A Semantic Query Interface for the OGO Platform

José Antonio Miñarro-Giménez, Mikel Egaña Aranguren,
Francisco García-Sánchez, and Jesualdo Tomás Fernández-Breis

Facultad de Informática, Universidad de Murcia,
CP 30100 Murcia, Spain
jose.minyarro@um.es, mikel.egana.aranguren@gmail.com,
frgarcia@um.es, jfernand@um.es

Abstract. In the last years, a number of semantic biomedical systems have been developed to store biomedical knowledge in an accessible manner. However, their practical usage is limited, since they require expertise in semantic languages by the user, or, in the other hand, their query interfaces do not fully exploit the semantics of the knowledge represented. Such drawbacks were present in the OGO system, a resource that semantically integrates knowledge about orthologs and human genetic diseases, developed by our research group. In this paper, we present an extension of the OGO system for improving the process of designing advanced semantic queries. The query module requires the users to know and to manage only the OGO ontology, which represents the domain knowledge, simplifying the process of query building.

Keywords: Ontology, Semantic Web, Semantic Querying, Biomedical Informatics.

1 Introduction

An ontology is a formalisation of a knowledge domain, a set of concepts and their relationships, on which machines can perform automated reasoning. In the last years, there has been an increasing interest in the development and application of ontologies in biomedical domains, in order to computationally represent and efficiently manage biomedical knowledge. As a representative example, the OBO Foundry community attempts to develop a set of orthogonal biomedical ontologies to support biomedical research[1].

The Knowledge Representation (KR) technology used for building biomedical ontologies is closely related to the Semantic Web. The Semantic Web is a next generation web in which automated processing of information will deliver more concise results to the user, and allow machines to perform time consuming tasks [1]. For the Semantic Web to work, information must be codified with precise semantics, thus in a computationally accessible manner, *via* ontologies.

[1] http://www.obofoundry.org/

S. Khuri, L. Lhotská, and N. Pisanti (Eds.): ITBAM 2010, LNCS 6266, pp. 128–142, 2010.

Semantic Web technologies have been increasingly used for data integration in life sciences and provide a useful framework for translational medicine (see, for instance, [3]). Different Semantic Web technologies such as RDF[2], OWL[3] and SPARQL[4] have been used for developing such semantic biological solutions. OWL, the Web Ontology Language, a W3C[5] official recommendation, is one of the most widely used ontology languages. OWL is designed to act as a standard for producing web interoperable ontologies in the Semantic Web, and it provides an optimal balance between decidability and expressive power.

The Orthology Gene Ontological (OGO) system, developed by our research group, also exploits semantic technologies to integrate life sciences information [6]. The OGO system connects data about orthologous genes and human genetic diseases, facilitating the labour of life scientists. The OGO Knowledge Base (KB) is based on a domain ontology that guides the semantic integration of the information collected from the repositories about orthology and human genetic diseases. The original implementation of the OGO system provided a keyword-based query interface and the semantic query was automatically generated. However, this query interface did not allow users to make complex queries exploiting the semantics of the domain.

Most biomedical semantic systems provide query interfaces based on powerful semantic query languages such as OWL DL or SPARQL, since such languages allow the exploitation of the semantics of the represented domain at query time, by incorporating all the restrictions modelled in the ontology in the queries. These languages are not specially difficult for users with a computing background or with some expertise in relational query languages such as SQL. However, they are currently not usable by the average biomedical researcher, who should be the final user of these semantic systems. Therefore, mechanisms for making query construction simpler for such users are required.

It has already been noted that "*the casual user is typically overwhelmed by the formal logics of the Semantic Web*" [2]. This is due to the fact that ontology users have to be familiar with [14]: (1) the ontology syntax (*e.g.* RDF, OWL), (2) some formal query language (*e.g.* SPARQL), and (3) the structure and vocabulary of the target ontology. In recent years, Controlled Natural Languages (CNL) [10] have received much attention due to their ability to reduce ambiguity in natural language. CNLs are characterised by two properties [11]: (1) their grammar is more restrictive than that of the general language, and (2) their vocabulary only contains a fraction of the words that are permissible in the general language. These restrictions aim at reducing or even eliminating both ambiguity and complexity. In recent years, the use of CNLs in the context of the Semantic Web has received much attention. Several platforms have been developed to work as either natural language ontology editors or natural language query systems. Two important examples in the first category are CNL Editor [8] (formerly OntoPath [5]) and

[2] `http://www.w3.org/RDF`
[3] `http://www.w3.org/TR/owl-features`
[4] `http://www.w3.org/TR/rdf-sparql-query`
[5] `http://www.w3c.org`

GINO [2]. Moreover, our research group has recently developed OWLPath [13]. Such systems rely on the existence of a grammar that guides the querying process and also constrains the expressivity of the queries, therefore not providing rich results for complex domains. As a result, we decided to extend the functionality of the OGO query interface by using the ontology to guide the users in the design of the queries, providing an interface that has the same expressivity of a semantic query language. Such extension is described in this paper.

2 The OGO System

OGO is a system based on Semantic Web technologies that integrates and stores up to date information about orthology and human genetic diseases, continuously updating it. The orthology information (clusters of ortholog genes) is collected and integrated from KOG[6], Inparanoid[7], OrthoMCL[8] and Homologene[9], reducing the heterogeneity and redundancy of such information. The data about genetic diseases is collected from OMIM[10]. The current version of the OGO system[11] contains more than 90,000 ortholog clusters, more than a million of genes, *circa* a million of proteins, approximately 16,000 human genetic disorders and more than 17,000 references to PubMed articles.

The KB is based on the OGO ontology, which imports[12] other bio-ontologies in order to organise, in classes, the information provided by individuals (orthologs, genetic diseases, *etc.*). The imported semantics can be exploited at query time. The imported bio-ontologies are listed as follows:

- Gene Ontology (GO) [7], linking ortholog genes to their molecular function, cellular component or biological process.
- Evidence Codes Ontology[13] (ECO), providing a hierarchy for classifying GOA (Gene Ontology Annotation) evidence codes [4].
- NCBI taxonomy (NCBI) [9], providing a hierarchy for the taxonomic classification of organisms.
- Relationship Ontology (RO) [12], providing a common set of relationships for bio-ontologies, to facilitate integration of the OGO system with other "RO friendly" bio-ontologies and resources.

The OGO ontology[14] (Figure 1) contains the links between orthologous genes but also provides information about other gene-related concepts like organisms, proteins and GO terms. The information about genetic diseases was integrated

[6] http://www.ncbi.nlm.nih.gov/COG/grace/shokog.cgi
[7] http://inparanoid.sbc.su.se/cgi-bin/index.cgi
[8] http://www.orthomcl.org/cgi-bin/OrthoMclWeb.cgi
[9] http://www.ncbi.nlm.nih.gov/sites/entrez?DB=homologene
[10] http://www.ncbi.nlm.nih.gov/sites/entrez?db=OMIM\&itool=toolbar
[11] http://miuras.inf.um.es/~ogo
[12] http://www.w3.org/TR/2009/REC-owl2-syntax-20091027
[13] http://www.berkeleybop.org/ontologies/owl/ECO
[14] http://miuras.inf.um.es/ontologies/OGO.owl

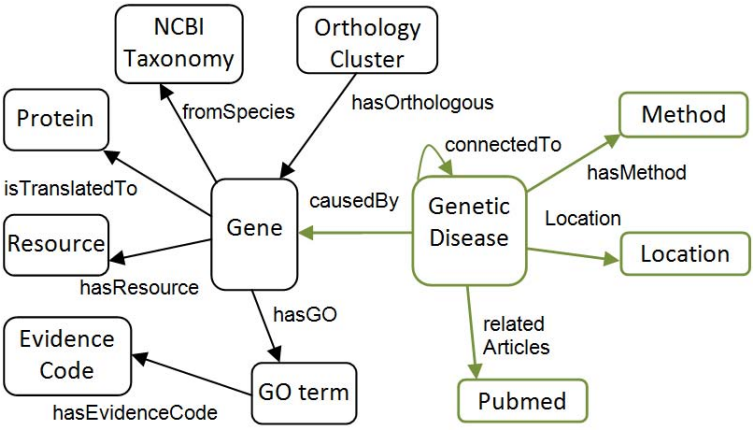

Fig. 1. The OGO ontology

with orthology information using the relationship *causedBy*. Cardinality constraints and disjoint axioms were also defined in the ontology, for example each gene can only belong to one species and also each genetic disease instance has to be caused by at least one gene.

The information was integrated using the OGO methodology [6], which requires the definition of mapping rules between the resources and the OGO ontology. These mapping rules indicate the correspondences between data fields in the source files and the concepts, properties and relations in the ontology. The rules must be defined manually as they depend on the structure of each file belonging to different repositories. The Jena Semantic Web framework[15] was used for manipulating the ontology, building the repository, executing the integration process and implementing the persistence of the model. Jena is an open source Java framework for building Semantic Web applications which provides a programmatic environment for RDF, OWL and SPARQL. The SPARQL functionality is provided by the ARQ module. Since the persistency of the KB is supported by a relational database, the ARQ module translates SPARQL queries into SQL ones in order to retrieve the information from the KB.

2.1 Keyword-Based Query Interface

The first query application interface, developed for querying the OGO repository, was based on keywords to be matched against genes or genetic diseases. Figure 2 shows this interface, which offers two different querying perspectives: orthology and genetic diseases.

The first perspective allows for querying using the ID or names of a gene. The results are its orthologous genes, and those can be filtered by species or the resource from which the orthologous gene was retrieved, using the check boxes. Once the keyword has been input by the user, a SPARQL query pattern (Table 1) is

[15] `http://jena.sourceforge.net/`

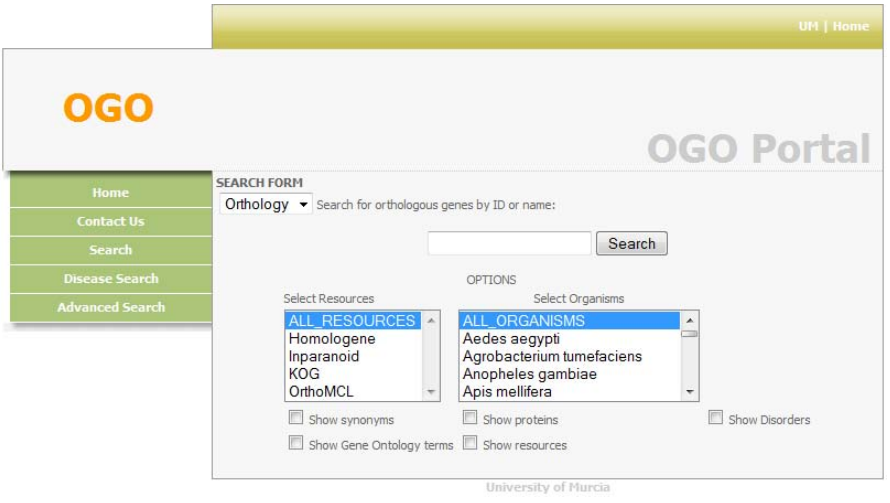

Fig. 2. Keyword-based query interface

automatically applied to create the query. The *<KEYWORD>* corresponds to the gene name or ID, the *<ORGANISMn>* corresponds to the URI of any of the organism in the NCBI taxonomy and the *<RESOURCEn>* corresponds to the URI of any of the orthology repositories.

The genetic diseases perspective allows to query the repository by disease name or OMIM identifier. The orthology information can be reached through this perspective by clicking on the gene name that appears in its results. Table 2 shows the SPARQL query pattern applied to the user input, where *<KEYWORD>* corresponds to the genetic disease name or OMIM identifier.

Table 1. Basic orthologs query pattern

```
@prefix ogo: <http://miuras.inf.um.es/ontologies/OGO.owl>.
SELECT
  ?Gene0
WHERE {
  ?Gene1 ogo:Name ?literal .
  FILTER (regex(?literal, <KEYWORD>)) .
  ?OrthologyCluster ogo:hasOrthologous ?Gene1 .
  ?OrthologyCluster ogo:hasOrthologous ?Gene0 .
  {
    ?Gene1 ogo:fromSpecies <ORGANISM1> .
    UNION
    ?Gene1 ogo:fromSpecies <ORGANISM2> .
    UNION
    ...
  }.
  {
    ?Gene1 ogo:hasResource <RESOURCE1> .
    UNION
    ?Gene1 ogo:hasResource <RESOURCE2> .
    UNION
    ...
  }.
}
```

Table 2. Genetic disease query pattern

```
@prefix ogo: <http://miuras.inf.um.es/ontologies/OGO.owl>.
SELECT
  ?Disease
WHERE {
  ?Disease ogo:Name ?literal .
  FILTER (regex(?literal, <KEYWORD>)) .
}
```

As it can be appreciated by looking at the previous query patterns, the expressivity of the queries is very limited. This was due to the fact that our initial work focused on obtaining the integrated repository and providing an user-friendly query interface for any user. Once these users have realized that the system provides semantically rich information, more powerful methods are required because queries such as "finding all genes orthologous to the gene which is translated to a particular protein" or "finding all the genes belonging to a particular species that are related to the gene that causes a certain disease" cannot be defined with these query interfaces, even though the ontological repository would answer such queries.

2.2 The Semantic Query Interface

In this section, we describe the semantic query interface that we have developed for the OGO system. The interface was developed to assist biomedical users to formulate complex semantic queries. Its architecture is depicted in Figure 3, comprising (1) the communication interface, (2) the Jena module for accessing the semantic RDF/OWL repository, (3) the query ontology, and (4) the guided search subsystem.

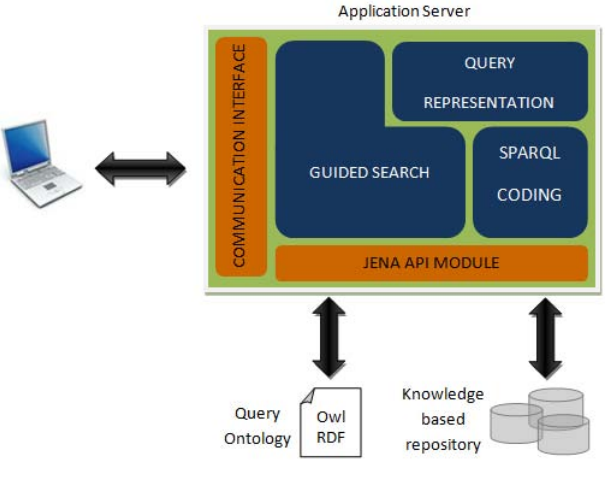

Fig. 3. The architecture of the semantic flexible query system

The web user interface was implemented using Ajax, Javascript, Servlets and JSP. The communication interface module facilitates the communication between users and the server application, *via* Web. The Jena API module allows the combination of the KB with the ontology used for guiding the definition of the semantic query. The query ontology is the same ontology that was used for defining the KB. However, the query ontology can also be customised for improving the user experience. These changes permit to define user-friendly names in the ontology elements to help users identify such elements in the query interface. It is also possible to hide concepts and relationships of the ontology that might not be interesting for querying purposes.

The modules that implement the ontology guided search method are described as follows:

- *Query representation module:* It stores the query defined by the users through the web interface. This module is responsible for ensuring the consistency of the query since it verifies that the values and conditions introduced are valid for the ontology. Thus, this module has to cooperate with the guided search module, which limits the options offered during the definition of the query, and with the SPARQL coding one, which generates the final SPARQL queries (see below).
- *Guided search module:* This module guides users throughout the different stages of the definition of the query and it is also responsible for gathering, formatting and showing the query results to the users. It assists users in the design of the query by managing the available knowledge from the query ontology. Thus, only allowed queries are defined and executed in the repository. Currently, the queries that can be defined form a subset of the potential SPARQL expressivity. The grammar describing the types of SPARQL queries that can be defined is depicted in Table 3. However, some axioms defined in the ontology, such as cardinality or disjointness, and properties such as *label* or *comment*, are not allowed to be retrieved. These elements of the ontology are excluded to simplify the query interface and thus provide users with only the elements of the ontology containing information about the domain.
- *SPARQL coding module:* This module transforms the user-defined query into an optimised SPARQL query. The process is divided in two phases. Firs, the user-defined query is translated into a SPARQL-based query, and then, the SPARQL query is optimised in order to reduce the response time. The way that users define the query condition clauses is very similar to the basic graph pattern of SPARQL, in which the subject, predicate and object of the RDF triple may be a variable. Therefore, the translation is mainly focused on identifying the proper URI of the ontology elements and satisfying the grammar rules of the query language. In terms of optimisation, we sort the different types of condition clauses to make the SPARQL queries run faster. The sorting takes into account the details of our KB (it contains few concepts and relationships compared to the number of individuals). Thus, conditions which contain less variables are planned to be executed first during the sorting phase.

Table 3. Query Grammar

Query::="SELECT" **ListVar** (**WhereClause**)?
ListVar::=Var (**Var**)*
WhereClause::="WHERE {" **ConditionClause** (**ConditionClause**)* "}"
ConditionClause::=[VarCondition | **LiteralCondition]** "."
VarCondition::=[Var | **Individual] Property [Var** | **Individual]**
LiteralCondition::=[Var | **Individual] Property [Var** | **Individual]** "."
"FILTER (regex (" **Var** "," **Literal** "))"
Var → This term represents a variable in the query which can be matched to any concept or individual in the ontology.
Individual → This term represents a concept or individual identified by an URI in the ontology.
Property → This term represents a relationship or property identified by an URI in the ontology.
Literal → This term represents any data value defined by the user.

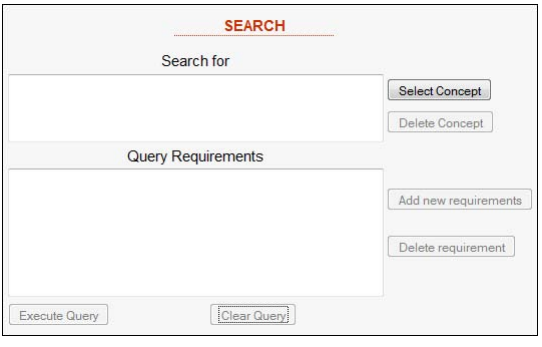

Fig. 4. The web interface for the design of the queries

The ontology guided search user interface is shown in Figure 4. The "Search for" text area contains the variables that are returned by the server application. By clicking on "Select Concept", the ontological tree is shown to the user, who selects the concept to add into the query. An example of the ontology tree is shown in the Figure 5. Later, each selected concept is associated with a unique variable in the query. Therefore, it is possible to select the same concept several times. These selected concepts can be removed from the "Search for" area by clicking on "Delete concept". This is only possible if the concept does not participate in any condition clause defined in the query. Figure 4 also contains the "Execute Query" and "Clear Query" buttons. The "Execute Query" button is activated once the user has added a concept in the query and allows its execution.

Once the concepts have been selected, the users have to define the conditions the data must meet to be included in the list of results. The conditions defined for the current query are shown in the "Query Requirements" area, and they can be added by clicking on "Add new requirement". When this occurs, the list of all allowed conditions for each variable node are shown. The left part of Figure 6 shows the allowed conditions for a query about the variable that represents the *Gene* concept of the ontology. The concept *Gene[0]* has some data type properties associated, such as *Name* and *Identifier*, for which the only allowed values are strings. The *Gene[0]* has also relationships that link a gene to other

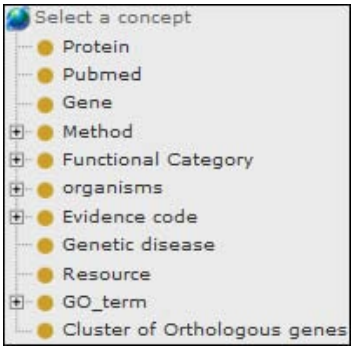

Fig. 5. Example of an ontology tree

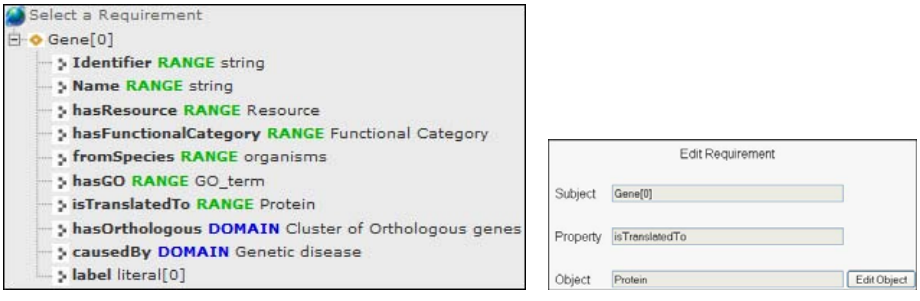

Fig. 6. Defining a requirement

ontology concepts, such as *hasResource* which associates a gene to the repository from which it was collected; the *hasFunctionalCategory* relationship relates a gene to a Functional Category in the ontology; the *fromSpecies* relationship identifies the organism that belongs to the gene; the *hasGO* relationship associates genes with Gene Ontology terms; and the *isTranslatedTo* relationship associates instances of genes with instances of proteins. Finally, the *Gene[0]* participates in some relationships but in its range, such as *hasOrthologous* which relates a *Cluster of Orthologous genes* to its gene instances; and the *causedBy* relationship which associates a *Genetic disease* and the genes which are involved in. Once a property is selected, new options are offered to the user to specify the value condition. This is shown in the right part of Figure 6. In case the property has associated a simple data type such as a string, the user can input the value, such as in *Name* or *Identifier* properties of the Figure 6. In case a relationship relates the selected concept to another one and this has subclasses or instances, the user is given the possibility of choosing one by clicking on "Edit Object", such as in *fromSpecies* or *isTranslatedTo* relationships in Figure 6. Even more, when defining the domain of a relationship, a subclass or individual may also be selected by clicking on "Edit Subject". If the condition adds new

variable concepts in the "Query Requirement" area, then new conditions for these variables will be generated when clicking on the "Add new requirement" button. The variables which only appear in conditions but not in the "Search for" area do not appear in the results table.

Once the query is executed, the results are displayed in a table on which columns are the ontology concepts shown in the "Search for" area, and rows are the clickable and browseable results. Examples of such output are shown in Figures 7 and 8, on Section 3.

3 Query Examples

This section describes how advanced semantic queries can be designed and executed in the OGO system using the new query interface. In previous sections, we mentioned two queries that could not be formulated with the previous interface: (1) Finding all genes orthologous to a gene which is translated to the protein "NP_008816.3" protein; and (2) finding all the genes belonging to a particular species that are related to the gene that causes a certain disease.

The first query illustrates how to obtain orthology information without using gene names. The second query shows how translational bioinformatics can be done with the semantic integration of the repository. It also shows how semantic query languages can be used for retrieving the suitable instances by filtering the repository results with relationships or properties of ontology concepts. Each example can be defined in a single query and therefore researchers can obtain the suitable results in one step avoiding querying several repositories and manually integrating the information.

3.1 Finding All Genes Orthologous to a Gene Which Is Translated to the Protein "NP_008816.3"

Table 4 shows the representation of the query after the selection of the concepts and properties in the user interface. In this query, we want to retrieve all the instances of *Gene* that belong to the same cluster of orthologous genes. Such cluster must contain a gene that is translated into the protein *"*NP_008816.3*"*. This query is translated into the following conditions: (1) A gene, which is represented by the variable *Gene[1],* is translated to the *Protein[2]*; (2) the variable *Protein[2]* has the property *Name* with the *"NP_008816.3"* string value; (3) the

Table 4. Query one: Find all genes orthologous to the gene which is translated into "NP_008816.3" Protein

Search for:
Gene[0]
Query requirements:
Gene[1]→is translated to→Protein[2]
Protein[2]→Name→"NP_008816.3"
Cluster of orthologous genes[3]→Has Orthologous→Gene[1]
Cluster of orthologous genes[3]→Has Orthologous→Gene[0]

variable *Gene[1]* belongs to a cluster of orthologous genes, which is identified by the variable *Cluster of orthologous genes[2]*; and finally, another variable of a gene concept, that is called *Gene[0],* belongs to the same cluster variable, *Cluster of orthologous genes[2].* The [i] suffixes indicate that they refer to different instances among requirements. For instance, the two Gene instances *Gene[0]* and *Gene[1]* refer to different instances but *Cluster of orthologous genes[2]* refers to the same instance.

Therefore, the user defines the query without knowing the full URIs and the amount of manual typing is minimised, thus reducing the possibility of making mistakes. The corresponding automatically generated SPARQL query is shown in Table 5. For the query generation, the "Search for" and "Query requirements" areas are changed into SELECT and WHERE SPARQL clauses. Each concept and property graphically selected are identified in the ontology by means of its URI and are constrained by the corresponding condition clauses in the WHERE area.

Table 5. Query one in SPARQL

```
@prefix ogo: <http://miuras.inf.um.es/ontologies/OGO.owl>.
SELECT
  ?Gene_0
WHERE {
  ?Protein_2 ogo:Name ?literal_4 .
  FILTER (regex(?literal_4, "NP_008816.3")) .
  ?Gene_1 ogo:isTranslatedTo ?Protein_2 .
  ?Cluster_of_Orthologous_genes_3 ogo:hasOrthologous ?Gene_1 .
  ?Cluster_of_Orthologous_genes_3 ogo:hasOrthologous ?Gene_0 .
}
```

The left-side of Figure 7 shows part of the results obtained for this query. It contains the identifiers of all gene instances that belong to the same cluster of orthologous genes and this cluster has a gene that is translated into the protein *"NP_008816.3"*. The interface allows us to navigate through the results and view their properties. An example of this navigation is shown on the right-side of Figure 7: the information about the human gene "*atbf1*", which is translated into the protein "*NP_008816.3*", is shown.

3.2 Finding All the Orthologous Genes of the Gene That Causes Prostate Cancer and belong to *Rattus Norvegicus*

The representation of this query in the interface is shown in Table 6. To define this query, we first added the variables *Genetic disease[0]*, which corresponds to the Genetic disease concept, and *Gene[1]*, which corresponds to the Gene concept. We then linked *Genetic disease[0]* to *Gene[2]* through the *caused by* relationship, and we defined the value "prostate cancer, susceptibility to" for its property *Name*. Finally, the orthologous genes were obtained by means of the definition of the requirements for *Cluster of orthologous genes[3], Gene[1]*

Fig. 7. Query one: Results

Table 6. Query Two: Find all genes orthologous to the genes that cause the Prostate cancer disease and belongs to the *Rattus norvegicus* organism

Search for:
Genetic disease[0]
Gene[1]
Query requirements:
Gene[1]→from species→Rattus norvegicus
Genetic disease[0]→caused by→ Gene[2]
Genetic disease[0]→Name→"prostate cancer, susceptibility to"
Cluster of orthologous genes[3]→Has Orthologous→Gene[2]
Cluster of orthologous genes[3]→Has Orthologous→Gene[1]

Table 7. Query 2 in SPARQL

```
@prefix ncbi: <http://um.es/ncbi.owl>.
@prefix ogo: <http://miuras.inf.um.es/ontologies/OGO.owl>.
SELECT
  ?Genetic_disease_0
  ?Gene_1
WHERE {
  ?Gene_1 ogo:fromSpecies ncbi:NCBI_10116
  ?Genetic_disease_0 ogo:Name ?literal_4 .
  FILTER (regex(?literal_4,"Prostate cancer, susceptibility to")) .
  ?Genetic_disease_0 ogo:causedBy ?Gene_2 .
  ?Cluster_of_Orthologous_genes_3 ogo:hasOrthologous ?Gene_2 .
  ?Cluster_of_Orthologous_genes_3 ogo:hasOrthologous ?Gene_1 .
}
```

and *Gene[2]*, as in the first example. Table 7 shows the SPARQL query that is automatically generated for this particular query. Some results are shown in Figure 8. The left column contains the identification of the *Genetic diseases*, and the right column shows the names of the genes that belong to *Rattus norvegicus* and are orthologous to the genes that cause the genetic disease.

Genetic disease[0]	Gene[1]
104155, prostate cancer, susceptibility to, zinc finger homeobox 3	pex12, 116718
	rgd1560268, zfhx3.predicted, 307829
	rgd1563022, zfhx4.predicted, 310250
600020, neurofibrosarcoma, prostate cancer, susceptibility to, max-interacting protein 1	clec5a, 679787
	ensmog00000026306, loc684510
	loc689617, 689617
	loc689629, 689629
	max, mgc124611, 60661
	mxi.wr, mxi1, 25701
	tgap1, 294892

Fig. 8. Query Two: Results

4 Conclusions

Providing usable and flexible mechanisms for querying biomedical semantic repositories is fundamental for the success of the application of the Semantic Web in life sciences. To date, many efforts have been invested on the semantic representation and integration of biomedical data, so very interesting semantic repositories have been generated. However, most of the currently available systems have problems for facilitating the definition of expressive, semantic queries. This is due to the fact that they require knowing complex languages such as SPARQL or limit the expressivity of the queries, as it happens with most CNLs.

We have experienced ourselves this problem in the development of the OGO project, which has generated a large integrated KB of biomedical knowledge relating orthology information with human genetic diseases. However, the query and exploitation mechanisms of such knowledge and data were limited in the initial version of the OGO system. To overcome this flaw, we have presented in this paper a system that guides users in the design of powerful semantic queries. Our main goal was to facilitate the exploitation of our semantic repositories by biomedical researchers.

In this work, we present our initial results, moving the complexity of using the OGO system from mastering semantic languages such as SPARQL to working with a domain ontology. Although the researchers need to know the ontology and its structure, they do not have to define the query manually. In fact, using a SPARQL-based query interface does not only require expertise in SPARQL but also expertise in the underlying ontology. Our system reduces the need to know precisely the concepts and how they are related since these are displayed to the user during the query definition. In addition, the need to be an expert on SPARQL is eliminated because the interface guides the definition of the query by enabling/disabling options. This query design assistance is performed by using the OGO ontology. Consequently, the learning curve for executing advanced queries on the semantic repository is reduced. Even more, the interface avoids

the misuse of the relations and properties between concepts in the ontology and typos in queries. Therefore, the interface guides the researchers for defining the query properly.

An interesting property of this query module is the fact that it is a generic solution. Although it has been developed for and applied to the OGO system, it is possible to use it for any semantic system that uses SPARQL-based queries. This is possible because we would only need to change the ontology that guides the process and to connect the system to the new semantic repository. We are applying it to our repository of clinical archetypes. We would like to allow users to define semantic queries without noticing that they are using an ontology, by reducing the number of clicks and selections for adding a particular ontological entity or requirement into the query. We are also exploring the possibility of using a reduced version of the OGO ontology for guiding the process due to the overload produced by handling large amounts of data managed and transferred during the design of the query. The ontology can also be simplified to constraint the queries to a particular set of concepts and properties. Furthermore, the definition of labels in the ontology concepts, relations and properties would make it possible to show alternative names of these entities, so friendlier names would be shown to the users and multilingual interfaces could be easily developed.

Acknowledgements

This work has been funded by the Spanish Ministry for Science and Education through grant TSI2007-66575-C02-02 and the Comunidad Autónoma de la Región de Murcia through grant BIO-TEC 06/01-0005. JA Miñarro is supported by the Fundación Séneca and the Servicio de Empleo y Formación through the grant 07836/BPS/07.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (2001)
2. Bernstein, A., Kaufmann, E.: GINO - A Guided Input Natural Language Ontology Editor. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 144–157. Springer, Heidelberg (2006)
3. Bodenreider, O., Sahoo, S.S.: Semantic Web for Translational Biomedicine: Two Pilot Experiments. In: Proceedings of the AMIA Summit on Translational Bioinformatics, vol. 148 (2008)
4. Camon, E., Magrane, M., Barrell, D., Lee, V., Dimmer, E., Maslen, J., Binns, D., Harte, N., Lopez, R., Apweiler, R.: The Gene Ontology Annotation (GOA) Database: sharing knowledge in Uniprot with Gene Ontology. Nucleic Acids Res. 32, D262–D266 (2004)
5. Jiménez-Ruiz, E., Berlanga, R., Nebot, V., Sanz, I.: OntoPath: A Language for Retrieving Ontology Fragments. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 897–914. Springer, Heidelberg (2007)

6. Miñarro Giménez, J., Madrid, M., Fernández-Breis, J.: OGO: an ontological approach for integrating knowledge about orthology. BMC Bioinformatics 10, S10–S13 (2009)
7. GO, C.: Gene Ontology: tool for the unification of biology. Nature Genetics 23, 25–29 (2000)
8. Namgoong, H., Kim, H.G.: Ontology-based Controlled Natural Language Editor Using CFG with Lexical Dependency. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 351–364. Springer, Heidelberg (2007)
9. Sayers, E., Barrett, T., Benson, D., Bolton, E., Bryant, S., Canese, K., Chetvernin, V., Church, D., Dicuccio, M., Federhen, S., Feolo, M., Helmberg, W., Geer, L., Kapustin, Y., Landsman, D., Lipman, D., Lu, Z., Madden, T., Madej, T., Maglott, D., Marchler-Bauer, A., Miller, V., Mizrachi, I., Ostell, J., Panchenko, A., Pruitt, K., Schuler, G., Sequeira, E., Sherry, S., Shumway, M., Sirotkin, K., Slotta, D., Souvorov, A., Starchenko, G., Tatusova, T., Wagner, L., Wang, Y., Wilbur, J., Yaschenko, E., Ye, J.: Database resources of the National Center for Biotechnology Information. Nucleic acids research (2009)
10. Schwitter, R.: A controlled natural language layer for the semantic web. In: Zhang, S., Jarvis, R.A. (eds.) AI 2005. LNCS (LNAI), vol. 3809, pp. 425–434. Springer, Heidelberg (2005)
11. Smart, P.: Controlled natural languages and the semantic web. Tech. rep., School of Electronics and Computer Science, University of Southampton, Technical Report ITA/P12/SemWebCNL (2008)
12. Smith, B., Ceusters, W., Klagges, B., Kohler, J., Kumar, A., Lomax, J., Mungall, C., Neuhaus, F., Rector, A., Rosse, C.: Relations in Biomedical Ontologies. Genome Biology 6, R:46 (2005)
13. Valencia-García, R., García-Sánchez, F., Castellanos-Nieves, D., Fernández-Breis, J.: OWLPath: An OWL Ontology-Guided Query Editor. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans (2010), doi:10.1109/TSMCA.2010.2048029
14. Wang, C., Xiong, M., Zhou, Q., Yu, Y.: PANTO: A Portable Natural Language Interface to Ontologies. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 473–487. Springer, Heidelberg (2007)