProfessor.

Indique (par inférence) que ?X est de type GraduateStudent et aussi de type AssistantProfessor ce qui n'est pas conforme à l'ontologie.

De plus, même si la requête avait été correcte, il se peut aussi qu'elle eût été trop restrictive et donc, dans ce cas, il peut aussi être intéressant de proposer des solutions alternatives. Dans ces deux cas, les techniques de 'relaxation' des requêtes trouveront leur utilité.

Design et méthode graphique :

L'article commence donc par sa manière de réaliser graphiquement un requête SPARQL.
Le panneau 'browser' présente l'ontologie comme un graphe. (voir figure ci-dessous).

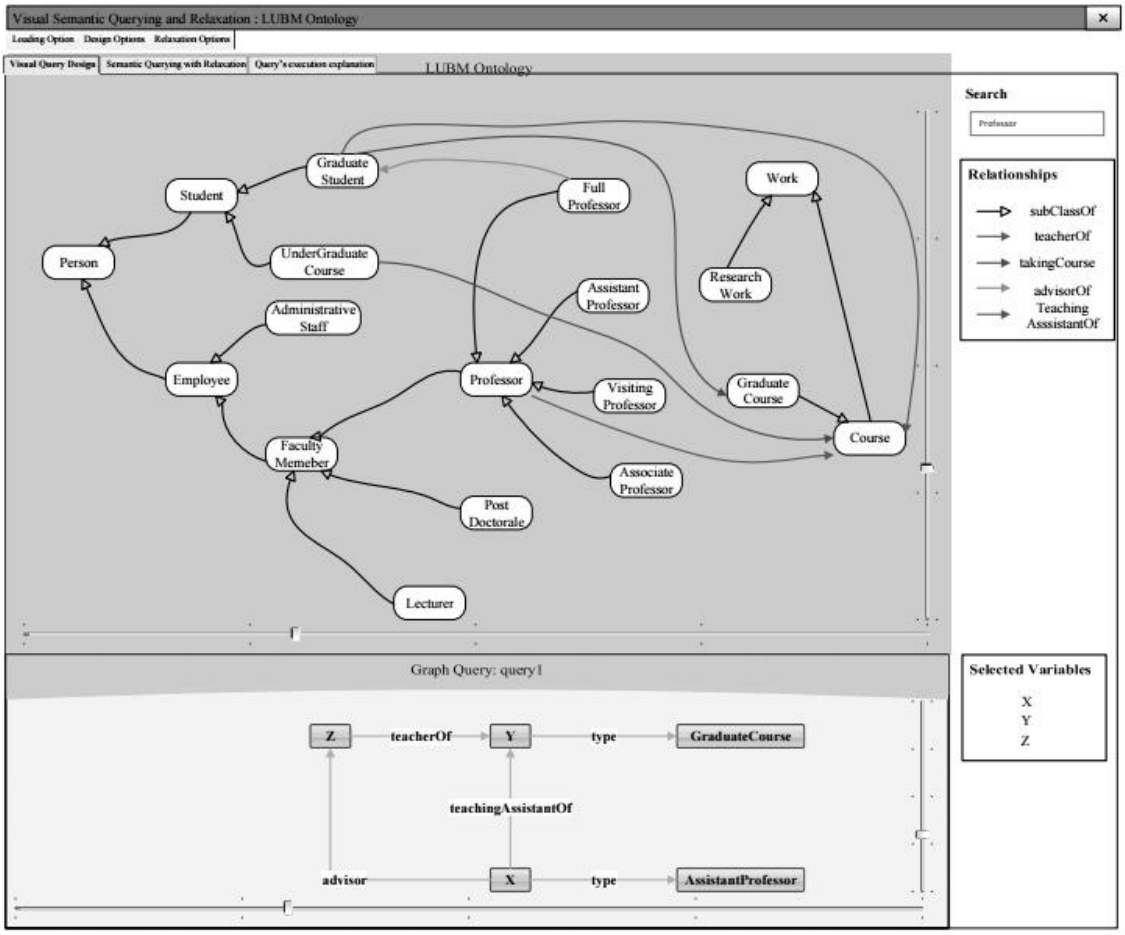


Figure 32 Présentation de l'ontologie dans le panneau graphique

Comme dans QueryVOWL, une boîte de recherche avec auto-complétion est disponible afin de trouver des classes et des propriétés dans des ontologies de grande taille. Quand une classe ou une propriété est choisie, le graphe est centré dessus afin de présenter tous ses concepts et propriétés relatifs.

L'article décrit ensuite la construction d'une requête entrois étapes :

- Drag & Drop des classes ou propriétés à partir du panneau de browser d'ontologie vers le panneau de construction de requête(en bas).
    - « Mettre une classe C dans un panneau crée un graphe correspondant au triplet (?Vi rdf :type C) « avec Vi qui est une variable qui n'a pas encore été utilisée.
    - De la même manière, déposer une propriété P crée un graphe de type (?Vi P ?Vj)
- Liaison des triplets patterns défini à la première étape en identifiant les variables qu'ils partagent. Cela se fait ici en (drag & drop) prenant une variable dans un triplet et en la déposant sur une autre variable d'un autre triplet. Cela indiquera que les deux variables sont les mêmes. A ce stade Qars teste s'il n'y a pas d'inconsistance dans la requête notamment au niveau des domaines et images des propriétés. Notre requête précédente aurait donc déjà mis en surbrillance la dernière ligne de la requête, celle qui révélait une inconsistance de type par rapport à l'ontologie étudiée.
- Ajout des opérations 'FILTER, OPTIONAL, UNION' en sélectionnant avec le bouton droit de la souris les parties de la requête sur lesquelles ces options doivent agir (variable, triplet ou ensemble de triplets). Chacun de ses opérateurs sera identifié par une couleur spécifique sur le browser de requête. Les modifications peuvent se faire aussi bien au niveau textuel que graphique et les modifications répercutées en parallèle.

Les méthodes de 'relaxation' (révision) de requêtes :

Aspect intéressant de Qars et non proposé par exemple surQueryVOWL, les auteurs décrivent la mise en place de 3 modes de fonctionnement et de mise en oeuvre de ces algorithmes :

- ***Mode automatique et itératif***

Lors de la construction précédente de requête, l'utilisateur a la possibilité d'indiquer qu'il attend k réponses. Si le nombre de réponses est inférieur à k alors Qars enclenche son système de modification de requête ('relaxation')comme suit :

- Traitement d'un ensemble de requêtes dérivées de la requête initiale.
- Le système les ordonne ensuite avec le degrè de similarité le plus proche de la requête initiale
- Le système les exécute ensuite selon l'ordre précédemment établi jusqu'à obtenir le nombre k de réponses attendues. Si les réponses sont nombreuses, elles sont classées selon leur degrè de pertinence à la requête initiale.

- ***Mode manuel : processus de substitution par des préférences utilisateurs***

Les utilisateurs peuvent manuellement définir des contraintes sur des parties de requêtes dans le panneau de requête.

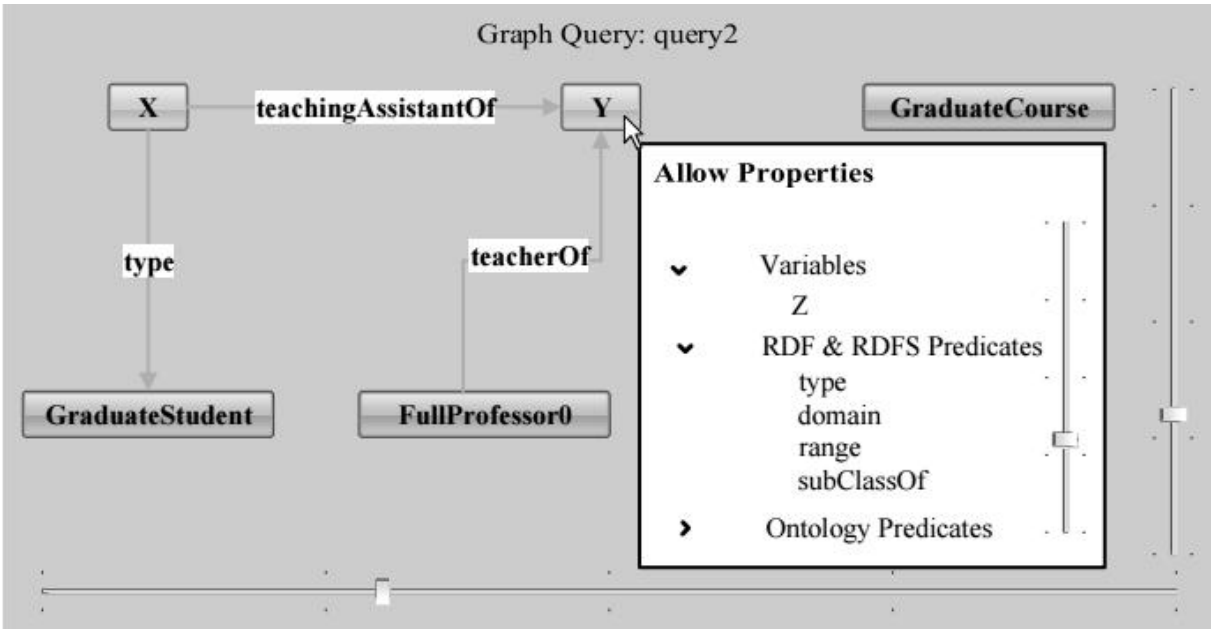


Figure 33 Réglage des préférences uilisateurs

Trois types de contrainte peuvent être définies (accessibles avec le bouton droit sur l'élement concerné) :

-i) Les pattern de triplets qui ne doivent pas être 'relaxés'

-ii) Les classes ou propriétés admises dans la hiérarchie de la classe qui peuvent être utilisées à la place de la classe de base (classes mères) . Ci-dessous, on permet à ce que le type GraduateStudent soit élargi au type Person en cas d'échec de la requête initiale. A cette étape, Qars teste si la classe sélectionnée reste cohérente avec le reste de la requête.

-iii) Définition des valeurs de tolérance pour les filtres selon leur type de donnée.

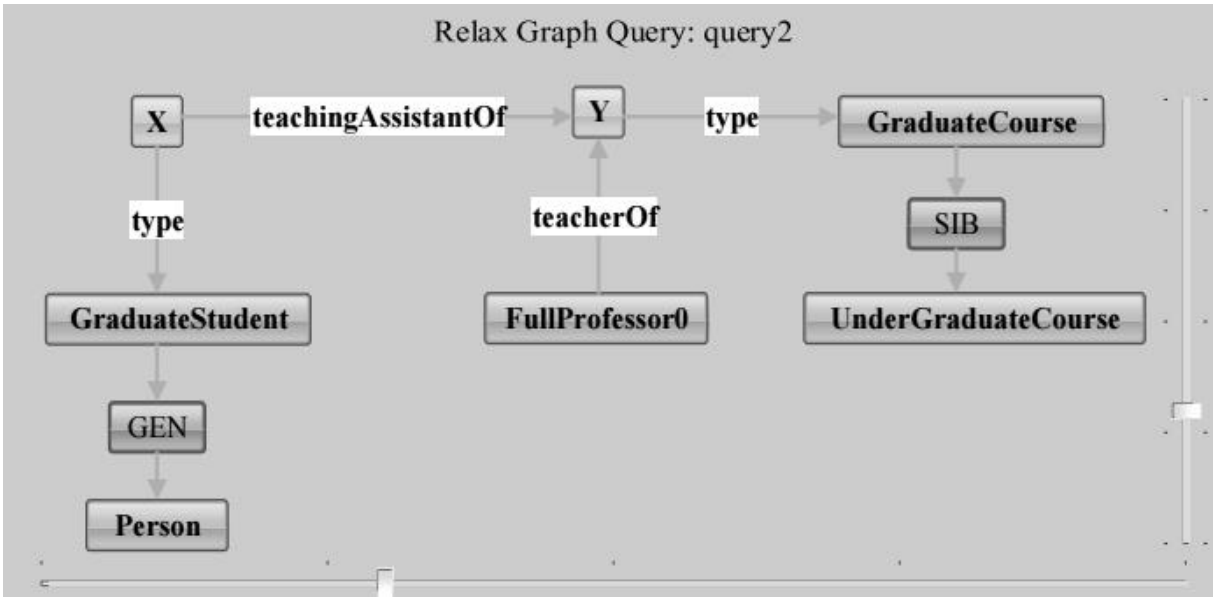


Figure 34 Réglage des réglages de 'relaxation' de classes.

Reste ensuite à définir le nombre k de réponses attendues.

- ***Mode interactif avec Feedback utilisateurs***

Dans les modes précédents, l'utilisateur ne connaît pas les causes de l'échec de sa requête. Ce mode fournit des explications à l'utilisateur en affichant l'ensemble des MFS (ou Minimal Failing SubQueries dont nous verrons succintement la détermination) .

« Chaque MFS est :

i) Une sous-requête défaillante de la requête initiale
ii) N'inclut pas elle-même une sous-requête défaillante. »

Logiquement, si une MFS n'est pas 'rectifiée' (relaxée), la requête initiale ne retournera que des réponses vides. La rectification se fera dans ce cas en deux phases :

Phase 1 : Qars affiche l'ensemble des MFS de la requête initiale
Phase 2 : L'utilisateur peut automatiquement ou manuellement rectifier chacune des MFS. Par défaut, QARS rendra OPTIONAL les patterns triplets concernés par ces MFS.

La **similarité** est la notion clef de la méthode Qars. Utilisée d'un côté pour mesurer la similarité entre la requête initiale et les requêtes modifiées et d'un autre côté pour calculer le degrè de satisfaction des solutions alternatives présentées en accord avec la requête initiale. Ce dernier point permet de proposer à l'utilisateur un ensemble de solutions sélectionnées desquelles il pourra choisir ce qu'il jugera être les k meilleures réponses. « Quand à la similarité des requêtes, le fait de classer les requêtes et de choisir un ensemble de requêtes appropriées pour obtenir les meilleures k-réponses aidera l'utilisateur. »
Ces requêtes exécutées pourront être vues par l'utilisateur comme une aide, une approche, lui permettant d'atteindre les réponses attendues.

Architecture de QARS :

L'architecture est illustrée dans la figure ci-dessous dans l'article :

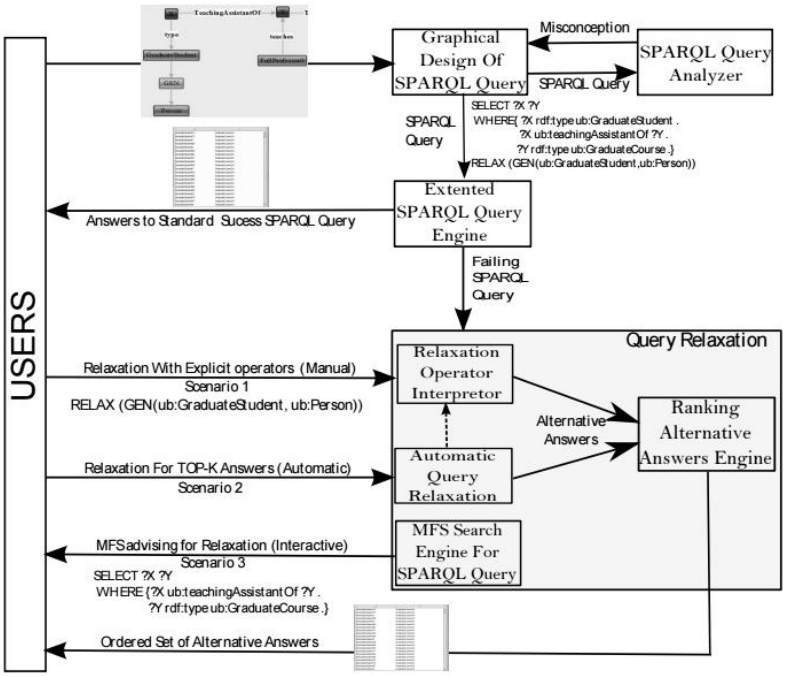


Figure 35 Architecture de QARS

Deux grandes parties constituent cette architecture : Le design graphique de SPARQL et l'Analyseur de requête SPARQL.

1) Le design graphique de SPARQL (GDSQ):

GDSQ offre une interface intuitive pour l'utilisateur afin de créer ses requêtes . Cette interface teste en simultané la syntaxe et la consistance de la requête graphiquement créée. Come vu plus haut cette intrface propose un champ d'autocomplétion et de suggestion de classes et de propriétés pour construire la requête.

2) L'analyseur de requête

Il constitue le cœur de la ré-évaluation de requêtes et est composé des quatre modules suivants :

a) L'interpréteur d'opération de 'relaxation'.

Ce moteur interprête les trois opérateurs de relaxation GEN, SIB et PRED et génére les les ensembles correspondants de requêtes étendues.
L'opérateur GEN prend un concept Ci en entrée et son super concept Cf. Le système génère alors la requête en remplaçant dans celle-ci la classe Ci par la classe Cf étendant ainsi le type et également la portée de la requête.

Pour l'opérateur SIB, le système remplace successivement une classe C0 par ses classes cibles C1,C2,…,Cm.

L'opérateur PRED 'relâche' petit à petit, par incrémentation les filtres qui impliquent des types de données littéraux (numérique, string, …).

b) La ré-évaluation de requête automatique.

Ce module est déclenché lorsque l'utilisateur choisit de vouloir obtenir k réponses sans agir sur les opérateurs de relaxation. Le moteur génère alors toutes les requêtes en utilisant les trois opérateurs vus précédemment. Ces requêtes sont alors classées selon leur similarité avec la requête modifiée. La mesure suivante entre classes est utilisée :

$$Sim(Ci,Cf) = IC(msca(Ci,Cf)) / (IC(Ci) + IC(Cj) - IC(msca(Ci,Cf)))$$

Avec Ci :classe initiale, Cf : classe finale,

IC : quantité d'informations dans une classe,
msca(C1,C2) : le plus proche ancêtre commun de C1 et C2.

Les requêtes modifiées sont alors exécutées en commençant par la plus similaire à la moins similaire et les réponses obtenues sont envoyées vers le moteur de classification des requêtes et réponses.

c) Le moteur de classification des requêtes et réponses.

QarS donne à l'utilisateur un ensemble de réponses alternatives. « Chaque réponse hi est associée avec un score de satisfaction calculé comme suit :

SatQ(hi)=min(Sim(Q',Q),SatQ'(hi)) » avec Sim, fonction de similarité définie ci-dessus telle que sim(Q,Q') mesure l'indice de similarité entre une requête Q et sa forme 'relâchée' Q'.

SatQ'(hi) est la satifiabilité de la réponse hi engendrée par la requête Q'.

d) Le moteur MFS pour SPARQL.

MFS (Minimal Failing Subqueries) est un procédé qui permet d'identifier la cause d'un échec d'une requête.
«

- La requête est formalisée sous forme de conjonction de conditions : C1 $\wedge$ C2 $\wedge$ …Cn
- Recherche est faite de la plus petite sous liste de conditions qui annule la requête »

« Pour trouver les autres MFS, un ensemble de sous-requêtes significatives (SSQ) est calculé. Chaque SSQ est caractérisée par 3 propriétés

- Elle ne contient pas les MFS déjà trouvées
- Elle n'est pas incluse dans ces MFS
- Elle n'inclut pas d'autres SSQ

Ce procédé est réalisé récursivemet sur chaque élément de l'ensemble des SSQ.
Chaque MFS trouvées par ce procédé sont alors montrées à l'utilisateur comme explication de l'échec de sa requête. »

4- VIZIQUER 3

La première version de Viziquer fût créée en 2010. [art-25] [web –33]
Viziquer permet de se connecter à un endpoint SPARQL ou à un fichier d'ontologie local et de construire graphiquement des requêtes SPARQL pour retrouver les données. Le logiciel permet également d'explorer la structure de l'ontlogie avec son explorateur OWLGred qui s'appuie sur la syntaxe Manchester d'OWL (https://www.w3.org/TR/owl2-manchester-syntax/) pour la symbolisation des classes (concepts) .
Notre parcours : comprendre la syntaxe Manchester pour comprendre la visualisation OWLGred pour comprendre enfin le fonctionnement de Viziquer .

a- Syntaxe Manchester pour OWL

En pré-requis de ce qui suivra, un bref examen de ce qu'est la syntaxe Manchester [web -34] est nécessaire.

Manchester est une des syntaxes servant à décrire une ontologie OWL. (nous avons aussi OWL/XML, Turtle et RDF/XML).
Nous reprendrons ici le parcours décrivant les notions de base de ce langage d'ontologie OWL issues du cours de Guy Lapalme ,Université de Montréal.

RDF/XML est le format habituel, format d'échange. Manchester permet une meilleure lisibilité de la structure de l'ontologie.

Description d'une instance

Marie est une personne :
Individual : Marie
Types : Person

membreX est une instance de la classe C :
Individual : membreX
Types : C

Propriétés d'objets (relations entre instances)

Marie est femme de Joe
Individual : Marie
Facts : est_femme_de Joe

Ou négatif :

Marie n'est pas la femme de Joe :
Individual : Marie
Facts : not est_femme_de Joe

Hiérarchies de classes

Sous-classes :

Class : tigre
SubClassof : mammifères

Classes disjointes :

DisjointClasses : animaux, végétaux

Hiérarchies de propriétés

ObjectProperty : a_pour_couleur
SubPropertyOf : a_pour_description
EquivalentProperties :
Est_decrite_par, autre_onto :has_description

Restrictions sur les propriétés

ObjectProperty : a_pour_epouse
Domain : homme
Range :femme

Egalité et inégalité des individus

Différence :
Individual : Marc
DifferentFrom : Jerome

Identique :
Individual : John
SameAs : Jean

Types de valeurs de propriétés :

Positif :

Individual : Jean
Facts : est_age_de « 41 »^^xsd :integer

Négatif :

Individual : Annie
Facts : not est_age_de « 41 »^^xsd :integer

Domaine et portée des propriétés :

DataProperty : est_age_de
Domain : Personne
Range : xsd :nonNegativeInteger

Opérations de classes complexes :

Intersection :

Class : Mère
EquivalentTo : Femme et Parent

Union :

Class : Parent
EquivalentTo : Pere ou Mere

Complement :

Class : ChildlessPerson
EquivalentTo : Personne and not Parent

Définition de sous-classes :

Class : GrandParent
SubClassOf

Restrictions de propriétés :

Quantification existentielle :

Class : Parent
EquivalentTo : a_pour_enfant some Personne

*Quantification universelle :*Une personne est heureuse si tous ses enfants sont heureux

Class : Personne_contente
Equivalent_to : a_pour_enfant only Personne_contente

Restrictions de cardinalité :

Cardinalité maximum :

Individual : John

Types : a_pour_enfant max 4 Parent

(John fait partie de la classe des gens qui ont au plus 4 enfants qui sont parents

Cardinalité minimum :

Individual : John
Types : a_pour_enfant min 2 Parent

Cardinalité exacte :

Individual : Jean
Types : a_pour_enfant exactly 3 Parent

Enumeration d'individus :

Class : MesAmis
EquivalentTo : {Marc, Luc, Gino, Isabelle}

Caractéristiques des propriétés :

Inverse :

ObjectProperty : a_pour_parent
InverseOf : est_enfant_de

Symétrie :

ObjectProperty : a_pour_epouse
Characteristics : Symmetric

Asymétrie :

ObjectProperty : a_pour_enfant
Characteristics : aSymmetric

Disjonction :

DisjointProperties : a_pour_parent, a_pour_epoux

Réflexivité :

ObjectProperty : a_pour_frere
Characteristics : Reflexive
ObjectProperty : parentOf
Characteristics : Irreflexive

Unicité de valeur :

ObjectProperty : a_pour_mari
Characteristics : Functional

ObjectProperty : a_pour_mari
Characteristics : InverseFunction

Transitivité :

ObjectProperty : a_pour_ancetre
Characteristics : Transitive

Chaîne de propriétés :

ObjectProperty : a_pour_grand_parent
Characteristics : a_pour_parent o a_pour_parent

ObjectProperty : a_pour_oncle
Characteristics : a_pour_parent o a_pour_frere

Annotation : Décrit la classe énoncée. Ne rentre pas dans la logique de l'ontologie

Class : Person
Annotations :
Rdfs : comment : « Représente l'ensemble de toutes les personnes »

b- OWLGred [art-26]

Nous nous attarderons donc quelques instants sur la formalisation et la représentation utilisée afin de mieux comprendre par la suite le fonctionnement de Viziquer 3.

OWLGrEd permet la visualisation ou l'édition graphique d'une ontologie OWL en utilisant une représentation intuitive qui combines des diagrammes de classes UML avec la syntaxe Manchester pour les expressions de classe.

Il est installable et disponible en format .jar sur http://owlgred.lumii.lv/get_started#Download#Download .

Dans son article « OWLGrEd: a UML Style Graphical Editor for OWL", Jānis Bārzdiņš & al expliquent leur démarche en remarquant que les précédentes visualisations d'ontologies proposées (IsaViz, OWLvi, GrOWL, WELKIN) ont le défaut d'être trop 'fouillis', se contentant de présenter tous les triplets comme deux nœuds (classes) associées par un lien (propriété) ce qui rend la structure difficile à concevoir par une telle représentation étant donné le nombre important de triplets que peut comporter une même ontologie.

L'article de Kārlis Čerāns décrit l'intérêt de cette nouvelle version de Viziquer de par l'ajout du traitement des agrégats dans les requêtes SPARQL (count, distinct, group…).
Pour qu'une visualisation graphique soit efficace, elle doit regrouper les éléments communs ensemble, approche qui a été utilisée avec succés dans les diagrammes UML. Beaucoup de concepts d'OWL sont très proches de ceux exprimés par les diagrammes UML. Toutefois OWL a plus de souplesse et de fonctionnalités que les diagrammes UML comme les descriptions de classes (enumeration, conjonction, disjonction, négation), les classes anonymes, les axiomes de classe, les restrictions de propriétés, etc…

La proposition de l'article est donc d'étendre les notations de diagrammes de classes UML avec une syntaxe similaire à la syntaxe Manchester pour pallier aux fonctionnalités OWL manquantes et rendre cette notation compacte et compréhensible.

Pour la plupart des fonctionnalités il s'agira d'un mapping point à point d'OWL vers les concepts UML :
- Ontologies vers paquetages
- Classes OWL vers les classes UML
- Les propriétés de données vers les attributs de classes
- Les propriétés d'objets vers les associations

- Les instances vers les objets

Ensuite des fonctionnalités non présentes dans UML sont ajoutées :

- On ajoute aux classes UML des champs où des expressions OWL peuvent être insérées : classes équivalentes, expressions de super-classes, disjonction de classes, annotations, commentaires.

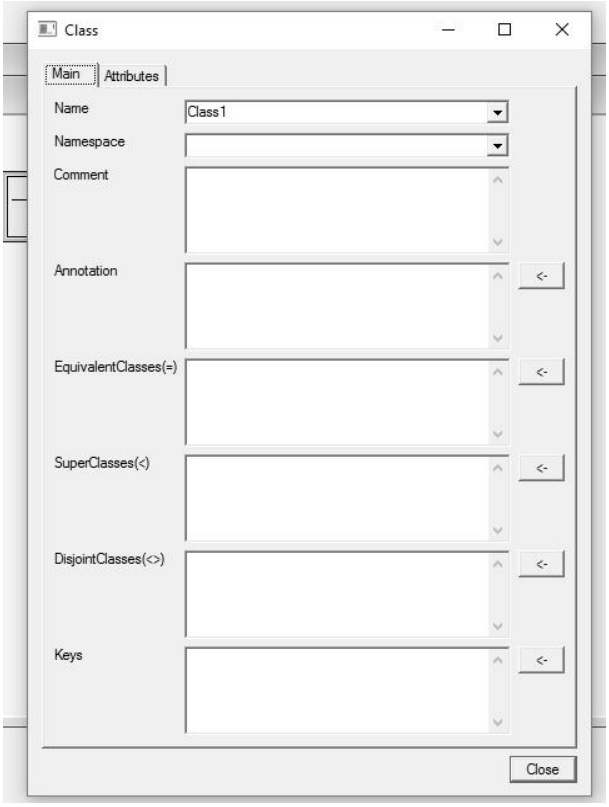


Figure 36 Ajout d'expressions et de propriétés OWL à une classe

- Des champs similaires sont ajoutés aux associations et aux attributs.

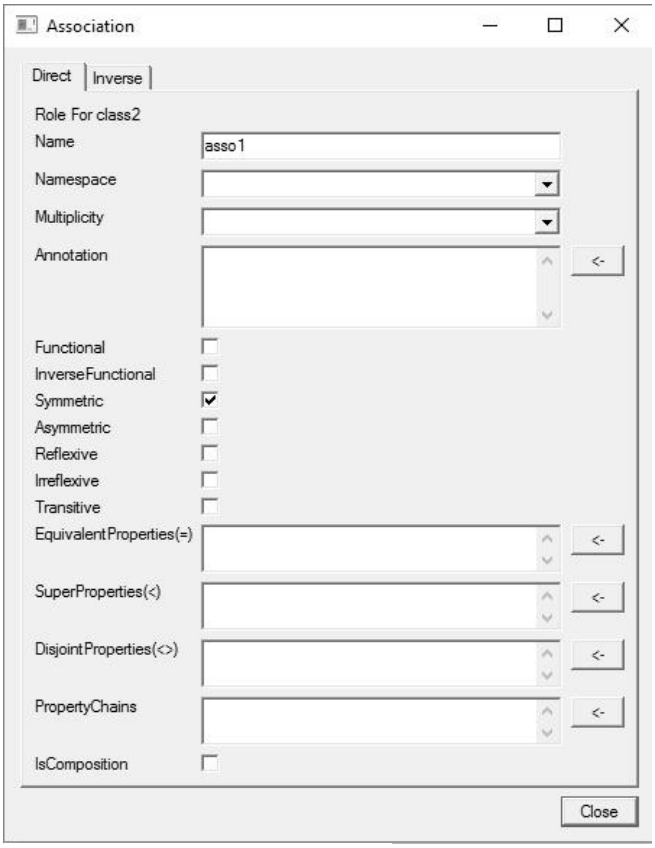


Figure 37 Ajout d'attributs de propriété

- Les classes anonymes sont montrées dans des boîtes avec uniquement les possibilités de définitions de classes équivalentes. (voir construction ci-dessous)

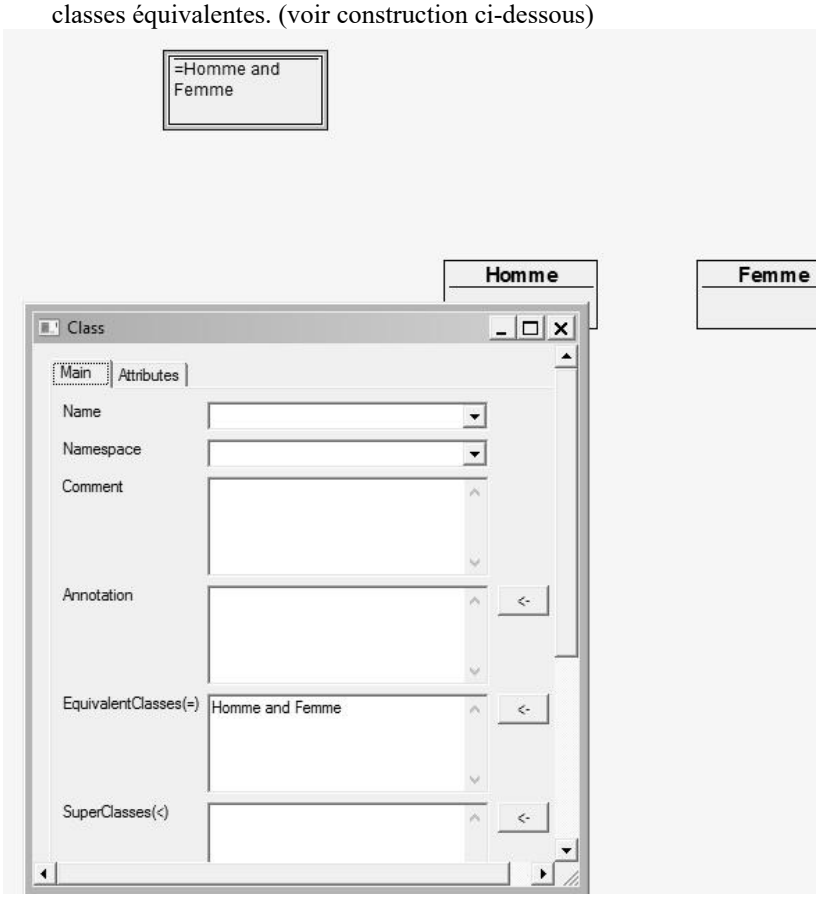


Figure 38 Boîte représentant une classe anonyme

« Il y a plusieurs façons de visualiser les superclasses anonymes :

1- Comme des expressions textuelles
2- Comme une ligne de généralisation de la sous-classe vers la classe anonyme correspondante
3- Comme contraintes multiples dans une association
4- Comme une ligne de restriction vers la classe correspondante » (voir la figure ci-dessous, en particulier la ligne rouge ‘eats’ entre les classes Lion et Herbivore.

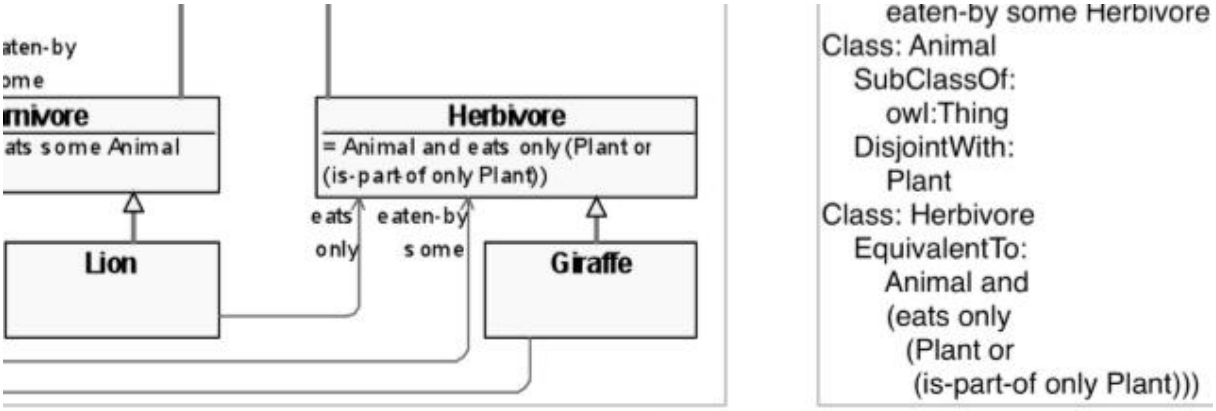


Figure 39 Plusieurs façons de visualiser les superclasses anonymes

L’ontologie peut être divisée en plusieurs diagrammes à l’intérieur de paquetages dans lesquels chaque diagrammes montrera une vue différente de l’ontologie ou un sous-ensemble. (figure ci-dessous)

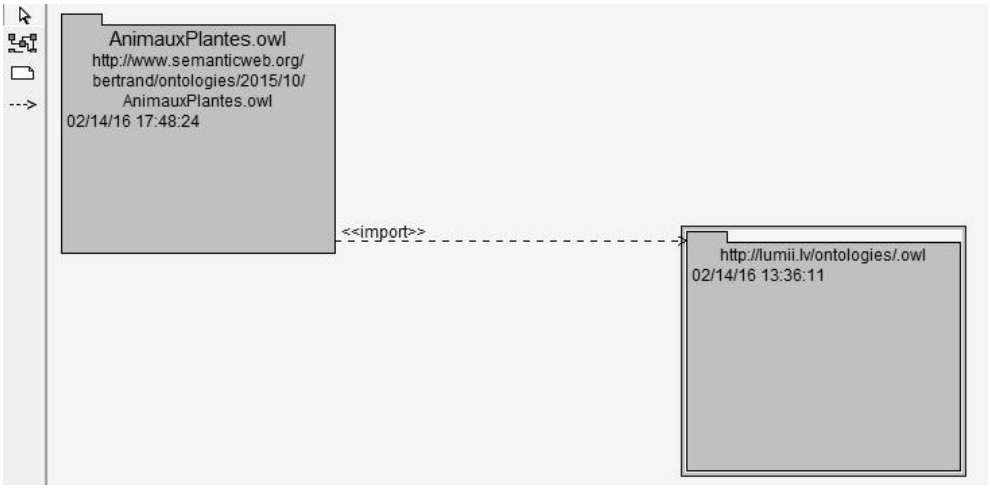


Figure 40 Paquetages

Les lignes multiples de généralisation peuvent être fusionnées avec un symbole de fourche pour réduire le nombre de lignes arrivant dans une super-classe ou par une notation en mode texte(figure ci-dessous) :

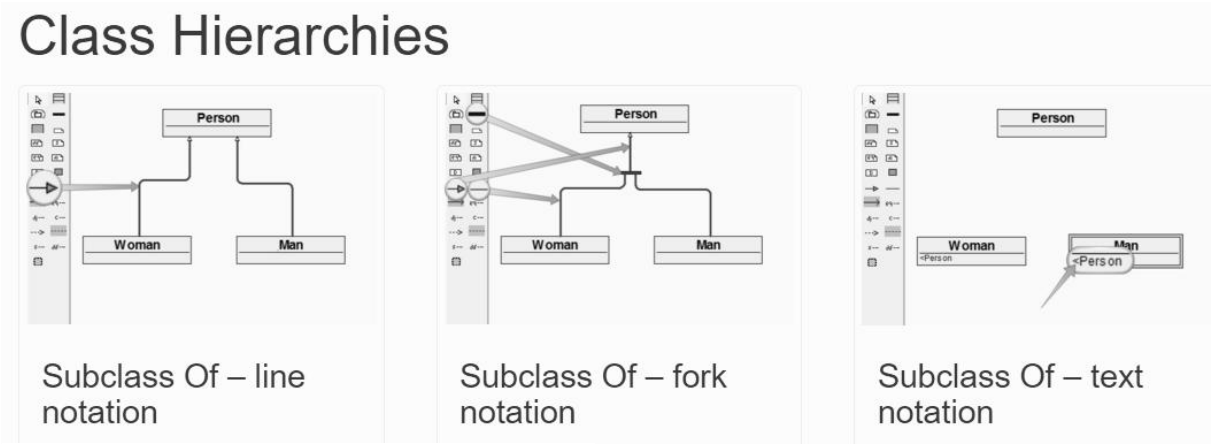


Figure 41 Différentes symbolisation de hiérarchie de classes.

Afin d'illustrer ces notations et les liens que celles-ci ont avec l'écriture Manchester vue ci-dessus, l'article propose le schéma d'ontologie suivant suivant :

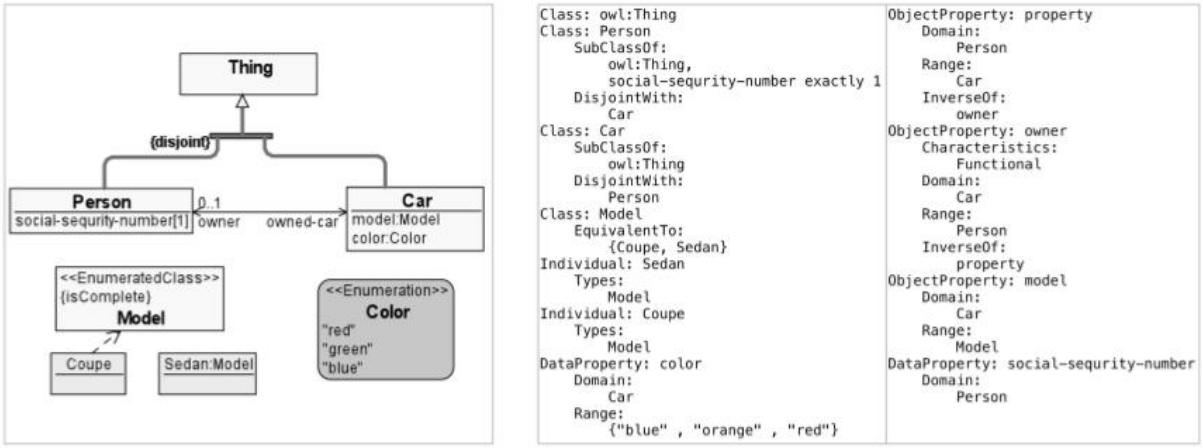


Figure 42 Ecriture Manchester décrivant une ontologie

Cette visualisation utilise uniquement les spécifications standarts d'UML, les classes étant représentées par des boîtes rectangulaires et les propriétés de données indiqués comme des labels dans la boîte de la classe. Les propriétés d'objets sont représentées par des lignes entre les boites correspondants à leurs classes de domaine et d'image (range). Si la propriété d'objet a un inverse alors elles sont toutes deux représentées par la même ligne (cf owner et owned-car) avec une flèche aux deux extrémités.

Les restrictions de cardinalité d'une propriété sont montrés au dessus de la ligne près du nom de la propriété (comme usuellement dans UML). Si l'image d'une propriété de données est une énumération de valeurs, celle ci est alors schématisée comme un rectangle avec des bords arrondis (Color dans notre exemple). Si une classe est définie par ses instances (comme 'Model'), cela est indiqué par le label <<EnumeratedClass>>.

Pour illustrer ensuite les nouvelles notations utilisées afin de compléter les diagrammes UML habituels, l'article propose le schéma suivant :

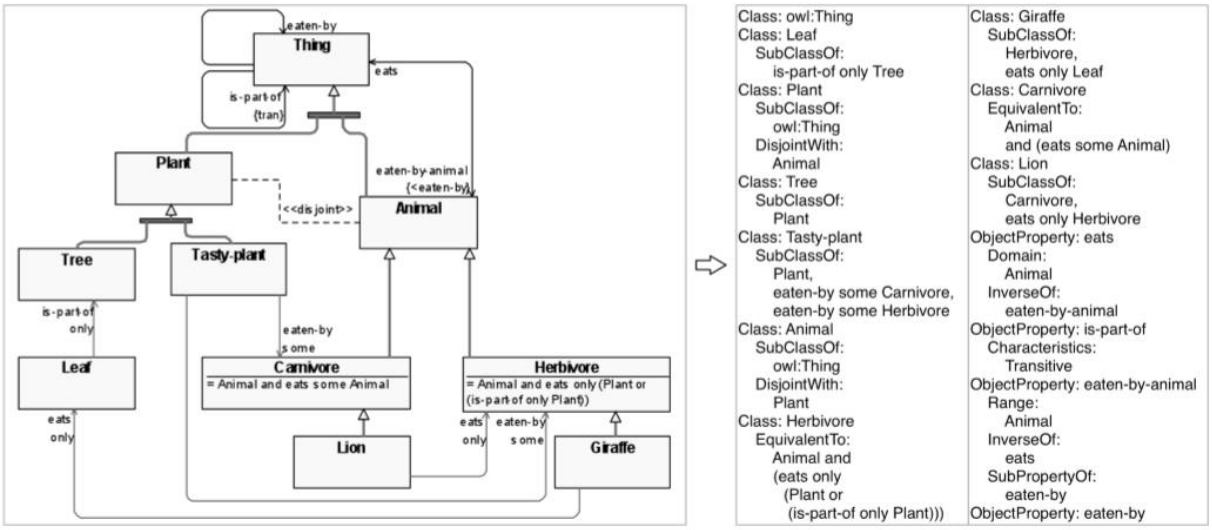


Figure 43 Illustration des notations utilisées pour compléter les diagrammes UML usuels.

Les lignes rouges symbolisent les restrictions 'some values from' et 'only values from' encodées comme sous-classes dans OWL. Les super-propriétés sont dépeintes comme un texte près du nom de la sous-propriété (exemple : <eaten-by près de la case Animal, sous eaten-by-animal). Le symbole '<' correspond à 'sous-propriété' dans la notation UML.

Concernant les ontologies et leur largeur devenant souvent importante, le logiciel propose un layout automatique qui s'adapte au format . Les ontologies devenat vite imposantes, il est souvent utile d'avoir un accès aisé etrapide à un élément. Le logiciel est doté d'une fonction de recherche permettant de retrouver, d'après un pattern de recherche, une classe, un label, une propriété.

c- ViziQuer 3

Après ces pré-requis nous pouvons maintenant aborder le fonctionnement de ViziQuer 3.

A la différence des premières versions de ViziQuer, ViziQuer 3 permet les interrogation avec les opérateurs d'agrégation disponibles sous SPARQL à savoir : COUNT, SUM, MIN/MAX,AVG,SAMPLE,GROUP_CONCAT,GROUP BY,HAVING, FILTER.

ViziQuer se base donc sur le visuel et les diagrammes basés sur OWLGred vu précédemment. L'article utilise la mini-ontologie suivante afin d'expliquer le fonctionnement de construction d'une requête. (voir figure ci-dessous)

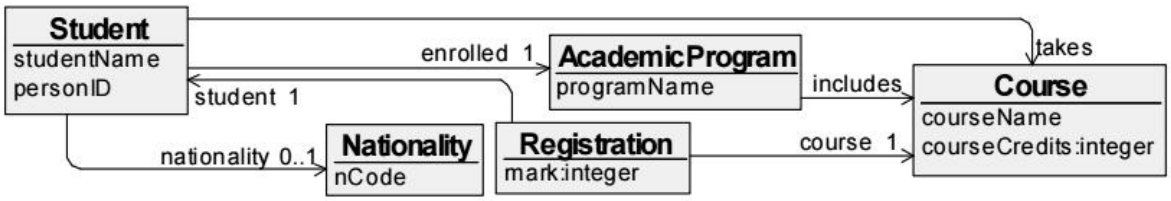


Figure 44 Viziquer mini-ontologie exemple

« Une requête dans ViziQuer est un graphe de nœuds d'instances de classes connectées avec des liens ,correspondants aux triplets connectant ces instances. Chaque nœud montre le nom d'instance de classe (par exemple Registration, Student dans la *figure 49* ci-dessous), éventuellement une référence explicite de nom d'instance (de variable) (par exemple ici R ou S), des conditions (comme mark>=4) et des sélections d'attributs (exemple : R et mark dans le rectangle Registration). »

L'un des rectangles de classe est indiqué, par sa couleur orange) comme la classe de base de la requête tandis que les autres classes (appelées classes de condition) sont indiquées en couleur violette. « La sémantique d'une requête de base est de trouver tous les graphes d'instances correspondants au pattern défini par la requête et de lister les instances sélectionnées et leurs attributs pour chaque instance de graphe.

Les instructions LIMIT, ORDER BY peuvent être indiqués dans les rectangles de requête principale.
Il y a différents symboles de liens :

- En noir, ligne solide : affirmatifs
- En bleu, ligne discontinue : optionnel
- En rouge, ligne continue avec l'étiquette {not} : négation

L'interprétation par défaut du lien optionnel ou de négation est de marquer l'intégralité du sous-graphe placé derrière le lien (du point de vue de la classe principale de la requête) comme optionnel ou nié (respectivement).
Un lien de négation avec une étiquette de {condition} est interprêté comme la non existence (FILTER NON EXISTS) des triplets engendrés entre le nœud et son instance image.

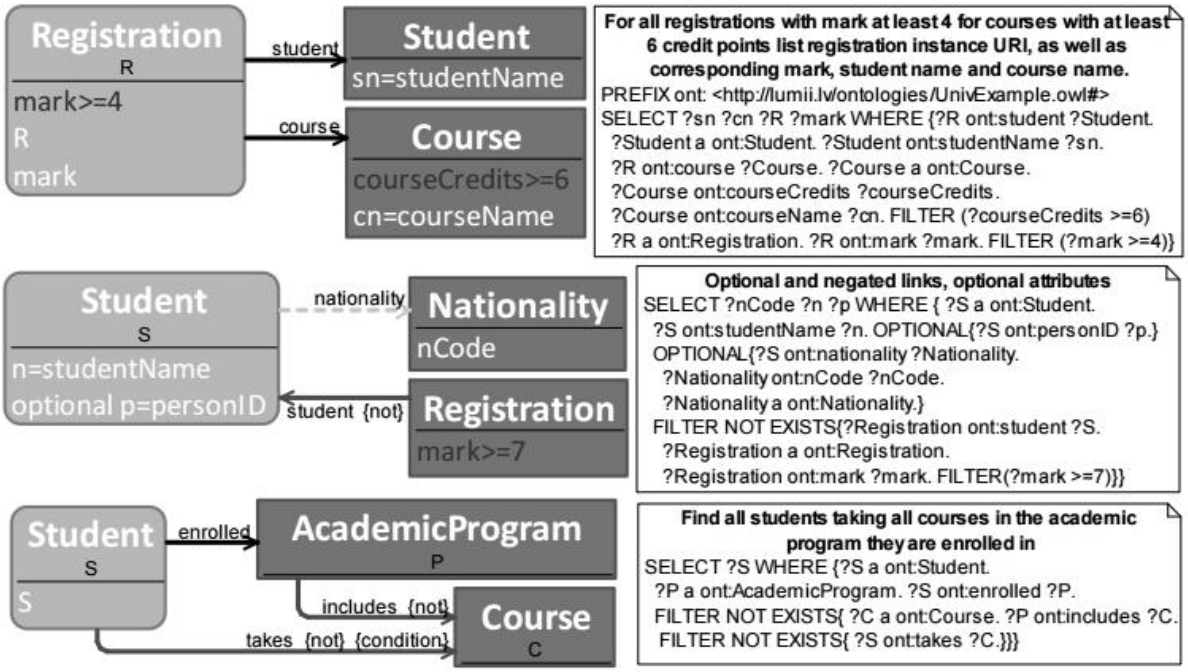


Figure 45 Exemples de représentation de requêtes Viziquer

Manipulation des agrégations dans les requêtes :

« Les options d'agrégation peuvent être inclues dans les requêtes juste en introduisant dans l'instance de classe des expressions d'agrégation dans la liste des attributs où une fonction d'agrégation SPARQL (count, sum, avg) est appliquée à une expression d'attributs non agrégée (par exemple sum(courseCredits)) » dans la figure *fig-viz2* ci dessous.

L'idée est de traiter les valeurs agrégées en prenant comme ensemble de regroupement tous les attributs non agrégés spécifiés dans la requête. Si tous les attributs agrégés sont placés dans la même seule classe dans la requête, celle-ci sera appelée 'classe d'agrégation'. Dans le cas d'attributs agrégés placés dans des classes placées dans des classes différentes, des sous-requêtes séparées sont à réaliser pour chaque classe agrégée avec leur résultats fusionnés.

L'article explique ensuite que SPARQL suit trois étapes pour traiter une requête avec opérateurs d'agrégation :

- « La requête brute avec les arguments de la fonction d'agrégation (attributs simples) au lieu des attributs d'agrégats est générée.
- La liste distincte pour l'opération d'agrégation opérée sur la requête brute est formée, consistant en tous les attributs (aussi bien non agrégés que les agrégés) et les instances de la classes principale de la requête et de la classe de regroupement. L'ensemble peut être étendu en assignant les clauses <<all>> à une classe dans la requête et une classe pouvant être exclue de l'ensemble avec la clause <<exists>>
- L'agrégation sur la liste distincte sélectionnée de la requête brute est opérée en agrégeant les attributs à agréger et en groupant tous les attributs non soumis à agrégation. »

La figure *fig-viz2* présente deux variantes de la requête suivante : « trouver toutes les nationalités et la somme des credits points des cours obtenue par les étudiants de cette nationalité ».

La première requête compte chaque cours une fois par nationalité, tandis que la seconde une fois par nationalité et étudiant, puisque la classe Student est dans l'ensemble de classe multiplicative pour la requête et qu'en conséquent une variable ?S apparaît dans la liste distincte de la requête (menant hypothétiquement à compter les credit points our un simple cours plusieurs fois par nationalité).

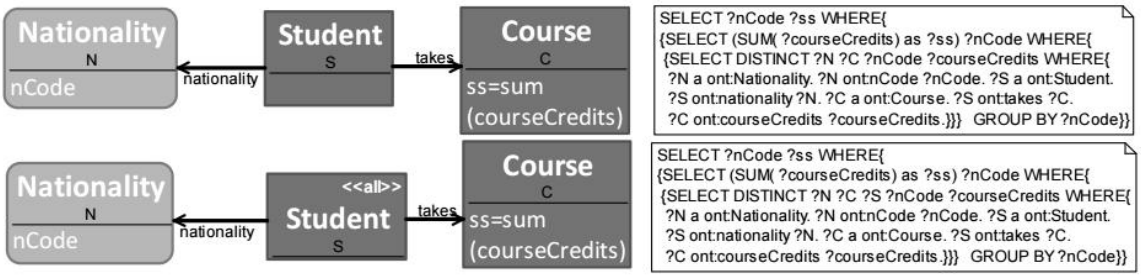


Figure 46 Deux variantes de la requête: « trouver toutes les nationalités et la somme des credits points des cours obtenue par les étudiants de cette nationalité »

Analyse :

En analyse finale de Viziquer 3, nous remarquons que son *point fort* est sa proposition de calcul d'agrégats dans les requêtes SPARQL assez innovante d'après les autres solutions existantes.
Son *point faible* ?
Il nécessite d'avoir une vision de l'ontologie, une connaissance de celle-ci avant de créer sa requête . Le but est pourtant de simplifier la création de requête et de mettre cet outil à portée de non initiés.
En effet, en partant d'une classe, il n'est pas possible d'avoir accès aux propriétés d'objets (juste les propriétés de type de données à travers le système d'attributs sont disponibles) dont elle constitue le domaine.
De plus, une fois deux classes posées sur le canevas, si celles-ci ne sont pas liées par une propriété d'objets, aucune propriété n'est proposée. Il aurait été intéressant de créer un algorithme permettant de voir si un chemin (voie possible entre deux nœuds) de dimension prédéfinie N peut lier les deux classes ou non, et dans le cas positif, de proposer celui-ci.

Partie 3

Proposition d'une technique : Aide à la création de requêtes SPARQL à l'aide d'un browser contextuel d'IRI

L'état de l'art précédent nous a permis d'étudier quelques techniques de facilitation de création de requêtes ainsi que leurs principales faiblesses.

Pour ce qui est des techniques :

- *Browser graphique de l'ontologie*
- *Analyse sémantique de questions langage naturel pour former une requête automatiquement.*
- *Notations graphiques de type UML améliorées.*
- *Visualisation globale de l'ontologie.*
- *Mapping d'éléments graphiques vers SPARQL.*
- *Relaxation de requêtes en cas d'ensemble résultats nuls ,insuffisants ou insatisfaisants.*

Les faiblesses :

- *Rigidité de la formation de la requête graphique.*
- *Visualisation globale des ontologies plus importantes confuse.*
- *Analyse sémantique automatique dont la pertinence de résultats n'est encore qu'assez faible.*

L'idée de base vient du constat suivant : lors de la consultation d'une ontologie la difficulté principale est l'acquisition des IRI (identificateurs de ressources) afin de réaliser la requête (dont la structure est proche de ce que l'on voit en SQL).

Autre idée est basée sur le fait que la pauvreté actuelle dans un consensus pour un dictionnaire commun permettant à des requêtes en langages naturels d'apporter un taux de satisfaction intéressant nous amène à abandonner cette solution et à décider de laisser à l'utilisateur le choix des concepts que nous lui proposerons dans une liste de pertinences construite à partir de mots clefs.

Enfin, la visualisation qui elle, sera amenée dans des tableaux dont le contenu sera orienté également par les mots clefs de l'utilisateurs. (fractionnement de l'information générale).

A partir de ces idées, nous nous proposons d'établir un logiciel qui procédera à la construction de requêtes complexes en plusieurs étapes :

- *Etape de recherche des IRI* ***(Internationalized Resource Identifier)*** *dans l'ontologie par mots clefs*
- *A partir de ces IRI, construction des requêtes simples (SELECT et Jointures)*
- *A partir de ces requêtes simples :*
    - *Création des requêtes complexes d'union*
    - *Création des requêtes d'agrégation*

A- Outils utilisés : Java/Jena.

JAVA est particulièrement adapté à notre situation de par la richesse de ses bibliothèques et ses outils d'implémentation solides et standards. Nous développerons avec les librairies pour SPARQL (JENA). La notion d'ontologie étant indissociable de celle de 'partage', JAVA (notamment avec J2EE, JSP et ses servlets) offrent un cadre idéal de développement futur (non développé ici) de pages web pouvant utiliser les moteurs créés pour requêter les ontologies (voire même aussi pour les éditer ou les créer).

1- Présentation générale de Apache Jena

Dans la construction de notre outil d'aide à la requête SPARQL nous utiliserons le framework JENA. [web -35]
Jena est un set d'outils originellement développé par HP Labs (Bristol, UK en 2000) permettant la construction, la lecture et la modification de base de connaissances sémantiques. Il fournit des librairies JAVA permettant au développeur de concevoir des applications utilisant les standards RDF,RDFS, OWL et leur interrogation avec SPARQL.
JENA possède également un moteur d'inférence permettant d'utiliser le raisonnement logique sur les ontologies OWL et RDFS ainsi que des stratégies de stockage pour sauvegarder les triplets RDF sur le disque.

2- Aspects techniques utiles à notre projet

Les aspects principaux que nous utiliserons de JENA seront liés à une utilisation d'interrogation de la base. Il est toutefois utile de souligner que JENA permet aussi de :

- Créer une ontologie
- Créer une classe
- Créer une classe d'union/intersection OWL
- Créer les propriétés de données et des propriétés d'objet
- Créer des relations hiérarchiques entre des classes/

Nous nous intéresserons pour notre part à l'aspect consultatif de JENA.

a. Charger une ontologie

Deux cas de figure se présentent :

- *Chargement d'une ontologie locale dans un modèle :*

Cela nécessite l'utilisation d'un objet OntModel propre à l'API de JENA.
Cela se fait comme suit : Création du modèle d'ontologie voulue,

OntModel mode ;

mode=ModelFactory.*createOntologyModel*(OntModelSpec.***OWL_MEM***);

Cette instruction (factory instituée depuis la version 2.0 de JENA) crée un modèle d'ontologie avec la configuration par défaut qui est définie pour une compatibilité maximale avec la version précédente de JENA.

OWL_MEM permet de manier une ontologie OWL Full et sans moteur d'inférence.

D'autres options permettant de préciser le modèle créé sont pré-définies, lesquelles sont :

OntModelSpec	Language profile	Storage model	Reasoner

OntModelSpec	Language profile	Storage model	Reasoner
OWL_MEM	OWL full	in-memory	none
OWL_MEM_TRANS_INF	OWL full	in-memory	transitive class-hierarchy inference
OWL_MEM_RULE_INF	OWL full	in-memory	rule-based reasoner with OWL rules
OWL_MEM_MICRO_RULE_INF	OWL full	in-memory	optimised rule-based reasoner with OWL rules
OWL_MEM_MINI_RULE_INF	OWL full	in-memory	rule-based reasoner with subset of OWL rules
OWL_DL_MEM	OWL DL	in-memory	none
OWL_DL_MEM_RDFS_INF	OWL DL	in-memory	rule reasoner with RDFS-level entailment-rules
OWL_DL_MEM_TRANS_INF	OWL DL	in-memory	transitive class-hierarchy inference
OWL_DL_MEM_RULE_INF	OWL DL	in-memory	rule-based reasoner with OWL rules
OWL_LITE_MEM	OWL Lite	in-memory	none
OWL_LITE_MEM_TRANS_INF	OWL Lite	in-memory	transitive class-hierarchy inference
OWL_LITE_MEM_RDFS_INF	OWL Lite	in-memory	rule reasoner with RDFS-level entailment-rules
OWL_LITE_MEM_RULES_INF	OWL Lite	in-memory	rule-based reasoner with OWL rules
RDFS_MEM	RDFS	in-memory	none
RDFS_MEM_TRANS_INF	RDFS	in-memory	transitive class-hierarchy inference

OntModelSpec	Language profile	Storage model	Reasoner
RDFS_MEM_RDFS_INF	RDFS	in-memory	rule reasoner with RDFS-level entailment-rules

Tableau 10 Options de création de la factory model de Jena

Ce modèle une fois défini, on crée un flux de lecture dans le fichier local contenant l'ontologie étudiée :

```
FileInputStream in = null;
try
{
  in=new FileInputStream(arg);
}
catch (FileNotFoundException ex)
{
  Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
```

Le chargement de l'ontologie dans le modèle s'effectue ensuite en utilisant la méthode 'read'.
Différents constructeurs existent :

```
read( String url )
read( Reader reader, String base )
read( InputStream reader, String base )
read( String url, String lang )
read( Reader reader, String base, String Lang )
read( InputStream reader, String base, String Lang )
```

Nous utiliserons ici :

mode.read(in,**null**, "RDF/XML-ABBREV");

RDF/XML-ABBREV permet une partie lisible quoique moins efficace que RDF/XML.
Cette variante de read permet une lecture sur un flux d'entrée d'un document en utilisant l'URI de base passée (dans notre cas, la base est nulle, signifiant qu'il n'y a pas conversion des URI relatives).

L'ontologie locale est alors chargée.

Reste ensuite à créer des requêtes à l'aide des classes Query et QueryExecution

```
private QueryExecution qexec;
private Query query;

Query query = QueryFactory.create(queryst);
qexec = QueryExecutionFactory.create(query,mode);
```

On utilise le parser pour transformer une chaîne String (ici queryst) en forme exécutable. Une fois cette forme créée, on peut la passer à la QueryFactory pour produire un moteur d'exécution de requête sur le modèle ontologique précédemment créé.

- *Chargement d'une ontologie distante (endpoint)*

```
this.query = QueryFactory.create(queryst);
qexec = QueryExecutionFactory.sparqlService(arg, query);
```

De manière similaire, on transforme la forme String en une forme exécutable.
Puis, à l'aide de la forme de méthode :

static **QueryExecution**	**sparqlService(String** service, **Query** query) Create a QueryExecution that will access a SPARQL service over HTTP

On crée un moteur d'exécution connecté sur le point d'accès distant (dans notre cas 'arg')
(http://dbpedia.org/sparql par exemple).

b. Installation du moteur d'inférence

Afin de pouvoir appliquer un raisonnement sur l'ontologie chargée (par exemple transitivité) il nous faut définir un moteur d'inférence. [web -36]
Celui-ci est implanté dans la classe connection_ontology en important les librairies :

```
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ReasonerRegistry;
```

et le moteur d'inférence implémenté de la manière suivante :

```
mode = ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM );
//implantation du reasonner
InfModel inf = ModelFactory.createInfModel(reasoner, mode);
```

Le queryexecutionfactory est ensuite basé sur ce moteur d'inférence avec la ligne :

```
qexec = QueryExecutionFactory.create(query,inf);
```

Pour résumer l'opération, il s'agit de la pose d'un filtre d'inférence sur les données collectées.

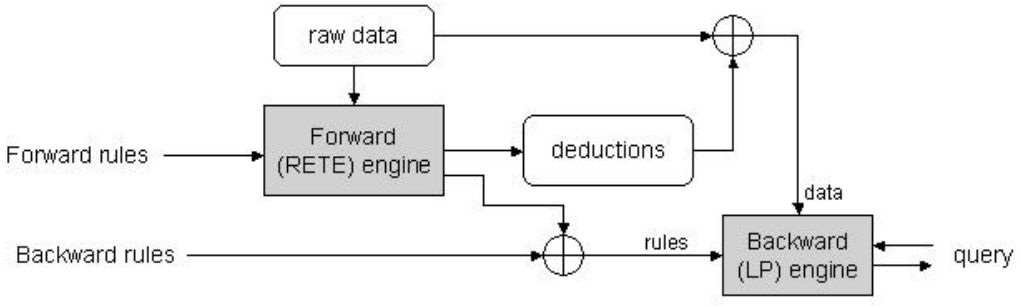


Figure 47 Architecture de la pose d'un filtre d'inférence

c. Exploitation des résultats de requête

A partir de notre objet de type QuerySolution qexec, nous récupérons un Resultset en appliquant la commande.execSelect().

```
try {
```

```
                ResultSet results = qexec.execSelect();
                System.out.println(results.getRowNumber());
                ret=results;


        }
```

Le Resultset présente les résultats émanant d'une requête sous forme de table dont les colonnes sont les variables contenues dans la clause SELECT. Chaque rangée correspond à un ensemble de liens issus de l'ontologie interrogée qui correspondent aux conditions de la requête. L'accès aux résultats se fait alors par lenom de chaque variable.

L'affichage graphique se sert d'un modèle de table qui prend pour paramètre d'une part les données du résultset, d'autre part, le nom des variables isssues de la clause SELECT qui correspondront aux colonnes du tableau.

```
DefaultTableModel tableModel = new DefaultTableModel(getData(res_set), getColumns(res_set));
```

Les colonnes sont récupérées dans une liste de type 'Vector' par la commande .getResultVars appliquée à notre objet de type Resultset.

```
private Vector getColumns(ResultSet results)
{
   Vector cols = new Vector();
   cols.addAll(results.getResultVars());
   return cols;
}
```

Les données sont ensuite récupérées par deux boucles while imbriquées, l'une récupérant les lignes de solutions (QuerySolution), l'autre découpant ces lignes (objet Querysolution issu d'une application de la fonction .nextSolution de l'objet de type Resultset) et récupérant les données indexées par le nom de la variable correspondante .

```
        private  Vector getData(ResultSet data)
{
   Vector results = new Vector();
   for (; data.hasNext(); )
   {
      QuerySolution qs = data.nextSolution();
      results.add( getRowData(data, qs) );
   }
   return results;
}

private Vector getRowData(ResultSet results, QuerySolution qs)
{
   Vector row = new Vector();
   for (Iterator iter = results.getResultVars().iterator() ; iter.hasNext() ; )
   {
      String var = (String)iter.next();
      row.add( getValueAsString(qs, var) );
   }
   return row;
}
```

La fonction getValueAsString prenant en paramètre la ligne de solution et la variable déterminant la colonne à traiter permet alors d'extraire desobjets de type RDFNode afin de les convertir en chaînes de caractère.

Cette étape permet aussi d'effectuer divers traitements sur les nœuds (ici, en l'occurrence, une détermination du type de nœud, littéral ou non).

3- Utilisation de l'ontologie Dbpedia

Le but de l'outil sera une consultation générique d'ontologies et non pas d'une seule ontologie. Nous choisissons toutefois de baser l'étude de notre logiciel sur l'ontologie Dbpedia. Dbpedia est [web -37] issu de l'effort participatif d'une communauté visant à extraire des informations structurées de wikipedia et de les rendre disponibles sur le web (pour réutilisation par d'autres lecteurs logiciels) en permettant l'utilisation de requêtes SPARQL complexes . DBpedia est une base de données sémantique de taille très importante comportant 4,58 millions d'objets. Nous partons donc de l'évidence que si nous pouvons faciliter une navigation, une requête sur une ontologie des plus importantes, il nous sera ensuite aisé d'explorer des structures sémantiques de moindre importance.

B- Implémentation d'un panneau de recherche d'IRI

L'idée de base est de faciliter la recherche d'éléments (Entités, classes, propriétés) . Utiliser ces éléments précédemment trouvés pour affiner la recherche. Interface Graphique, mode texte, langage naturel + choix de l'utilisateur.La difficulté essentielle étant pour un utilisateur, non pas de créeer une requête ou de la structurer mais de retrouver les IRI traduisant sa volonté de requête. Le but est que l'utilisateur puisse choisir au mieux, en toute connaissance de cause, les Entités, classes et propriétés qui correspondent le mieux à sa demande.

Outils : Librairie JENA pour JAVA, WindowsBuilder sous Eclipse (pour la création rapide des formulaires).

Nous baserons encore notre étude sur l'ontologie dbpedia du fait de sa taille, de sa complexité et de la profusion de classes et de propriétés qui y existent.

Etude de cas de figures :

Ces recherches sont bien sûr améliorables (notamment par des recherches collatérales sur les labels par exemple).

Présentation de l'interface :

Chaque recherche sera représentée dans une liste dont chacun des résultat pourra être stocké dans un tableau réutilisable aisément par l'utilisateur (notammentdans les listes combo de l'interface). Le principe n'est pas des plus complexe mais il a l'avantage certain d'aider l'utilisateur à trouver les entités, classes et relation qui pourront l'aider à constituer sa requête. Ce système consiste à trouver et sélectionner les IRI qui correspondent à notre demande, de les stocker pour réutilisation. A cette fin, trois listes sont disponibles dans l'ensemble du programme :

- la liste des URI entités stockées
- la liste des URI classes stockées
- la liste des URI propriétés stockées

Ces IRI une fois stockées, on peut ensuite passer à la construction de requêtes simples ou à des jonctions de requêtes simples. Ces requêtes une fois construites, permettent à leur tour et à partir d'elles même de construire des requêtes d'union ou des requêtes de groupement (group by, having…). C'est donc un procédé en trois niveaux qui laisse toujours à l'utilisateur sa liberté d'intervention et de modification manuelle des requêtes créées.

figure 52 : Panneau de recherche

Petit rappel sur les triplets . Il sont de la forme <sujet><propriété><objet>.
Le sujet pouvant être une IRI de classe, une IRI d'instance de classe, un nœud blanc.
La propriété étant une IRI de propriété d'objet ou de classe.

L'objet pouvant être une IRI de classe, une IRI d'instance, un nœud blanc ou un littéral (String, integer, date…)

C- Fonctionnement global et menu général

Le menu général reprend les rubriques de bases utiles à l'utilisation de l'ontologie.
Le système se base sur des listes globales accessibles par tous les panneaux (Listes des requêtes simples, listes des entités, des classes, des propriétés stockées par l'utilisateur).
L'utilisateur se voit faciliter la tâche et peut découvrir les URI et les inter dépendances de l'ontologie de manière instinctive et interactive.

L'interface est composée de 6 fenêtres principales :

- Choix de l'ontologie.
- Le menu général
- Une console, permettant de voir les messages du programme (redirection des erreurs, des messages de connection etc…)
- Fenêtre de recherche d'entités, de classes et de de propriétés par filtrage simple ou l'une par rapport à l'autre (cas précédents étudiés)
- Fenêtre de création de requêtes et de jointure de requêtes en se servant des URI trouvées dans la fenêtre de recherche
- Fenêtre d'union de requêtes utilisant des requêtes créées et stockées dans le panel de création/jointure
- Fenêtre de construction de requêtes utilisant des fonctions de groupage (sum, count, etc…) (basée sur des requêtes précédemment créées)

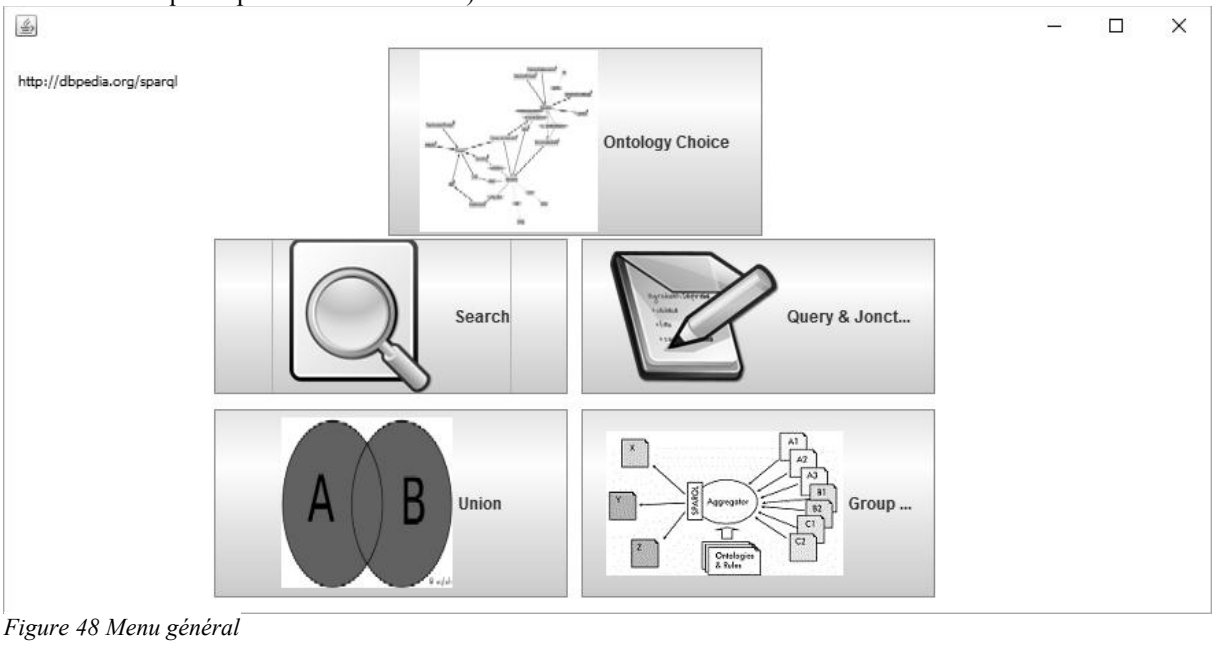


Figure 48 Menu général

1- Etape préliminaire : choix de l'ontologie (locale ou distante)

Elle est effectuée via le formulaire suivant :

Figure 49 Panneau de choix et chargement d'ontologie

Radio bouton pour choisir l'ontologie locale ou distante.
ComboBox Adresse pour choisir l'adresse du fichier local ou du point d'accès distant . Celui-ci contient par défaut quelques adresses de base mais peut aussi être directement modifié pour saisir l'adresse .
Le bouton « ok » assure la connection et ouvre ensuite les fenêtres servant à la recherche et création de requêtes.

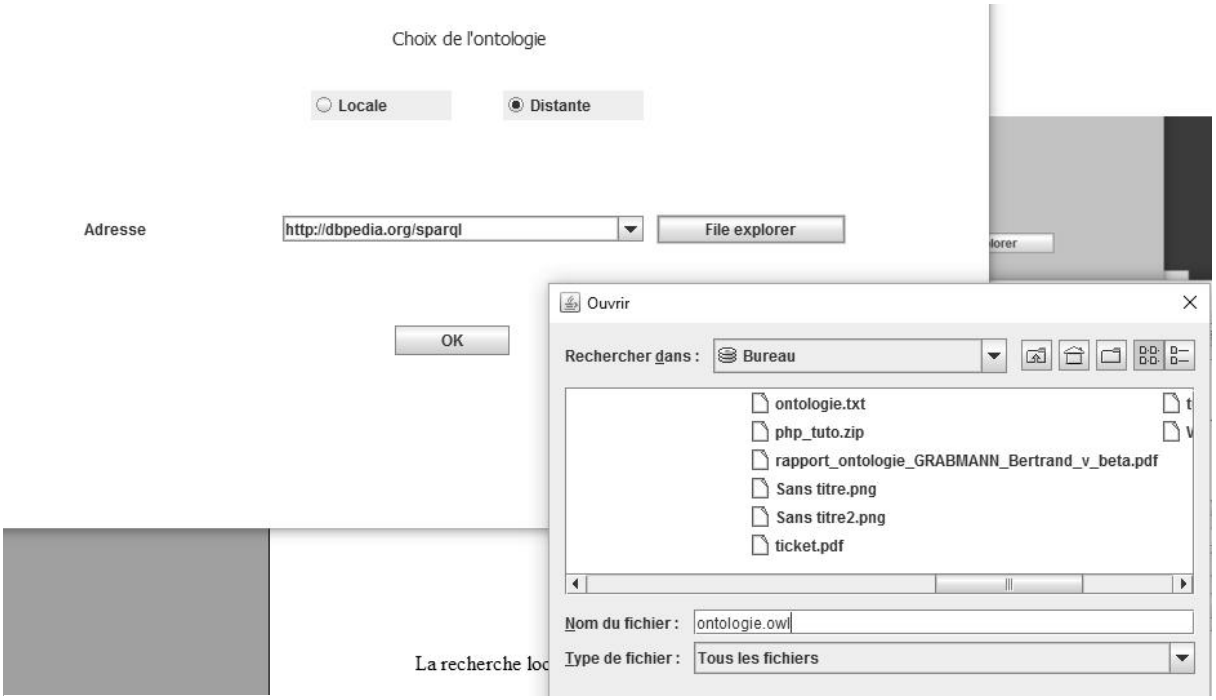


Figure 50 Choix d'une ontologie locale

La console :

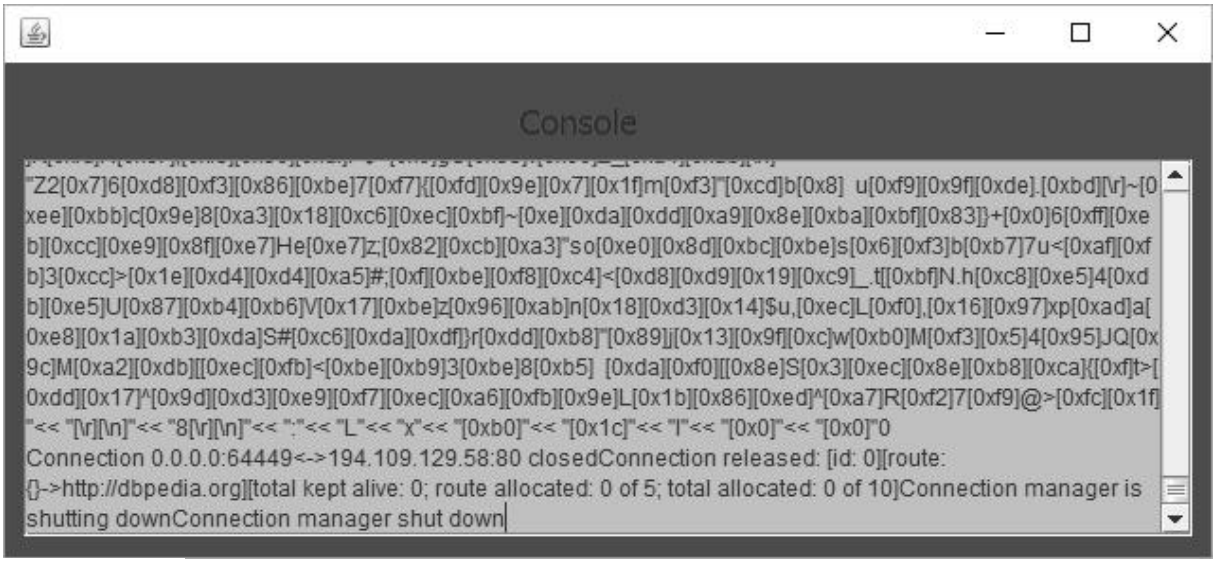


Figure 51 Console

2- Recherche des IRI utiles à la construction des requêtes.

La recherche locale se fait via un formulaire de recherche de fichier :

Figure 52 Bouton de recherche d'IRI

Le panneau de recherche se base essentiellement su la notion des triplets RDF.

Petit rappel sur ces triplets . Ils sont de la forme <sujet><propriété><objet>.
Le sujet pouvant être une IRI de classe, une IRI d'instance de classe, un nœud blanc.
La propriété étant une IRI de propriété d'objet ou de classe.
L'objet pouvant être une IRI de classe, une IRI d'instance, un nœud blanc ou un littéral (String, integer, date…)

Le **panneau de recherche** se présente comme suit :

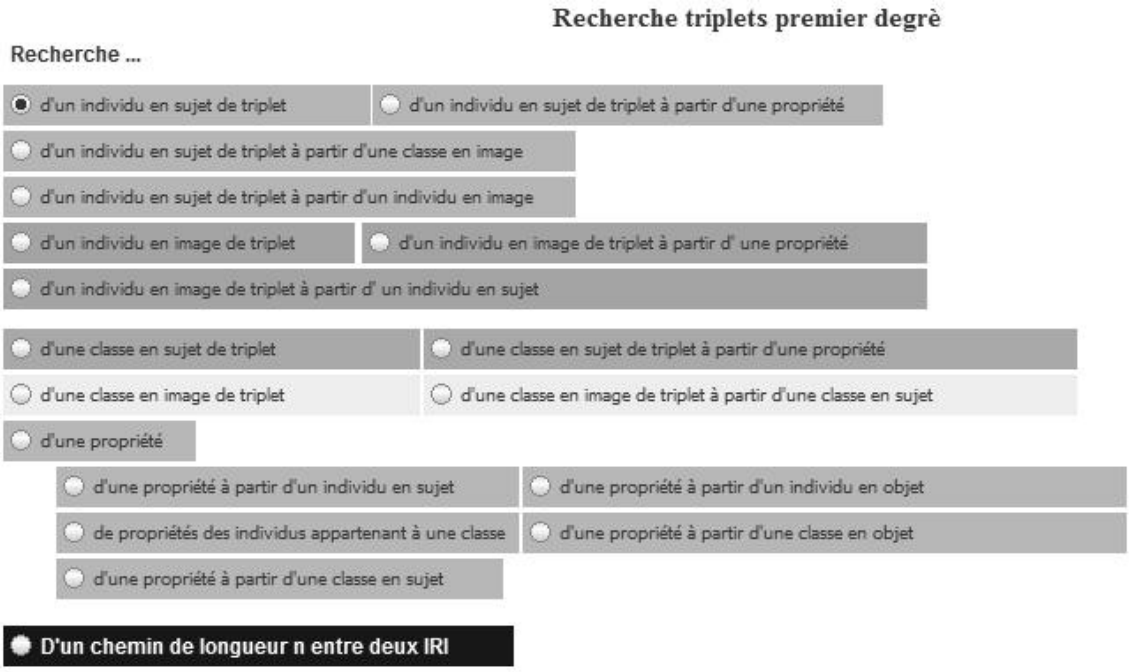


Figure 53 Panneau de recherche d'IRI

Il a pour but de faciliter la recherche des IRI (d'individu, de classe, de propriété) dans l'ontologie étudiée. Ces IRI une fois déterminées pourront être stockée et enregistrée dans les variables globales du programme.

Ces adresses pourront ensuite être réutilisées afin de constituer les requêtes complexes (notamment d'union, d'intersection).

Ce panneau reprend la plupart des cas de figure étudié plus haut. En fonction du choix de l'utilisateur, celui ci sera orienté vers un panneau lui demandant les informations utiles et adaptées à sa recherche.

Les différents types de recherche facilitant lanavigation de l'utilisateur pour retrouver ses I.R.I. sont :

Recherche :

1) D'un individu en sujet de triplet : ?suj tel que ?suj est un individu dans le triplet ?suj ?prop ?img . Une fenêtre s'ouvre alors, offrant un champ permettant de filtrer la variable ?suj.

Figure 54 Sous-panneau de recherche d'entité

.

2) D'un individu ?suj en sujet de triplet à partir d'une propriété : ?suj <IRIprop> ?img
La fenêtre qui s'ouvre permet la saisie d'un filtre sur ?suj et le choixde l'IRI à fixer dans la requête.

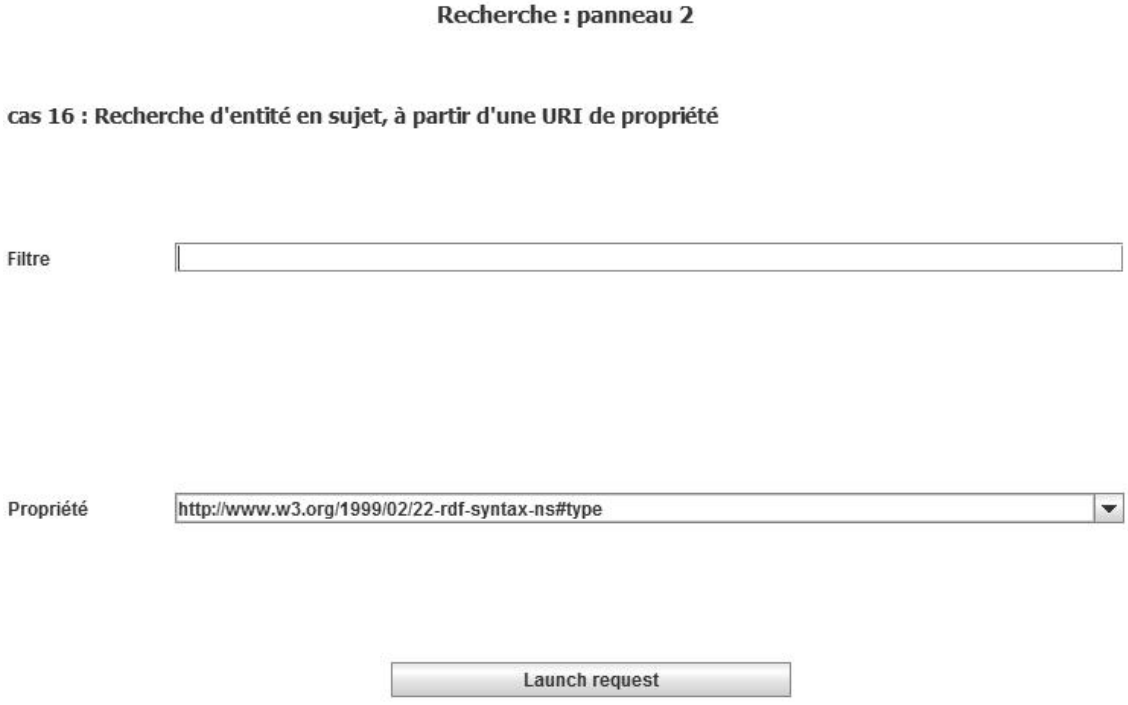


Figure 55 Sous-panneau de recherche d'entité à partir d'une URI de propriété

3) recherche d'un individu ?suj en sujet de triplet à partir d'une classe en image : ?suj ?prop <IRIclasse>

La fenêtre de recherche suivante s'ouvre :

Figure 56 Sous-panneau de recherche d'individu à partir d'une URI de classe

4) Recherche d'un individu ?suj en sujet de triplet à partir d'un individu en image : ?suj ?prop <IRIindiv>

Qui lance le panneau suivant offrant la possibilité de filter ?suj et de fixer le choix de <IRIindiv> dans une liste prenant ses IRI sources dans des IRI précédemment enregistrées ou entrées manuellement par l'utilisateur.

Figure 57 Sous-panneau de recherche d'entité à partir d'une URI d'entité

5) Recherche d'un individu ?img en image de triplet : ?suj ?prop ?img

Le panneau lancé propose alors de filter ?img

6) Recherche d'un individu ?img en image de triplet à partir d'une propriété fixée : ?suj <IRIprop> ?img

Possibilité de filtrer ?img et choix de l'IRI dans une liste.

7) Recherche d'un individu ?img en image de triplet à partir d'un individu fixé en sujet : <IRIindiv> ?prop ?simg

Possibilité de filtrer ?img et choix de IRIindiv dans une list.

8) Recherche d'une classe ?suj en sujet de triplet : ?suj ?prop ?img

Possibilité de filtrage de ?suj

9) Recherche d'une classe ?suj en sujet de triplet à partir d'une propriété : ?suj <IRIProp> ?img

Possibilité de filtrer ?suj et choix de IRIprop dansune liste (ou saisie manuelle).

10) Recherche d'une classe ?img en image de triplet :

Possibilité de filtrer ?img

11) Recherche d'une classe ?img en image de triplet à partir d'une classe en sujet : <IRIsuj> ?prop ?img

Possibilité de filtre sur ?img, choix de l'IRIsuj dans une liste.

12) Recherche d'une propriété, formulaire avec filtre sur la propriété recherchée.

13) Recherche d'une propriété ?prop à partir d'un individu en objet : ?suj ?prop <IRIindiv>

Filtrage sur ?prop et choix de IRIindiv dans une liste .

14) Recherche des propriétés des individus appartenant à une classe :
Cette propriété se base sur la donnée d'une classe et le requêtage suivant :

```
Select ?prop where {
        ?suj a <IRLclass>.
        ?suj ?prop ?img.
}
```

Mise à disposition : un filtre, une liste contenant des IRI de classes (enregistrées précédemment par l'utilisateur).

15) Recherche d'une propriété ?prop à partir d'une classe en objet : ?suj ?prop <IRIclass>

Panneau généré : Filtre sur ?prop, liste de classes.

16) Recherche d'une propriété ?prop à partir d'une classe en sujet :<IRIclass> ?prop ?img

Panneau généré : Filtre sur ?prop, liste de classes.

17) Recherche d'un chemin de longueur n entre deux individus .
Cette option permet d'établir s'il existe un chemin de longueur n entre deux entités et donc, quelle est la relation entre elles.

Le panneau résultant se présente comme suit :

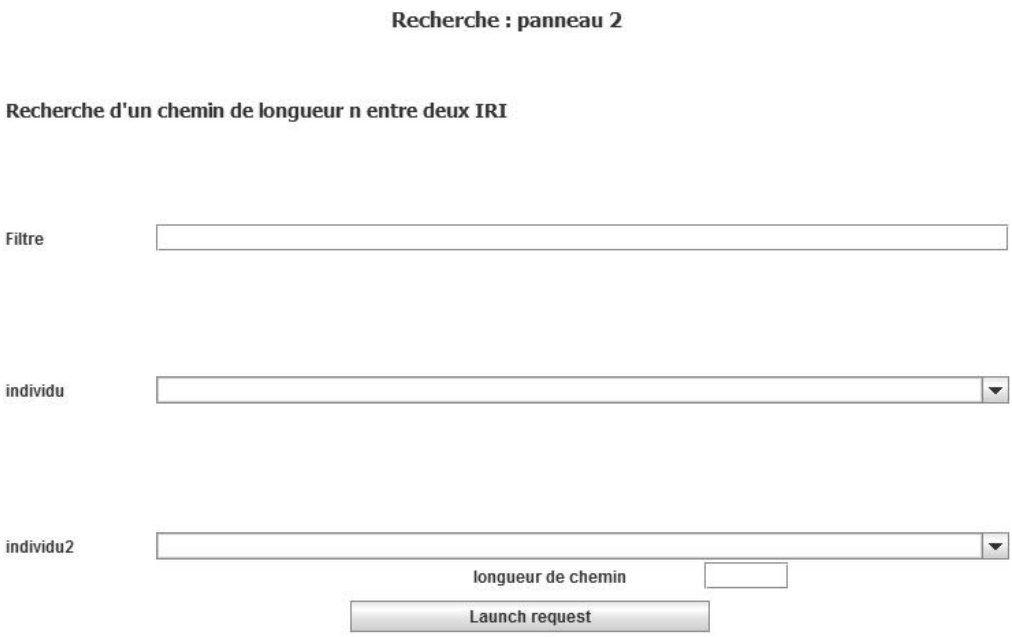
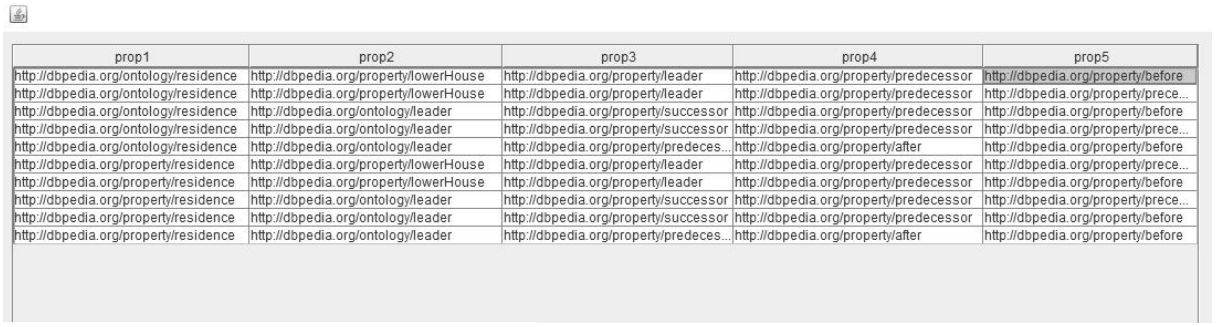


Figure 58 Sous-panneau de recherche d'un chemin de longueur n entre deux individus

La recherche est (pour le moment) simple et sans heuristique, se bornant à explorer tous les triplets possibles dans un périmètre n de l'entité de départ. (intéressant à établir : parcours 'intelligent' à partir d'indices laissés par l'utilisateur ou autres heuristiques de parcours de graphe sémantique).

Voici une petite démonstration simple de cette fonctionnalité.

Soit à rechercher le lien que l'ontologie étudiée (ici dbpedia) entre http://dbpedia.org/resource/Hulk_Hogan et http://dbpedia.org/resource/Lawrence_Hogan avec un rang n=5 (les rangs 1,2,3 et 4 ont été testés et sans réponse). On obtient :



prop1	prop2	prop3	prop4	prop5
http://dbpedia.org/ontology/residence	http://dbpedia.org/property/lowerHouse	http://dbpedia.org/property/leader	http://dbpedia.org/property/predecessor	http://dbpedia.org/property/before
http://dbpedia.org/ontology/residence	http://dbpedia.org/property/lowerHouse	http://dbpedia.org/property/leader	http://dbpedia.org/property/predecessor	http://dbpedia.org/property/prece...
http://dbpedia.org/ontology/residence	http://dbpedia.org/ontology/leader	http://dbpedia.org/property/successor	http://dbpedia.org/property/predecessor	http://dbpedia.org/property/before
http://dbpedia.org/ontology/residence	http://dbpedia.org/ontology/leader	http://dbpedia.org/property/successor	http://dbpedia.org/property/predecessor	http://dbpedia.org/property/prece...
http://dbpedia.org/ontology/residence	http://dbpedia.org/ontology/leader	http://dbpedia.org/property/predeces...	http://dbpedia.org/property/after	http://dbpedia.org/property/before
http://dbpedia.org/property/residence	http://dbpedia.org/property/lowerHouse	http://dbpedia.org/property/leader	http://dbpedia.org/property/predecessor	http://dbpedia.org/property/prece...
http://dbpedia.org/property/residence	http://dbpedia.org/property/lowerHouse	http://dbpedia.org/property/leader	http://dbpedia.org/property/predecessor	http://dbpedia.org/property/before
http://dbpedia.org/property/residence	http://dbpedia.org/ontology/leader	http://dbpedia.org/property/successor	http://dbpedia.org/property/predecessor	http://dbpedia.org/property/prece...
http://dbpedia.org/property/residence	http://dbpedia.org/ontology/leader	http://dbpedia.org/property/successor	http://dbpedia.org/property/predecessor	http://dbpedia.org/property/before
http://dbpedia.org/property/residence	http://dbpedia.org/ontology/leader	http://dbpedia.org/property/predeces...	http://dbpedia.org/property/after	http://dbpedia.org/property/before

Figure 59 Résultats de requête de recherche de chemin

Ces lignes sont plusieurs chemins reliant les deux entités étudiées. (on pourrait aussi rajouter les nœuds (classes ou individus) intermédiaires.

Les chemins nous révèlent que les seuls liens liant les deux hommes (politique et catcheur) ne sont que le fait de résider dans un pays dont le dirigeant leur était commun (on pourrait faire plus simple en disant qu'ils habitent le même pays…ce qui inciterait à apporter plus d'éléments pour rendre cette recherche 'intelligente').

3- Affichage et sélection des éléments construction des requêtes.

Les résultats de requête s'affichent dans un tableau dont les cases sont sélectionnables une à une et enregistrablesdans une des quatre catégories mentionnées au dessous : Classe, individu, propriété ou Littéral (String, int, etc…).

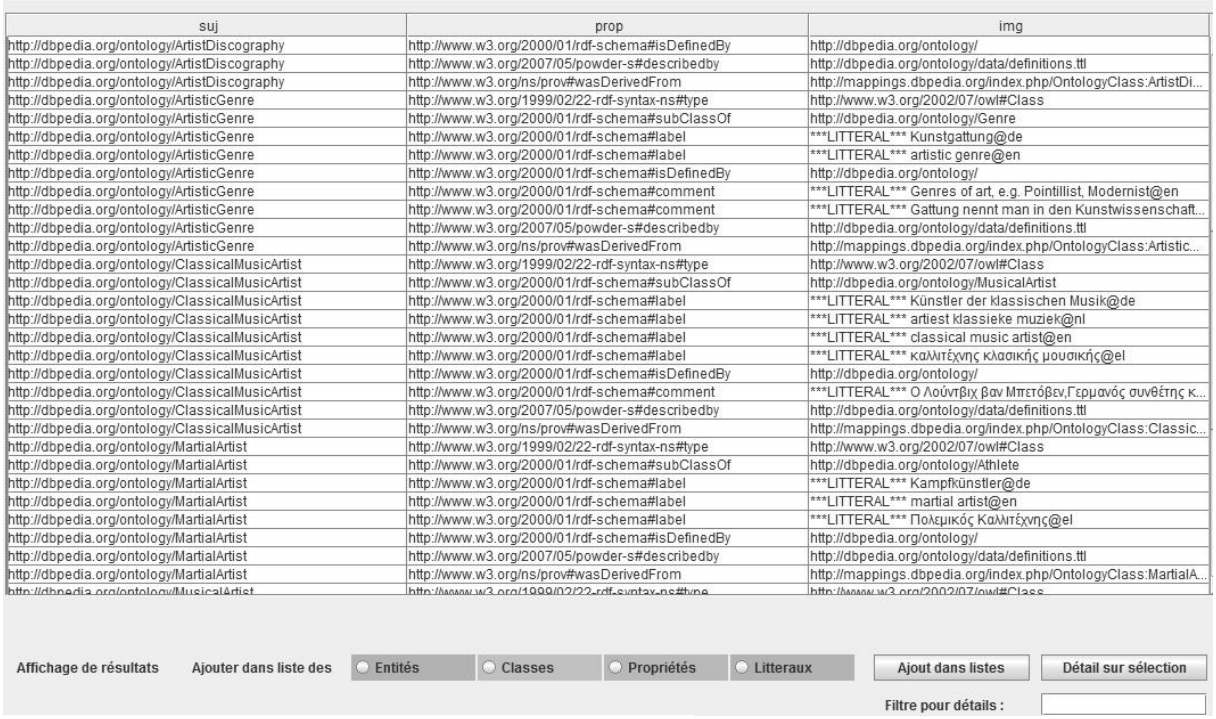


Figure 60 Affichage et sélection des éléments construction des requêtes

Le nombre de colonnes est automatiquement ajusté en fonction des variables demandées dans la clause 'SELECT' de notre requête.
Le bouton 'ajout dans liste', associé à l'un des boutons radio permet l'enregistrement de la valeur contenue dans la case sélectionnée . (Les littéraux sont automatiquement classés dans une liste des littéraux, quelque soit le bouton radio coché).
Le bouton 'détail sur sélection' permet de relancer une requête de type :

```
SELECT  ?prop ?img WHERE {
        <IRI case sélectionnée> ?prop ?img.
}
```

Donc, de pouvoir voir les propriétés et images attachées à l'élément sélectionnés . Par exemple ici :

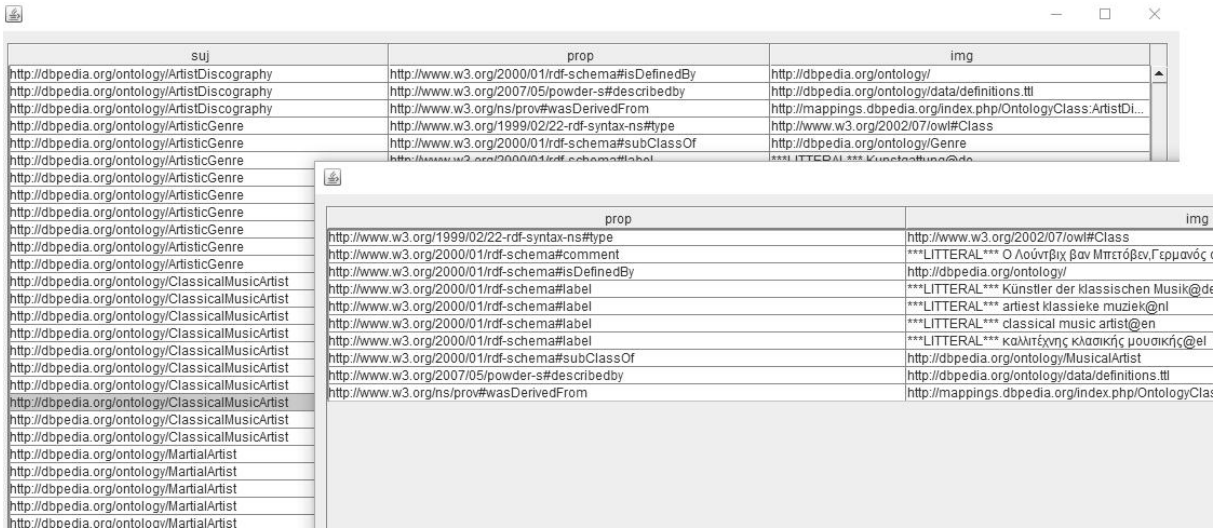


Figure 61 Recherche 'détaillée' sur un élément.

Chaque case dans le nouveau formulaire peut ensuite elle même être à nouveau 'détaillée' (sauf les feuilles terminales, les littéraux par exemple).

4- Exemples d'utilisation du panneau de facilitation de recherche

Exemple 1 : Requête : Trouver le lieu d'habitation de Bruce Willis.

Etape 1 : Trouver l'URI d'entité représentant Bruce Willis (s'il y en a une) :

Menu général>> Panneau Search>>Sélection de recherche : d'un individu (instance) en sujet de triplet >> ok >> avec un filtre finissant par la chaîne « willis » ($, expression des regexp).

Une fenêtre donnant le choix d'entités répondant à ce filtre est proposé, on en choisit une et on la stocke dans la liste des entités. Cette fenêtre propose une table dont les colonnes dépendent des variables selectionnées et dont les cases peuvent être sélectionnées individuellement pour avoir leur contenu stocké soit dans la liste des URI entités, des URI de classes ou des URI de propriétés.

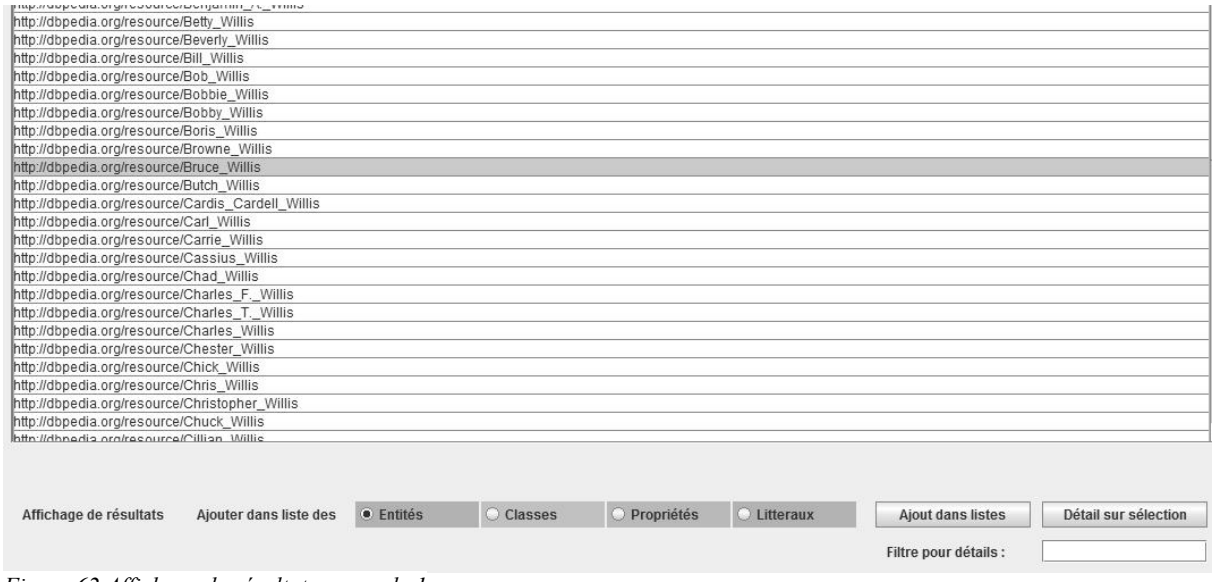


Figure 62 Affichage de résultats :exemple 1

L'entité est stockée et disponible dans la liste des URI d'entités (si l'on coche le bouton radio entité).

Etape 2 : trouver les relations liées et disponibles pour cette entité « bruce willis » et ayant trait à son lieu d'habitation :

3 méthodes peuvent se proposer avec le panneau de recherche :

- Sélection de la case de l'IRI de Bruce Willis >> Bouton 'détails sur sélection' >> et choix/visualisation dans le panneau d'affichage

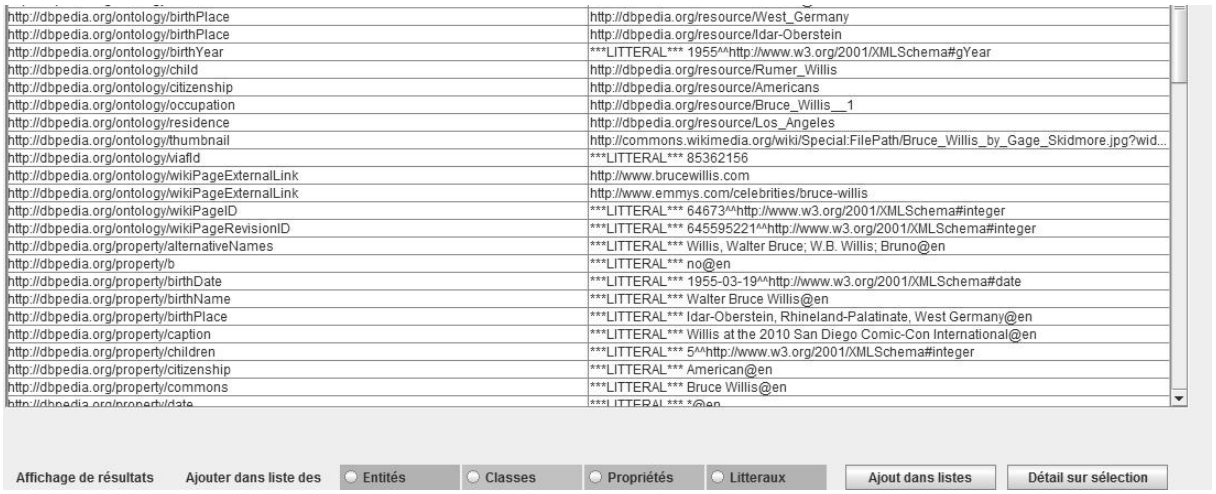


Figure 63 Résultats issus de la sélection de la case de l'IRI de Bruce Willis >> Bouton 'détails sur sélection'

- Deuxième solution, réduisant le nombre de lignes affichées, saisir un mot clef dans le champ 'filtre' (ce qui requiert alors ici peut être une certaine connaissance sémantique de l'ontologie…les méthodes de relaxation vues plus haut pourraient être ici utiles) . Saisisons ici 'reside'. On obtient :

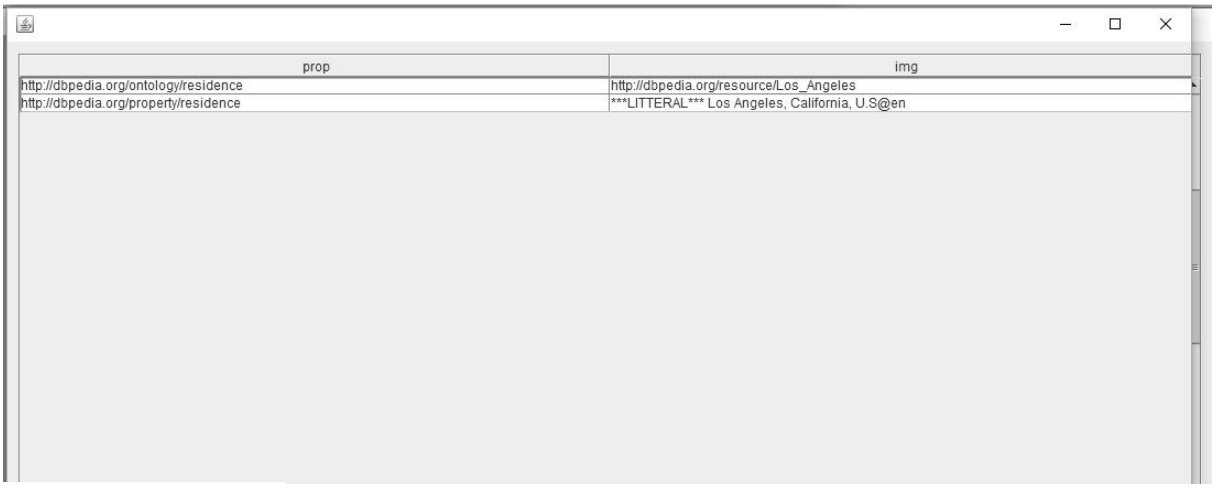


Figure 64 Deuxième solution

Exemple 2 : Requête : Trouver toutes les propriétés des individus qui sont des artistes.

Etape 1 : recherche de l'IRI de la classe traduisant notre concept d'artiste.

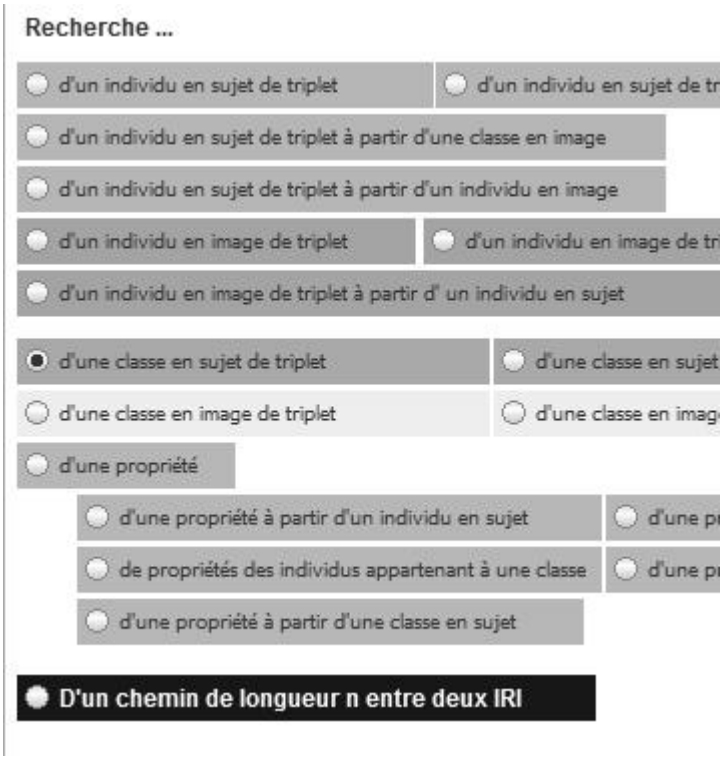


Figure 65 Recherche d'une classe en sujet de triplet

Bouton ok, puis le panneau suivant, entrer 'artist ' dans le filtre .

Figure 66 Sous formulaire issu de la recherche d'une classe en sujet de triplet

Bouton 'launch query'

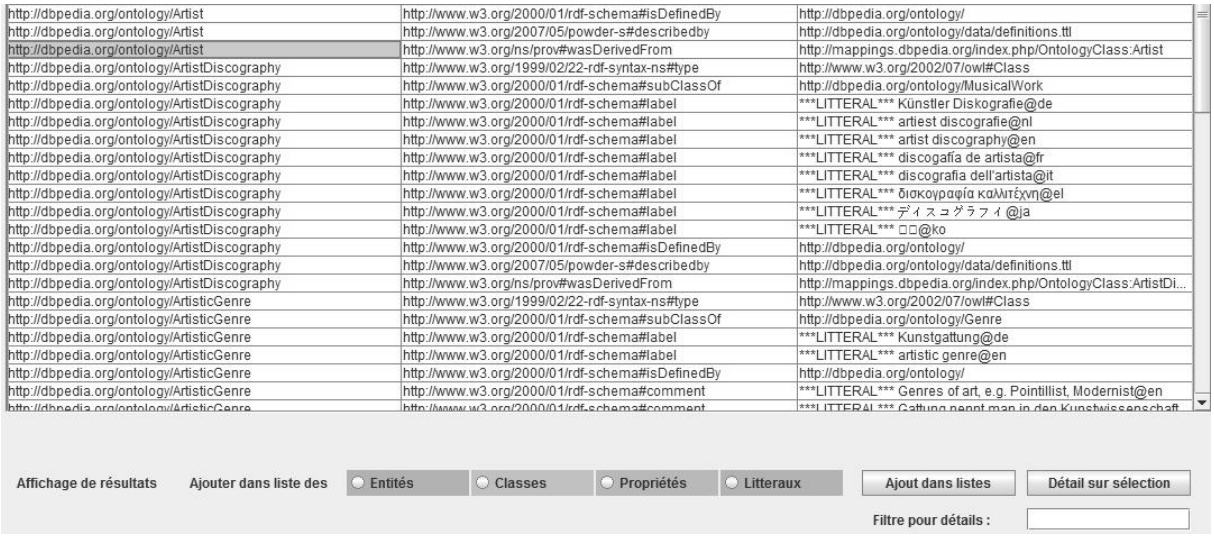


Figure 67 Résultats de « Trouver toutes les propriétés des individus qui sont des artistes »

Choix de l'IRI voulue, puis retour dans le panneau principal . (remarque : aller dans le bouton 'détail sur sélection' fournirait les propriétés de la classe en elle-même mais pas celles des entités appartenant à cette classe.

'Recherche de propriétés des individus appartenant à une classe'.

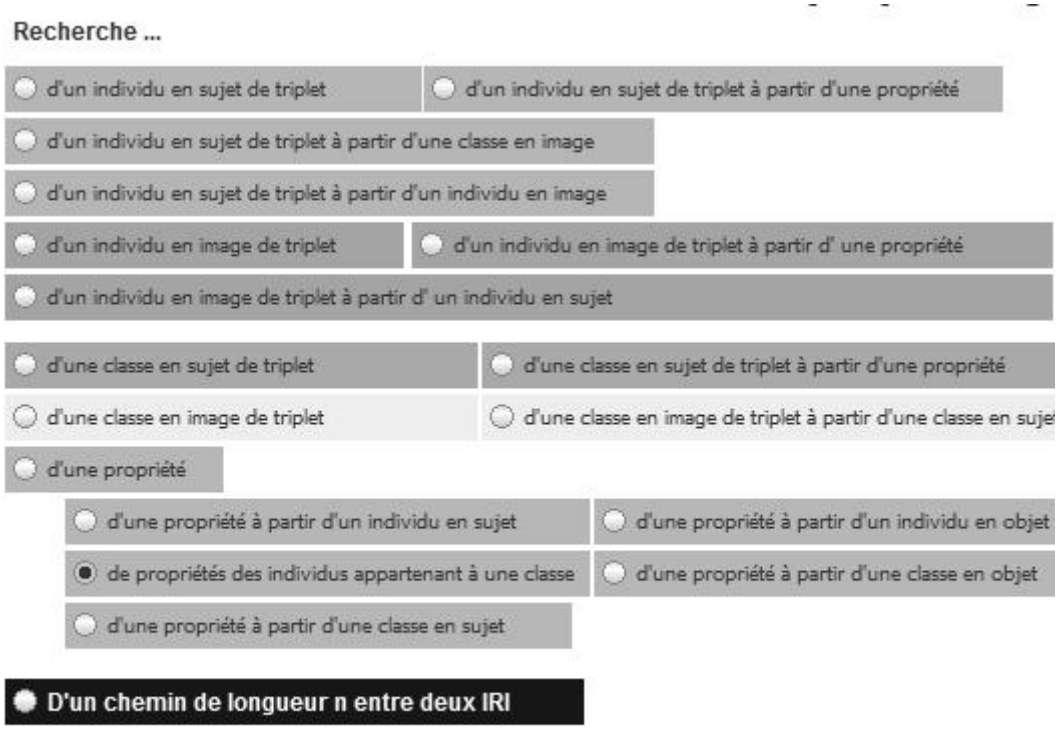


Figure 68 Recherche de propriétés des individus appartenant à une classe

Le panneau suivant s'ouvre et donne la possibilité de filtrer le nom des propriétés et le choix de la classe voulue (ici Artist).

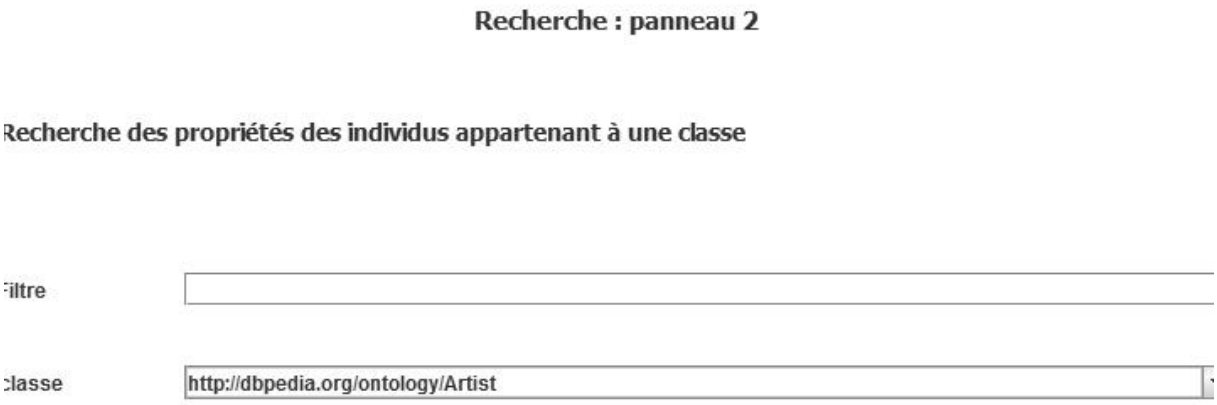


Figure 69 Possibilité de filtrer le nom des propriétés et le choix de la classe 'Artist'

Le lancement de la requête donne la liste des propriétés attendues :

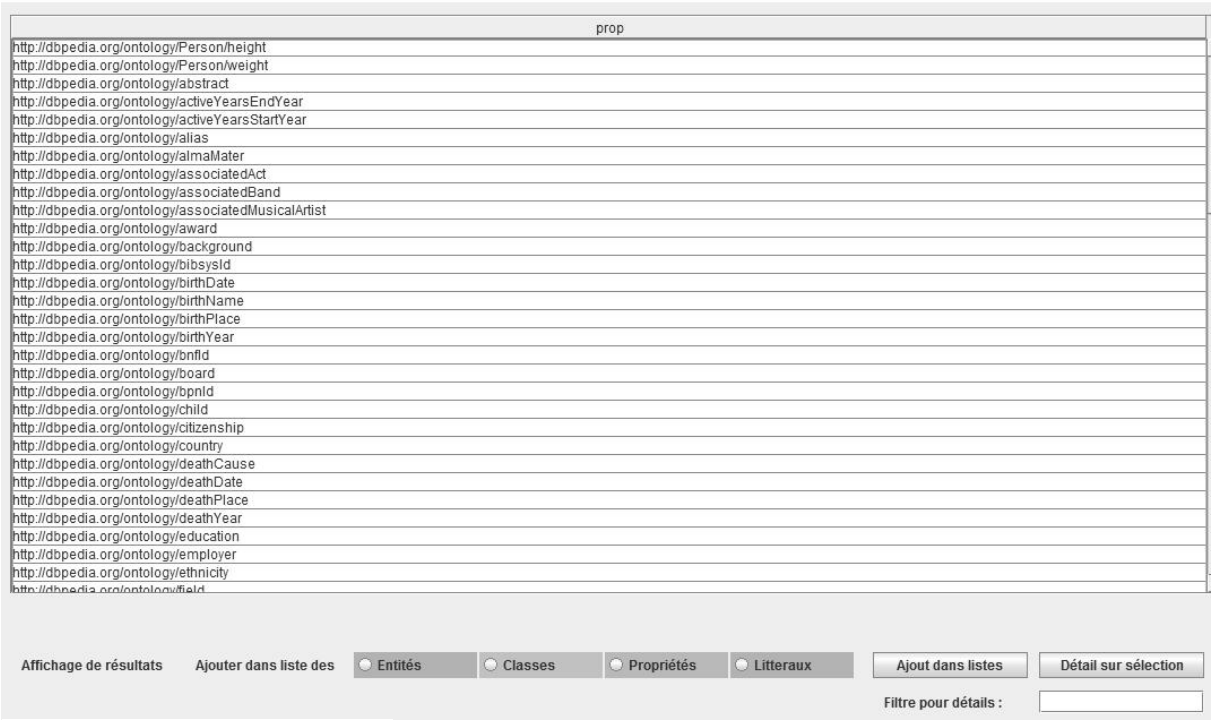


Figure 70 Liste des propriétés attendues

Sachant maintenant retrouver des IRI instinctivement et les ayant enregistrées, nous pouvons passer à l'étape de construction de requêtes complexes.

5- Construction des requêtes complexes : Jointure

Nous illustrerons cette étape par un exemple :

Soit donc à ***Trouver toutes les personnes dont le lieu d'habitation est également celui de Arnold Schwarzenneger et qui sont des artistes dont le nom contient la chaîne 'rob'.***

Nous sommes dans un cas de jointure et d'établissement de requête. Le logiciel offre une souplesse quand aux moyens d'arriver à la formalisation de la requête. Il requiert toutefois un raisonnement de l'utilisateur en terme de triplets RDF: sujet propriété image, c'est à dire que propriété(sujet) = image.

Etapes :

- Trouver l'URI d'Arnold Schwarzenneger

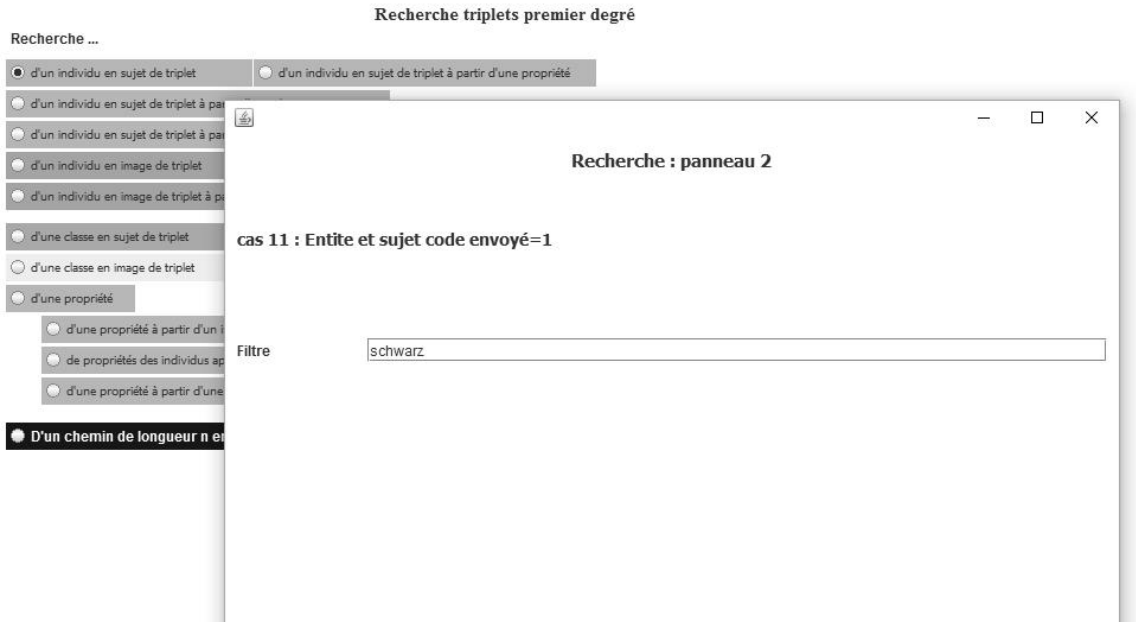


Figure 71 Trouver l'URI d'Arnold Schwarzenneger

Sélection de l'URI qui nous intéresse dans la liste des résultats et stockage dans la liste des entités (ajout dans listes>> radio bouton Entités) :

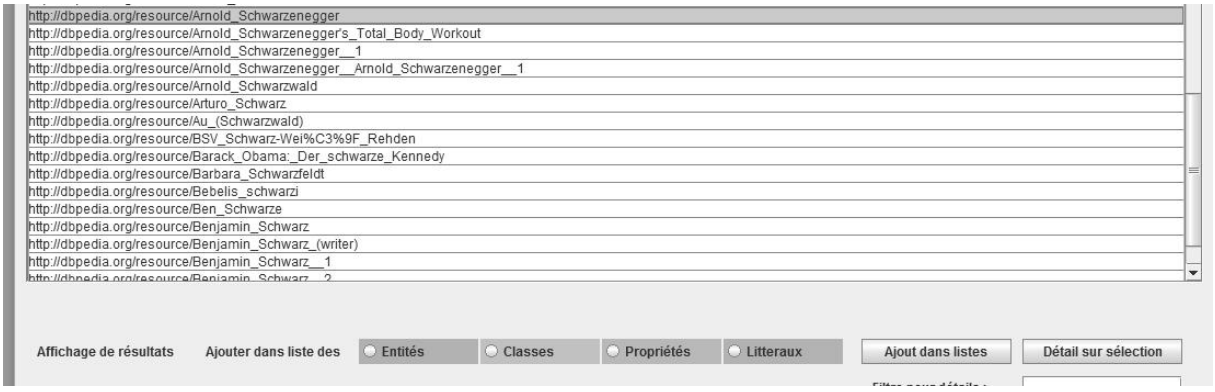


Figure 72 Sélection de l'URI dans la liste des résultats et stockage dans la liste des entités

- Trouver, à partir de cette URI, les propriétés relatives à la notion d'habitation.>>bouton 'détail sur sélection '.

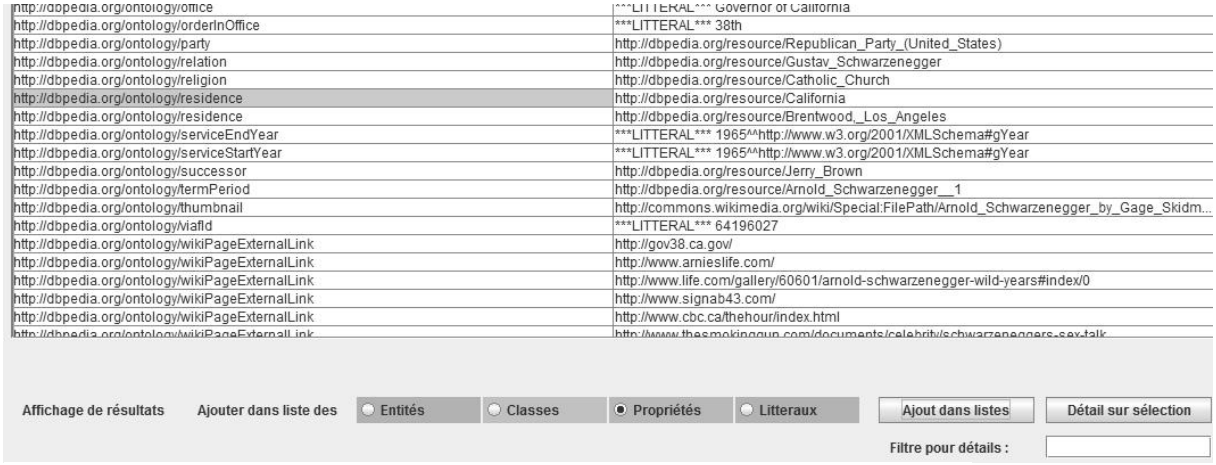


Figure 73 Trouver les propriétés relatives à la notion d'habitation.>>bouton 'détail sur sélection '.

Sélection de la propriété voulue et stockage dans la liste des URI de propriétés .

- Trouver ensuite l'IRI liée au concept d''artiste' :

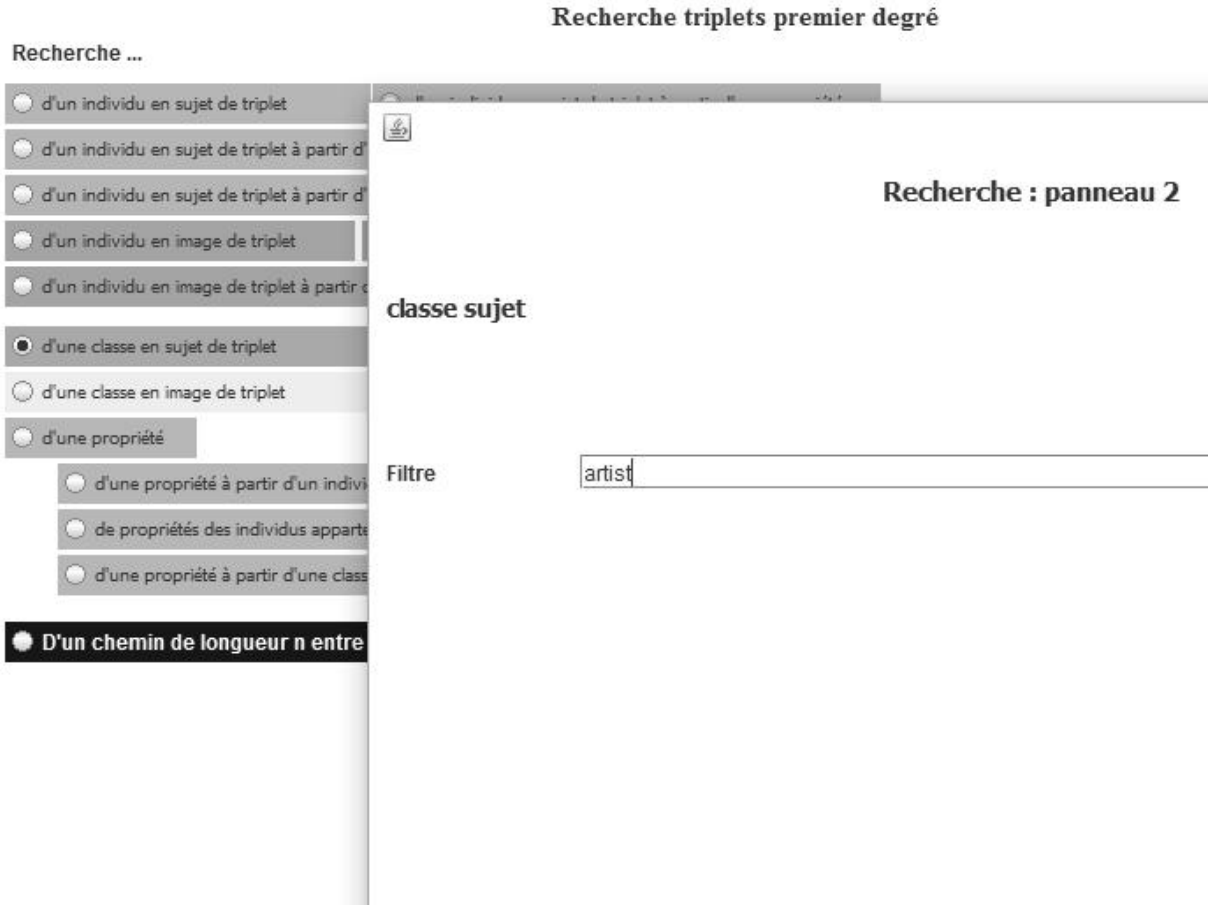


Figure 74 Trouver ensuite l'IRI liée au concept d''artiste'

Choix de l'URI parmi les choix proposés et ajout dans la liste des classes :

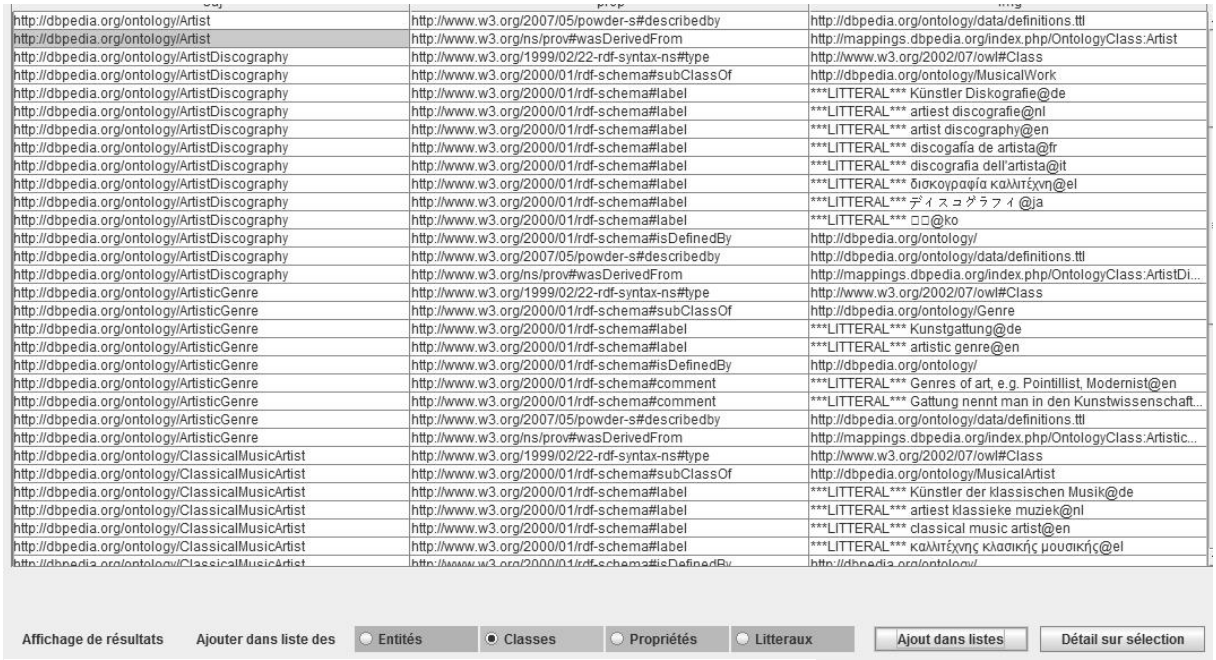


Figure 75 Choix de l'URI parmi les choix proposés et ajout dans la liste des classes

- Effectuer la requête de jointure des deux conditions

Panneau Request/jonction

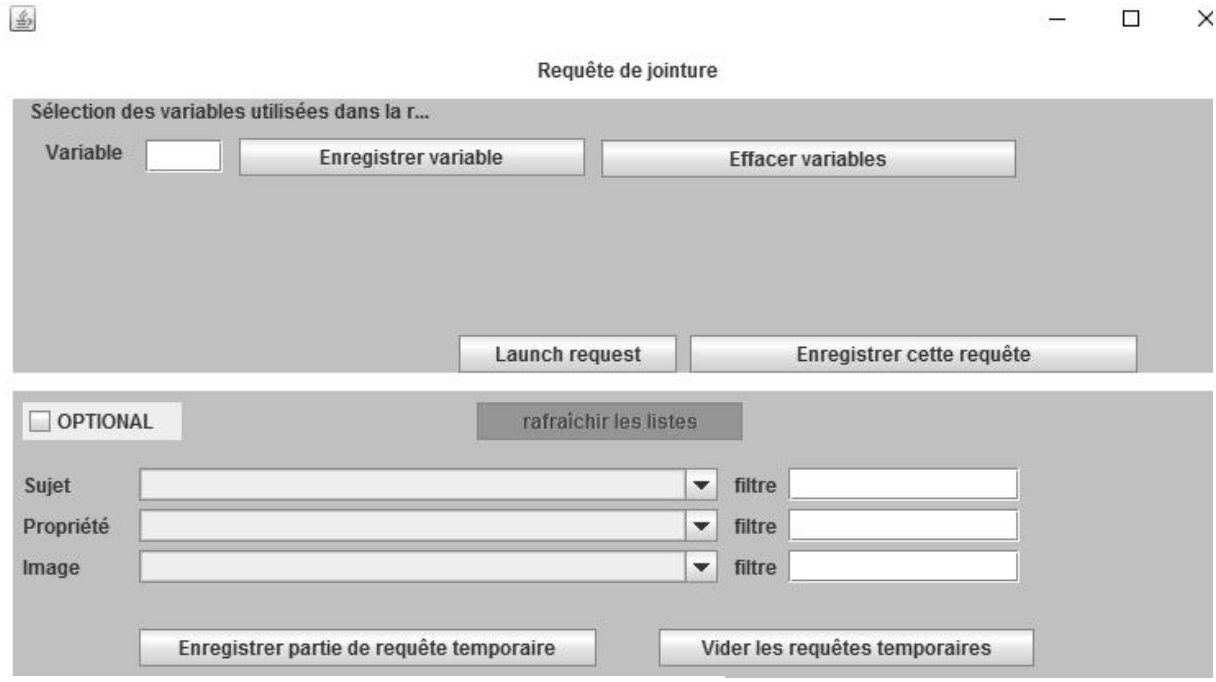


Figure 76 Panneau de construction de requêtes et de jointures de requêtes

- Les variables se rentrent sans le caractère '?'
- 'enregistrer variable' permet sa prise en compte dans la selection
- 'effacer variables' remet à zéro les variables
- 'launch request ' lance la requête affichée dans le text-panel (qui peut aussi être modifié manuellement par l'utilisateur)
- 'enregistrer requête' permet d'enregistrer la requête affichée dans la liste globales des requêtes (persistance après fermeture de la fenêtre actuelle). (on pourrait envisager aussi un enregistrement des requêtes sur un support physique).

- La case ‘optionnal’ permet d’activer une partie de jointure ‘optionnal’ (voir les spécifications SPARQL plus haut).
- Le bouton ‘rafraichir les listes’ permet de réactualiser les listes. (possibilité d’ajouter un design pattern ‘observateur’ pour éviter cela par la suite).
- Le bouton ‘Enregistrer partie de requête’ permet de générer la partie constituée par les combos . Cet enregistrement n’est que local au temps de vie du formulaire.
- Vider les requêtes temporaires permet de ‘flusher’ la mémoire liée au formulaire.

Première variable inconnue ?place, lieu de résidence de Schwarzenneger.>>enregistrer variable
Deuxième variable ?sujet, nom des individus habitant à ?place.>>enregistrer variable

Figure 77 Définitions des variables

la création de la première partie de la requête de jointure se traduira ainsi sur le panneau :

(Entrée de « place » dans variable >> enregistrer variable >> entrée de « suj » dans variable >> enregistrer variable >>
Choisir l’URI de schwarzenegger dans le combo sujet >> choix de l’URI de propriété ‘residence’ >> choix de la variable ?place dans le combo image >> enregistrer partie de requête

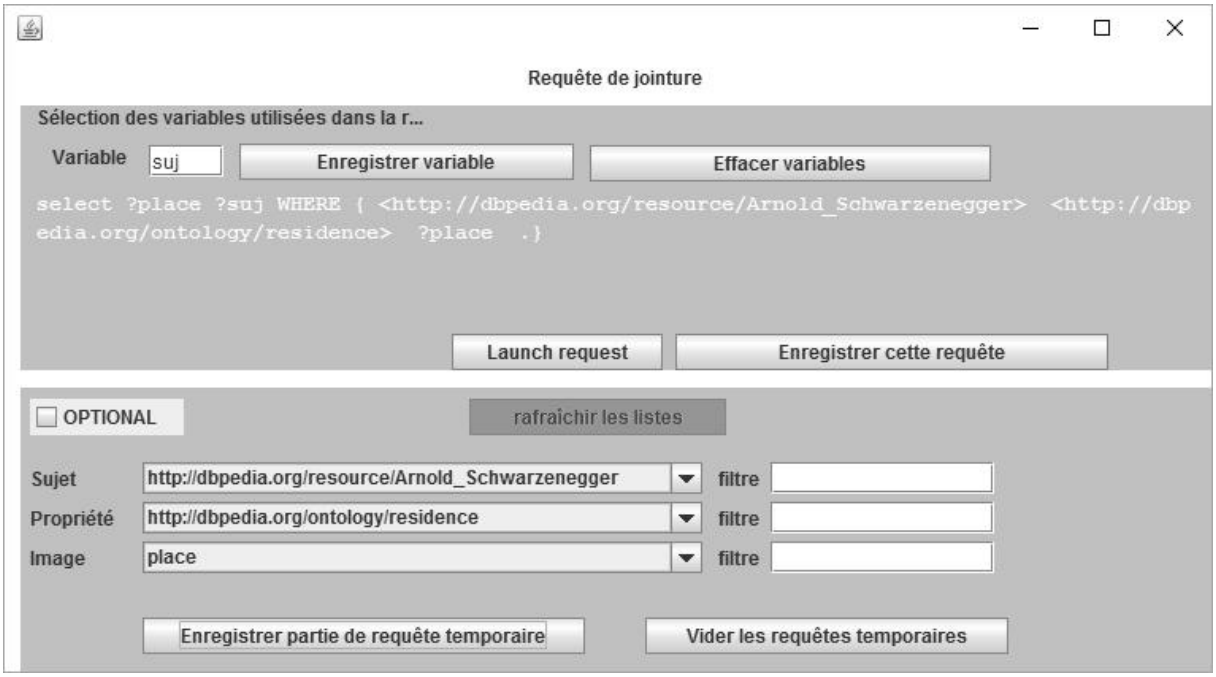


Figure 78 Construction de la requête

Deuxième partie de la requête :
Choix de ?suj dans le combo Sujet >> Choix de l'URI de résidence dans le combo Propriété >> choix de la variable ?place dans le combo Image >> Enregistrer partie de requête

Figure 79 Construction de la deuxième partie de requête

Troisième partie de la requête :
Choix de ?suj dans le combo Sujet >> Choix de l'URI de 'type' dans le combo Propriété >> choix de l'IRI du concept 'artist' dans le combo Image >> Enregistrer partie de requête

Figure 80 Construction de la troisième partie de requête

Pour voir les résultats <<launch request>>

Figure 81 Résultats issus d'un appui sur <<launch request>>

Qui traduit la requête : (deux jointures)

```
select ?place ?suj WHERE {
        <http://dbpedia.org/resource/Arnold_Schwarzenegger>  <http://dbpedia.org/ontology/residence>
        ?place  .
        ?suj  <http://dbpedia.org/ontology/residence>  ?place  .
        ?suj  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>  <http://dbpedia.org/ontology/Artist> .
         FILTER regex(str(?suj),"rob","i") .
}
```

Nous allons, pour répondre au cas suivant, 'enregistrer cette requête'.

6- Construction des requêtes complexes : Union

Exemple 1 : Trouver toutes les personnes dont le lieu d'habitation est également celui de Arnold Schwarzenneger et qui sont des artistes dont le nom contient la chaîne 'rob'. OU toutes les personnes qui sont des présidents et dont le nom contient 'joel'.

Notre première partie de requête a été précédemment créé.
Nous devons, de la même manière créér la requête qui donne les IRI des personnes dont le nom contient 'john'.

- Recherche de l'iri de la classe President

- Puis, panneau de requête/jointure et recherche de ?suj a <IRIsinger>

Figure 82 Construction de la deuxième requête

Enregistrement de la requête, puis passage dans le formulaire d'union.

Pour unifier, l'utilisateur doit veiller à avoir des variables affichées qui soient cohérentes . Si ?suj est choisi dans la première requête, elle le sera dans la deuxième.

Suj >> enregistrer variable
Choix de notre première requête dans la liste 1
Choix de la deuxieme requête dans la liste 2

Puis 'UNION'

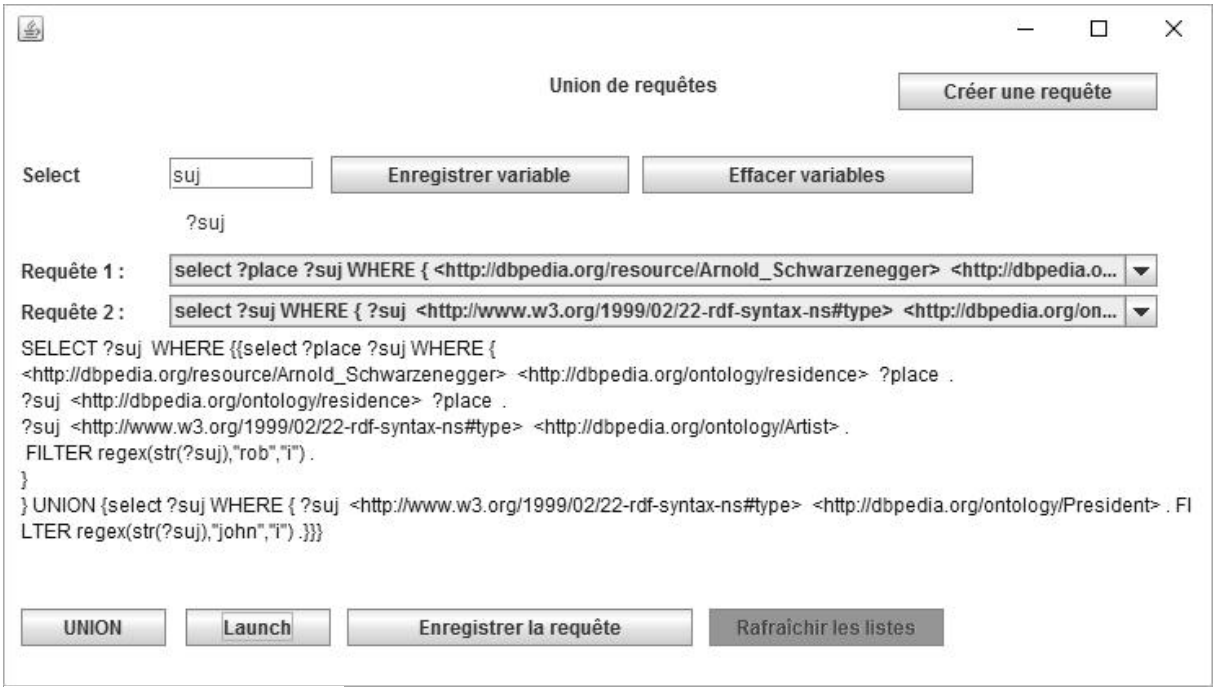


Figure 83 Union des deux requêtes

et launch :

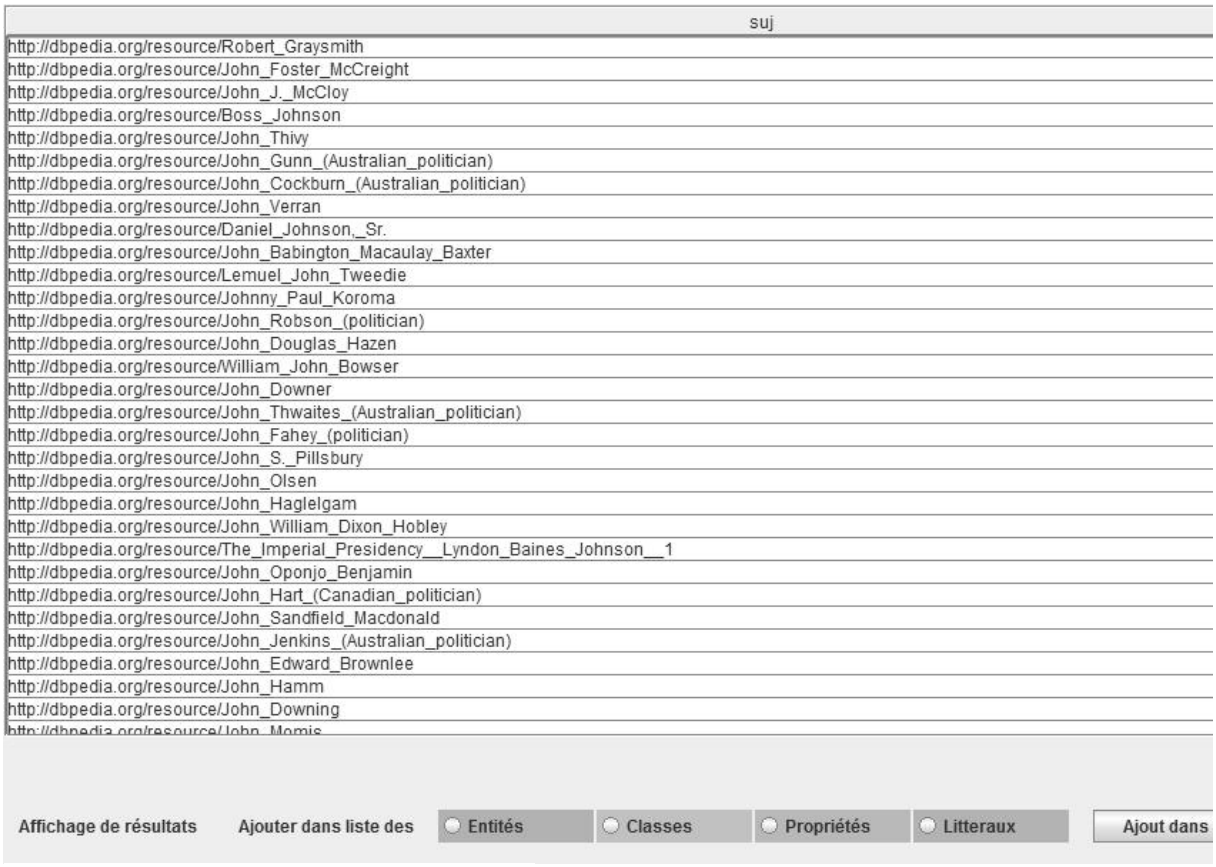


Figure 84 Résultat de l'exemple de l'union de requêtes

On peut ensuite également enregistrer cette requête dans la liste des requêtes et s'en servir à nouveau dans une requête d'union.

Exemple 2 : Trouver toutes les personnes dont le lieu d'habitation est soit la France, soit l'Angleterre.

Requête d'union :
-Trouver l'URI de la France
-Trouver l'URI de l'Angleterre
-Trouver l'URI de la propriété habite ou réside
- Créer les deux requêtes simples
- Unir ces deux requêtes

On se sert ensuite du panneau d'union de requêtes :

Choix de la variable utilisée dans les requêtes >> '+' >>
Choix dans le combo requête 1 de la première requête intervenant dans l'union>>
Choix dans le combo requête 2 de la seconde requête intervenant dans l'union>> 'UNION'

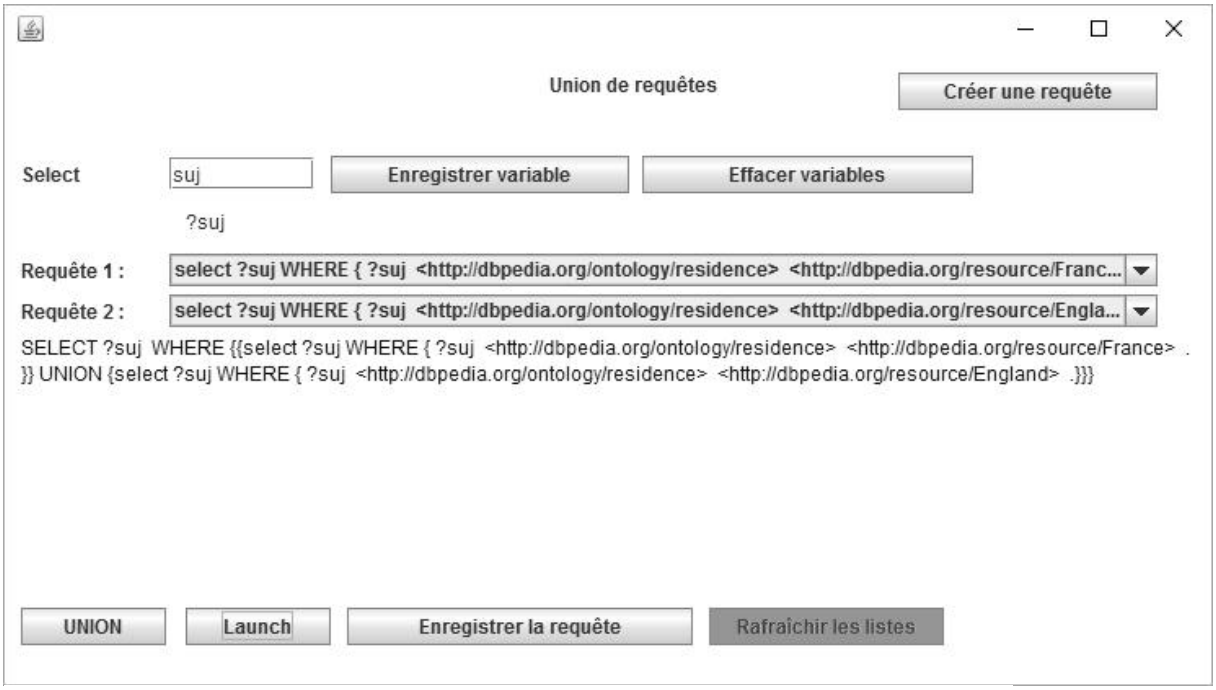


Figure 85 « Trouver toutes les personnes dont le lieu d'habitation est soit la France, soit l'Angleterre. »

>>'Launch'

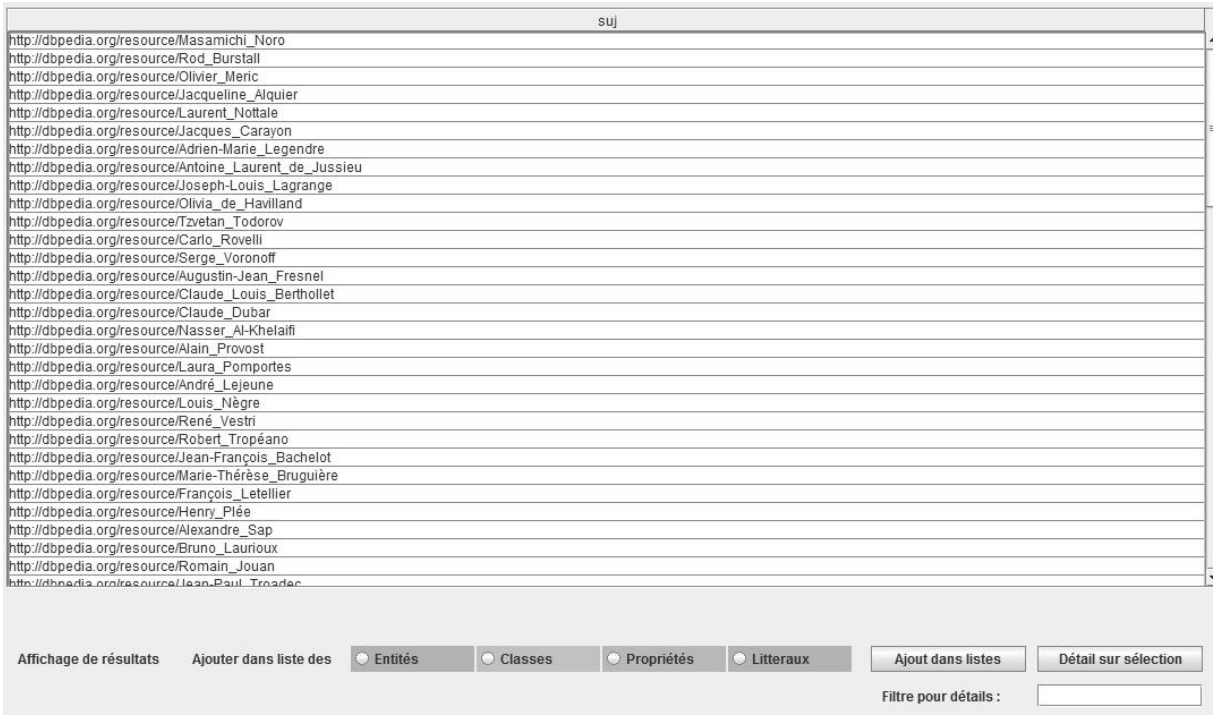


Figure 86 Résultat de la requête : « Trouver toutes les personnes dont le lieu d'habitation est soit la France, soit l'Angleterre. »

7- Construction de requêtes complexes : Agrégats

Ce volet permet la construction, à partir de requêtes préalablement créées, de requêtes utilisant les opérateurs de type 'group by …. Having…sum, count, avg'.

Il réutilise des requêtes précédemment créées et leur adjoint les opérateurs d'aggrégation (SUM, COUNT, MIN)

Partie 4

Phases de test : Interrogation d'une ontologie sur les particularités touristiques de Phuket

---

Pour la conception du logiciel, je me suis appuyé sur l'ontologie Dbpedia en raison de sa taille et de sa complexité importantes ('qui peut le plus, peut le moins'). Le travail consistait cependant à développer un outil générique. Le présent test permet d'évaluer également ce critère d'adaptabilité.

Dans ce sens, une ontologie que je ne connaissais pas a donc été créée par un membre de l'équipe CARTOON (Kitsiri Chochiang), ontologie relative au tourisme dans la ville de Phuket en Thaïlande et assortie de questions relatives à ce thème dont les réponses devaient être trouvées avec l'outil créé.

Cette ontologie se présente se structure autour de trois ontologies qui se complètent l'une l'autre :

1- Une ontologie de langage permettant la traduction des termes (Anglais, Thaïlandais).
2- Une ontologie de profil d'utilisateurs
3- L'ontologie principale , de tourisme basée sur les classes principales suivantes :
    a. Evènements et festivals
    b. Attractions
    c. Magasins
    d. Restaurants

L'objectif sera donc de voir si nous arrivons, à l'aide du logiciel, à extraire les renseignements de cette ontologie pour répondre à des questions relatives à des évènements, des horaires, des lieux, etc…

A- Requêtes sur l'ontologie, méthodes pour obtenir les réponses

Queries are based on the file 'Phuket2.owl' and concern the ontology 'Phuket'

http://www.h2dal.com/ontology/Phuket2.owl

Preliminary step : Ontology choice :

Ontology Choice >> Local System File >> File Explorer

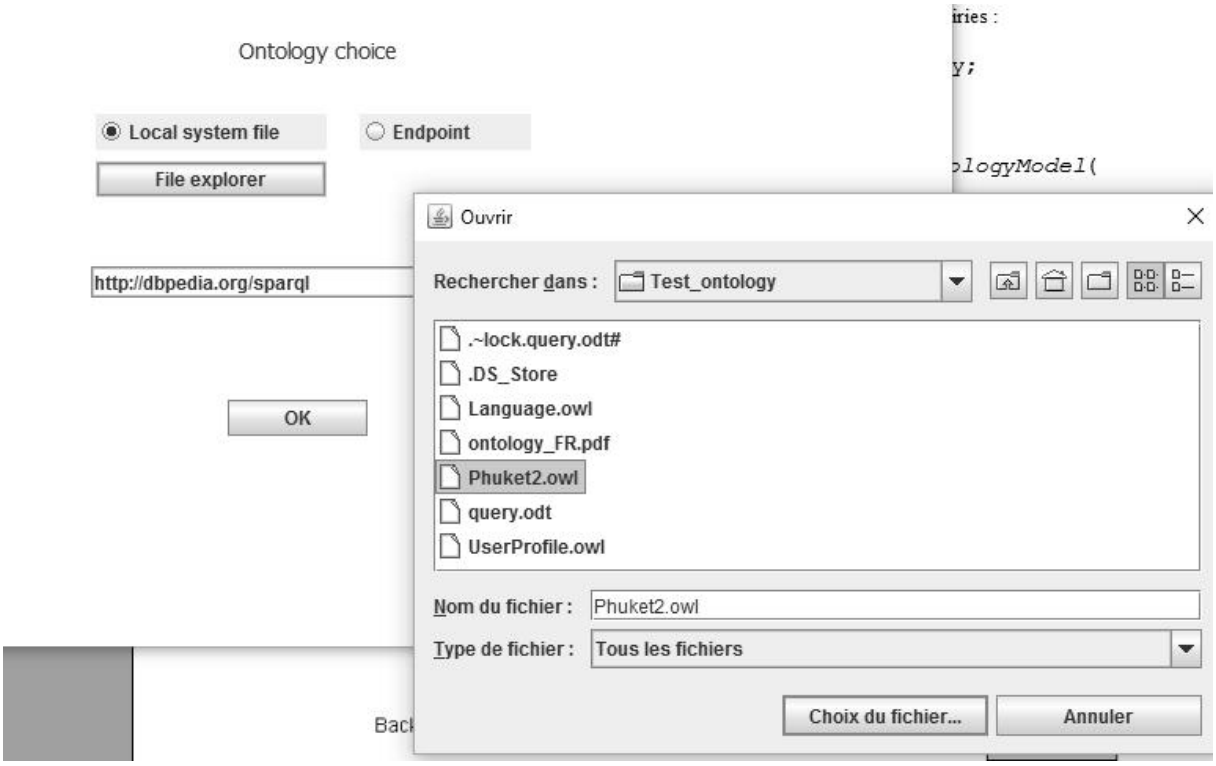


Figure 87 Choix de l'ontologie Phuket2.owl

Button ok.

1- Find the japanese restaurants :

- We must first find the IRI relied to the concerned class.

Figure 88 Bouton de recherche d'IRI

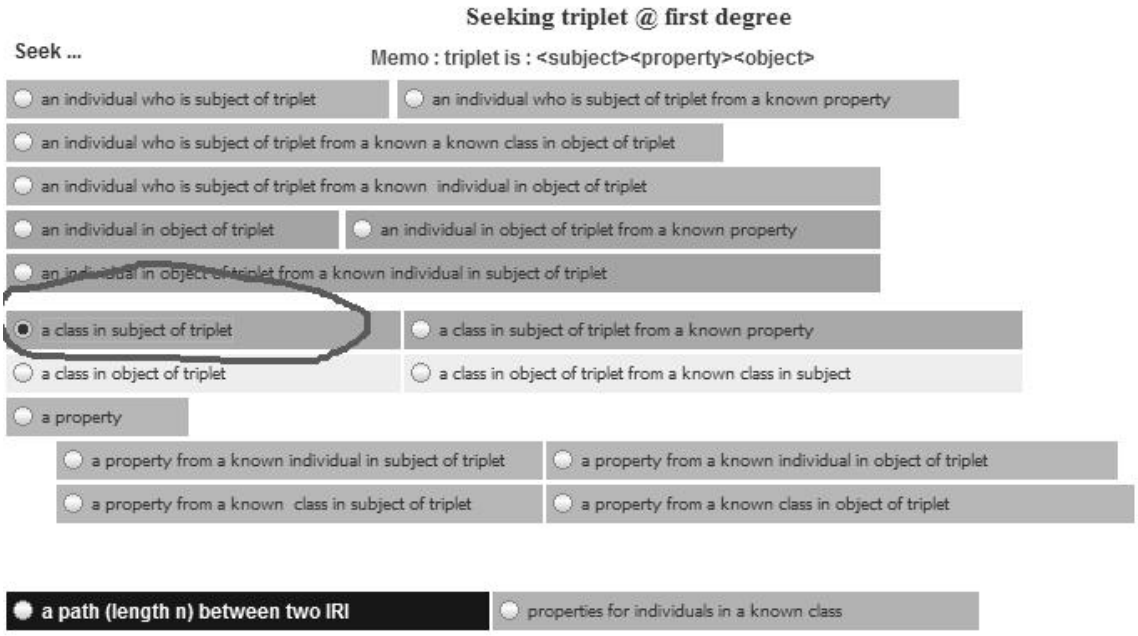


Figure 89 Recherche d'une classe contenant les restaurants japonais

We are looking for a class, this which contains the japanese restaurants. The retained keyword will be "japan" to realize the research.

Figure 90 Sous formulaire de recherche d'une classe contenant les restaurants japonais

Launch query >> The following panel will propose several choices.

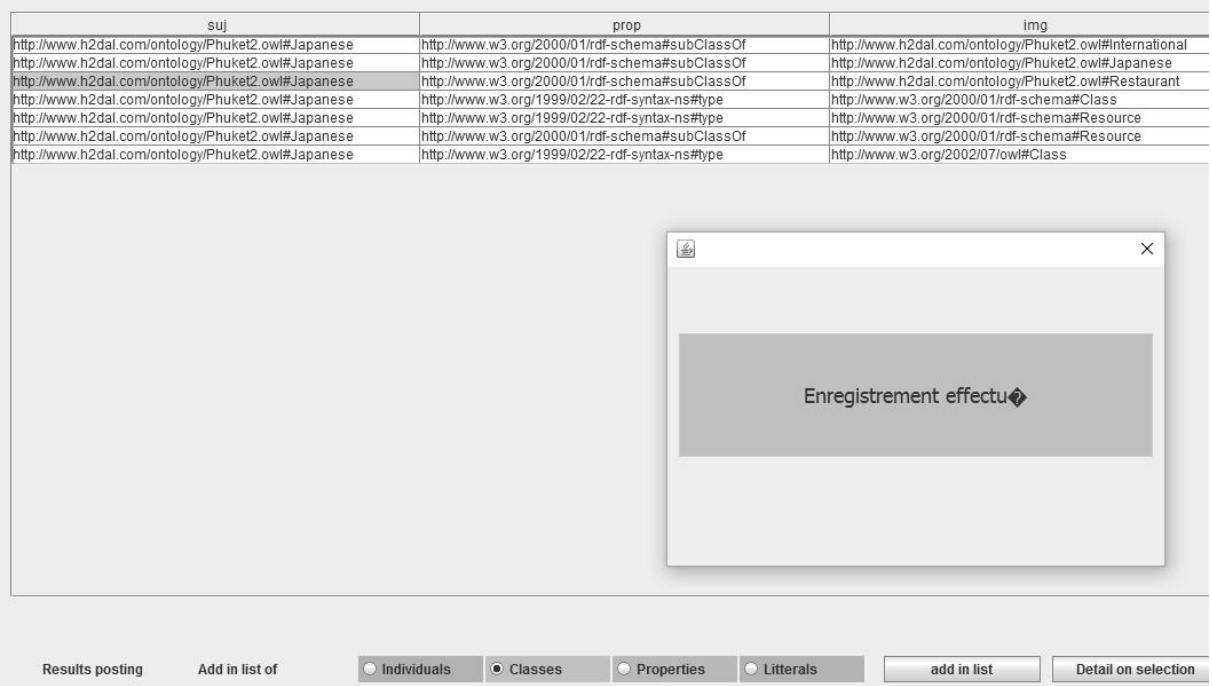


Figure 91 Panneau de résultats permettant le choix de l'IRI voulue.

We select the IRI we find the best >> and the record with 'add in list' to have this IRI disponible in our comboList in the others panels. Then, close the window.

We have our conceptuel IRI elements to construct our query. It will be sufficient to ask for the elements who belong to the class 'japanese restaurant' (for which we have the IRI recorded).

Menu>>Query & jonction

Figure 92 Bouton de construction de requêtes/jointures de requêtes

In the 'variable' textfield, enter 'suj' (or another name of variable you want to have)>>then, 'record variable' button.

In the above part of the window, 'refresh list'.

In the combolist 'sujet' : choose our variable (suj)

In the combolist 'propriété' : Choose the type property 'is_a' (abbréviation 'a' ou IRI http://www.w3.org/1999/22-rdf-syntax-ns#type) This rdf property structure is pre-recorded in our listes (It is possible to pre-record some other usefull one like 'subclassof' by exemple).

In the combolist 'image' : choose the IRI of the class of the japanese restaurants we recorded the step before .

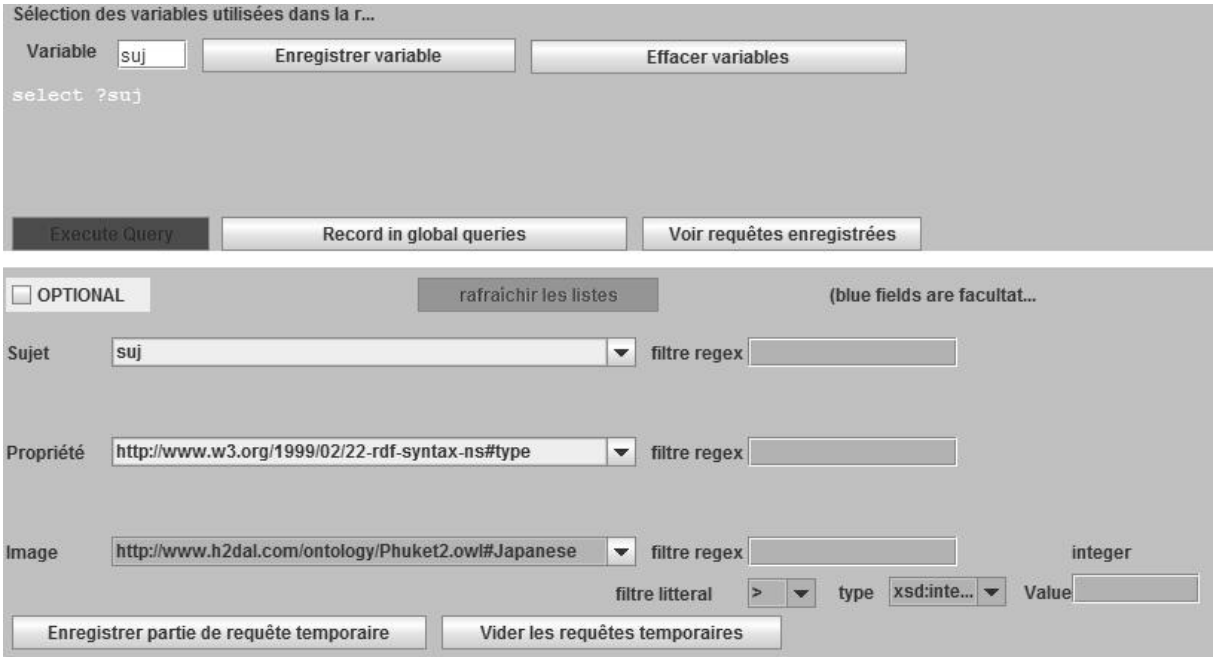


Figure 93 Construction de la requête pour trouver les membres appartenant à la classe des restaurants japonais

Then, « record part of temporary query (to put it in the screen) »

The above part of screen display the sparql query (it can be manually modified if the user want to) :

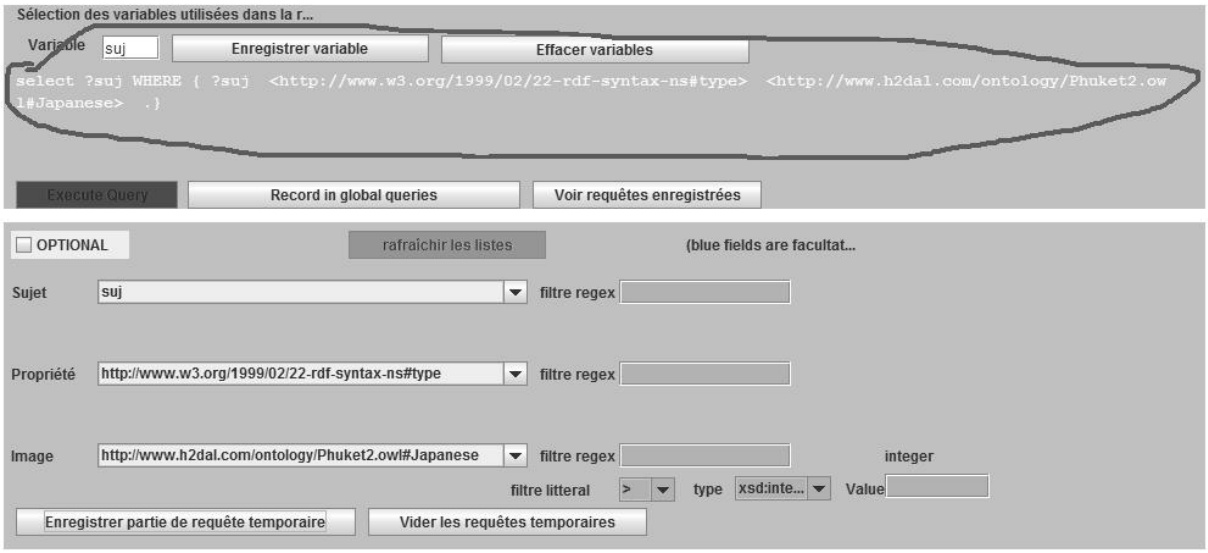


Figure 94 La partie supérieure donne le code SPARQL de la requête

On peut alors enregistrer notre requête dans les requêtes globales (record in global queries), ou la lancer tout simplement (execute query).

Then we can record our query in the global queries (to retrieve it later after closing this window by example), or to execute it (execute query)

The result is then :

suj
http://www.h2dal.com/ontology/Phuket2.owl#Zen
http://www.h2dal.com/ontology/Phuket2.owl#FUJI3
http://www.h2dal.com/ontology/Phuket2.owl#FUJI2
http://www.h2dal.com/ontology/Phuket2.owl#FUJI1

Figure 95 Panneau de résultats listant les restaurants japonais

We have the query SPARQL:

select ?suj WHERE { ?suj <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.h2dal.com/ontology/Phuket2.owl#Japanese> .}

2- Which districts have a FUJI restaurant :

The idea is to firstly look for a property with the keyword ‘district’.

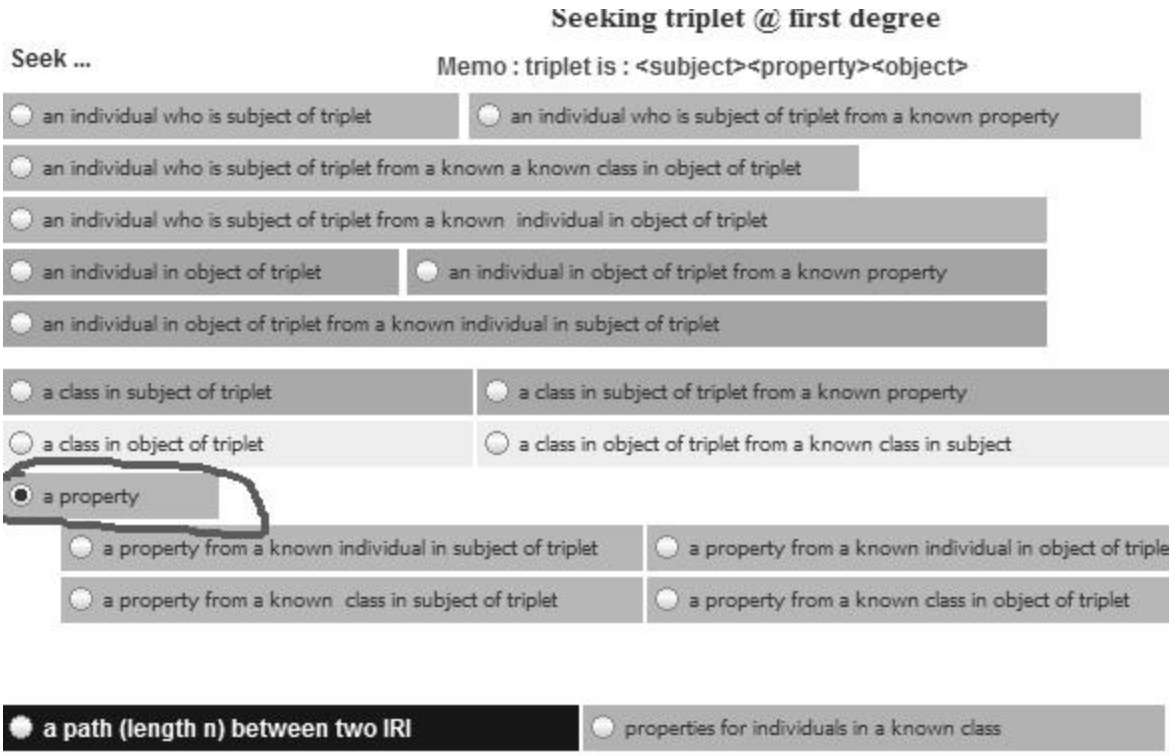


Figure 96 Recherche d’une IRI de propriété

Then, ‘ok’ and put the keyword ‘district’ in the textfield :

Figure 97 Sous formulaire de recherche d'une IRI de propriété.

Then 'launch query'.

We obtained the following results :

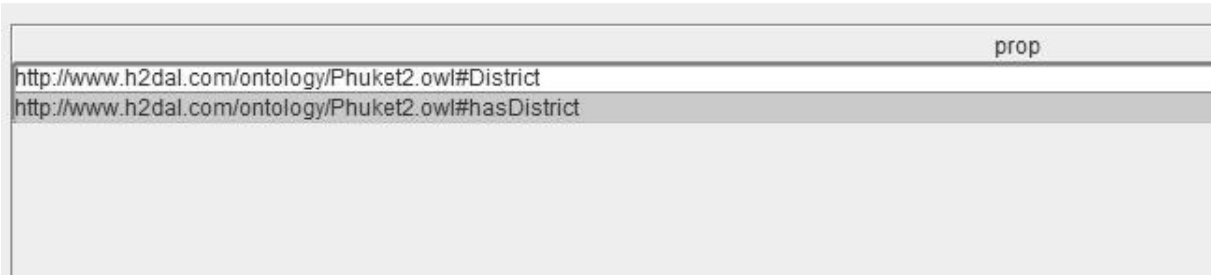


Figure 98 Liste de IRI de propriétés proposées

We choose the 'hasDistrict' property and add it in the list of properties.

Figure 99 Choix de L'IRI et ajout de celle-ci dans la liste de mémorisant les IRI de propriétés.

Then, close the window.

We will then realize the query with the Query/jonction panel .

Variables : suj, distict

List sujet : variable suj + filtre 'fuji'
List property : IRI 'hasdistrict '
List image : variable district

Then, button 'record part of temporary query' and then record and execute.

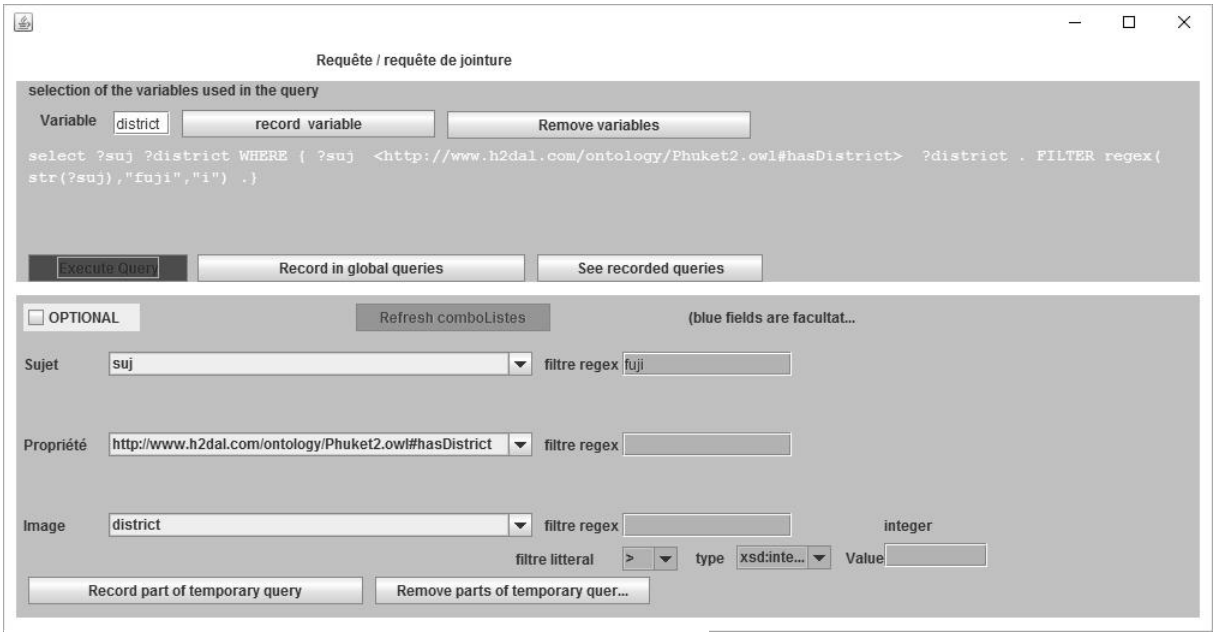


Figure 100 Création de la requête àpartir des IRI précédemment trouvés

The result is then :

suj	district
http://www.h2dal.com/ontology/Phuket2.owl#FUJI2	http://www.h2dal.com/ontology/Phuket2.owl#Kathu
http://www.h2dal.com/ontology/Phuket2.owl#FUJI1	http://www.h2dal.com/ontology/Phuket2.owl#Mueang
http://www.h2dal.com/ontology/Phuket2.owl#FUJI3	http://www.h2dal.com/ontology/Phuket2.owl#Kathu

Figure 101 Résultat, liste des restaurants japonais et de leurs districts

And the SPARQL query :
select ?suj ?district WHERE { ?suj <http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict> ?district .
FILTER regex(str(?suj),"fuji","i") .}

Suite du test : voir annexe.

B- Analyse du test

1- Aspect validation/test de l'outil

La phase de test s'est effectuée sur une ontologie dont je n'avais pas connaissance au moment de la conception du logiciel et fournie par un membre du laboratoire de recherche, assortie d'une série de questions auxquelles le logiciel devait répondre.

- *Au niveau ergonomique :* l'application s'avère très instinctive, à condition que l'utilisateur ait une notion de triplets (sujet, propriété, images) ainsi que des notions de concepts, d'instances et de propriétés. Le principe se rapprocherait le plus fortement de celui de NITELIGHT avec cependant une possibilité plus libre et plus souple de la création de la requête une fois les IRI trouvés. Au lieu du panneau de construction graphique, nous avons notre interface de création dynamique de requête SPARQL, requête qui peut aussi être modifiée 'à la main'. L'utilisateur a donc une impression moindre d'être enfermé dans un schéma graphique parfois un peu opaque.

- *Au niveau prise de connaissance de l'ontologie* et de recherche des IRI (briques essentielles de la constitution des requêtes), le logiciel s'avère également très efficace. Moyennant quelques mots clefs bien choisis, les concepts et objets sont aisément retrouvables dans l'ontologie . Par rapport aux autres solutions graphiques étudiées, l'outil proposé permet d'avoir plus d'aisance et de recul, l'utilisateur n'étant pas restreint dans ses choix précédents pour constituer sa requête ou trouver les éléments qu'il désire. La visualisation globale via un tableau de tableau de données semble être plus confortable que la visualisation proposée par QueryVowl qui s'avère vite assez confuse lorsqu'une ontologie est composée de centaines d'éléments, et ce, en dépit, des techniques basées sur les cardinalités de propriétés par classe influant sur la taille des éléments les représentants.

- *Au niveau visualisation et souplesse de la création de la requête :* Notre outil s'avère là aussi plus 'libre', laissant à l'utilisateur la possibilité d'utiliser les IRI qu'il veut, voire même de modifier la requête SPARQL manuellement. L'accès au code généré SPARQL est toujours aisément disponible pour l'utilisateur, ce qui est intéressant dans le cas ou celui-ci désire réutiliser ces requêtes dans d'autres applications.Les différents cadres proposés et la graduation de la création de requête, à savoir :
    - *Recherche des IRI et stockage*
    - *Construction des requêtes de bases à partir des IRI stockées et stockage*
    - *Construction des requêtes complexes à partir des requêtes basiques stockées*
    - *Opération de regroupement sur les requêtes*

permettent un raisonnement sur la création de la requête et l'objectif à atteindre, par étapes. Les techniques étudiées dans l'état de l'art ne permettent souvent pas cette étude fractionnée.

2- Apport pour le travail

Le besoin était de guider à la rédaction des requêtes SPARQL, ce qui sous-entendait un accès aisé et constant au code SPARQL généré. Cela est le cas, chaque formulaire proposant un cadre éditable dans lequel figure la construction de la requête effectuée tant au niveau de la recherche des IRI que de la construction des requêtes simples ou complexes. Le logiciel ne se propose donc pas seulement de fournir des résultats mais également le code construit qui a permis de les obtenir. Le code généré peut ainsi être réutilisé afin de créer des requêtes ou des requêtes types pouvant être intégrées par exemple dans la représentation d'ontologies bien définies sur un support informatif (WEB par exemple).

Le cadre du travail au sein du laboratoire de recherche consistait à fournir un outil générique permettant à un membre de l'équipe de pouvoir interroger les ontologies qu'il élaborerait, lui permettant ainsi d'extraire le code SPARQL généré une fois les résultats voulus obtenus.

Le logiciel créé apporte donc deux contributions principales :

- Dans un cadre d'utilisation plus étendu, permettre à un utilisateur d'obtenir des résultats d'une ontologie à partir de mots clefs et d'aide à la structure de ses requêtes

- Dans le cadre de ma collaboration avec Kitsiri Chochiang (doctorante) , permettre la génération de code SPARQL ayant mené aux résultats pour réutilisation ultérieure.

Cet outil se situera donc en aval d'une ontologie créée et en amont de création d'une interface de représentation de cette ontologie ou d'outil consultatif générique.

Conclusion et perspectives

Lors de ce mémoire, nous avons, au préalable, étudié le support de nos sources de données, à savoir l'ontologie, sa raison d'être, sa structure, ses fonctionnalités et ses implémentations pour les supports informatiques, nous avons ensuite précisé en quoi consistait SPARQL, au niveau de ses spécifications, de sa sémantique et de sa syntaxe et avons ainsi pu ainsi constater la complexité des écritures des requêtes, principalement du fait de la recherche des IRI qui y sont utilisées.

Nous avons ensuite établi un état de l'art des quelques techniques proposées ces dernières années visant à simplifier et rendre plus instinctives les requêtes sur une ontologie dont la structure et les IRI ne nous sont pas connus ou dont la complexité ne permettrait pas toujours de s'y familiariser pour retrouver les entités (IRI) dont nous aurions besoin.

C'est ainsi que nous avons d'abord étudié des techniques basées sur l'analyse lexicale et sémantique de questions posées en langage naturel. Le problème constaté résidait dans le fait que plus le procédé était automatisé, plus la marge d'erreur des interprétations du moteur de raisonnement était importante, ce raisonnement étant implanté et mû par des dictionnaires(comme Wordnet) permettant l'appariement de termes consistant en extraction d'entités, de relations, par le rapprochement et l'analyse des interactions entre celle-ci pour tenter une réponse cohérente. Les bases de connaissances sémantiques examinées sont souvent elles-mêmes fautives du fait d'inconsistances ou d'une construction incohérente (tant qu'une norme de construction ontologique ne sera pas standardisée, le problème se posera. Certaines ontologies de construction comme foaf permettent déjà de réaliser des liaisons standards dans un concept (ici la personne) et se révèlent de plus en plus comme des futures normes. (et donc de permettre déjà des automatisations beaucoup plus efficaces et fiables).

De ce constat, nous avons ensuite examiné quelques techniques de constructions de requêtes par visualisation graphique de l'ontologie. Nous avons pu en relever quelques techniques intéressantes comme le fait de retrouver instinctivement des concepts dans la base, d'en associer les propriétés et littéraux concernés. Toutefois, l'élaboration de requêtes plus complexes devient assez vite, à l'usage assez peu visualisable et peu engendrer un certain 'fouillis' dû parfois à la taille des ontologies examinées et donc la représentation graphique n'est pas parlante .

Nous avons donc de là, proposé une technique s'inspirant de la recherche d'objets (IRI) par un panneau de recherche offrant diverses méthodes pour déterminer et trouver au mieux les concepts dont nous aurions besoin dans une requête, puis, d'autres panneaux permettant des constructions plus complexes de requêtes.

Le test révèle un avantage indéniable, le fait que l'utilisateur puisse vite prendre conscience des objets et relations qui existent dans l'ontologie étudiée et associer, sans connaître SPARQL,les entités qu'il aura trouvées. Le test révèle néanmoins que l'utilisateur doit avoir un raisonnement à base de triplet RDF, ce qui ne sera pas forcément le cas d'un utilisateur non initié. Est révélé également par ce test un nombre de manipulations qui peut vite se révèler rebutant si l'utilisateur recherche beaucoup de réponses de types différents. Il révèle toute son efficacité pour trouver des IRI et monter des 'squelettes' ou structures de requêtes complexes que l'utilisateur pourra ensuite modifier 'à la main'.

Au niveau des perspectives, les évolutions futures seraient les suivantes :

- Simplifier les interrogations de plusieurs ontologies liées en proposant un système de propriétés équivalentes (l'une appartenant à une ontologie 1, l'autre à l'ontologie 2), de faire en sorte qu'un littéral

trouvé dans le résultat d'une requête puisse servir de filtre pour une autre (chose faite manuellement actuellement et provoquant des étapes supplémentaires).

- Tirer profit des techniques de 'relaxations de requête' et ainsi d'assouplir' le champ contextuel des requêtes n'apportant pas de réponse sur un filtre en élargissant la recherche sur des mots dont le sens conceptuel est similaire à ce filtre (sur un dictionnaire local par exemple défini et rempli par l'utilisateur), ou l'élargissement automatique de la recherche du terme sur les propriétés classiques de description d'un concept comme 'keyword' ou 'label' (la possibilité est ici aussi offerte par le panneau de recherche mais encore avec l'intervention de l'utilisateur qui teste les différentes possibilités).
- Travailler sur l'enregistrement et la persistance des IRI et des requêtes enregistrées .
- Au niveau présentation et diffusion de l'application sur un support WEB, la migration vers une application J2EE peut s'avèrer intéressant tant par sa capacité à générer les pages web en gardant la force de la librairie JENA que par ses possibilités de développement collaboratif.
- Pour ce qui est de la communication entre plusieurs ontologies, des relations d'équivalences pourraient être définies en dépendance avec l'ontologie particulière étudiée. On pourrait alors proposer une interface destinée à ce que l'utilisateur définisse ces équivalences(classes, propriétés, objets).

Les ontologies étant, de par leur utilisation dans le monde sémantique, destinées à voir une évolution forte durant les années qui viendront, il en sera de même des outils de consultation de celles-ci. Des standards futurs établis par des consortium comme le W3C viendront sans doute aider le développement de ces outils en fournissant une 'colonne vertébrale ontologique' ou des dictionnaires communs permettant ainsi d'automatiser les requêtes jusqu'à permettre l'expression de celles-ci en langage naturel avec un taux de succès quasi-total. Pour le moment, nous devons adopter un compromis entre automatisme et généricité.

Références bibliographiques

[art -1] Mika Viinikkala, 2004,Ontology in Information Systems, 2004, 22.03.2004
[art -2] Quine, 1948,On what there is, , Review of Metaphysics,
[art -3] Nicola Guarino, 2009, Daniel Oberle, and Steffen Staab,What Is an Ontology
[art -4] B. Chandrasekaran and John R. Josephson, 1999,What Are Ontologies, and Why Do We Need Them, Ohio State University
[art -5] G . Van Heijst,1997,Using explicit ontologies in KBS development, University of Amsterdam
[art -6] Anne Steiner, 2003,Réalisation d'une Topic Map sur l'introduction des TICs dans la GRH, 11.01.2003
[art -7] Bénédicte Le Grand, Michel Soto,2002,Visualisation of the Semantic Web: Topic Maps, Information Visualisation, Proceedings. Sixth International Conference on
[art -8] R. J. Brachman and J. G. Schmolze,1985,An overview of the KL-ONE knowledge Representation System. Cognitive Science, p171–216, April 1985.
[art -9] Mohamed Chaabani, 2009,Formalisation de la logique de description ALC dans l'assistant de preuve Coq, Université deToulouse
[art -10] Aimilia Magkanaraki, 2002, Ontology storage and querying, Foundation for Research and Technology Hellas, Avril 2002
[art -11] Jeen Broekstra, 2003, SeRQL: A Second Generation RDF Query Language, 4/11/2003
[art -12] Bastian Quilitz and Ulf Leser Humboldt, 2008, Querying Distributed RDF Data Sources with SPARQL ,Universit¨at zu Berlin,
[art -13] Jorge Perez, Marcelo Arenas, and Claudio Gutierrez, 2006,Semantics and Complexity of SPARQ,Universidad de Talca 2 Pontificia Universidad Catholica de Chile 3 Universidad de Chile
[art -14] Olaf Hartig1, Christian Bizer2, and Johann-Christoph Freytag ,2008,Executing SPARQL Queries over the Web of Linked Data
[art -15] Atanas Kiryakov, Vassil Momtchev, 2009, Triplesets: Tagging and Grouping in RDF Ontotext, AD Sofia Bulgaria
[art-16] Marcelo Arenas, Claudio Gutierrez, Jorge Perez, 2009, On the Semantics of SPARQL
[art -17] Romain Beaumont, 2015, Représentation sémantique de questions pour interroger le Web sémantique ,LIMSI-CNRS, France 2 Université Paris-Sud, France 3 ENSIIE
[art -18] Anthony Fader Luke Zettlemoyer Oren Etzioni, 2013, Paraphrase-Driven Learning for Open Question Answering Computer Science & Engineering University of Washington Seattle
[art -19] Russell, Alistair, Smart, Paul R., Braines, Dave and Shadbolt, Nigel R, 2008, NITELIGHT: A Graphical Tool for Semantic Query Construction, University of Southampton
[art -20] Florian Haag, Steffen Lohmann, Stephan Siek, and Thomas Ertl, 2015, Visual Querying of Linked Data with QueryVOWL Institute for Visualization and Interactive Systems, University of Stuttgart
[art -21] Steffen Lohmann, 2014, Visualizing Ontologies with VOWL,University of California, Santa Barbara, US
[art -22] Géraud Fokou ,2015 ,QaRS: A User-Friendly Graphical Tool for Semantic Query Design and Relaxation,LIAS/ISAE-ENSMA University of Poitiers
[art -23] Geraud Fokou ,2015,Cooperative Techniques for SPARQL Query Relaxation in RDF Databases, LIAS/ISAE-ENSMA - University of Poitiers
[art -24] Geraud Fokou, 2014, Conception d'un framework pour le traitement coopératif des requêtes sémantiques
[art -25] Kārlis Čerāns & al ,2015,Towards Graphical Query Notation for Semantic Databases Kārlis Čerāns & al
[art -26] Jānis Bārzdiņš, 2010, OWLGrEd: a UML Style Graphical Editor for OWL

Webographie

[web -1] **http://logic.stanford.edu/kif/dpans.html**, Knowledge Interchange Format, **1998**.
[web -2] **http://www.w3.org/TR/daml+oil-reference** ,DAML+OIL Reference Description. W3C Note 18 December **2001**.
[web -3] **http://www.w3.org/TR/rdf -schema/** ,Dan Brickley and R. V. Guha, Editors. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, 10 February **2004**
[web -4] **http://www.w3.org/TR/owl-guide/**,Michael K. Smith, Chris Welty, and Deborah L. McGuinness, Editors, OWL Web Ontology Language Guide,W3C Recommendation, 10 February **2004**
[web -5] **http://www.w3.org/TR/rdf-syntax-grammar/**, W3C, 25 February **2014**.
[web -6**] http://www.w3.org/TR/n-triples/**, W3C,9 January **2014**.
[web -7] **http://www.w3.org/TR/turtle/**, W3C,. 9 January **2014**
[web -8] **http://www.w3.org/TR/json-ld/**,W3C, 16 January **2014**
[web -9] **http://www.w3schools.com/xml/xml_rdf.asp**
[web -10] **http://www.w3.org/TR/rdf-schema/**
[web -11] **http://www-ksl.stanford.edu/software/jtp/doc/owl-reasoning.html**
[web -12] **https://www.cambridgesemantics.com/semantic-university/rdfs-vs-owl**
[web -13] **http://www.infowebml.ws/rdf-owl/DatatypeProperty.htm**
[web -14] **http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.3**
[web -15] **http://www.obitko.com/tutorials/ontologies-semantic-web/owl-dl-semantics.html**
[web -16] **http://webdam.inria.fr/Jorge/html/wdmch9.html**
[web -17] **http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/**
[web -18] **http://www.ercim.eu/publication/Ercim_News/enw51/alexaki.html**
[web -19] **http://www.w3.org/TR/rdf-sparql-query/**
[web -20] **http://www.w3.org/TR/rdf-sparql-query**/
[web -21] **http://www.w3.org/TR/sparql11-query/**
[web -22] **http://www.fil.univ-lille1.fr/~caronc/WS/sparqlPar4.pdf**
[web -23] **https://corese.inria.fr/srv/tutorial/rdf**
[web -24] **http://wimmics.inria.fr/soft**
[web -25] **http://www.w3.org/2009/Talks/0615-qbe/**
[web -26] **http://jplu.developpez.com/tutoriels/web-semantique/nouveautes-sparql-1-1/**
[web -27] **http://www.w3.org/TR/sparql11-property-paths/**
[web -28] **http://www.iro.umontreal.ca/~lapalme/ift6281/sparql-1_1-cheat-sheet.pdf**
[web -29] **http://www.w3.org/TR/rdf-sparql-query/#rOptionalGraphPattern**
[web -30] **http://opentox.org/data/documents/development/RDF%20files/JavaOnly/query-reasoning-with-jena-and-sparql**
[web -31] **http://vowl.visualdataweb.org/webvowl/index.html**
[web -32] **http://vowl.visualdataweb.org/queryvowl/v1/index.html#toc**
[web -33] **http://viziquer.lumii.lv/**
[web -34] **http://www.iro.umontreal.ca/~lapalme/ift6282/OWL/OWL-Syntaxe.pdf**
[web -35] **https://jena.apache.org**
[web -36] **https://jena.apache.org/documentation/inference/#owl**
[web -37] **http://wiki.dbpedia.org/about**
[web -38] **https://corese.inria.fr/srv/tutorial/rdf**
[web -39] **http://blog.clever-age.com/fr/2006/06/08/le-web-semantique-en-entreprise-comment-et-a-quels-niveaux/**
[web –40] **http://data.culture.fr/thesaurus/static/technos-web-semantique**

Annexes

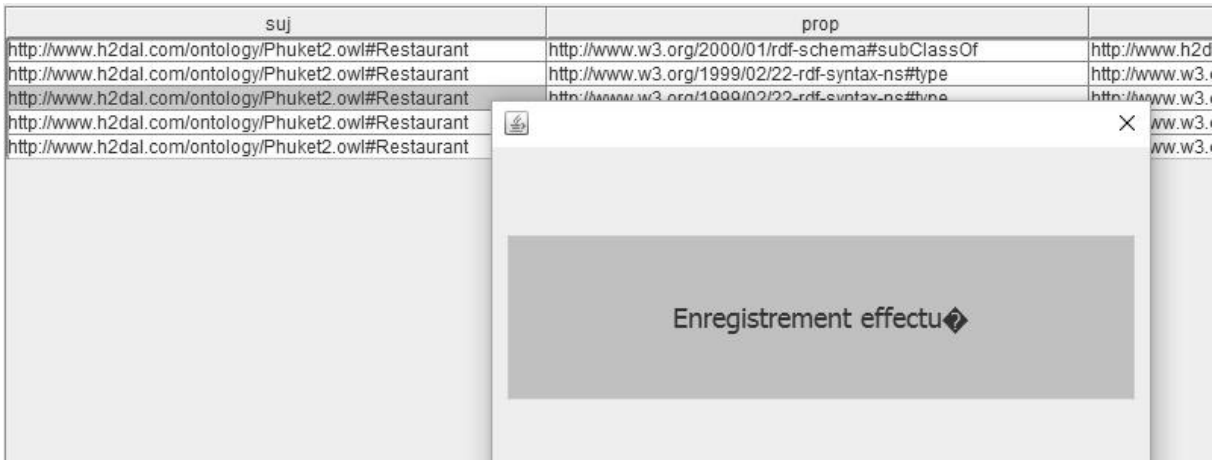
Annexe 1:

Queries based on the file ‘Phuket2.owl’

Which restaurant close on Monday ?

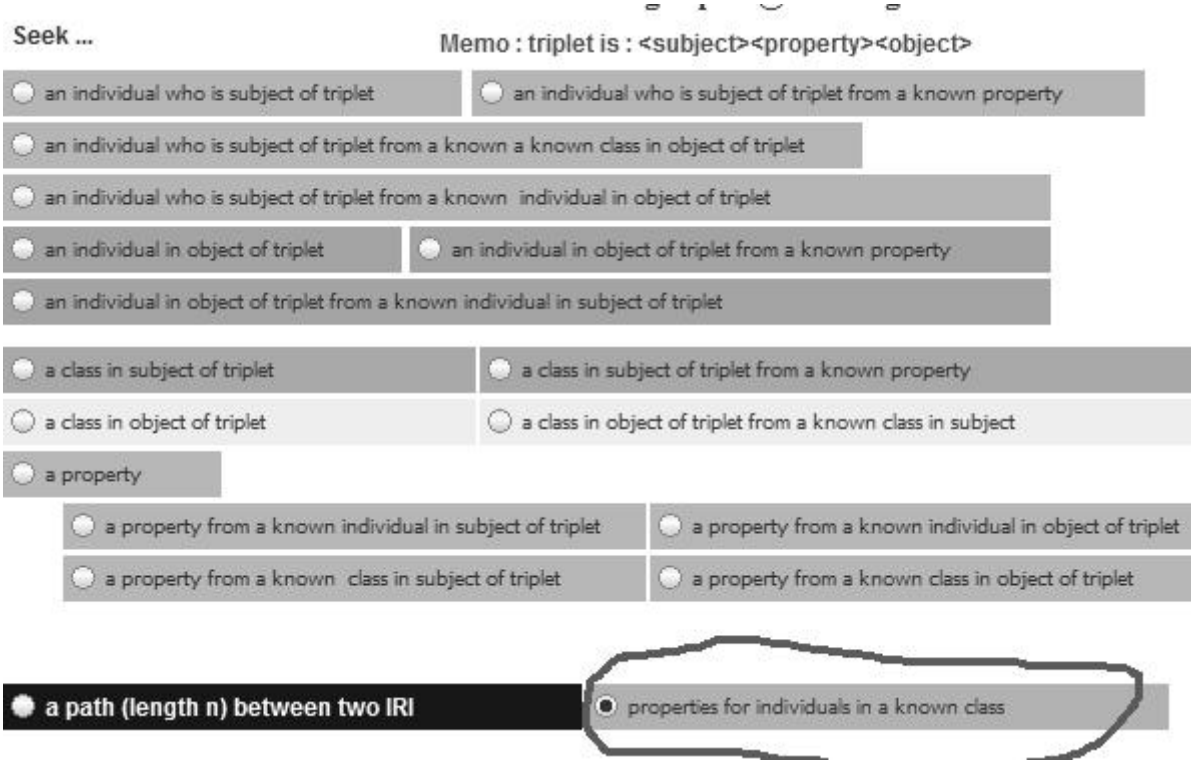
Step 1 : looking for the IRI of the restaurant

Panel search >> radio button ‘a class in subject of triple’ >> ok >> key word restaurant and ok



Then record the IRI in the list of classes.

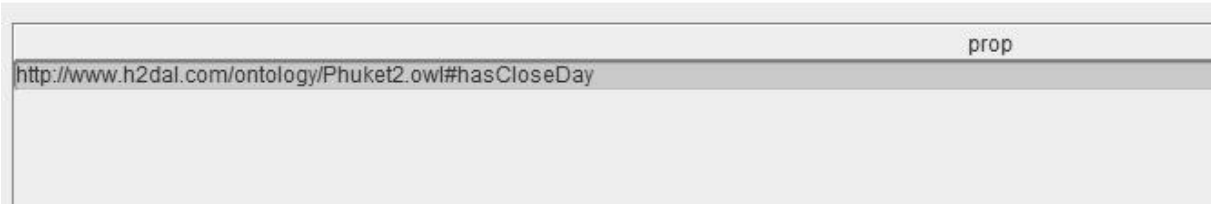
Step 2 : looking for a property which mention the idea of ‘close’ or ‘open’



'properties for individuals i a known class'>>ok

Then, choose the IRI of restaurant in the list and type 'close' in filter.

We have the following results :



We, then, record it in the list of properties and close the window.(we either could have a quicker solution by looking for a property with the keyword 'close'…possible here because of the ontology size).

Step 2: In the panel query /jonction, we type the variable suj, day >> record variables

Then, in the 1st list : - suj

In the 2nd list : IRI of property hascloseday

Int the 3rd list : - day + filter monday

Then, 'record part of temporary query', then record or execute query :

And we obtain the following results :

suj	day
http://www.h2dal.com/ontology/Phuket2.owl#Halal1	http://www.h2dal.com/ontology/Phuket2.owl#Monday

With the query :

select ?suj ?day WHERE { ?suj <http://www.h2dal.com/ontology/Phuket2.owl#hasCloseDay> ?day . FILTER regex(str(?day),"monday","i") .}

We already can see that the user dexterity to retrieve concept and to organize his query construct is usefull too. The software is generic, usefull to discover the conceptual structure of any ontology by instinctiv immersion and retrieving easyli the IRI and their concerned properties.

Which restaurant close on tuesday ?

Same method as 3 but with a filter 'tuesday'. (remember if you don't close the panel 'query/jonction', it may be some residuals queries…to avoid that, the remove buttons are here.)

By example, if you don't close your window the last time,

'remove parts of temporary query' >> then change the filter and type tuesday>>' record part of temporary query' then execute query (there is no result for this time of construct of this ontology).

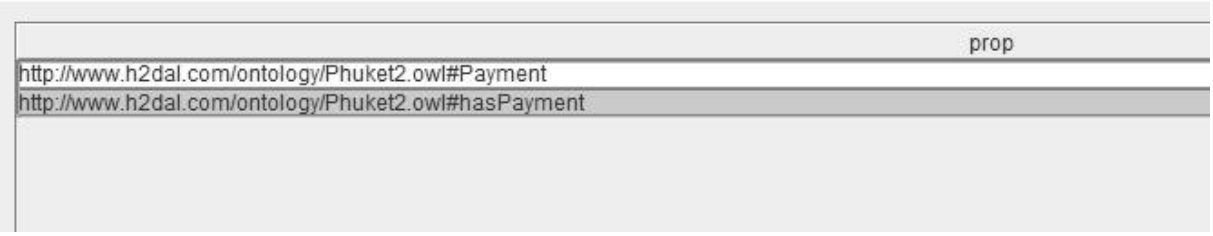
The generated query is:

select ?suj ?day WHERE { ?suj <http://www.h2dal.com/ontology/Phuket2.owl#hasCloseDay> ?day . FILTER regex(str(?day),"tuesday","i") .}

Which restaurant I can pay by CreditCard

IRI to look for :

- Restaurant (we already have it)
- Payment property : panel search >> radio button 'property'>>ok>>filter 'pay'



prop
http://www.h2dal.com/ontology/Phuket2.owl#Payment
http://www.h2dal.com/ontology/Phuket2.owl#hasPayment

Select the correct IRI, then record in property list. Close the window.

Then, go to the panel Query/jonction

Variable suj, payment >>record variables

Refresh

List 1 : suj

List 2 : IRI haspayment

List 3: variable payment + filtre ‘credit’

‘record of temporary query’>>the record/execute

We get :

suj	payment
http://www.h2dal.com/ontology/Phuket2.owl#Halal1	http://www.h2dal.com/ontology/Phuket2.owl#CreditCard
http://www.h2dal.com/ontology/Phuket2.owl#Flying1	http://www.h2dal.com/ontology/Phuket2.owl#CreditCard
http://www.h2dal.com/ontology/Phuket2.owl#Bungy1	http://www.h2dal.com/ontology/Phuket2.owl#CreditCard

with the SPARQL query :

```
select ?suj ?payment WHERE { ?suj  <http://www.h2dal.com/ontology/Phuket2.owl#hasPayment>  ?payment .
FILTER regex(str(?payment),"credit","i") .}
```

How many district in Phuket and the name?

That question will need the agregate functionnality of our software.

- Step 1 : we look for the IRI of the class ‘District’

  Panel search >> a class in subject of triplet >> OK >>filter : “district”

  Then, we choose and record the correct IRI.

suj	prop	
http://www.h2dal.com/ontology/Phuket2.owl#District	http://www.w3.org/2000/01/rdf-schema#domain	http://www.h2dal.c
http://www.h2dal.com/ontology/Phuket2.owl#District	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.h2dal.c
http://www.h2dal.com/ontology/Phuket2.owl#District	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/
http://www.h2dal.com/ontology/Phuket2.owl#District	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/
http://www.h2dal.com/ontology/Phuket2.owl#District	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/
http://www.h2dal.com/ontology/Phuket2.owl#District	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/
http://www.h2dal.com/ontology/Phuket2.owl#District	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/
http://www.h2dal.com/ontology/Phuket2.owl#District	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/
http://www.h2dal.com/ontology/Phuket2.owl#District	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/

- Step 2 : we constitute the simple query of finding the distinct elements in this class

  Panel query >> variable dist >> ‘record variable’

  Refresh combolists.

  List 1 : variable dist

List 2 : IRI of #type :

List 3 : IRIof district

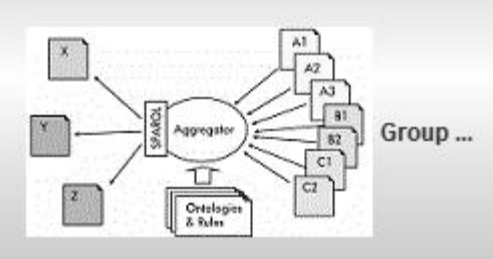
‘record part of temporary query’

‘record in global query’ (that will be usefull to make the agregation)

If we launch the query, we obtain :

dist
http://www.h2dal.com/ontology/Phuket2.owl#Thalang
http://www.h2dal.com/ontology/Phuket2.owl#Mueang
http://www.h2dal.com/ontology/Phuket2.owl#Kathu

- Step 3: we agregate the precedent query in the agregation panel



Then, a list is proposed. Choose the precedent query and ‘construct aggregate query’

Choose select COUNT dist.

Then select ‘ok group’ button.-without checking the ‘group by’ clause, because we don’t need it here.

Then ‘execute query’ :

COUNT_dist
LITTERAL 3^^http://www.w3.org/2001/XMLSchema#integer

The resulting sparql query is :

SELECT (COUNT(?dist) AS ?COUNT_dist) WHERE { ?dist <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.h2dal.com/ontology/Phuket2.owl#District> .}

Which beach I can play water activity?

This query is a typical intersection query between :

?place is_a beach

?place has_activity ?activity

?activity is_a water_activity

So, to do that we need the IRI of the concept of the class 'Beach', the IRI of the concept of the property has_activity relied to a beach and the IRI of the class 'water activity'.

IRI of the 'beach' :

Panel search >> a class in subject of triplet >> ok >> enter keyword beach >> launch query >>

suj	prop	im
http://www.h2dal.com/ontology/Phuket2.owl#Beach&Bay	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.h2dal.com/ontology/Ph
http://www.h2dal.com/ontology/Phuket2.owl#Beach&Bay	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.h2dal.com/ontology/Ph
http://www.h2dal.com/ontology/Phuket2.owl#Beach&Bay	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.h2dal.com/ontology/Ph
http://www.h2dal.com/ontology/Phuket2.owl#Beach&Bay	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-sch
http://www.h2dal.com/ontology/Phuket2.owl#Beach&Bay	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-sch
http://www.h2dal.com/ontology/Phuket2.owl#Beach&Bay	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-sch
http://www.h2dal.com/ontology/Phuket2.owl#Beach&Bay	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#C

Select the right IRI and record it in classes. (classes >>add in list)

Close the window, then return to the 'search panel'.

Choose : 'property for individuals in a known class' >> ok

Then, choose the class you recorded lately in the list and 'launch query'

prop
http://www.h2dal.com/ontology/Phuket2.owl#Latitude
http://www.h2dal.com/ontology/Phuket2.owl#Longtitude
http://www.h2dal.com/ontology/Phuket2.owl#Name
http://www.h2dal.com/ontology/Phuket2.owl#PathPic
http://www.h2dal.com/ontology/Phuket2.owl#hasActivity
http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict
http://www.h2dal.com/ontology/Phuket2.owl#hasRating
http://www.w3.org/1999/02/22-rdf-syntax-ns#type

IRI 'hasActivity' seems to be a good candidate. Choose it and record it in the property list.

Then, close the window and go to the 'search panel' again to find the class relative to the water activities.

A class in subject of triplet >> ok >> enter the keyword 'activity' by example, then 'launch query'.

http://www.h2dal.com/ontology/Phuket2.owl#WaterActivity	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://w
http://www.h2dal.com/ontology/Phuket2.owl#WaterActivity	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://w

Select the good IRI, record it in the classes, then close the window.

We have all IRIs we need, we go now in the Query and Junction panel.

Then enter variable ‘place’>>record variable button>> enter variable ‘activity’>>record variable button

Refresh comboListes button >>

List1 : choose variable ‘place’

List2: choose IRI of the #type

List3: choose IRI of Beach and Bays

Button ‘record part of temporary query’

Now,

List1 : choose variable ‘place’

List2 : choose IRI of the’ activity’ property

List3: choose the variable ‘activity’

Button ‘record part of temporary query’

And to finish :

List1 : choose variable ‘activity’

List2: choose IRI of the #type

List3: choose the IRI of the class ‘water_activity’

Button ‘record part of temporary query’

The query is ready, you can record it /or to execute it .

We obtain :

place	activity
http://www.h2dal.com/ontology/Phuket2.owl#Beach2	http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingB
http://www.h2dal.com/ontology/Phuket2.owl#Beach1	http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingA
http://www.h2dal.com/ontology/Phuket2.owl#Beach1	http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingB

And the SPARQL query is :

```
select ?place ?activity WHERE { ?place  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.h2dal.com/ontology/Phuket2.owl#Beach&Bay>  . ?place
<http://www.h2dal.com/ontology/Phuket2.owl#hasActivity>  ?activity  . ?activity
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.h2dal.com/ontology/Phuket2.owl#WaterActivity>  .}
```

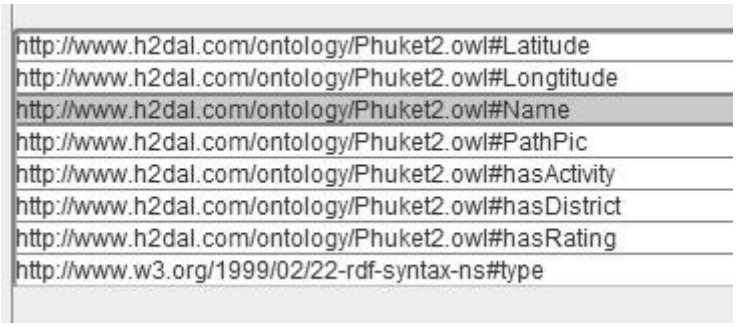
Which sport I can play at Kamala Beach

The idea is :

- a) Looking for the IRI who give the name of a beach.

  - Looking for the IRI of class beach
  - With the search panel ‘property for individuals in a known class

We get :



We choose the #name property and record it.

We profit to choose the property #hasActivity (that is the answer of the step c)) too.

-b) Looking for the beach whose name is ‘kamala’

-c) Looking for the property who designed a sport activity.(or the nearest notion).: see a)

-d) Panel Query/junction>>variables plage and name

Refresh lists>>

List 1: var plage

List 2 : #type

List 3 : IRI of beach

Button record part of temporary query

Then

List 1: var plage

List 2 : IRI #name

List 3 : var name + filtre ‘kamala’

Button record part of temporary query

Execute query :

plage	
http://www.h2dal.com/ontology/Phuket2.owl#Beach2	***LITTERAL*** Kamala Beach

We record the #Beach2 IRI in the individuals.

-e) Last step :

In the Query Panel:

Variable act

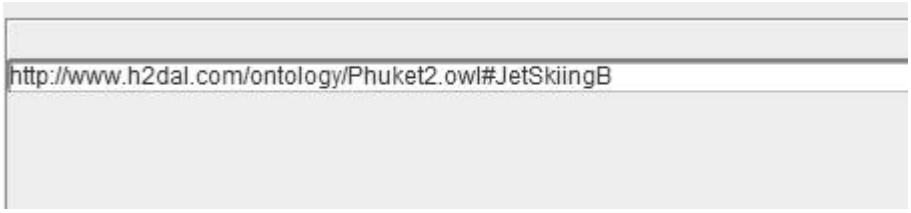
Refresh ComboList :

List1: IRI #Beach2

List2: IRI #hasActivity

List3: variable act

We get :



Can I watch Bangla BoxingStadium Patong on Saturday?

- Look for the IRI class relative to the boxe
- Look for the IRI name property of the individuals belonging to the class Boxe
- With the help of panel Query/junction Looking for the IRI entity whose name contains ‘patong’

We realise :

```
select ?box ?name WHERE { ?box  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.h2dal.com/ontology/Phuket2.owl#Boxing>  . ?box
<http://www.h2dal.com/ontology/Phuket2.owl#Name>  ?name . FILTER
regex(str(?name),"patong","i") .}
```

That give the result :

box	name
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	***LITTERAL*** Bangla BoxingStadium Patong

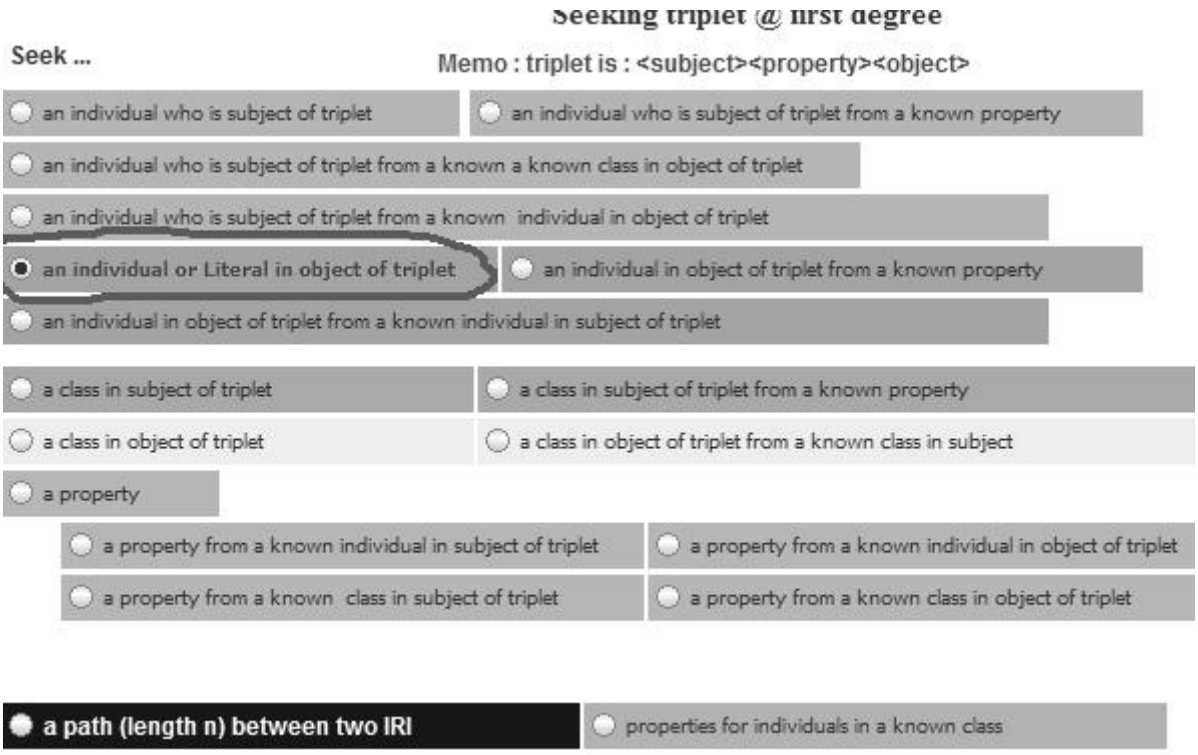
We record the entity #boxing1 .

Then, on the same window, with the button ‘detail on selection’ we obtain :

ttp://www.h2dal.com/ontology/Phuket2.owl#Latitude	***LITTERAL*** 7.891495
ttp://www.h2dal.com/ontology/Phuket2.owl#Longtitude	***LITTERAL*** 98.301756
ttp://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** Bangla BoxingStadium Patong
ttp://www.h2dal.com/ontology/Phuket2.owl#StartDateTime	***LITTERAL*** 2016-03-29T21.00
ttp://www.h2dal.com/ontology/Phuket2.owl#Street	***LITTERAL*** Ratutit
ttp://www.h2dal.com/ontology/Phuket2.owl#Telephone	***LITTERAL*** +66897261112
ttp://www.h2dal.com/ontology/Phuket2.owl#hasCloseDay	http://www.h2dal.com/ontology/Phuket2.owl#Monday
ttp://www.h2dal.com/ontology/Phuket2.owl#hasCloseDay	http://www.h2dal.com/ontology/Phuket2.owl#Saturday
ttp://www.h2dal.com/ontology/Phuket2.owl#hasCloseDay	http://www.h2dal.com/ontology/Phuket2.owl#Thursday
ttp://www.h2dal.com/ontology/Phuket2.owl#hasCloseDay	http://www.h2dal.com/ontology/Phuket2.owl#Tuesday
ttp://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Kathu
ttp://www.h2dal.com/ontology/Phuket2.owl#hasPayment	http://www.h2dal.com/ontology/Phuket2.owl#CreditCard
ttp://www.h2dal.com/ontology/Phuket2.owl#hasRating	http://www.h2dal.com/ontology/Phuket2.owl#5star
ttp://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Activity
ttp://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Attraction
ttp://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Boxing
ttp://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#DateTime

We can see that Saturday is in list of Closing days.

Another method, the shortest is to choose the option “individual or Literal in object of triplet in the Search panel :



Then, enter the keyword ‘Bangla’ in the textfield.

suj	prop	img
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	http://www.h2dal.com/ontology/Phuket2.owl#Content	***LITTERAL*** Choice of one of boxing. Bangl
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	http://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** Bangla BoxingStadium Patong
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Real filght, Stadium, Midnight, E
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	http://www.h2dal.com/ontology/Phuket2.owl#Email	***LITTERAL*** banglaboxing@gmail.com

Choose the #boxing1 then 'detail on selection' reveals you all the informations.

Which activity has 4 or 5 star? (search from hasRating)

- Look for the IRI of class Activity
- Look for the IRI property hasRating of the individuals belonging at this class
- Ideally, to filter on Literals, the ontology must mention a int literal (*int+string like "3star" is not a standart defined type*) to design the number of stars and to allow the query to filter like that by example : FILTER (?star > "3"^^xsd:integer) .} Here it is not possible .

So the approximated solution is with the query panel :

We can't use the literal filter on the image star for the reasons we evocated before.

On obtient la requête sparql :

select ?act ?star WHERE { ?act <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.h2dal.com/ontology/Phuket2.owl#Activity> . ?act
<http://www.h2dal.com/ontology/Phuket2.owl#hasRating> ?star .}

et le résultat :

act	star
http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingB	http://www.h2dal.com/ontology/Phuket2.owl#4star
http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingA	http://www.h2dal.com/ontology/Phuket2.owl#3star
http://www.h2dal.com/ontology/Phuket2.owl#Flying1	http://www.h2dal.com/ontology/Phuket2.owl#5star
http://www.h2dal.com/ontology/Phuket2.owl#Bungy1	http://www.h2dal.com/ontology/Phuket2.owl#4star
http://www.h2dal.com/ontology/Phuket2.owl#Zoo	http://www.h2dal.com/ontology/Phuket2.owl#4star
http://www.h2dal.com/ontology/Phuket2.owl#Halal1	http://www.h2dal.com/ontology/Phuket2.owl#4star
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	http://www.h2dal.com/ontology/Phuket2.owl#5star
http://www.h2dal.com/ontology/Phuket2.owl#Beach1	http://www.h2dal.com/ontology/Phuket2.owl#5star
http://www.h2dal.com/ontology/Phuket2.owl#Theater1	http://www.h2dal.com/ontology/Phuket2.owl#5star
http://www.h2dal.com/ontology/Phuket2.owl#Beach2	http://www.h2dal.com/ontology/Phuket2.owl#3star
http://www.h2dal.com/ontology/Phuket2.owl#Theater2	http://www.h2dal.com/ontology/Phuket2.owl#3star

We could apply a filter 4 or 5 on the field star.

To obtain a first query for the 4 stars :

select ?act ?star WHERE { ?act <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.h2dal.com/ontology/Phuket2.owl#Activity> . ?act
<http://www.h2dal.com/ontology/Phuket2.owl#hasRating> ?star . FILTER regex(str(?star),"4","i") .}

and another one for the 5 stars

and to make the union with the panel UNION:

and we obtain :

SELECT ?act ?star WHERE {{select ?act ?star WHERE { ?act <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.h2dal.com/ontology/Phuket2.owl#Activity> . ?act <http://www.h2dal.com/ontology/Phuket2.owl#hasRating> ?star . FILTER regex(str(?star),"4","i") .}} UNION {select ?act ?star WHERE { ?act <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.h2dal.com/ontology/Phuket2.owl#Activity> . ?act <http://www.h2dal.com/ontology/Phuket2.owl#hasRating> ?star . FILTER regex(str(?star),"5","i") .}}}

This could have been obtained too without union with simply the regex x|y

select ?act ?star WHERE { ?act <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.h2dal.com/ontology/Phuket2.owl#Activity> . ?act <http://www.h2dal.com/ontology/Phuket2.owl#hasRating> ?star . FILTER regex(str(?star),"4|5","i") .}

act	star
http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingB	http://www.h2dal.com/ontology/Phuket2.owl#4star
http://www.h2dal.com/ontology/Phuket2.owl#Bungy1	http://www.h2dal.com/ontology/Phuket2.owl#4star
http://www.h2dal.com/ontology/Phuket2.owl#Zoo	http://www.h2dal.com/ontology/Phuket2.owl#4star
http://www.h2dal.com/ontology/Phuket2.owl#Halal1	http://www.h2dal.com/ontology/Phuket2.owl#4star
http://www.h2dal.com/ontology/Phuket2.owl#Flying1	http://www.h2dal.com/ontology/Phuket2.owl#5star
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	http://www.h2dal.com/ontology/Phuket2.owl#5star
http://www.h2dal.com/ontology/Phuket2.owl#Beach1	http://www.h2dal.com/ontology/Phuket2.owl#5star
http://www.h2dal.com/ontology/Phuket2.owl#Theater1	http://www.h2dal.com/ontology/Phuket2.owl#5star

Which activity pay over 1000 THB?

Here the price must be a type double. (for the moment, in the ontology the type is a non standart one)

A solution is to have a property has_price and another one has_unity_of_money by example.

August is high season?

Panel search : individual who is subject of triplet >> ok

Type ‘season’ in textfield filter

We obtain :

suj
http://www.h2dal.com/ontology/Phuket2.owl#HighSeason
http://www.h2dal.com/ontology/Phuket2.owl#LowSeason

We choose the #highseason and push the button “detail on selection”

We have the results :

prop	ling
http://www.h2dal.com/ontology/Phuket2.owl#MonthNumber	***LITTERAL*** 9,10,11,12,1,2,3,4
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#DateTime
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Month
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Season
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual

We have the answer and see that High Season doesn’t contains the 8th month.(august).

Which month is High season?

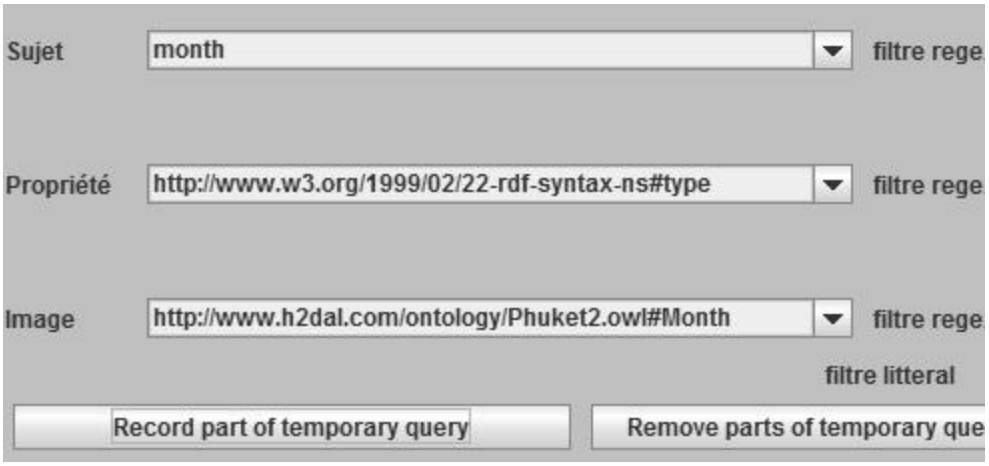
The precedent answer give the 9,10,11,12,1,2,3,4

It is then possible to realise a search on the class ‘month’, on the property of the individuals belonging to this class :

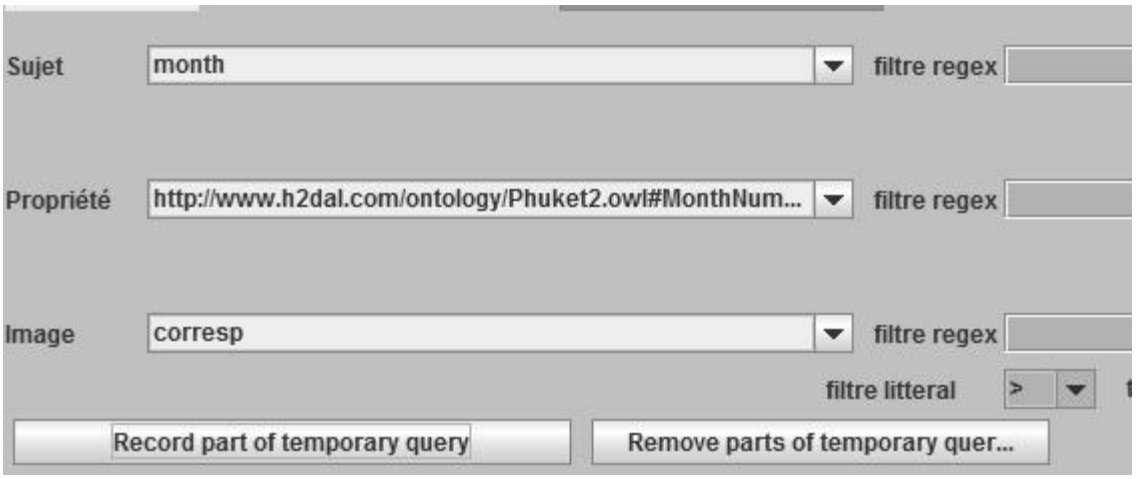
http://www.h2dal.com/ontology/Phuket2.owl#Latitude
http://www.h2dal.com/ontology/Phuket2.owl#Longtitude
http://www.h2dal.com/ontology/Phuket2.owl#MonthNumber
http://www.h2dal.com/ontology/Phuket2.owl#Name
http://www.h2dal.com/ontology/Phuket2.owl#NoMonth
http://www.h2dal.com/ontology/Phuket2.owl#PersonContact
http://www.h2dal.com/ontology/Phuket2.owl#PhotoDescription

and to realise the correspondance with the query/junction panel :

with variables ‘corresp’, ‘month’



and



SPARQL is then :

select ?corresp ?month WHERE { ?month <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.h2dal.com/ontology/Phuket2.owl#Month> . ?month <http://www.h2dal.com/ontology/Phuket2.owl#MonthNumber> ?corresp .}

and result :

corresp	month
"**LITTERAL*** 3	http://www.h2dal.com/ontology/Phuket2.owl#March
"**LITTERAL*** 1	http://www.h2dal.com/ontology/Phuket2.owl#January
"**LITTERAL*** 9	http://www.h2dal.com/ontology/Phuket2.owl#September
"**LITTERAL*** 10	http://www.h2dal.com/ontology/Phuket2.owl#October
"**LITTERAL*** 8	http://www.h2dal.com/ontology/Phuket2.owl#August
"**LITTERAL*** 5	http://www.h2dal.com/ontology/Phuket2.owl#May
"**LITTERAL*** 4	http://www.h2dal.com/ontology/Phuket2.owl#April
"**LITTERAL*** 12	http://www.h2dal.com/ontology/Phuket2.owl#December
"**LITTERAL*** 2	http://www.h2dal.com/ontology/Phuket2.owl#February
"**LITTERAL*** 7	http://www.h2dal.com/ontology/Phuket2.owl#July
"**LITTERAL*** 6	http://www.h2dal.com/ontology/Phuket2.owl#June
"**LITTERAL*** 11	http://www.h2dal.com/ontology/Phuket2.owl#November
"**LITTERAL*** 5,6,7,8	http://www.h2dal.com/ontology/Phuket2.owl#LowSeason
"**LITTERAL*** 9,10,11,12,1,2,3,4	http://www.h2dal.com/ontology/Phuket2.owl#HighSeason

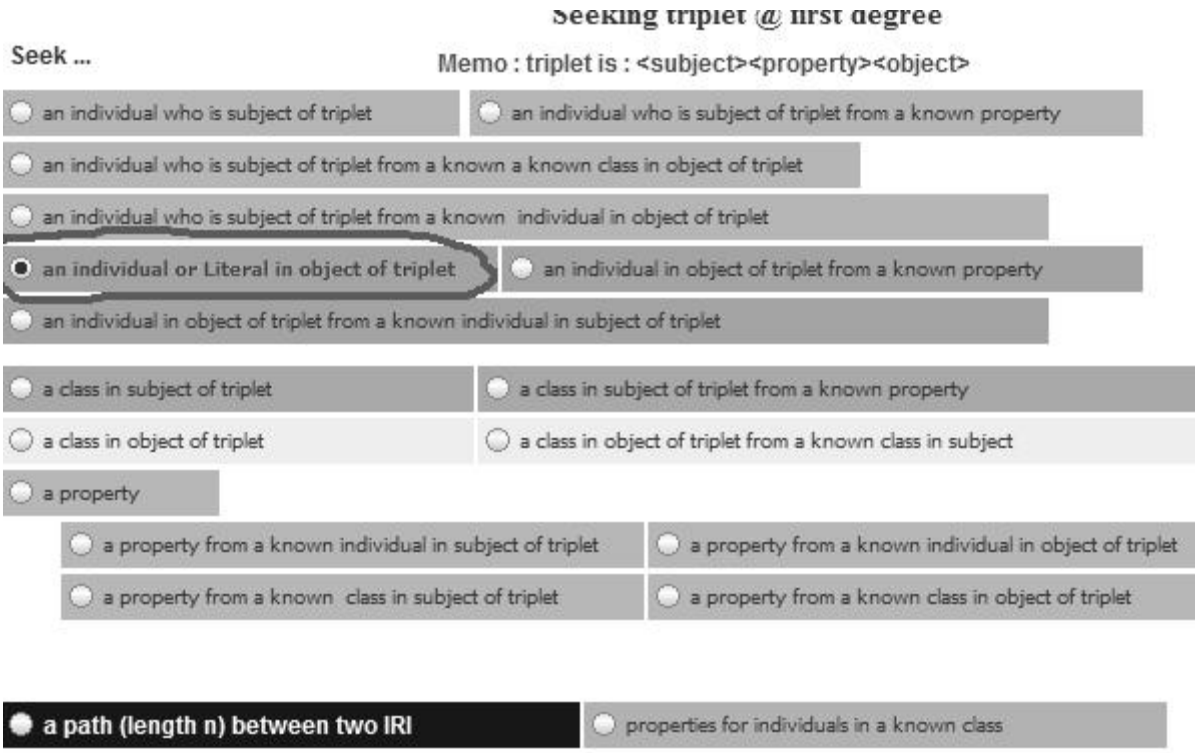
Suggest the shop for Art&Craft

Look for the IRI of class ‘art&craft’, then with panl query :

select ?suggestion WHERE { ?suggestion <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.h2dal.com/ontology/Phuket2.owl#Art&Craft> .}

suggestion
http://www.h2dal.com/ontology/Phuket2.owl#JimThomson

Can I pay money with MasterCard for movie?



Type ‘movie’ in the textfield.

suj	prop	img
http://www.h2dal.com/ontology/Phuket2.owl#Theater1	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Central Phuket, Movie, Popcorn, C
http://www.h2dal.com/ontology/Phuket2.owl#Theater2	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Movie, Popcorn, Cinema

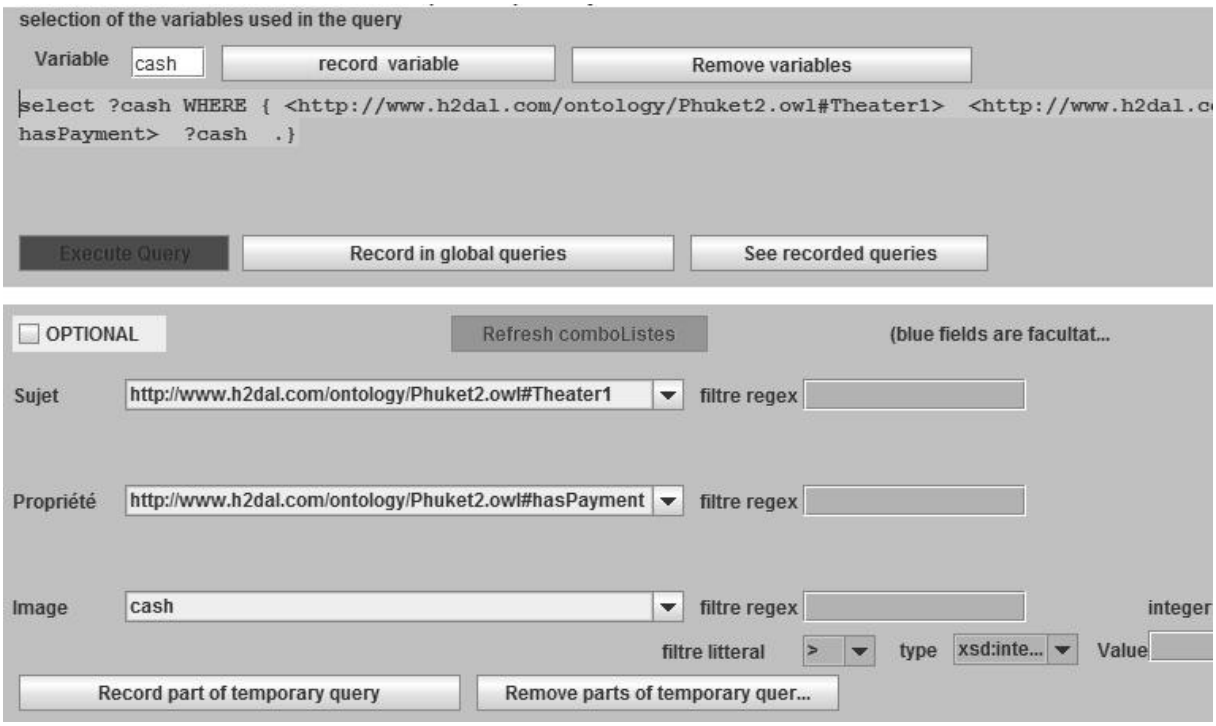
’movie’ concerns two individuals : #Theater1 and #Theater2

We record the IRIs.

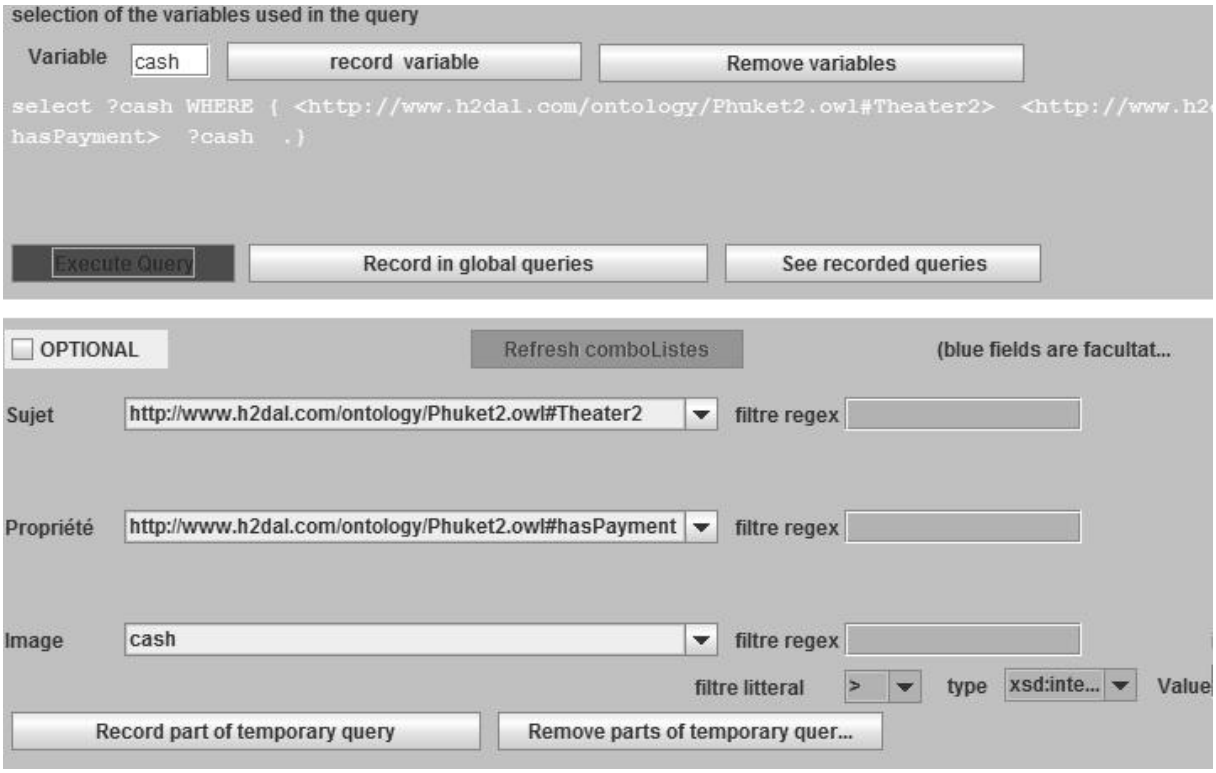
Then, we record the IRI of #hasPayment.(easily findable with the ‘details on selection’ button).

Then, go in the Query Panel :

Constitute the first query and record it.



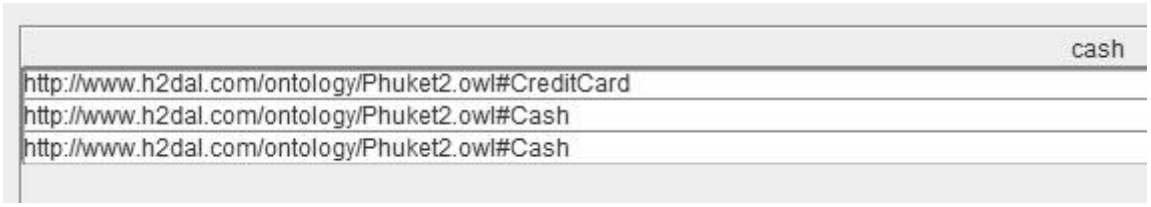
The same for theater 2 :



record the query too.

Then go to the union panel :

And execute query :



We get all possible payments to see a movie.

Which theater I can pay by cash or creditcard?

We make the search with the ‘theater’ term.

- Look for the IRI of class theater
- Look for the properties relative to the individuals belonging to this class and choice of the #haspayment IRI
- With the query panel we’ve got :
- select ?th ?pay WHERE { ?th <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.h2dal.com/ontology/Phuket2.owl#Theater> . ?th <http://www.h2dal.com/ontology/Phuket2.owl#hasPayment> ?pay .}

and the result :

th	pay
http://www.h2dal.com/ontology/Phuket2.owl#Theater2	http://www.h2dal.com/ontology/Phuket2.owl#Cash
http://www.h2dal.com/ontology/Phuket2.owl#Theater1	http://www.h2dal.com/ontology/Phuket2.owl#CreditCard
http://www.h2dal.com/ontology/Phuket2.owl#Theater1	http://www.h2dal.com/ontology/Phuket2.owl#Cash

Theater1 can be paid with Credit Card.

Which waterfall is in Kathu?

select ?waterf ?name WHERE { ?waterf <http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict> <http://www.h2dal.com/ontology/Phuket2.owl#Kathu> . ?waterf <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.h2dal.com/ontology/Phuket2.owl#Waterfall> . ?waterf <http://www.h2dal.com/ontology/Phuket2.owl#Name> ?name .}

results :

waterf	name
http://www.h2dal.com/ontology/Phuket2.owl#Waterfall1	***LITTERAL*** Kathu waterfall

Which restaurant open daily?

We must find the restaurant which not have close days.

So, the list of restaurants MINUS the restaurant who have a closing day.

- Looking for the IRI of class restaurant
- Looking for the IRI of property relative to the closing days
- Realise the query to find all the restaurant
- Realise the query to find all the restaurant with a closing day.
- In the panel 'UNION', realise the 'MINUS' of the two precedent queries.

We have the following results :

rest
http://www.h2dal.com/ontology/Phuket2.owl#Zoo
http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingA
http://www.h2dal.com/ontology/Phuket2.owl#FUJI3
http://www.h2dal.com/ontology/Phuket2.owl#FUJI2
http://www.h2dal.com/ontology/Phuket2.owl#Bungy1
http://www.h2dal.com/ontology/Phuket2.owl#Boxing2
http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingB
http://www.h2dal.com/ontology/Phuket2.owl#Aquarium
http://www.h2dal.com/ontology/Phuket2.owl#Beach1
http://www.h2dal.com/ontology/Phuket2.owl#Theater1
http://www.h2dal.com/ontology/Phuket2.owl#ChineseNewYear
http://www.h2dal.com/ontology/Phuket2.owl#Zen
http://www.h2dal.com/ontology/Phuket2.owl#Beach2
http://www.h2dal.com/ontology/Phuket2.owl#Theater2
http://www.h2dal.com/ontology/Phuket2.owl#FUJI1
http://www.h2dal.com/ontology/Phuket2.owl#Waterfall1
http://www.h2dal.com/ontology/Phuket2.owl#ShopB
http://www.h2dal.com/ontology/Phuket2.owl#ShopA
http://www.h2dal.com/ontology/Phuket2.owl#ChaoLayFestival
http://www.h2dal.com/ontology/Phuket2.owl#ChineseNewYear1
http://www.h2dal.com/ontology/Phuket2.owl#Halal2

(for the present ontology, item like #zoo seems to be classified with the inference in the restaurants…this ontology is for the moment in construct mode, so can have sometimes inconsistancies (see soft protégé and reasonner mode)

Which restaurant open at 10.00

Need the datatype in the ontology to be standard :

http://www.w3.org/2001/XMLSchema#time ?

see: https://www.w3.org/TR/rif-dtb/

https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/#isoformats

or property has_open_hour xsd:integer

Which shop has email and/or websiteURL

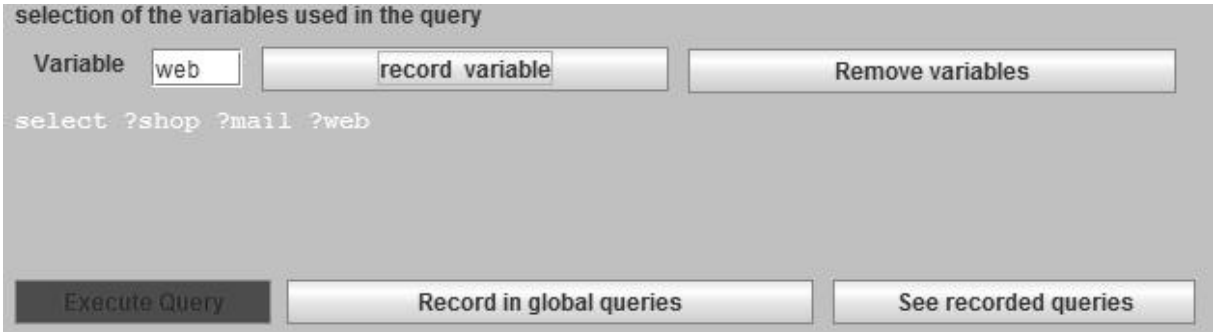
Panel ‘search’ >> seek a class in subject of triplet :

Choose #shopping and record classe.

Panel search >> seek properties for individuals in a known class

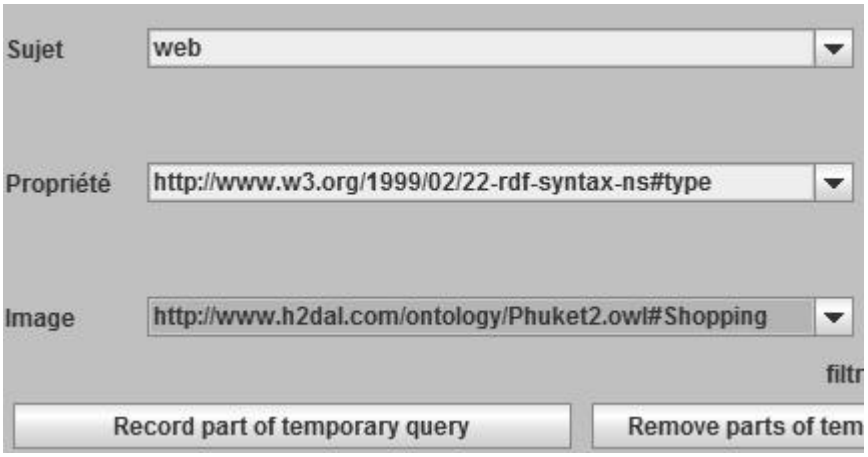
Choose #email and websiteURL and record in property list.

Now, 



Choose shop, mail & web variables.

1st part of query :



Record part of temporary query…

2nd part of query :

Record part of temporary query…

3rd part of query :

Record part of temporary query…then execute or record/modify the SPARQL query.

You have results :

shop	mail	
http://www.h2dal.com/ontology/Phuket2.owl#Zoo	***LITTERAL*** phuketzoo@hotmail.co.th	***LITTERAL*** www.phuketzoo.com
http://www.h2dal.com/ontology/Phuket2.owl#Halal1	***LITTERAL*** aa@gmail.com	***LITTERAL*** www.halal1.com
http://www.h2dal.com/ontology/Phuket2.owl#Boxing2	***LITTERAL*** camp@sinbi-muaythai.com	***LITTERAL*** http://www.sinbi-muaythai.co

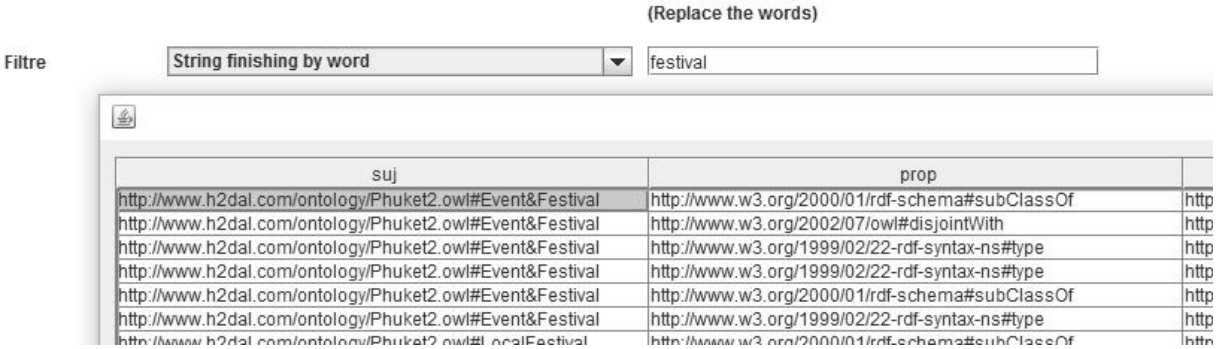
with the sparql :

select ?shop ?mail ?web WHERE { ?shop <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.h2dal.com/ontology/Phuket2.owl#Shopping> . ?shop
<http://www.h2dal.com/ontology/Phuket2.owl#Email> ?mail . ?shop
<http://www.h2dal.com/ontology/Phuket2.owl#WebsiteURL> ?web .}

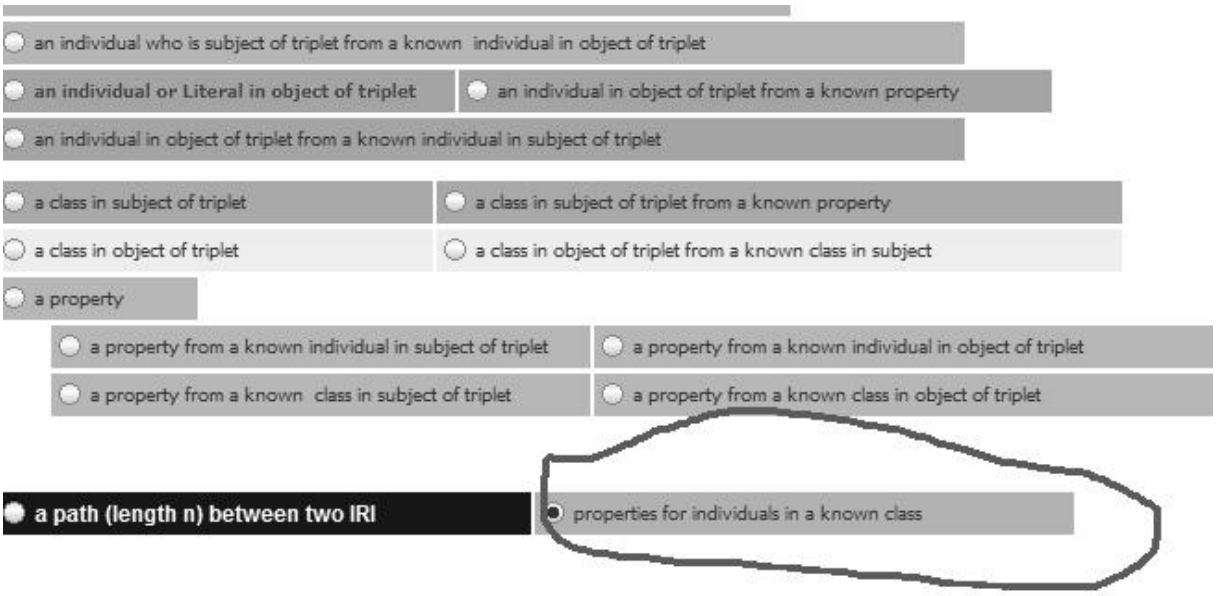
Which festival held in Thalang and When is it held?

Retrieve the IRI concept of Festival (in the class subject of triplet in the panel research)

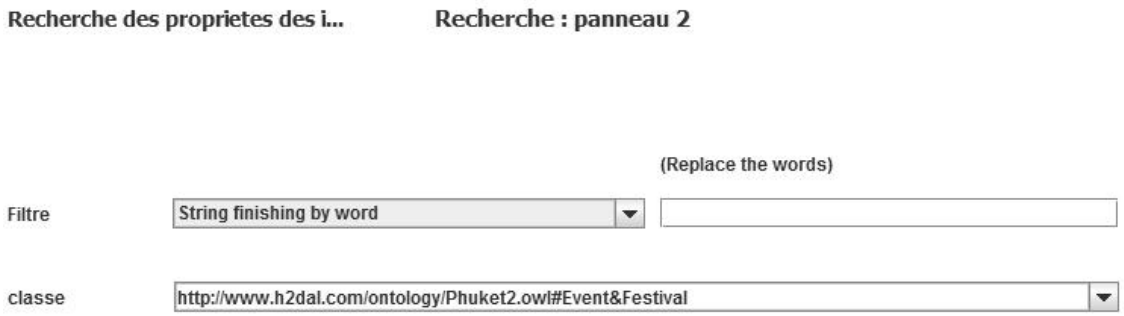
Sparql : PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl:
<http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd:
<http://www.w3.org/2001/XMLSchema#> SELECT DISTINCT ?suj ?prop ?img WHERE { ?suj a owl:Class .
?suj ?prop ?img . FILTER regex(str(?suj) ,"festival", "i") } ORDER BY ?suj LIMIT 100



Choose and record in class list.



To retrieve the properties of the entities belonging to the #festival class.

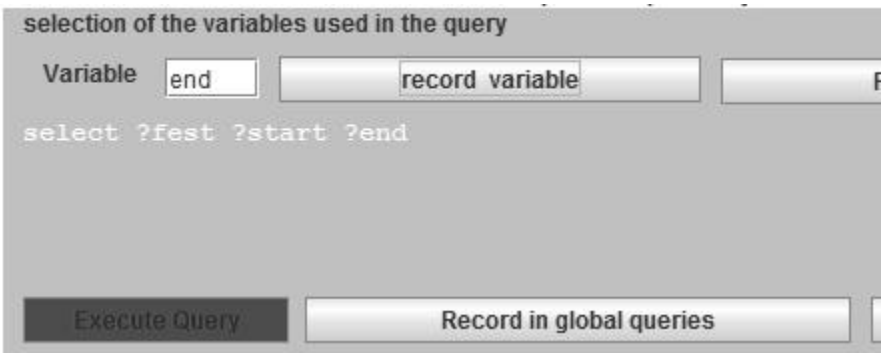


PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> SELECT DISTINCT ?prop WHERE { ?x a <http://www.h2dal.com/ontology/Phuket2.owl#Event&Festival> . ?x ?prop ?img. FILTER regex(str(?prop) ,"", "i") } ORDER BY ?prop LIMIT 100

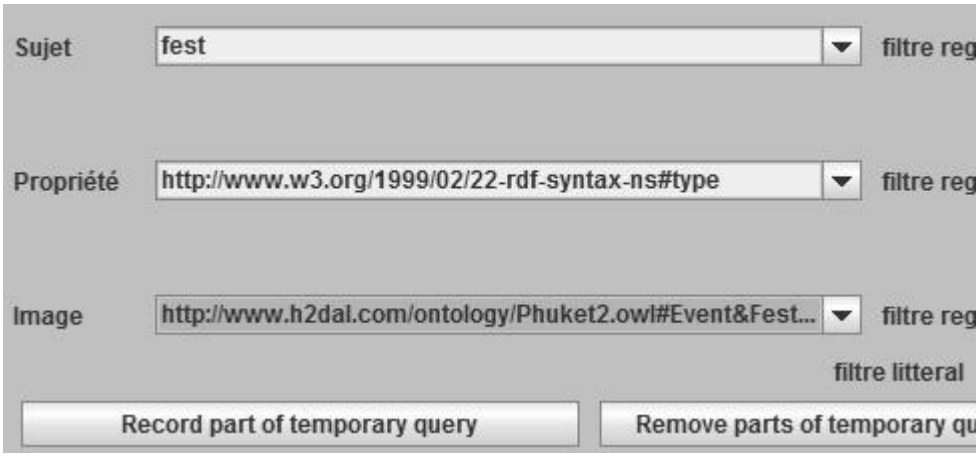
http://www.h2dal.com/ontology/Phuket2.owl#Address
http://www.h2dal.com/ontology/Phuket2.owl#Content
http://www.h2dal.com/ontology/Phuket2.owl#Email
http://www.h2dal.com/ontology/Phuket2.owl#EndDateTime
http://www.h2dal.com/ontology/Phuket2.owl#EndPeriod
http://www.h2dal.com/ontology/Phuket2.owl#Fax
http://www.h2dal.com/ontology/Phuket2.owl#Keyword
http://www.h2dal.com/ontology/Phuket2.owl#Latitude
http://www.h2dal.com/ontology/Phuket2.owl#Longtitude
http://www.h2dal.com/ontology/Phuket2.owl#Name
http://www.h2dal.com/ontology/Phuket2.owl#NoMonth
http://www.h2dal.com/ontology/Phuket2.owl#PathPic
http://www.h2dal.com/ontology/Phuket2.owl#PersonContact
http://www.h2dal.com/ontology/Phuket2.owl#PhotoDescription
http://www.h2dal.com/ontology/Phuket2.owl#Price
http://www.h2dal.com/ontology/Phuket2.owl#SpecialRequirement
http://www.h2dal.com/ontology/Phuket2.owl#StartDateTime
http://www.h2dal.com/ontology/Phuket2.owl#StartPeriod
http://www.h2dal.com/ontology/Phuket2.owl#Street
http://www.h2dal.com/ontology/Phuket2.owl#Telephone
http://www.h2dal.com/ontology/Phuket2.owl#WebsiteURL
http://www.h2dal.com/ontology/Phuket2.owl#hasActivity
http://www.h2dal.com/ontology/Phuket2.owl#hasAttraction
http://www.h2dal.com/ontology/Phuket2.owl#hasCloseDay
http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict
http://www.h2dal.com/ontology/Phuket2.owl#hasPayment
http://www.h2dal.com/ontology/Phuket2.owl#hasPopularSeason
http://www.h2dal.com/ontology/Phuket2.owl#hasPrice

Choose the properties #hasDistrict #EndPeriod #StartPeriod and record in property list.

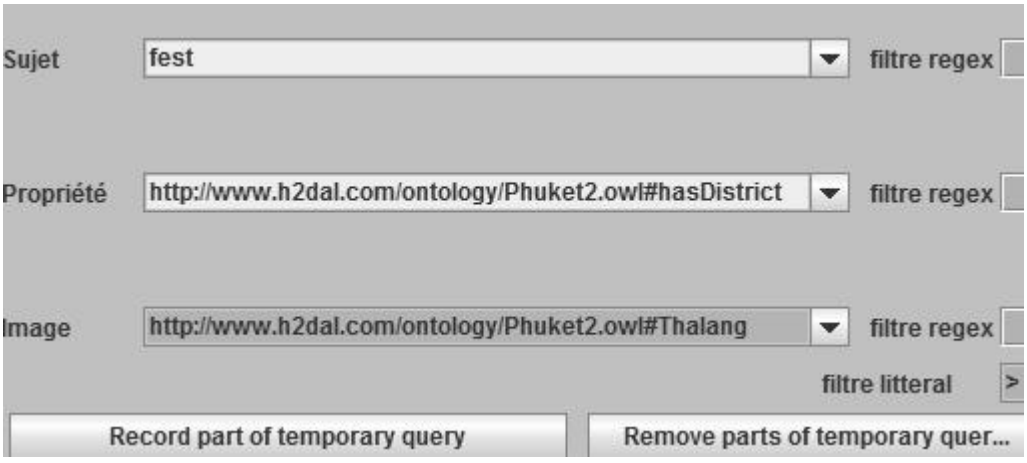
Now, we have all IRI useful to construct the final query :



1st part :



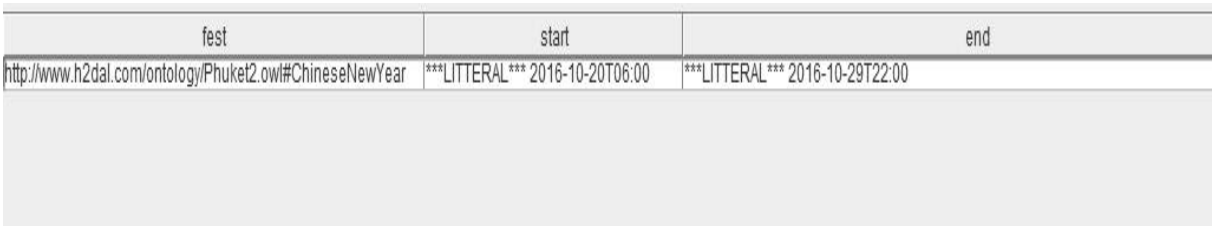
2nd part :



3rd part :

4th part :

Now Execute :



fest	start	end
http://www.h2dal.com/ontology/Phuket2.owl#ChineseNewYear	***LITTERAL*** 2016-10-20T06:00	***LITTERAL*** 2016-10-29T22:00

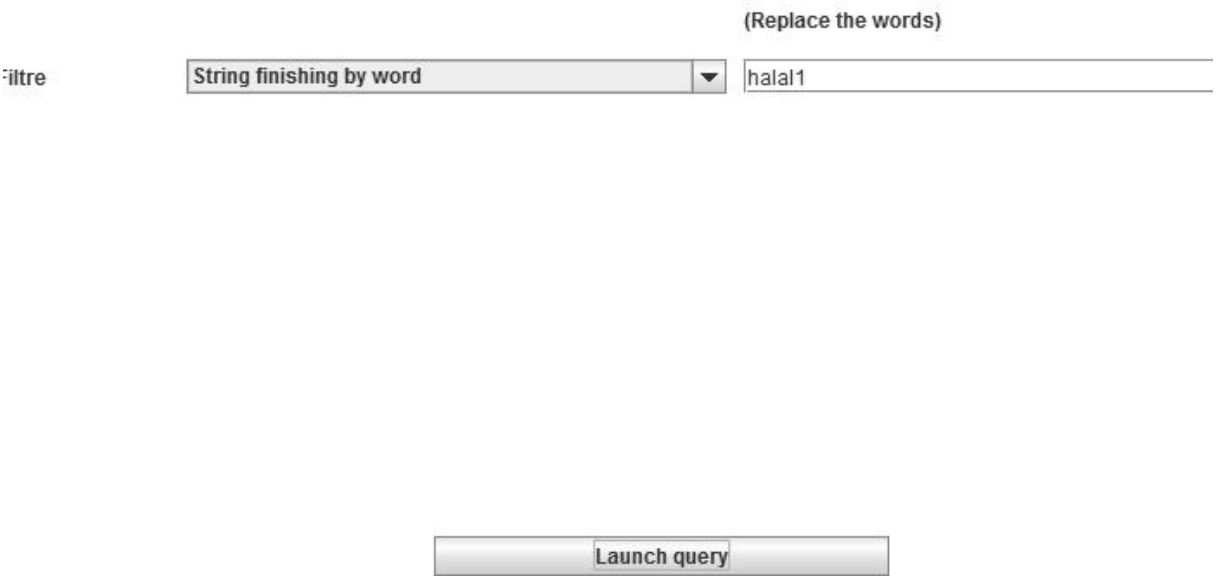
The associated SPARQL is

```
select ?fest ?start ?end WHERE { ?fest  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.h2dal.com/ontology/Phuket2.owl#Event&Festival>  . ?fest
<http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict>
<http://www.h2dal.com/ontology/Phuket2.owl#Thalang>  . ?fest
<http://www.h2dal.com/ontology/Phuket2.owl#StartPeriod>  ?start  . ?fest
<http://www.h2dal.com/ontology/Phuket2.owl#EndPeriod>  ?end  .}
```

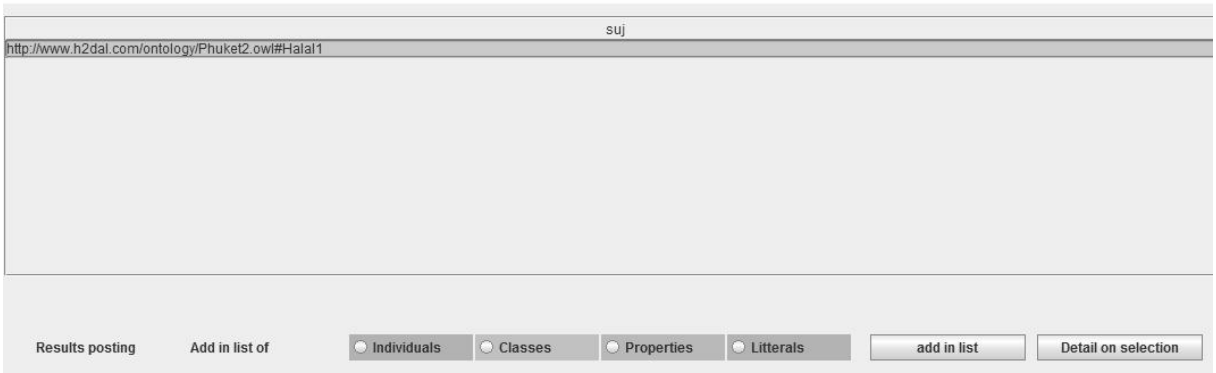
Please show information for Halal1

The shortest solution is :

Panel search >> seek and individual who is subject of triplet >>



PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> SELECT DISTINCT ?suj WHERE { ?suj a <http://www.w3.org/2002/07/owl#NamedIndividual> . ?suj ?prop ?img . FILTER regex(str(?suj) ,"halal1", "i") } ORDER BY ?suj LIMIT 100



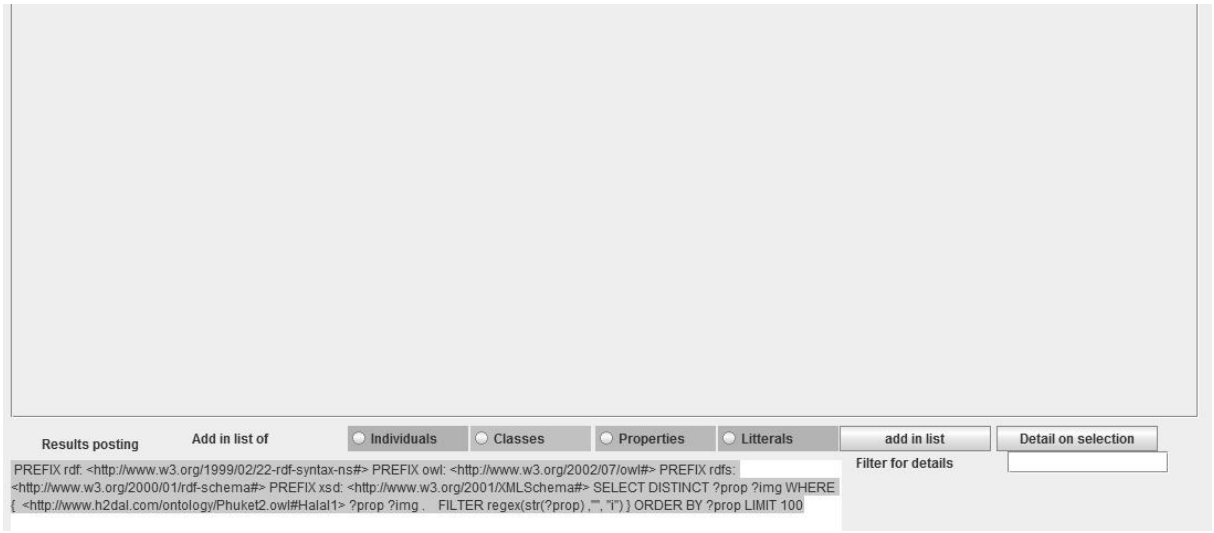
Select the #Halal1 IRI, then push the 'detail on selection' :

(that is SPARQL :

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd:

<http://www.w3.org/2001/XMLSchema#> SELECT DISTINCT ?prop ?img WHERE { <http://www.h2dal.com/ontology/Phuket2.owl#Halal1> ?prop ?img . FILTER regex(str(?prop) ,"", "i") } ORDER BY ?prop LIMIT 100)

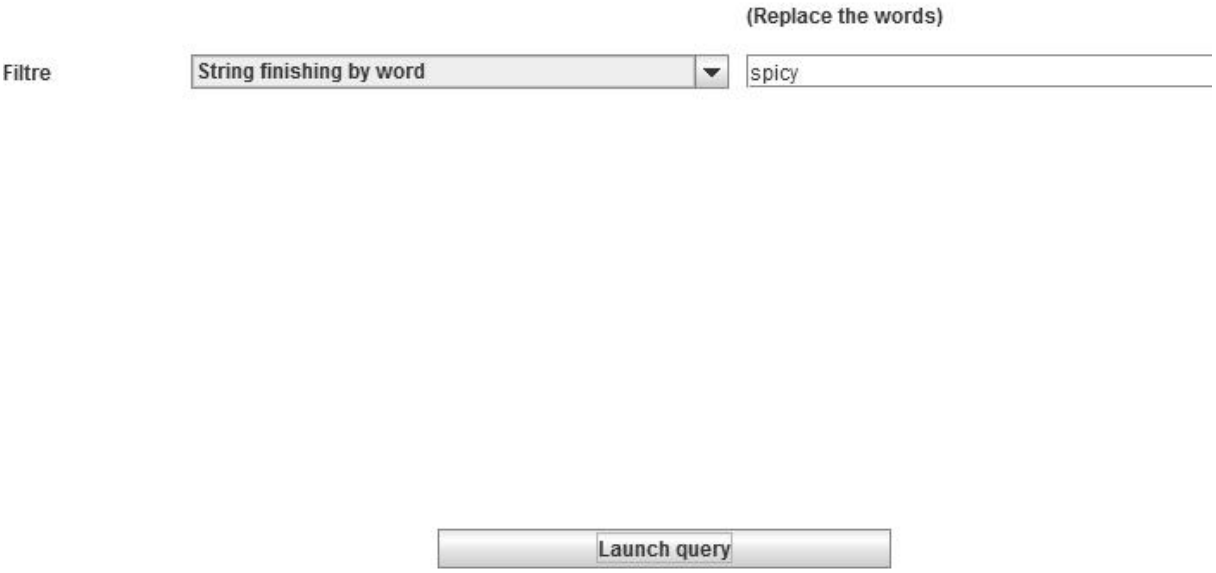
You can retrieve the SPARQL code in the inferior part of the panel :



prop	img
http://www.h2dal.com/ontology/Phuket2.owl#Address	***LITTERAL*** 256/9
http://www.h2dal.com/ontology/Phuket2.owl#Content	***LITTERAL*** Phuket's floating seafood restaurants ('krachang' in Thai) are a fun way to din.
http://www.h2dal.com/ontology/Phuket2.owl#Email	***LITTERAL*** aa@gmail.com
http://www.h2dal.com/ontology/Phuket2.owl#EndDateTime	***LITTERAL*** 2016-03-29T09:00
http://www.h2dal.com/ontology/Phuket2.owl#Fax	***LITTERAL*** 076285698
http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Food, Muslim, Beef [no(Restaurant, Thalang)]
http://www.h2dal.com/ontology/Phuket2.owl#Latitude	***LITTERAL*** 7.986576
http://www.h2dal.com/ontology/Phuket2.owl#Longtitude	***LITTERAL*** 98.331475
http://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** Bang_Mud_Floating_Restaurant
http://www.h2dal.com/ontology/Phuket2.owl#PathPic	***LITTERAL*** .....\onotlogy\1.jpg
http://www.h2dal.com/ontology/Phuket2.owl#PersonContact	***LITTERAL*** Somchai
http://www.h2dal.com/ontology/Phuket2.owl#PhotoDescription	***LITTERAL*** In front of the restaurant
http://www.h2dal.com/ontology/Phuket2.owl#StartDateTime	***LITTERAL*** 2016-03-29T22:00
http://www.h2dal.com/ontology/Phuket2.owl#Street	***LITTERAL*** Mahar
http://www.h2dal.com/ontology/Phuket2.owl#Telephone	***LITTERAL*** 081-273-2367
http://www.h2dal.com/ontology/Phuket2.owl#WebsiteURL	***LITTERAL*** www.halal1.com
http://www.h2dal.com/ontology/Phuket2.owl#hasCloseDay	http://www.h2dal.com/ontology/Phuket2.owl#Monday
http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Thalang
http://www.h2dal.com/ontology/Phuket2.owl#hasPayment	http://www.h2dal.com/ontology/Phuket2.owl#Cash
http://www.h2dal.com/ontology/Phuket2.owl#hasPayment	http://www.h2dal.com/ontology/Phuket2.owl#CreditCard
http://www.h2dal.com/ontology/Phuket2.owl#hasRating	http://www.h2dal.com/ontology/Phuket2.owl#4star
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Activity
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Attraction
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#DateTime
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#DateTimePeriod
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Event&Festival
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Halal

Which restaurant has spicy food?

Panel search >> seek an individual or literal in object of triplet >>



That is equivalent to the sparql we can retrieve :

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> SELECT DISTINCT ?suj ?prop ?img WHERE { ?suj ?prop ?img . FILTER regex(str(?img) ,"spicy", "i") . FILTER(NOT EXISTS { ?img a owl:Class }).} ORDER BY ?img LIMIT 100

We get :

suj	prop	
http://www.h2dal.com/ontology/Phuket2.owl#Halal2	http://www.h2dal.com/ontology/Phuket2.owl#SpecialRequirement	***LITTERAL*** Spicy Food

Which restaurant for Muslim?

Same method as 24) with the keyword ‘muslim’

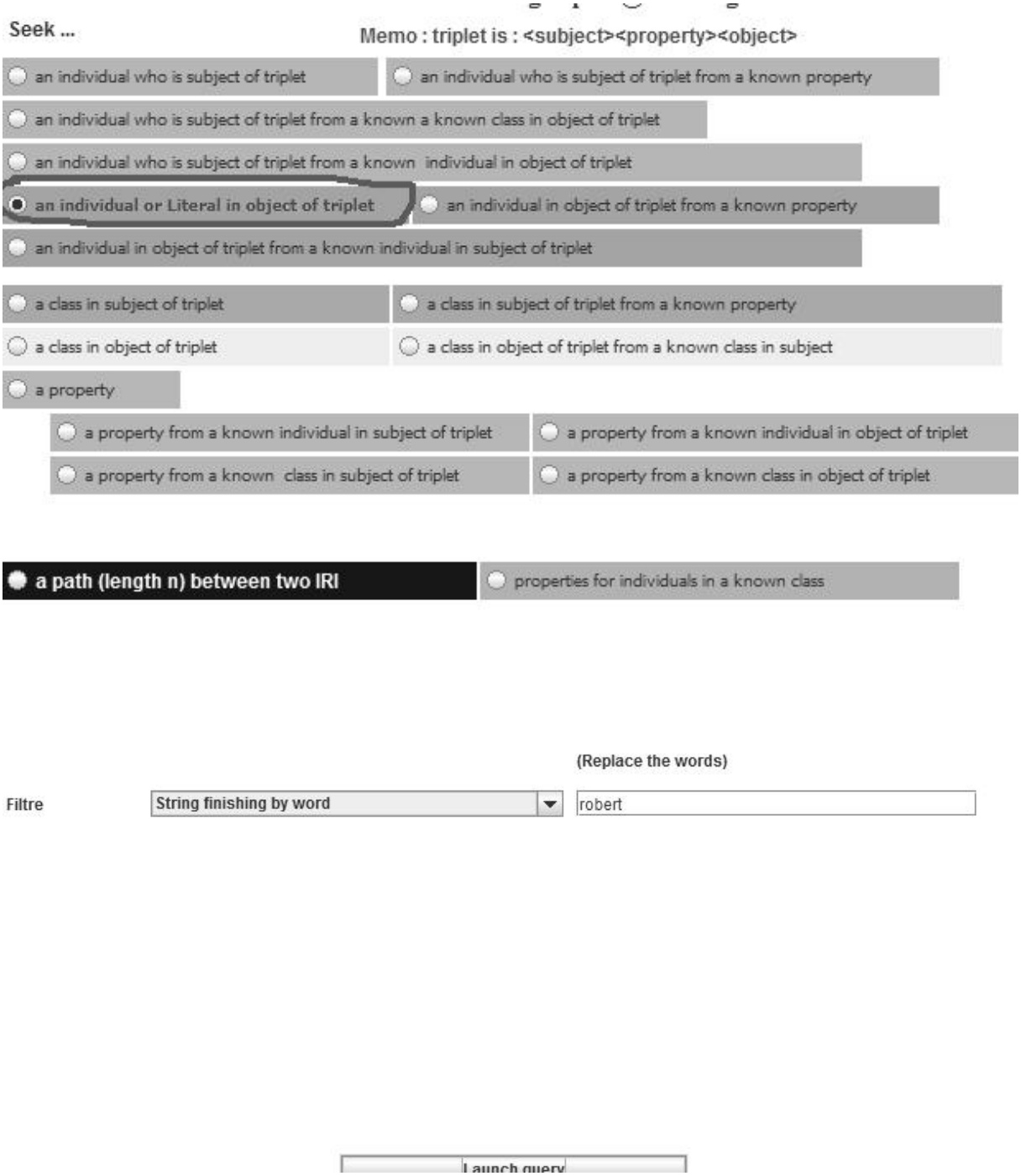
suj	prop	img
http://www.h2dal.com/ontology/Phuket2.owl#Halal1	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Food, Muslim, Beef [no(Restaurant, Thalang)]
http://www.h2dal.com/ontology/Phuket2.owl#Halal2	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Muslim

With the SPARQL query :

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> SELECT DISTINCT ?suj ?prop ?img WHERE { ?suj ?prop ?img . FILTER regex(str(?img) ,"muslim", "i") . FILTER(NOT EXISTS { ?img a owl:Class }).} ORDER BY ?img LIMIT 100

Which activity I must contact K.Robert

Like 24,25 :



We get :

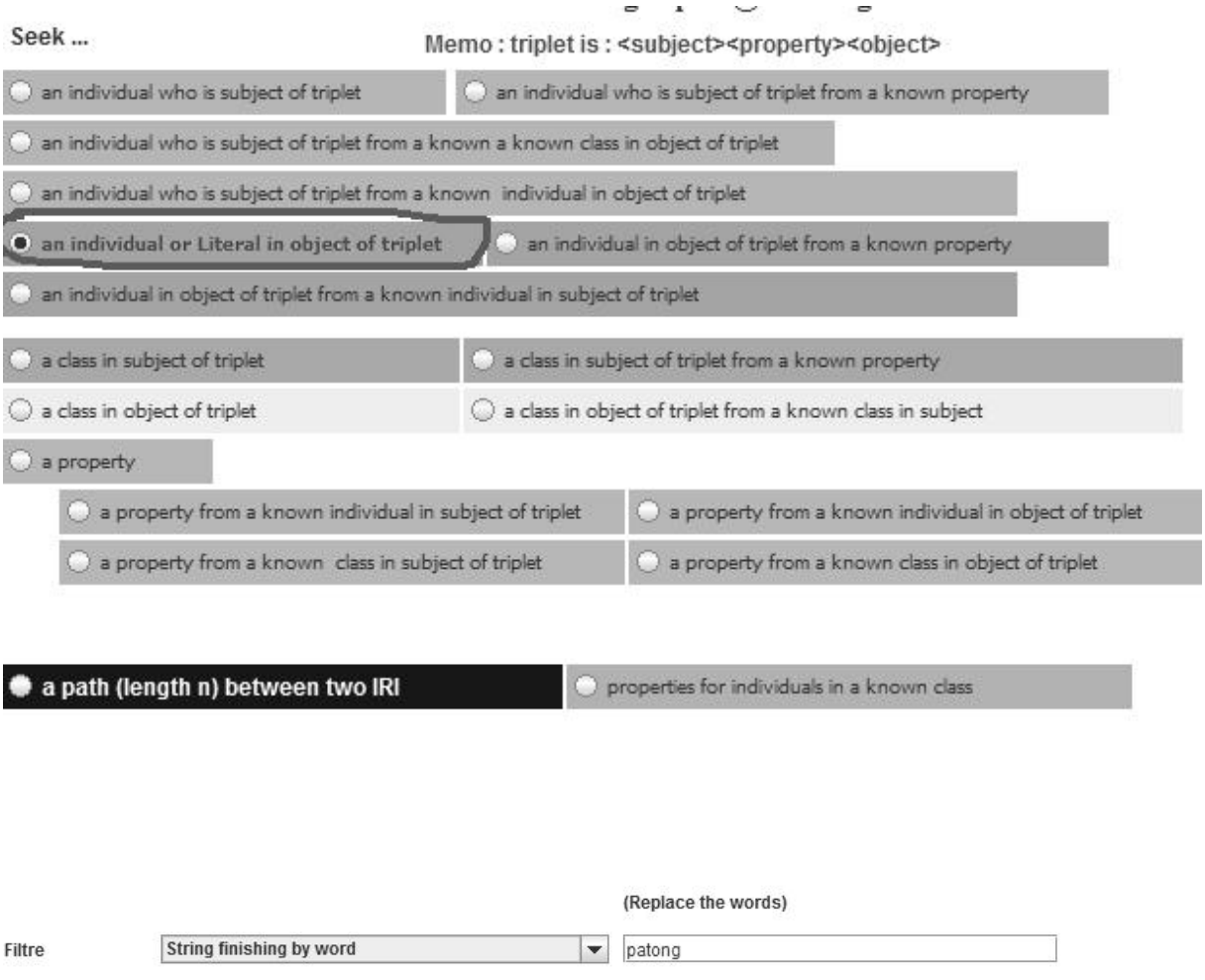
suj	prop	img
http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingA	http://www.h2dal.com/ontology/Phuket2.owl#PersonContact	***LITTERAL*** Robert

The sparql query is :

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> SELECT DISTINCT ?suj ?prop ?img WHERE { ?suj ?prop ?img . FILTER regex(str(?img) ,"robert", "i") . FILTER(NOT EXISTS { ?img a owl:Class }).} ORDER BY ?img LIMIT 100

What can I do at Patong?

Same method as 24,25,26



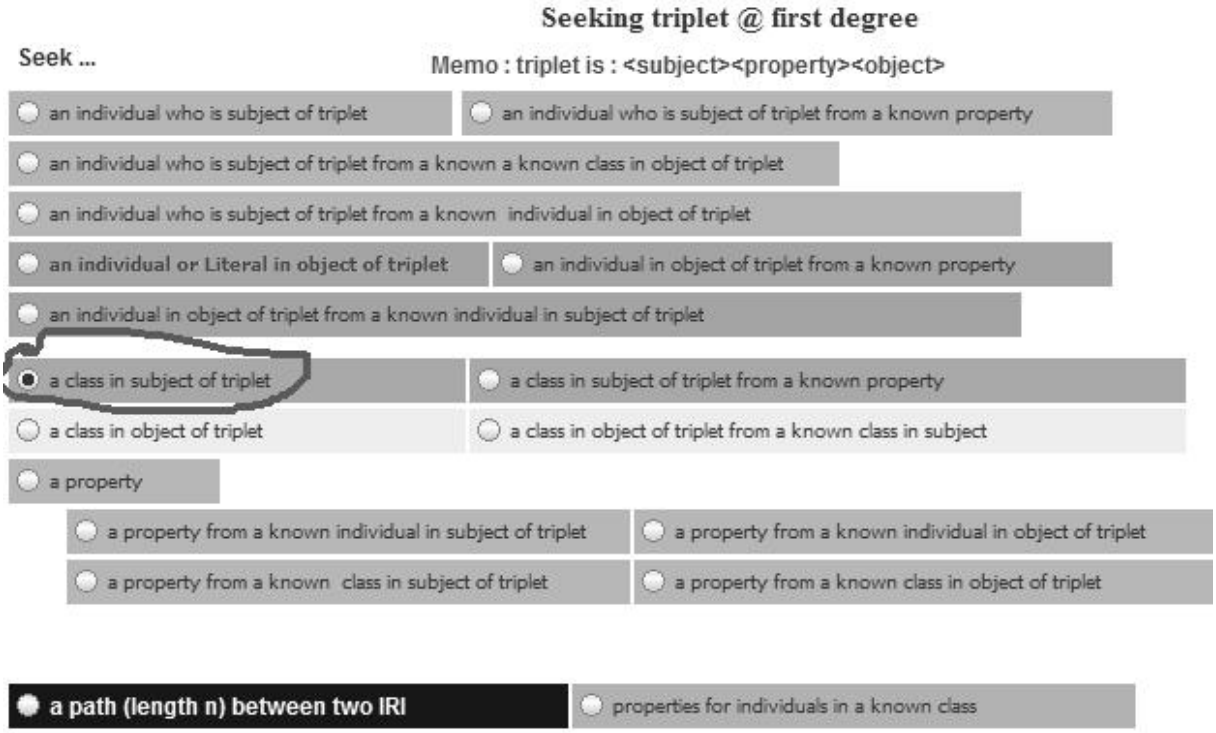
Then you get :

suj	prop	
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	http://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** Bangla Boxing
http://www.h2dal.com/ontology/Phuket2.owl#Beach1	http://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** Patong Beach
http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingA	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Patong Beach,
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Real filght, Stac

with the SPARQL :

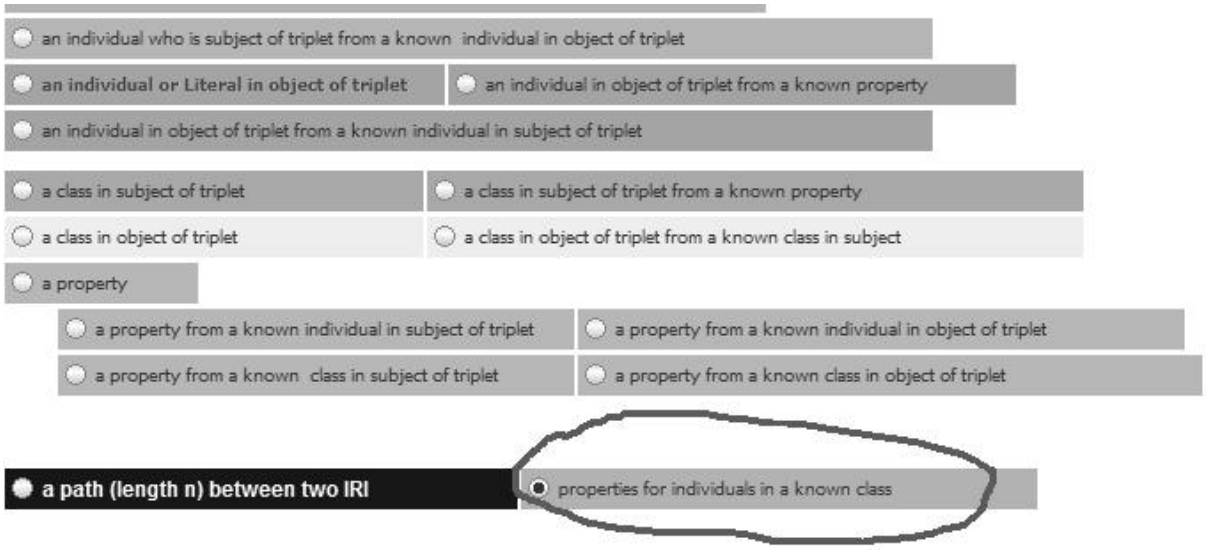
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> SELECT DISTINCT ?suj ?prop ?img  WHERE {  ?suj ?prop ?img . FILTER regex(str(?img) ,"patong", "i") . FILTER(NOT EXISTS { ?img a owl:Class } ).} ORDER BY ?img LIMIT 100
```

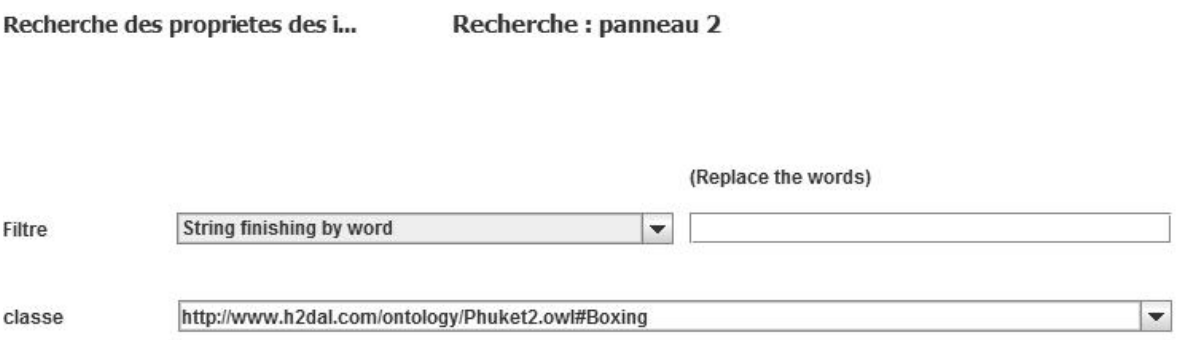
Please show the name for Boxing and show the address



Choose #Boxing and record in list of Classes.

Return in the Serach panel.





Choose #boxing in list then :

prop
http://www.h2dal.com/ontology/Phuket2.owl#Address
http://www.h2dal.com/ontology/Phuket2.owl#Content
http://www.h2dal.com/ontology/Phuket2.owl#Email
http://www.h2dal.com/ontology/Phuket2.owl#EndDateTime
http://www.h2dal.com/ontology/Phuket2.owl#Fax
http://www.h2dal.com/ontology/Phuket2.owl#Keyword
http://www.h2dal.com/ontology/Phuket2.owl#Latitude
http://www.h2dal.com/ontology/Phuket2.owl#Longtitude
http://www.h2dal.com/ontology/Phuket2.owl#Name
http://www.h2dal.com/ontology/Phuket2.owl#SpecialRequirement
http://www.h2dal.com/ontology/Phuket2.owl#StartDateTime
http://www.h2dal.com/ontology/Phuket2.owl#Street
http://www.h2dal.com/ontology/Phuket2.owl#Telephone
http://www.h2dal.com/ontology/Phuket2.owl#WebsiteURL
http://www.h2dal.com/ontology/Phuket2.owl#hasCloseDay
http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict
http://www.h2dal.com/ontology/Phuket2.owl#hasPayment
http://www.h2dal.com/ontology/Phuket2.owl#hasPrice
http://www.h2dal.com/ontology/Phuket2.owl#hasRating
http://www.w3.org/1999/02/22-rdf-syntax-ns#type

Select and record the properties you need :

Address, Street, hasDistrict.

Now, we create the main query with all elements we just collected.

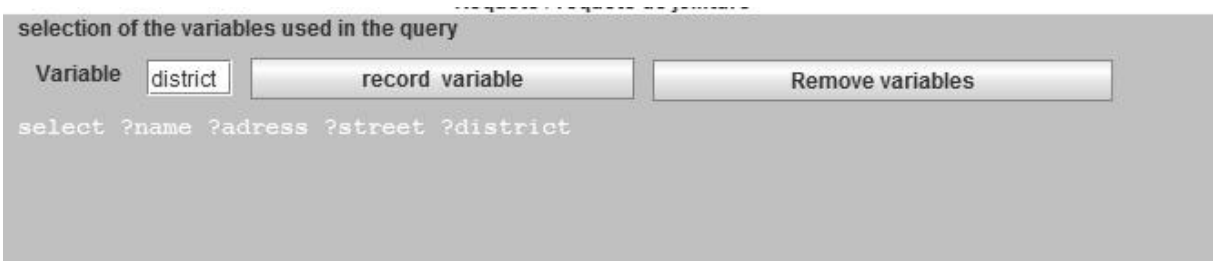
TxtField : name + record variable

TxtField : adress + record variable

TxtField : street+ record variable

TxtField : district + record variable

We get :



1st part of query :

“record part of temporary query’

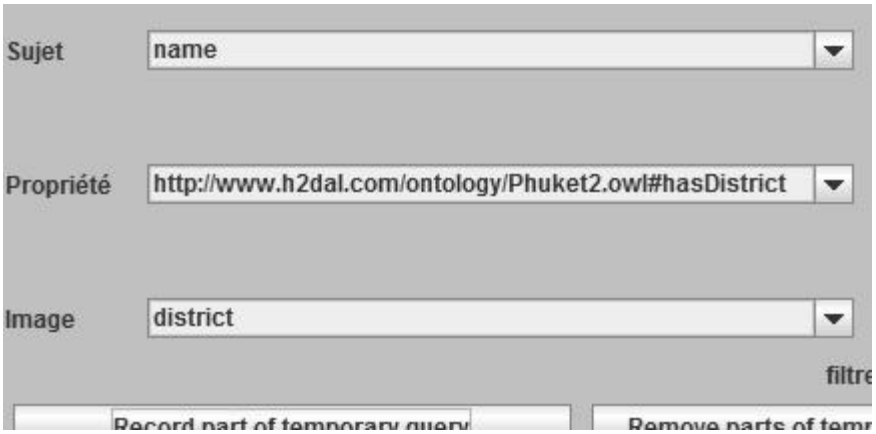
2nd part of query :

“record part of temporary query’

3rd part of query :

“record part of temporary query’

4th part of query :



“record part of temporary query’

Then Execute Query/record it :

name	adress	street	district
http://www.h2dal.com/ontology/Phuket2.owl#Boxing2	***LITTERAL*** 100/15	***LITTERAL*** Rawai	http://www.h2dal.com/ontology/Phuket2.owl#Mueang
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	***LITTERAL*** 198/4	***LITTERAL*** Ratutit	http://www.h2dal.com/ontology/Phuket2.owl#Kathu

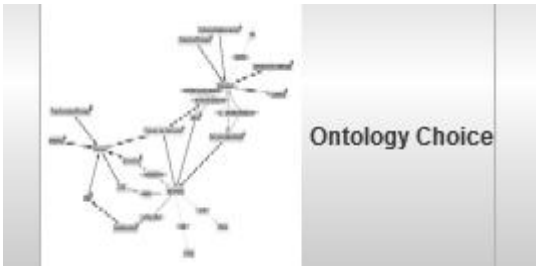
ANNEXE 2:

Queries From Phuket2.owl + Language.owl + UserProfile.owl

Preambule :

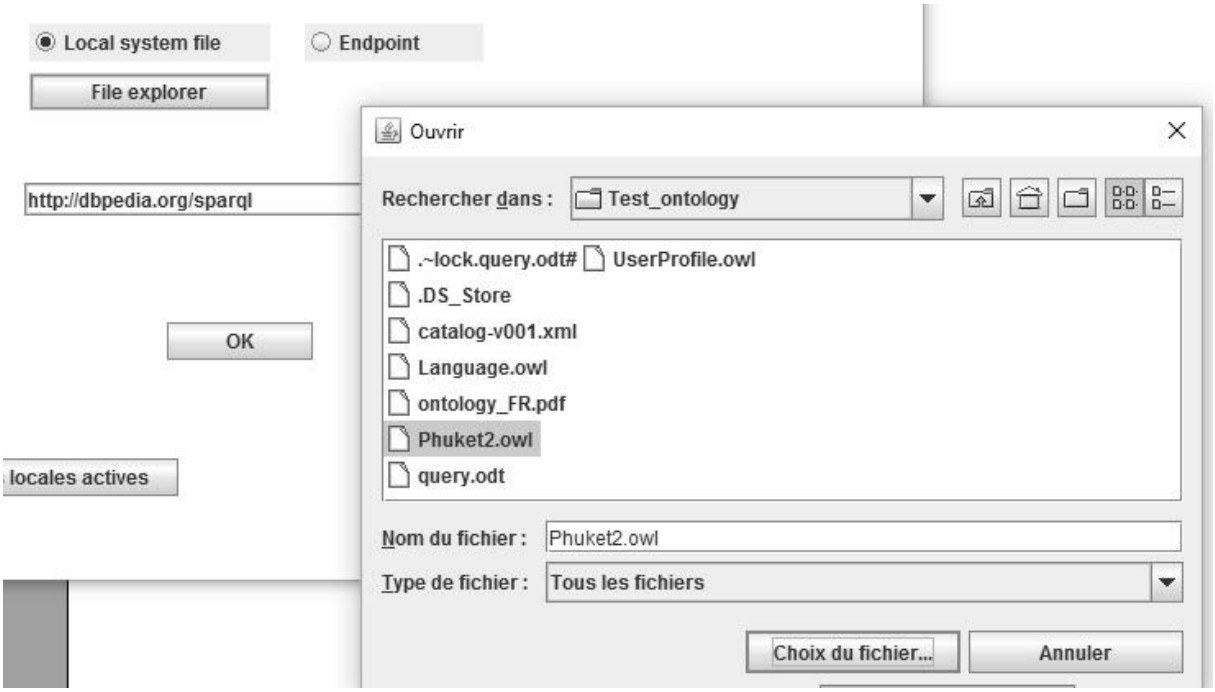
The software allows to consult several local ontologies.

This is the method to charge the 3 ontologies here :



For each ontology you must examine :

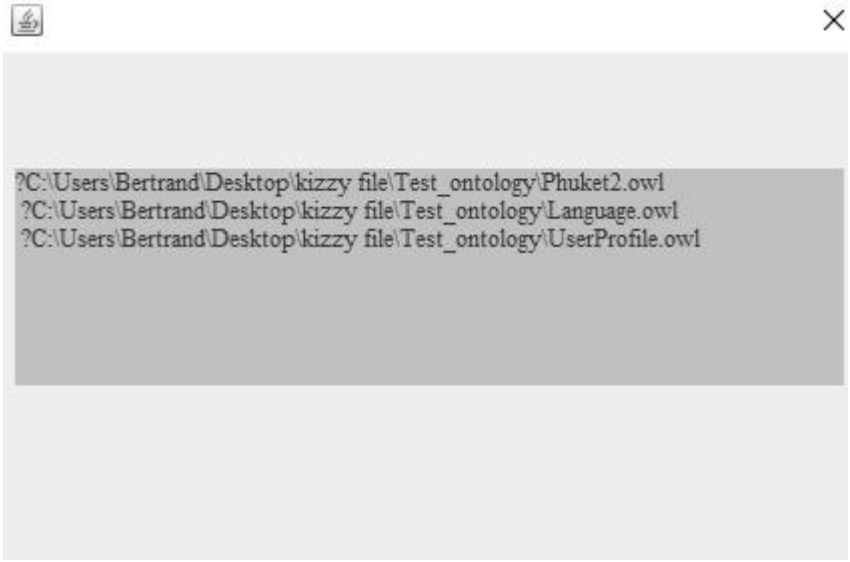
- 1)Choose ‘local system file’, then’ file explorer’ button.
- 2)Use the file explorer to find you owl file in your hard drive.



- 3) Then press ‘Choix du fichier’.
- 4) Then press ‘OK’.

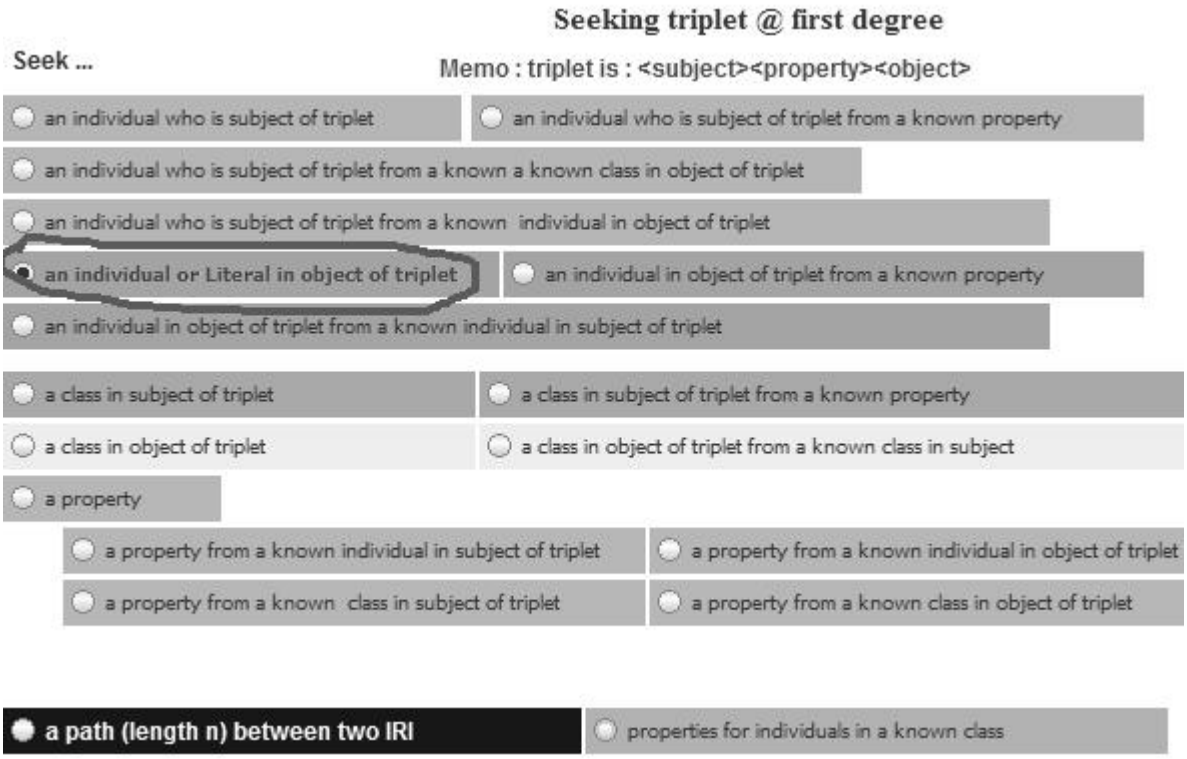
→ your file is loaded, get at the step 1) to load another ontology.

To see your actives ontologies, press ‘afficher ontologies locales actives’.



Who loves monkeys and suggest attraction for her and give the transliterated word(words_PO)?

Who loves monkeys :



We type ‘monkey’ in the textfield :

http://www.h2dal.com/ontology/UserProfile.owl#Man2	http://www.h2dal.com/ontology/UserProfile.owl#Like	***LITTERAL*** Outdoor activity, monkey
http://www.h2dal.com/ontology/UserProfile.owl#Man2	http://www.w3.org/2002/07/owl#topDataProperty	***LITTERAL*** Outdoor activity, monkey
http://www.h2dal.com/ontology/Phuket2.owl#Zoo	http://www.h2dal.com/ontology/Phuket2.owl#Content	***LITTERAL*** Phuket Zoo is a private zoo. It is not supported by the governm
http://www.h2dal.com/ontology/Phuket2.owl#Zoo	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** animals, Monkey Shows, Crocodile Shows, Elephant Shows,

We choose the first line which tells us that man2 like monkey.

To have details on man2, push button ‘detail on selection’

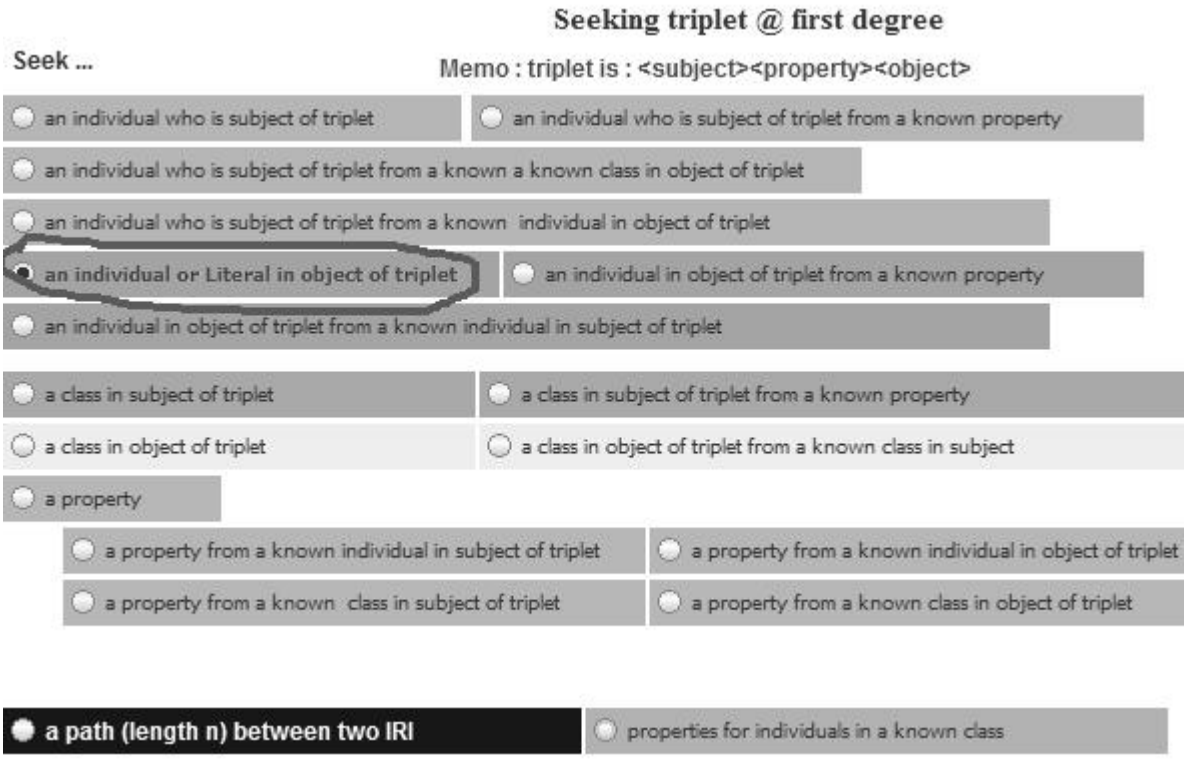
prop	
http://www.h2dal.com/ontology/UserProfile.owl#Depart	***LITTERAL*** 2016-04-10T06:00
http://www.h2dal.com/ontology/UserProfile.owl#Dislike	***LITTERAL*** adventure activity
http://www.h2dal.com/ontology/UserProfile.owl#Like	***LITTERAL*** Outdoor activity, monkey
http://www.h2dal.com/ontology/UserProfile.owl#Name	***LITTERAL*** Nana
http://www.h2dal.com/ontology/UserProfile.owl#NumberOfAdult	***LITTERAL*** 2
http://www.h2dal.com/ontology/UserProfile.owl#NumberOfChildren	***LITTERAL*** 1
http://www.h2dal.com/ontology/UserProfile.owl#Return	***LITTERAL*** 2016-05-01T06:00
http://www.h2dal.com/ontology/UserProfile.owl#hasDistrict	http://www.h2dal.com/ontology/UserProfile.owl#Kathu
http://www.h2dal.com/ontology/UserProfile.owl#hasTravelMember	http://www.h2dal.com/ontology/UserProfile.owl#Adult
http://www.h2dal.com/ontology/UserProfile.owl#hasTravelMember	http://www.h2dal.com/ontology/UserProfile.owl#Children
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.owl#Personal
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.owl#Travel
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.owl#TravelMer
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.owl#TripDate
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual
http://www.w3.org/2002/07/owl#topDataProperty	***LITTERAL*** Outdoor activity, monkey

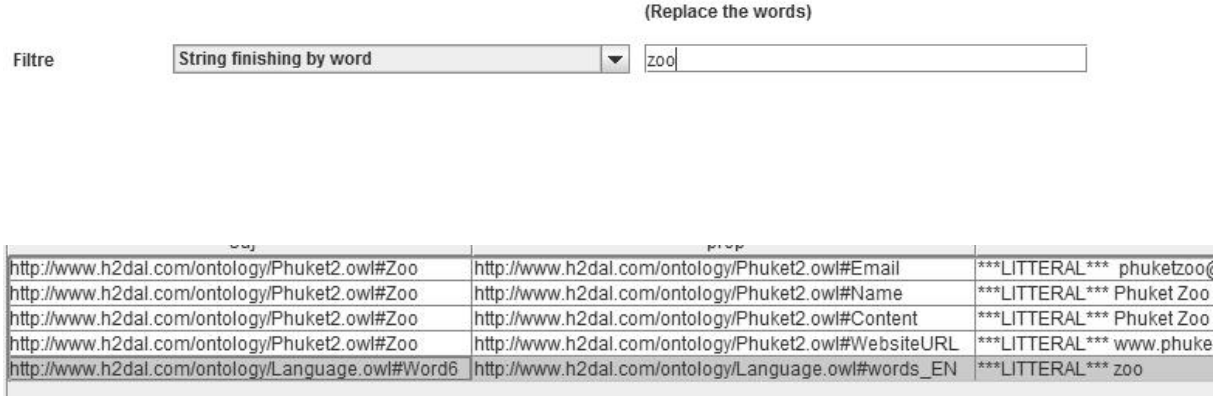
The name is ’nana’

Attraction for her :

The picture above shows the #zoo

Traduction de ce mot :





http://www.h2dal.com/ontology/Phuket2.owl#Zoo	http://www.h2dal.com/ontology/Phuket2.owl#Email	***LITTERAL*** phuketzoo@
http://www.h2dal.com/ontology/Phuket2.owl#Zoo	http://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** Phuket Zoo
http://www.h2dal.com/ontology/Phuket2.owl#Zoo	http://www.h2dal.com/ontology/Phuket2.owl#Content	***LITTERAL*** Phuket Zoo
http://www.h2dal.com/ontology/Phuket2.owl#Zoo	http://www.h2dal.com/ontology/Phuket2.owl#WebsiteURL	***LITTERAL*** www.phuke
http://www.h2dal.com/ontology/Language.owl#Word6	http://www.h2dal.com/ontology/Language.owl#words_EN	***LITTERAL*** zoo

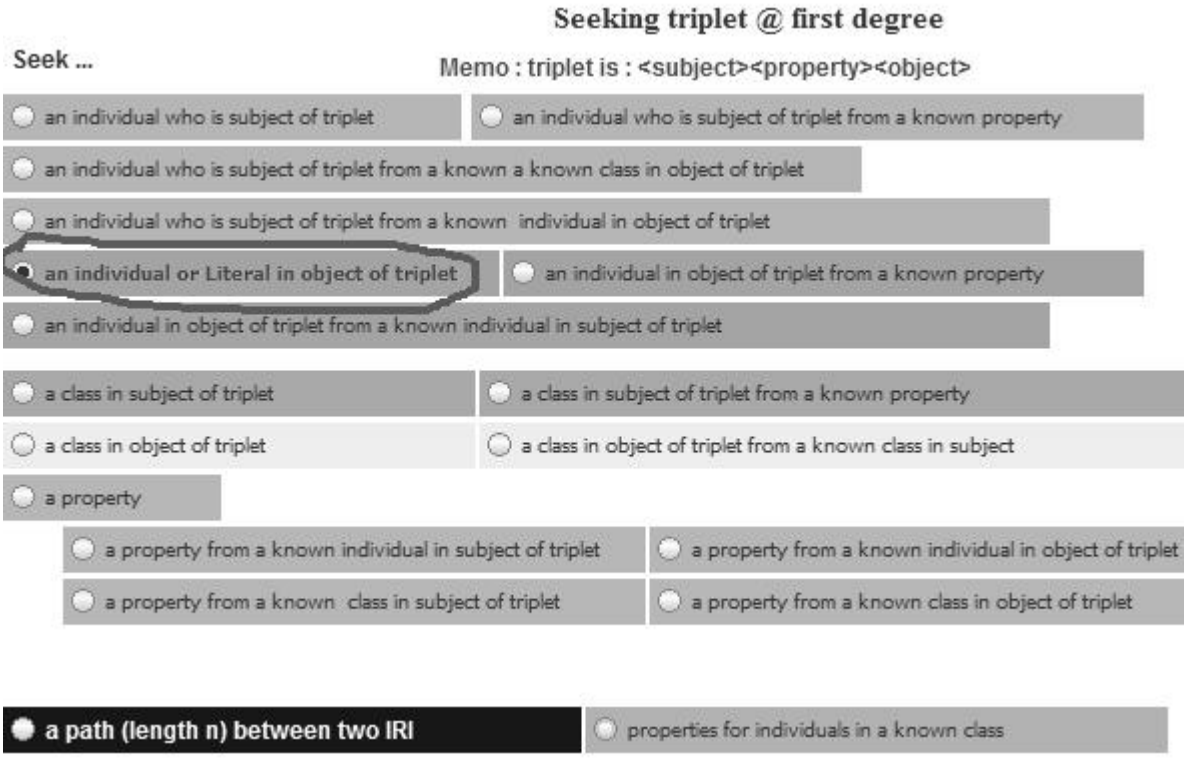
The last line is interesting for us :

Choose the #word6 and push ‘details’ button :

prop	
http://www.h2dal.com/ontology/Language.owl#hasGroup	http://www.h2dal.com/ontology/Language.owl#Attraction
http://www.h2dal.com/ontology/Language.owl#synonym	***LITTERAL*** animal
http://www.h2dal.com/ontology/Language.owl#words_EN	***LITTERAL*** zoo
http://www.h2dal.com/ontology/Language.owl#words_PO	***LITTERAL*** Suansat, sat
http://www.h2dal.com/ontology/Language.owl#words_TR	***LITTERAL*** Suansud, SuanSut, Sut, Sud
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Language.owl#Content
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual

The traduction is Suansat, sat (PO) ou Suansud, SuanSut, Sut,Sud (TR)

Which restaurant doesn't suitable for Amnart?

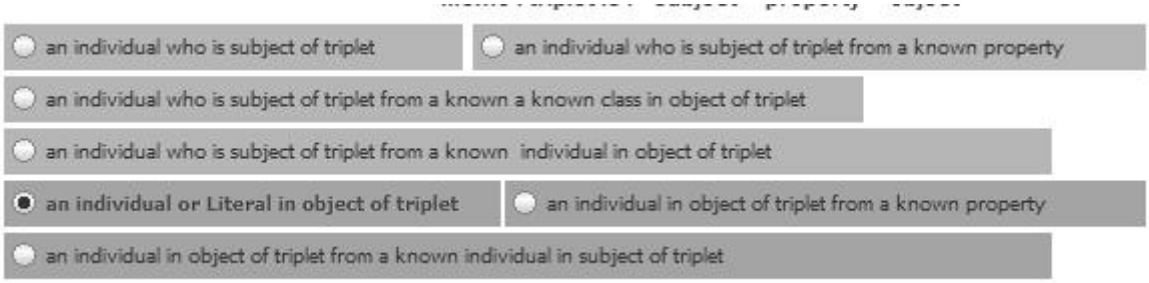


suj	prop	
http://www.h2dal.com/ontology/UserProfile.owl#Man1	http://www.h2dal.com/ontology/UserProfile.owl#Name	***LITTERAL*** Amnart

Choose #man1 IRI and press ‘detail’ :

prop	
http://www.h2dal.com/ontology/UserProfile.owl#Depart	***LITTERAL*** 2016-10-10T06:00
http://www.h2dal.com/ontology/UserProfile.owl#Dislike	***LITTERAL*** Spicy food
http://www.h2dal.com/ontology/UserProfile.owl#Email	***LITTERAL*** sa@gmail.com
http://www.h2dal.com/ontology/UserProfile.owl#Gender	***LITTERAL*** Male
http://www.h2dal.com/ontology/UserProfile.owl#Like	***LITTERAL*** Water Activity
http://www.h2dal.com/ontology/UserProfile.owl#Name	***LITTERAL*** Amnart
http://www.h2dal.com/ontology/UserProfile.owl#NumberOfA...	***LITTERAL*** 2
http://www.h2dal.com/ontology/UserProfile.owl#Return	***LITTERAL*** 2016-10-25T06:00
http://www.h2dal.com/ontology/UserProfile.owl#hasDistrict	http://www.h2dal.com/ontology/UserProfile.ow
http://www.h2dal.com/ontology/UserProfile.owl#hasInterest	http://www.h2dal.com/ontology/UserProfile.ow
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.ow
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.ow
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.ow
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.ow
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Reso
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndivic
http://www.w3.org/2002/07/owl#topDataProperty	***LITTERAL*** Water Activity

We see that Amnart doesn't like Spicy Food.



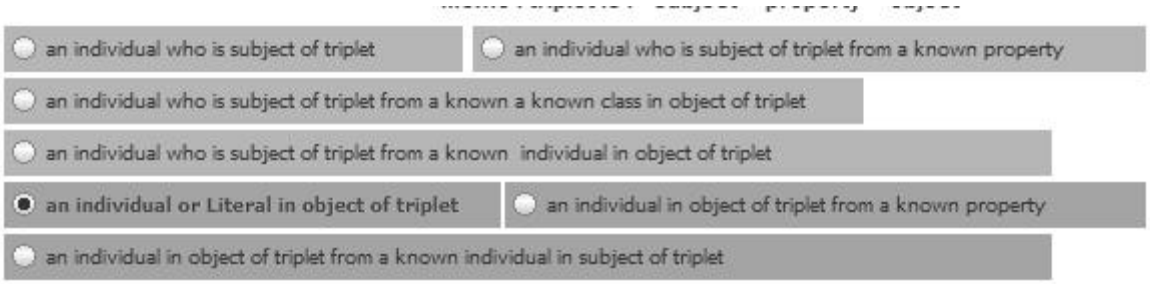
We get :

obj	prop	
http://www.h2dal.com/ontology/Phuket2.owl#Halal2	http://www.h2dal.com/ontology/Phuket2.owl#SpecialReq...	***LITTERAL*** Spicy Food
http://www.h2dal.com/ontology/UserProfile.owl#Man1	http://www.h2dal.com/ontology/UserProfile.owl#Dislike	***LITTERAL*** Spicy food

Halal2 is not suitable for Amnart.

Where I can watch the cinema?

Again panel

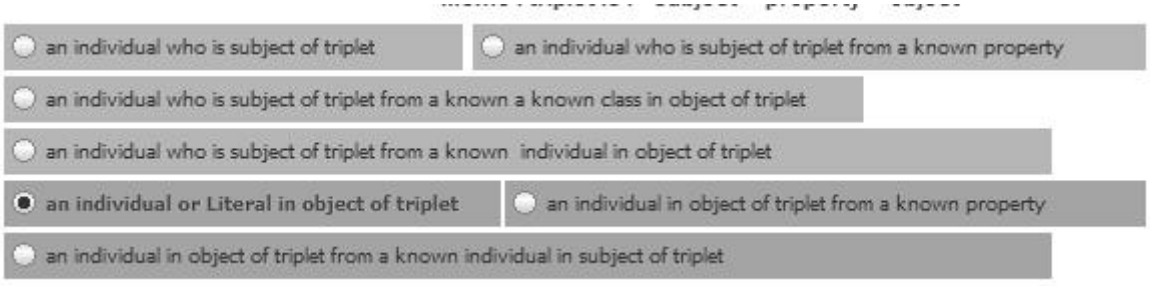


Then

we have the following results :

suj	prop	img
http://www.h2dal.com/ontology/Phuket2.owl#Theater1	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Central Phuket, Movie, Popcorn, Cinema
http://www.h2dal.com/ontology/Language.owl#Word2	http://www.h2dal.com/ontology/Language.owl#synonym	***LITTERAL*** Cinema
http://www.h2dal.com/ontology/Phuket2.owl#Theater2	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Movie, Popcorn, Cinema
http://www.h2dal.com/ontology/Phuket2.owl#Theater1	http://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** SF Cinema

What is Kathu? (Ans District, waterfall)



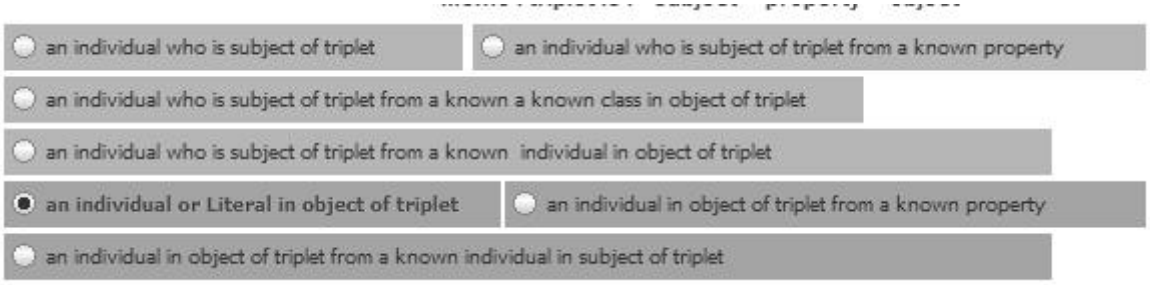
suj	prop	img
http://www.h2dal.com/ontology/Phuket2.owl#Beach1	http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Kathu
http://www.h2dal.com/ontology/Phuket2.owl#Beach2	http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Kathu
http://www.h2dal.com/ontology/Phuket2.owl#Boxing1	http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Kathu
http://www.h2dal.com/ontology/Phuket2.owl#Bungy1	http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Kathu
http://www.h2dal.com/ontology/Phuket2.owl#ChaoLayFestival	http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Kathu
http://www.h2dal.com/ontology/Phuket2.owl#FUJI2	http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Kathu
http://www.h2dal.com/ontology/Phuket2.owl#FUJI3	http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Kathu
http://www.h2dal.com/ontology/Phuket2.owl#Flying1	http://www.h2dal.com/ontology/Phuket2.owl#hasDistrict	http://www.h2dal.com/ontology/Phuket2.owl#Kathu

Select #Kathu then ‘detail on selection’

prop	img
http://www.h2dal.com/ontology/Phuket2.owl#Postal	***LITTERAL*** 83000
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#District
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual

Kathu is of type #district

What is Katu?



suj	prop	
http://www.h2dal.com/ontology/Language.owl#Word5	http://www.h2dal.com/ontology/Language.owl#words_TR	***LITTERAL*** Kratu, Krathu, Katu

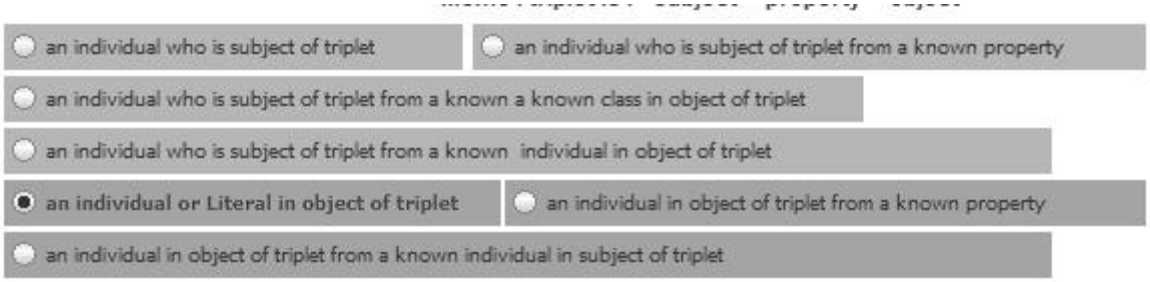
Katu is translitterated word of word 5.

We can select #word5 and ‘detail on selection’

prop	
http://www.h2dal.com/ontology/Language.owl#hasGroup	http://www.h2dal.com/ontology/Language.owl#Attraction
http://www.h2dal.com/ontology/Language.owl#hasGroup	http://www.h2dal.com/ontology/Language.owl#District
http://www.h2dal.com/ontology/Language.owl#words_PO	***LITTERAL*** Kathu
http://www.h2dal.com/ontology/Language.owl#words_TR	***LITTERAL*** Kratu, Krathu, Katu
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Language.owl#Content
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual

Katu is translitterated word of ‘kathu’.

What activity is adventure activity and who didn't like?



http://www.h2dal.com/ontology/Phuket2.owl#Flying1	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Eco Adventure, outdoor activity, Rainforest experience, Kathu
http://www.h2dal.com/ontology/Phuket2.owl#Flying1	http://www.h2dal.com/ontology/Phuket2.owl#Content	***LITTERAL*** Flying Hanuman is an adventure like no other on Phuket. It s
http://www.h2dal.com/ontology/Phuket2.owl#Bungy1	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Thrilling & Challenging Activity, Jungle Bungy Jump, extreme
http://www.h2dal.com/ontology/UserProfile.owl#Man2	http://www.h2dal.com/ontology/UserProfile.owl#Dislike	***LITTERAL*** adventure activity

The results indicate that adventure is relied to the activity #flying1 and #Bungy1 and that Man2 doesn't like this.

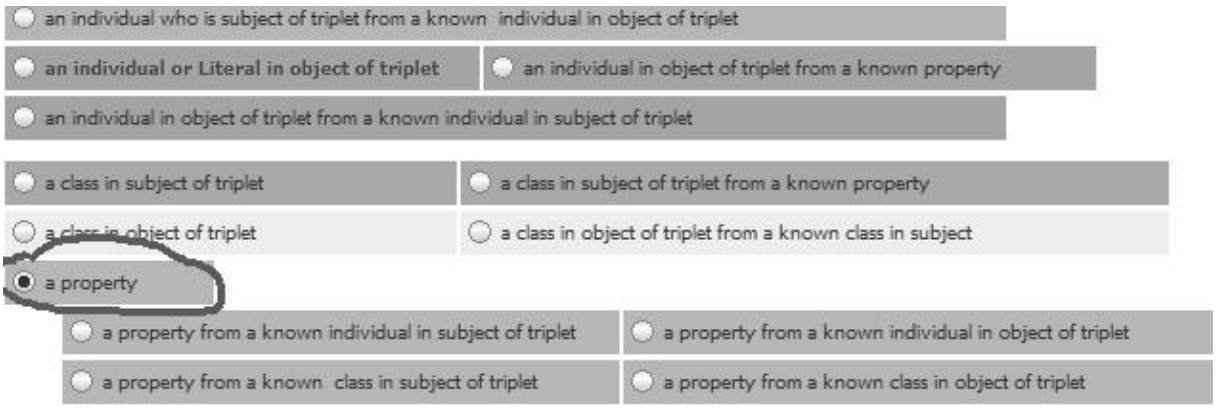
We can have details on #man2 by clicking 'detail on selection' :

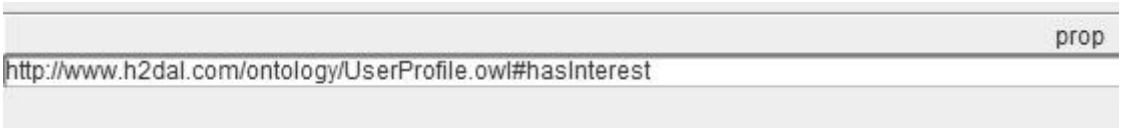
http://www.h2dal.com/ontology/UserProfile.owl#Depart	***LITTERAL*** 2016-04-10T06:00
http://www.h2dal.com/ontology/UserProfile.owl#Dislike	***LITTERAL*** adventure activity
http://www.h2dal.com/ontology/UserProfile.owl#Like	***LITTERAL*** Outdoor activity, monkey
http://www.h2dal.com/ontology/UserProfile.owl#Name	***LITTERAL*** Nana
http://www.h2dal.com/ontology/UserProfile.owl#NumberOfAdult	***LITTERAL*** 2
http://www.h2dal.com/ontology/UserProfile.owl#NumberOfChildren	***LITTERAL*** 1
http://www.h2dal.com/ontology/UserProfile.owl#Return	***LITTERAL*** 2016-05-01T06:00
http://www.h2dal.com/ontology/UserProfile.owl#hasDistrict	http://www.h2dal.com/ontology/UserProfile.owl#Kathu
http://www.h2dal.com/ontology/UserProfile.owl#hasTravelMember	http://www.h2dal.com/ontology/UserProfile.owl#Adult
http://www.h2dal.com/ontology/UserProfile.owl#hasTravelMember	http://www.h2dal.com/ontology/UserProfile.owl#Children
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.owl#Personal
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.owl#Travel
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.owl#TravelMember
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.owl#TripDate
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual
http://www.w3.org/2002/07/owl#topDataProperty	***LITTERAL*** Outdoor activity, monkey

Who interest in ChineseFestival?

We will look after a relation whois relative to a concept of interest.

Then, we will query ?suj #interest ?prop with a filter 'festival' on ?prop variable.





We record this IRI in the list of properties.

Enter ‘suj’ in text field, then ‘record variable’.

Same thing for variable ‘fest’

'record part of temporary query' then 'execute'

we get :

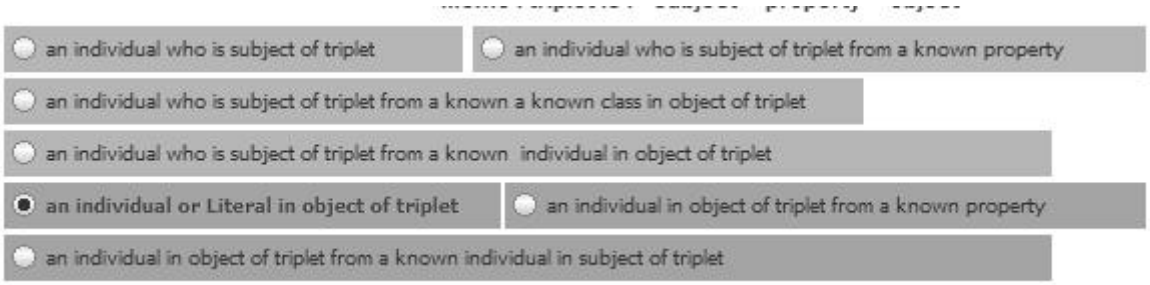
suj	fest
http://www.h2dal.com/ontology/UserProfile.owl#Man3	http://www.h2dal.com/ontology/UserProfile.owl#Event&Festival

We can then select details on these Man3 (we know he has interest on a festival but not specifically chinese one).

prop	
http://www.h2dal.com/ontology/UserProfile.owl#Gender	***LITTERAL*** Female
http://www.h2dal.com/ontology/UserProfile.owl#Like	***LITTERAL*** flower
http://www.h2dal.com/ontology/UserProfile.owl#Name	***LITTERAL*** Kai
http://www.h2dal.com/ontology/UserProfile.owl#hasDistrict	http://www.h2dal.com/ontology/UserProfile.owl#Thalang
http://www.h2dal.com/ontology/UserProfile.owl#hasInterest	http://www.h2dal.com/ontology/UserProfile.owl#Event&Festival
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/UserProfile.owl#Personal
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual
http://www.w3.org/2002/07/owl#topDataProperty	***LITTERAL*** flower

What activity can we play at thalay patong?

The 1st reflex is to looking for the word 'thalay'.



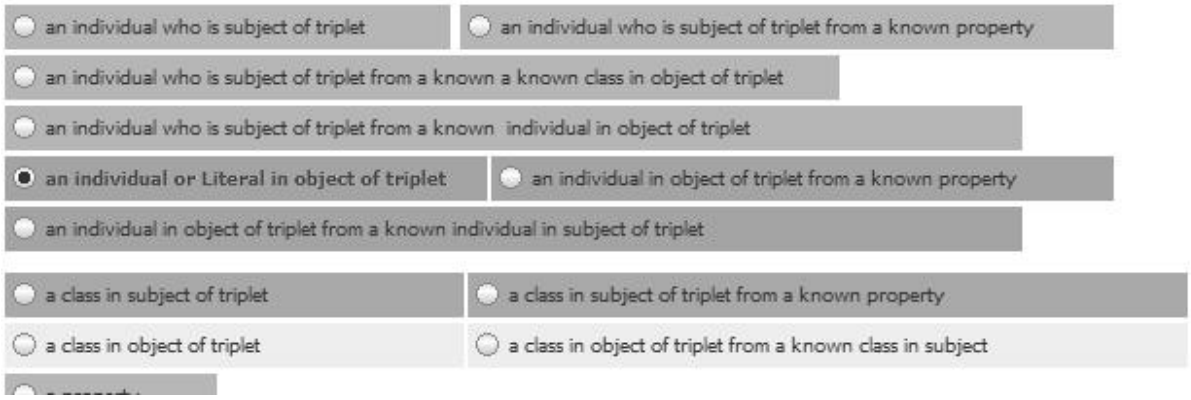
suj	prop	
http://www.h2dal.com/ontology/Language.owl#Word3	http://www.h2dal.com/ontology/Language.owl#words_PO	***LITTERAL*** Thalay, Hard

The results say that that refer to a #word3".

We select it and push 'detail on selection button'

prop	
http://www.h2dal.com/ontology/Language.owl#hasGroup	http://www.h2dal.com/ontology/Language.owl#Attraction
http://www.h2dal.com/ontology/Language.owl#synonym	***LITTERAL*** sea
http://www.h2dal.com/ontology/Language.owl#words_EN	***LITTERAL*** beach
http://www.h2dal.com/ontology/Language.owl#words_PO	***LITTERAL*** Thalay, Hard
http://www.h2dal.com/ontology/Language.owl#words_TR	***LITTERAL*** Talai, Talay, Taray, Tarai, Had, Hud
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Language.owl#Content
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual

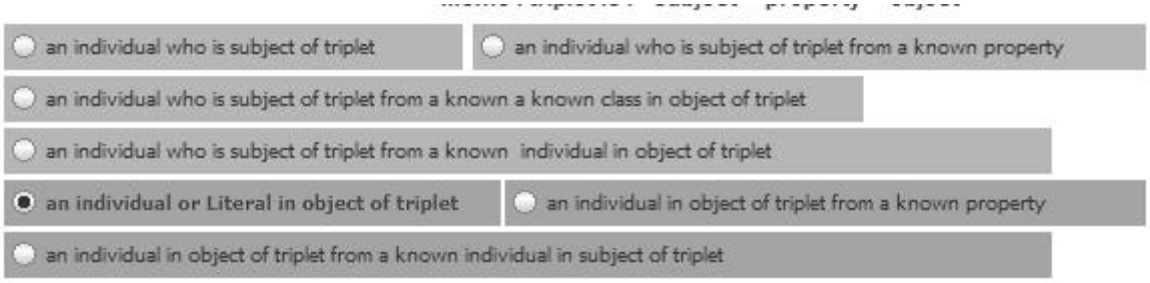
We know now that this word is a translitteration of the word 'beach'.



We get then the following results:

suj	prop	img
http://www.h2dal.com/ontology/Phuket2.owl#Beach1	http://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** Patong Beach
http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingA	http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Patong Beach, Water Sport

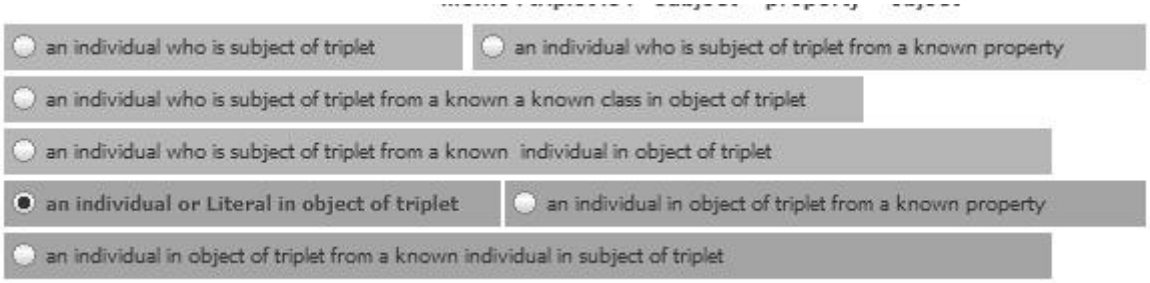
Can I pay with ngue in ABC Restaurant



suj	prop	
http://www.h2dal.com/ontology/Language.owl#Word4	http://www.h2dal.com/ontology/Language.owl#words_...	***LITTERAL*** Ngue, Nguea
http://www.h2dal.com/ontology/Language.owl#Word4	http://www.h2dal.com/ontology/Language.owl#words_...	***LITTERAL*** Nguen

With detailed selection :

prop	
http://www.h2dal.com/ontology/Language.owl#hasGroup	http://www.h2dal.com/ontology/Language.owl#Payment
http://www.h2dal.com/ontology/Language.owl#synonym	***LITTERAL*** Coin
http://www.h2dal.com/ontology/Language.owl#words_EN	***LITTERAL*** Cash
http://www.h2dal.com/ontology/Language.owl#words_PO	***LITTERAL*** Nguen
http://www.h2dal.com/ontology/Language.owl#words_TR	***LITTERAL*** Ngue, Nguea
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Language.owl#Content
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual



We look now after the ABC restaurant.

suj	prop	
http://www.h2dal.com/ontology/Phuket2.owl#Halal2	http://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** ABC R
http://www.h2dal.com/ontology/Phuket2.owl#JetSkiingA	http://www.h2dal.com/ontology/Phuket2.owl#Content	***LITTERAL*** Abcde

It refers to Halal2.

With ‘details’ we see:

prop	
http://www.h2dal.com/ontology/Phuket2.owl#Keyword	***LITTERAL*** Muslim
http://www.h2dal.com/ontology/Phuket2.owl#Name	***LITTERAL*** ABC Restaurant
http://www.h2dal.com/ontology/Phuket2.owl#SpecialRequirement	***LITTERAL*** Spicy Food
http://www.h2dal.com/ontology/Phuket2.owl#hasPayment	http://www.h2dal.com/ontology/Phuket2.owl#Cash
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Activity
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Attraction
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Event&Festival
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Halal
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#International
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Restaurant
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.h2dal.com/ontology/Phuket2.owl#Shopping
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual

We see in the #haspayment : Cash.

So, ABC Restaurant take the Ngue.

Annexe 3

fichier human_2007_09_11.rdfs

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf [
   <!ENTITY humans   "http://www.inria.fr/2007/09/11/humans.rdfs">
   <!ENTITY rdfs     "http://www.w3.org/2000/01/rdf-schema">
    ]>

<rdf:RDF
   xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="&rdfs;#"
   xmlns     ="&rdfs;#"
   xml:base  ="&humans;" >

<Class rdf:ID="Animal">
  <label xml:lang="en">animal</label>
  <comment xml:lang="en">a living organism characterized by voluntary movement.</comment>
  <label xml:lang="fr">animal</label>
  <comment xml:lang="fr">être vivant doué de sensibilite de mobilité.</comment>
</Class>

<Class rdf:ID="Male">
  <subClassOf rdf:resource="#Animal"/>
  <label xml:lang="en">male</label>
  <comment xml:lang="en">an animal that produces gametes (spermatozoa) that can fertilize female gametes (ova).</comment>
  <label xml:lang="fr">mâle</label>
  <comment xml:lang="fr">individu appartenant au sexe qui possède le pouvoir de fécondation.</comment>
</Class>

<Class rdf:ID="Female">
  <subClassOf rdf:resource="#Animal"/>
  <label xml:lang="en">female</label>
  <comment xml:lang="en"> an animal that produces gametes (ova) that can be fertilized by male gametes (spermatozoa).</comment>
  <label xml:lang="fr">femelle</label>
  <comment xml:lang="fr">animal appartenant au sexe apte à produire des ovules.</comment>
</Class>

<Class rdf:ID="Man">
  <subClassOf rdf:resource="#Person"/>
  <subClassOf rdf:resource="#Male"/>
  <label xml:lang="en">man</label>
  <comment xml:lang="en">an adult male person</comment>
  <label xml:lang="fr">homme</label>
  <comment xml:lang="fr">mâle adulte de l'espèce humaine.</comment>
</Class>

<Class rdf:ID="Person">
  <subClassOf rdf:resource="#Animal"/>
  <label xml:lang="en">person</label>
```

```
  <label xml:lang="en">human being</label>
  <label xml:lang="en">human</label>
  <comment xml:lang="en">a member of the human species</comment>
  <label xml:lang="fr">personne</label>
  <label xml:lang="fr">être humain</label>
  <label xml:lang="fr">humain</label>
  <label xml:lang="fr">homme</label>
  <comment xml:lang="fr">un membre de l'espèce humaine.</comment>
</Class>

<Class rdf:ID="Lecturer">
  <subClassOf rdf:resource="#Person"/>
  <label xml:lang="en">lecturer</label>
  <comment xml:lang="en">someone who lectures professionally</comment>
  <label xml:lang="fr">professeur</label>
  <comment xml:lang="fr">personne qui enseigne une discipline, une technique, un art.</comment>
</Class>


<Class rdf:ID="Researcher">
  <subClassOf rdf:resource="#Person"/>
  <label xml:lang="en">researcher</label>
  <label xml:lang="en">scientist</label>
  <comment xml:lang="en">a person who devotes himself to doing research</comment>
  <label xml:lang="fr">chercheur</label>
  <label xml:lang="fr">scientifique</label>
  <comment xml:lang="fr">personne adonnée à des recherches spécialisées.</comment>
</Class>

<Class rdf:ID="Woman">
  <subClassOf rdf:resource="#Person"/>
  <subClassOf rdf:resource="#Female"/>
  <label xml:lang="en">woman</label>
  <comment xml:lang="en">an adult female person</comment>
  <label xml:lang="fr">femme</label>
  <comment xml:lang="fr">femelle adulte de l'espèce humaine.</comment>
</Class>

<rdf:Property rdf:ID="hasAncestor">
  <domain rdf:resource="#Animal"/>
  <range rdf:resource="#Animal"/>
  <label xml:lang="en">has for ancestor</label>
  <comment xml:lang="en">relation between an animal and another animal from which it is descended.</comment>
  <label xml:lang="fr">a pour ancêtre</label>
  <comment xml:lang="fr">relation entre un animal et un autre animal duquel il descend.</comment>
</rdf:Property>

<rdf:Property rdf:ID="hasParent">
  <subPropertyOf rdf:resource="#hasAncestor"/>
  <label xml:lang="en">has for parent</label>
  <comment xml:lang="en">relation between an animal and another animal which gave birth to it.</comment>
  <label xml:lang="fr">a pour parent</label>
  <comment xml:lang="fr">relation entre un animal et un autre animal qui lui a donné naissance.</comment>
</rdf:Property>

<rdf:Property rdf:ID="hasChild">
  <label xml:lang="en">has for child</label>
  <comment xml:lang="en">relation between an animal and another animal to which it gave birth.</comment>
  <label xml:lang="fr">a pour enfant</label>
  <comment xml:lang="fr">relation entre un animal et un autre animal auquel il a donné naissance.</comment>
</rdf:Property>

<rdf:Property rdf:ID="hasFather">
  <subPropertyOf rdf:resource="#hasParent"/>
  <range rdf:resource="#Male"/>
  <label xml:lang="en">has for father</label>
  <comment xml:lang="en">to have for parent a male.</comment>
  <label xml:lang="fr">a pour père</label>
  <comment xml:lang="fr">avoir pour parent un mâle.</comment>
</rdf:Property>

<rdf:Property rdf:ID="hasMother">
  <subPropertyOf rdf:resource="#hasParent"/>
  <range rdf:resource="#Female"/>
  <label xml:lang="en">has for mother</label>
  <comment xml:lang="en">to have for parent a female.</comment>
```

```
  <label xml:lang="fr">a pour mère</label>
  <comment xml:lang="fr">avoir pour parent un femmelle.</comment>
</rdf:Property>

<rdf:Property rdf:ID="hasBrother">
  <domain rdf:resource="#Animal"/>
  <range rdf:resource="#Male"/>
  <label xml:lang="en">has for brother</label>
  <comment xml:lang="en">relation with a male who has the same parents.</comment>
  <label xml:lang="fr">a pour frère</label>
  <comment xml:lang="fr">relation avec un mâle ayant les mêmes parents.</comment>
</rdf:Property>

<rdf:Property rdf:ID="hasSister">
  <domain rdf:resource="#Animal"/>
  <range rdf:resource="#Female"/>
  <label xml:lang="en">has for sister</label>
  <comment xml:lang="en">relation with a female who has the same parents.</comment>
  <label xml:lang="fr">a pour soeur</label>
  <comment xml:lang="fr">relation avec une femelle ayant les mêmes parents.</comment>
</rdf:Property>

<rdf:Property rdf:ID="hasFriend">
  <domain rdf:resource="#Person"/>
  <range rdf:resource="#Person"/>
  <label xml:lang="en">has for friend</label>
  <comment xml:lang="en">relation between a person and another person he or she knows well and regards with affection and
trust.</comment>
  <label xml:lang="fr">a pour ami</label>
  <comment xml:lang="fr">relation entre une personne et une autre personne qui est l'objet d'un attachement privilégié.</comment>
</rdf:Property>

<rdf:Property rdf:ID="name">
  <label xml:lang="en">name</label>
  <label xml:lang="fr">nom</label>
  <comment xml:lang="en">designation of something.</comment>
  <comment xml:lang="fr">désignation de quelque chose.</comment>
</rdf:Property>

<rdf:Property rdf:ID="shoesize">
  <domain rdf:resource="#Person"/>
  <label xml:lang="en">shoe size</label>
  <label xml:lang="en">size</label>
  <label xml:lang="fr">pointure</label>
  <comment xml:lang="en">express in some way the approximate length of the shoes for a person.</comment>
  <comment xml:lang="fr">taille, exprimée en points, des chaussures d'une personne.</comment>
</rdf:Property>

<rdf:Property rdf:ID="age">
  <label xml:lang="en">age</label>
  <label xml:lang="fr">âge</label>
  <comment xml:lang="en">complete existence duration.</comment>
  <comment xml:lang="fr">durée complète d'existence.</comment>
</rdf:Property>

<rdf:Property rdf:ID="shirtsize">
  <domain rdf:resource="#Person"/>
  <label xml:lang="en">shirt size</label>
  <label xml:lang="en">size</label>
  <label xml:lang="fr">taille de chemise</label>
  <label xml:lang="fr">taille</label>
  <comment xml:lang="en">express in some way the approximate dimensions of the shirts of a person.</comment>
  <comment xml:lang="fr">dimensions approximatives des chemises portées par une personne.</comment>
</rdf:Property>

<rdf:Property rdf:ID="trouserssize">
  <domain rdf:resource="#Person"/>
  <label xml:lang="en">trousers size</label>
  <label xml:lang="en">size</label>
  <label xml:lang="fr">taille de pantalon</label>
  <label xml:lang="fr">taille</label>
  <comment xml:lang="en">express in some way the approximate dimensions of the trousers of a person.</comment>
  <comment xml:lang="fr">dimensions approximatives des pantalons portés par une personne.</comment>
</rdf:Property>

<rdf:Property rdf:ID="hasSpouse">
```

```
  <domain rdf:resource="#Person"/>
  <range rdf:resource="#Person"/>
  <label xml:lang="en">has for spouse</label>
  <label xml:lang="fr">est en ménage avec</label>
  <comment xml:lang="en">a person's partner in marriage.</comment>
  <comment xml:lang="fr">le partenaire d'une personne dans un mariage.</comment>
</rdf:Property>

</rdf:RDF>
```

Annexe 4
fichier human_2007_09_11.rdf

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY humans   "http://www.inria.fr/2007/09/11/humans.rdfs">
   <!ENTITY xsd      "http://www.w3.org/2001/XMLSchema#"> ]>

<rdf:RDF
   xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd ="&xsd;"
   xmlns     ="&humans;#"
   xml:base  ="&humans;-instances" >

<Man rdf:ID="Harry">
  <name>Harry</name>
  <hasChild rdf:resource="#John"/>
  <hasSpouse rdf:resource="#Sophie"/>
</Man>


<Person rdf:ID="John">
  <name>John</name>
              <shoesize>14</shoesize>
              <age>37</age>
</Person>

<Person rdf:ID="Mark">
  <name>Mark</name>
              <shoesize>8</shoesize>
              <age>14</age>
              <shirtsize>9</shirtsize>
              <trouserssize>36</trouserssize>
              <hasFather rdf:resource="#John"/>
</Person>

<Person rdf:ID="Eve">
  <rdf:type rdf:resource="&humans;#Lecturer"/>
              <hasSpouse rdf:resource="#David"/>
              <name>Eve</name>
              <hasFriend rdf:resource="#Alice"/>
</Person>

<Person rdf:ID="David">
  <rdf:type rdf:resource="&humans;#Researcher"/>
  <hasFriend rdf:resource="#Gaston"/>
              <name>David</name>
</Person>

<Woman rdf:ID="Alice">
              <hasFriend rdf:resource="#John"/>
              <name>Alice</name>
</Woman>

<Man rdf:ID="Jack">
              <hasFriend rdf:resource="#Alice"/>
              <hasChild rdf:resource="#Harry"/>
              <name>Jack</name>
</Man>

<Woman rdf:ID="Flora">
  <age>95</age>
              <hasSpouse rdf:resource="#Gaston"/>
```

```
		<hasChild rdf:resource="#Pierre"/>
		<name>Flora</name>
</Woman>

<Person rdf:ID="Laura">
		<hasFriend rdf:resource="#Alice"/>
		<name>Laura</name>
</Person>

<Woman rdf:ID="Jennifer">
		<hasSpouse rdf:resource="#John"/>
		<name>Jennifer</name>
</Woman>

<Man rdf:ID="Lucas">
		<shoesize>7</shoesize>
		<trouserssize>28</trouserssize>
		<age>12</age>
		<shirtsize>8</shirtsize>
		<name>Lucas</name>
		<hasMother rdf:resource="#Catherine"/>
</Man>

<Man rdf:ID="Gaston">
  <rdf:type rdf:resource="&humans;#Researcher"/>
		<shoesize>11</shoesize>
		<trouserssize>38</trouserssize>
		<age>102</age>
		<shirtsize>12</shirtsize>
		<name>Gaston</name>
		<hasChild rdf:resource="#Pierre"/>
		<hasChild rdf:resource="#Jack"/>
</Man>

<Lecturer rdf:about="#Laura"/>

<Person rdf:ID="William">
		<hasSpouse rdf:resource="#Laura"/>
		<shoesize>10</shoesize>
		<age>42</age>
		<trouserssize>46</trouserssize>
		<shirtsize>13</shirtsize>
		<name>William</name>
</Person>

<Man rdf:ID="Pierre">
		<shoesize>8</shoesize>
		<age>71</age>
		<trouserssize>30</trouserssize>
		<shirtsize>9</shirtsize>
		<name>Pierre</name>
</Man>

<Person rdf:ID="Karl">
		<hasSpouse rdf:resource="#Catherine"/>
		<hasFriend rdf:resource="#Sophie"/>
		<shoesize>7</shoesize>
		<age>36</age>
		<shirtsize>9</shirtsize>
		<trouserssize>40</trouserssize>
</Person>


<Woman rdf:ID="Catherine">
  <hasMother rdf:resource="#Laura"/>
</Woman>

<Researcher rdf:about="#Laura">
  <name>Laura</name>
</Researcher>

<Person rdf:about="#John">
		<shirtsize>12</shirtsize>
		<trouserssize>44</trouserssize>
		<hasParent rdf:resource="#Sophie"/>
</Person>
```

</rdf:RDF>