Enabling Keyword Search on Linked Data Repositories: An Ontology-Based Approach

Carlos Bobed* Guillermo Esteban
Eduardo Mena
IIS Department
University of Zaragoza
50018 Zaragoza, Spain
{cbobed, 529679, emena}@unizar.es

Abstract

The Web is experiencing a continuous change that is leading to the realization of the Semantic Web. Initiatives such as Linked Data have made a huge amount of structured information publicly available, encouraging the rest of the Internet community to tag their resources with it. Unfortunately, the amount of interlinked domains and information is so big that handling it efficiently has become really difficult for final users. Thus, we have to provide them with tools to search the needed resources in an easy way.

In this paper, we propose an approach to provide users with different domain views on a general data repository, enabling them to perform both keyword and refinement searches. Our system exploits the knowledge stored in ontologies to 1) perform efficient keyword searches over a specified domain, and 2) refine the user's domain searches. In this way, we enable the definition of different semantic views on Linked Data datasets without having to change the original semantics. We present a prototype of our approach that focuses on the case of DBpedia, which provides a semantic way to access to Wikipedia.

1 Introduction

The Web has made a huge and ever-growing amount of information available to its users. The appearance of new interaction paradigms that the so-called Web 2.0 introduced, in which the Web users become part of the information providers, has made this amount even bigger and more difficult to handle. At this point, the Semantic Web [17] has been proposed in order to relieve the user of the burden of processing the available information. By sharing definitions

*Corresponding author. Tel.: +34 976 76 24 72, fax: +34 976 76 19 14.
To be published in KES Journal, IOS Press, Netherlands ISSN:1327-2314

and tagging resources, the Internet is being made understandable to computers, thus, allowing them to process the information on behalf of users.

This progressive structuring is being made using ontologies (which offer a formal, explicit specification of a shared conceptualization [10]) and a set of different technologies built around them, promoted and supported by the W3C [1]. Initiatives such as Linked Data [6] have already made a huge amount of structured information publicly available, providing schemas and data in machine-readable formats; and, now it is the time to exploit all these information. Unfortunately, it is not realistic to expect that all the resources in the Internet will be perfectly described and annotated, as there are also huge amounts of information that are not semantically annotated as well. So, for the time being, both dimensions (*semantic* and *syntactic* ones) of the Web are condemned to coexist with each other. And so their different techniques and methods are.

In particular, the adoption of keyword-based search interfaces has spread widely in the last few years, mainly due to their ease of use. However, this ease of use comes from the simplicity of its query model, whose expressivity is low compared with other more complex query models [15]. This lack of expressivity leads to ambiguity when querying, and therefore, to non-satisfactory results. On the other hand, formal query languages, despite of offering the expressivity needed to describe the information need more accurately, are really hard to learn and we cannot expect final users to use them. The sweet spot would be keeping the usability of keyword interfaces, while achieving the expressivity and precision of formal languages. To manage it, several approaches have been proposed to perform *keyword query interpretation* [9], which is the process to translate the input set of keywords into a structured query. However, the current methods require highly expressive ontologies [8] or building large graphs out of the underlying data [18], which might not be completely available to certain users (e.g., when we only have access to a single data endpoint to which pose our queries). In the Linked Data scenario, the data to be searched is stored in RDF[2] format. The most basic representation data unit in RDF is the triplet, $< a\ R\ b >$, where a is the subject, b the object and R is the property that links them. So, at first sight, if we wanted to perform a keyword search on data in RDF format, we would have to search on all the elements of all the triples, which would lead to unbearable query processing times. Fortunately, we can take advantage of the structure of data stored in RDF to make keyword search manageable and enhance it.

In this paper, we present a system that adopts a hybrid search strategy which exploits the knowledge stored in ontologies to focus and enrich the search process on structured data. Our system builds on an external Linked Data repository (which might not be under our control) and takes as input an ontology which has two roles in the system: 1) to define the concept hierarchy of the search domain, guiding and narrowing the scope of the keyword-based search; and 2) to define the structure of the objects in the search domain, helping refining and

[1] `http://www.w3.org`

[2] RDF Resource Description Framework, `http://www.w3.org/TR/rdf-primer/`

suggesting further search results. With our approach, one can provide different views on a general data repository by just adapting externally the ontology provided. Moreover, our approach can be attached to any public SPARQL endpoint without overloading it (this is important in open scenarios, such as the one depicted by Linked Data). We use the DBpedia [7] as data repository example as it provides us with a semantic entrance to the Wikipedia[3].

The rest of the paper is as follows. In Section 2, we overview the architecture of our search system. The definition of the search domain using an ontology is explained in Section 3. In Section 4, we focus on how our system uses the ontology to enhance the search. In Section 5, we present the prototype we have developed over the DBpedia public endpoint. Some related work is discussed in Section 6. Finally, the conclusions and future work are drawn in Section 7.

2 Architecture of the System

In this Section, we overview the architecture that allows our system to exploit the information in the external Linked Data repository. As shown in Figure 1, there is a previous offline step that consists of defining the search domain and providing it modeled in an ontology. Our system uses an inner Description Logics reasoner [3] (DL reasoner from now on) to exploit the information in this ontology. Once it has the Domain Ontology, our system offers two different but complementary kinds of search depending on the user's input:

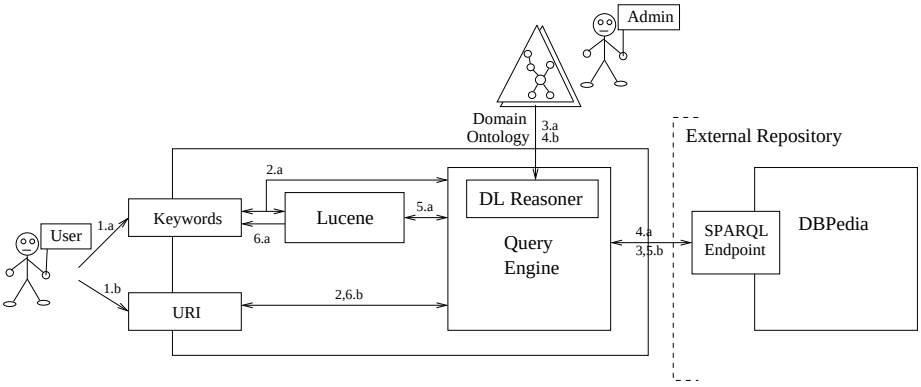

Figure 1: Our system provides two complementary search services: a) Keyword-based and b) URI refining services.

a) *Keyword-based Search.* This search service takes as input plain keywords (step 1.a) and checks out whether the search has been processed before (step 2.a). The system has an internal Lucene[4] repository that serves as

[3] http://wikipedia.org

[4] `http://lucene.apache.org/core/`

a cache memory to alleviate the workload of the external public endpoint (which is not under our control and might have limited availability). If the search has not been performed before, the keyword query is forwarded to our Query Engine which consults the concept hierarchy of the ontology to build a focused SPARQL query (step 3.a). This hierarchy includes only the objects we want to be searched, this is, the objects that we define in the search domain. If it is too large, there exists the possibility of specifying a class of the Domain Ontology to serve as top node of the focused search. Once the Query Engine has built the query, it poses it to the public endpoint to perform the actual search (step 4.a). When the results are retrieved, they are stored in our Lucene repository to cache them for future searches (step 5.a). The Lucene repository acts a cache for future searches and provides the system with relevance measures and ranking on the results. Finally, the results (a set of URIs) are returned ranked according to their relevance (step 6.a).

b) *URI Refining Search.* The results of the previous search service is a ranked set of URIs, which are presented to the user so s/he can explore them. When the user selects a URI (step 1.b), it is directly forwarded to the Query Engine (step 2.b). The Query Engine consults the data repository to obtain the type of the object behind the URI (step 3.b). Then, it consults the definition of its type (step 4.b) to build a set of specialized queries for that type of object (it consults the relevant properties[5] to retrieve the appropriate data and suggest other related objects) with the help of a DL reasoner. Finally, it forwards these queries to the data endpoint (step 5.b) and returns the data (step 6.b). This step is not cached as they are more specialized queries that are not so time consuming as a general search over the whole domain.

Notice that the Lucene repository is only used to cache and rank the results obtained from the actual query on the external Linked Data repository. We assume that the results can be cached as the Linked Data repositories are in fact quite stable in time (e.g.: there was a lapse of seven months from the release of version 3.6 of DBPedia to the 3.7 one, and a year between 3.7 and the latest one, 3.8). Anyway, our system can be easily adapted to another scenarios where data are more volatile by deactivating the caching mechanism. In the following sections we explain how the search domain has to be defined, how the system builds the queries and performs the actual searches, and finally, we present a prototype that we have developed applying our approach to the use case of the DBpedia.

3 Defining the Search Domain

The search domain is defined by using an annotated ontology provided by the administrator of the system (see Figure 1). This ontology provides an adaptable

[5]They are marked as being relevant during the ontology definition, so the Query Engine can be aware of them.

view on the underlying data and has to be aligned to the ontology that describes the actual data repository. In this way, our system can consider only part of the data while being able to access it properly. In fact, although it can be built from scratch, we advocate for using ontology extraction techniques [14] to obtain a module and, then, make our system work directly with a subontology of the repository's one.

The provided ontology tells our system which information is relevant to build the actual queries that are going to be posed to the external data repository. In particular, our system considers the following information:

- The *concept hierarchy* that is defined in the Domain Ontology. It contains the objects that are considered by our system in the searches. The size of this hierarchy is not a problem, as our system can receive an extra parameter to consider any concept as top node and thus focus the search only on its descendants.

- The *annotated properties*. We introduce four different annotations that make the system consider the defined properties to different purposes during the search process:

  - @dataRetrievableProperty: This annotation tells the system to retrieve the values of the property for the returned results. It allows to improve the information about the returned results.
  - @kwdSearchField: With this annotation, we mark the *keyword searchable* properties of the objects, i.e., the properties that have values that can be processed to perform keyword searches. These properties are the ones that are considered for the actual keyword search.
  - @relevantProperty: These properties are consulted in the refining step to further propose relevant results. The system uses them to build up relevant property chains that are the basis for the refinement queries.
  - @queryLanguage: The *language* that has to be considered when consulting the value of this property. When dealing with multilingual repositories, such as DBpedia, we can restrict the language to be considered with this annotation.

The Query Engine exploits this information to build scoped queries on the structured data. Depending on the underlying repository, we can specify via the annotations the different fields which we can perform the searches on and the objects that are relevant to the domain view we want to offer. In Figure 2, we can see an example of an annotated ontology[6].

The keyword search is marked to be performed on the descriptions of *Agent*, which subsumes both *Person* and *Company*. The values of the properties marked as retrievable will be returned along with the instances that conforms

[6]Note that it is only an excerpt of an sample ontology to illustrate the domain definition process, it is not meant to be an example of a complete domain definition.

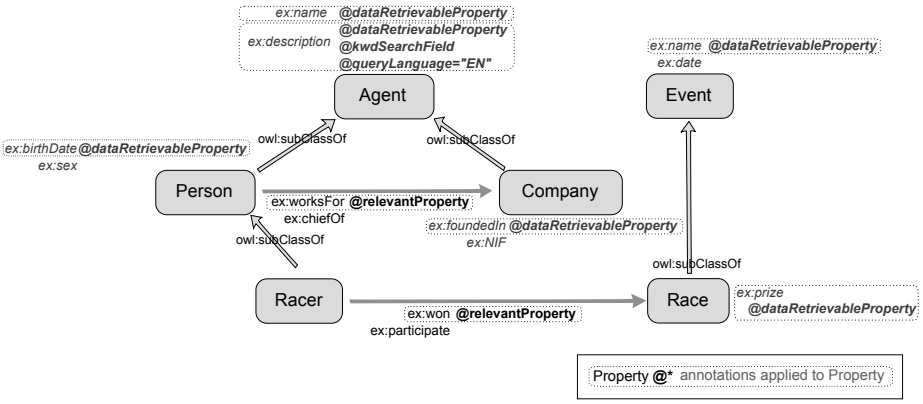

Figure 2: Example of an ontology defining a possible search domain (excerpt).

the answer. For example, when the resources our system returns belong to *Person*, their *ex:birthDate* will also be returned together. Finally, the properties such as *ex:won* and *ex:worksFor* will be used to further refine the queries and provide information about a particular instance of *Racer*.

4 Searching on Structured Data

Once we have provided the annotated ontology, our system is able to perform two types of searches on the external Linked Data repository: a keyword search and a URI refining search. Both searches exploit the ontological definition of the domain we have provided but in different ways, as we detail in the rest of the section.

4.1 Keyword Search

When the user poses a keyword query, the Query Engine consults the Domain Ontology to obtain information about:

- The hierarchy of the objects to look up. Exploiting the semantic structure of the Linked Data, the Query Engine is able to focus the search only on the objects that are relevant to the defined domain. Thus, our system pre-filters the triplet values to be checked against the keyword query.
- The properties whose values are to be retrieved along with the results. The system consults the annotations to know which properties have to be included along with the URI of the resources that will conform the answer.
- The properties on which it has to perform the keyword search. Due to the structure of Linked Data (it follows the RDF model based on triplets) the system would have to look for the keywords in every element of the

triplets (subject, property, and value) if we do not specify which properties to search on. This would lead to unbearable query processing times (note that our system builds on public endpoints which data are not under our control).

With this information, and for each of the input keywords, the Query Engine builds a SPARQL query to be posed to the public endpoint. This is done separately for each of the input keywords to be able to re-use results from one query to another as will be explained later. In Figure 3, the basic structure of these queries and how they are affected by the consulted information is shown:

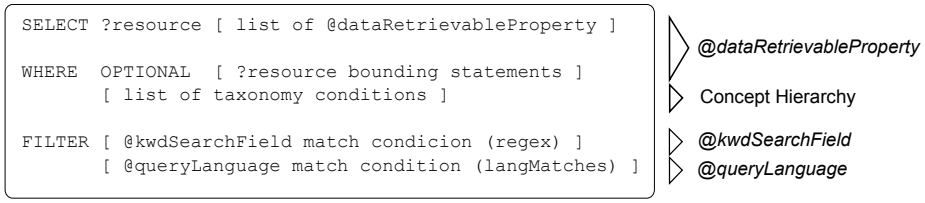

Figure 3: Basic structure of the keyword-based SPARQL query built by our system.

- The SELECT clause comes defined by a free variable for the URI of the answer resources and it is augmented with the set of properties marked as *retrievable* (@dataRetrievableProperty). To bind these properties to the returned resources, a condition for each of them is added to the WHERE clause (marked with the OPTIONAL flag to return the resources whose values for those properties remain unknown). The DL reasoner [3] allows us to check whether the properties to be consulted are compatible with the concepts that are consulted in the query.

- The WHERE clause is extended to focus the search only on the resources that belong to the concepts that are included in the concept hierarchy of the Domain Ontology. The system limits the possible candidate resources by forcing them to be instances of any of the objects included in it. Querying a smaller set of resources will provide more accurate results according to the selected knowledge context[7]. This limited search space will also impact the performance of the keyword search query, reducing the time required. The information of the concepts to be included is obtained with the help of a DL reasoner, which provides us with the actual concept hierarchy. Our system retrieves the relevant objects even when the underlying repository has not classifying capabilities. Finally, if the hierarchy is quite big, and without loss of generality, the system can accept a category/concept of the domain search to provide a more focused search scope.

[7]The resources to be checked are only the ones that are in the domain search, which are a strict subset of the whole dataset.

- A FILTER condition is added for each of the properties that are searchable on. To perform the actual keyword search, our system uses the *regex* function to look for such keywords in the property values. This FILTER is extended with a *lang* condition when the language that has to be considered for it is specified with the corresponding annotation.

Given the sample ontology of the previous section, an actual keyword-based query is shown in Figure 4. The three properties that were marked as @dataRetrievable for *Person* are bounded to the resources via the three first conditions on the WHERE clause. Note that, in spite of specifying *Person* as the root concept to focus the search, the DL reasoner has provided us with the retrievable properties that were defined for *Agent*, as it is its super class. Moreover, the conditions with the *is_a* property, retrieve all the objects that belong to *Person*.

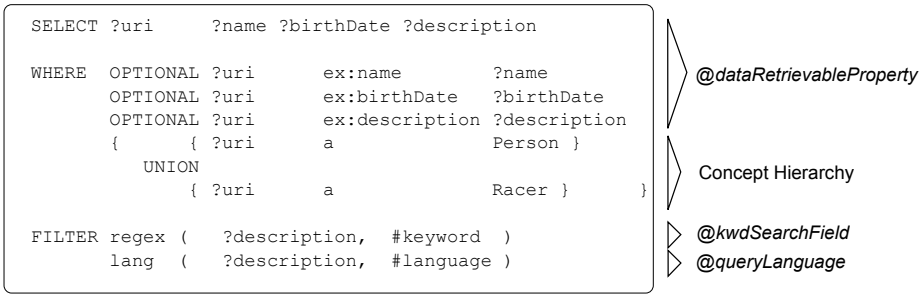

Figure 4: Example of an actual built query given the sample ontology (with *Person* as root concept).

The result of the built queries are a set of tuples $< URI, \{kwdSearchField_i\}, \{dataRetrievableProperty_j\} >$ for each keyword. Before presenting them to the user, all the results are inserted on a local Lucene repository along with the keyword query that has led to them. Lucene ranks them according to the relative frequency of the input keywords in the retrieved values of the properties that we marked in the ontology for the keyword search (this is, $\{kwdSearchField_i\}$). Our system, then, consults it with the whole set of input keywords, benefiting from this step in two ways:

- On the one hand, it obtains a ranked set of resources that have been retrieved due to their semantic belonging to the search domain. This is, our system uses Lucene to rank the results according to their relevance to the input keywords using the several well-known metrics it provides; however, note that this ranking is performed only on semantically related resources (the system has forced it via the query it has built for each keyword).
- On the other hand, the Lucene repository serves as a cache for further queries, alleviating the dependence on the processing resources of the public endpoints (which availability might be even compromised).

So, in the end, our system provides the user with a ranked set of semantically related resources, taking advantage of both semantic knowledge and information retrieval techniques.

4.2 URI Refining Search

The result of the previous keyword search is a set of semantic resources, identified each one by a URI. Once the user selects a resource, the system performs a URI refinement search to suggest further related results. This refinement is performed via the properties that have been annotated as relevant in the domain search definition.

Firstly, the Query Engine asks the underlying repository for the concept which the resource belongs to to know its defined properties. Then, it consults the DL reasoner to obtain the relevant properties[8] that are compatible with its definition. Once the system has the definition of the properties, it retrieves their values for the input resource. In the meantime, the system builds property compositions, that is, chains of properties whose consecutive ranges and domains are compatible[9]. These compositions are suggested as possible queries to retrieve further results. All the semantic checkings of the domains and ranges of the involved properties are performed with the help of the DL reasoner, and so, our system can avoid possible inconsistent queries (according to our Domain Ontology).

The results of this kind of search lead to more resources that can be navigated again and so on. In this way, our system keeps on providing results that are related to the resource without leaving the search domain. In Figure 5, two examples of refinement queries are shown.

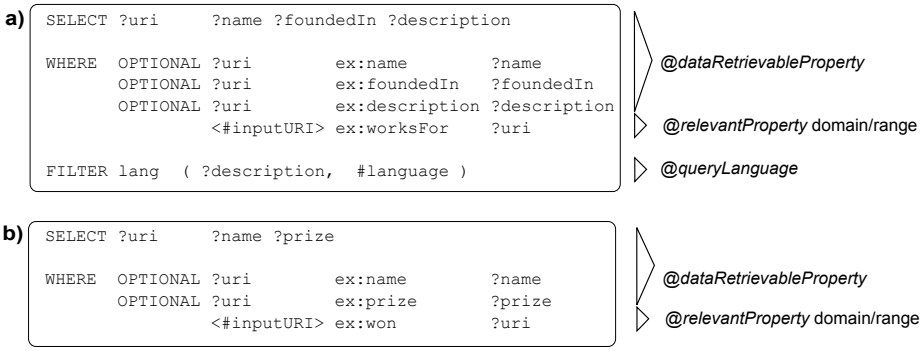

Figure 5: Example of refinement queries for a Racer URI: a) retrieve the companies s/he works for and b) retrieve the races s/he has won.

[8]Note that the Domain Ontology is mapped to the underlying repository, so we assume that we have the information needed to perform the proper translation.

[9]The length of the path is restricted to avoid infinite loops; it has been defined as a system configuration parameter.

Note how in the first query, as the resources to be returned are also *Agents*, the properties to be retrieved include *ex:description* and *ex:name*. As it was marked with the query language annotation, we restrict the returned values to be in English. In these examples, the related resources are a just one property distance from the refined resource, however, we could let the system form different property paths to explore the relationships at deeper levels.

5 Use Case: The DBpedia

In this Section, we present a prototype we have developed using the DBpedia [7] as external data repository and *'Mechanics'* as domain search. Firstly, we give an overview on the structure of DBpedia to explain how our system handles it to understand some design decisions we made during the development of the prototype. Secondly, we show the implementation details of the prototype and how it handles the special case of the DBpedia.

5.1 Structure of DBpedia

DBpedia is a project that extracts structured data from Wikipedia, and makes this information available on the Web under the principles of Linked Data. This extraction is performed automatically by exploiting the structure of the information stored in Wikipedia. However, the nature of the results of this extraction process differs from the sources in more ways than barely structural and format ones.

When moving from the *article world* of Wikipedia to the *semantic resources* in DBpedia, there are objects that might augment their descriptions as the new semantic model can represent more information about them. The articles extracted from Wikipedia, once in DBpedia, become *resources*. Each resource is represented by a URI and has a direct correspondence to its original Wikipedia's article, inheriting its categorization. The whole taxonomy of article categories of Wikipedia is included as an SKOS[10] ontology in DBpedia; thus, DBpedia provides a first view on the resources according their category.

Depending on the content of its corresponding article, a DBpedia resource might also be representing an object (see Figure 6). The classification of this object dimension of the resources is done via several general domain ontologies, being DBpedia Ontology[11] and YAGO[12] the most important ones. In this way, independently of the article categorization, DBpedia offers a second different view based on the nature of the underlying resources. However, this view does not cover all the DBpedia. There exist resources that, despite being categorized, do not have these descriptions as they are not defined in the used ontologies, as shown in Figure 6.

[10]SKOS Simple Knowledge Organization System, `http://www.w3.org/TR/skos-primer/`
[11]The DBpedia Ontology, `http://wiki.dbpedia.org/Ontology`
[12]YAGO Ontology, `http://www.mpi-inf.mpg.de/yago-naga/yago/`

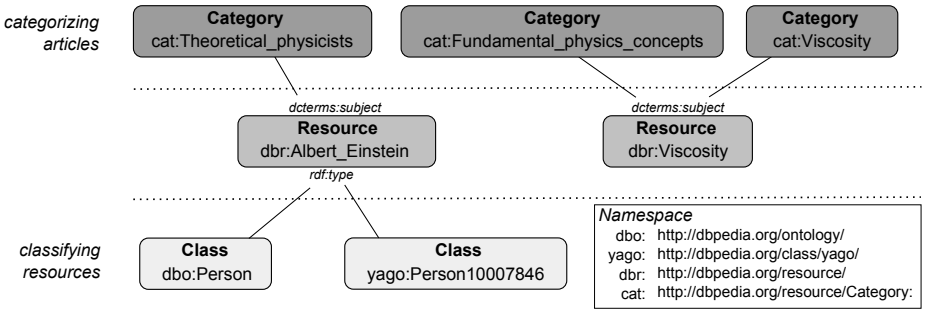

Figure 6: DBpedia excerpt of the descriptions of *Albert Einstein* and *Viscosity* resources.

Summing up, DBpedia organizes knowledge in two major ways: the SKOS categorization, and an ontological classification. Both of them can be used to define the domain hierarchy in our prototype, depending on what kind of resources we want to search on. However, as we are dealing with *Articles* (which are the most general type of resources in DBpedia), the natural way of focusing the search is through the SKOS categorization (i.e., the different categories of the articles).

5.2 Implemented Prototype

In our prototype, we have considered to perform the keyword search on *Articles*. Thus, as it works with the DBpedia as underlying data repository, we have considered the SKOS categorization as the general taxonomy that must be pruned to focus the keyword search. On the other hand, for refinement purposes, we obtain the ontological definitions from the DBpedia ontology (for those articles that are about resources which have an ontological classification). Far from being a drawback to our approach, this shows the flexibility that it gives us to define the search domain (the Domain Ontology in Figure 1 can be divided to focus on different aspects of the object definitions). In particular, we have pruned the SKOS categorization to deal with the categories under '*Mechanics*', and we are only interested, in the refinement process, in *People*, *Institutions* and the different articles that could be related to each resource. The keyword search is performed on the *abstract* property, which gives an excerpt of the Wikipedia entry associated to each resource.

The main adaptation that we have had to do to our prototype to deal with the special case of the DBpedia is to change how to handle the concept hierarchy. The properties that define the hierarchy depend on the modeling language that has been selected to express the ontology. In OWL[13] (e.g., when using the DBpedia ontology to obtain the hierarchy), the main property is the *is_a* property (subclassOf), while in SKOS the hierarchy is modeled via the *broader* and

[13]OWL Web Ontology Language, `http://www.w3.org/TR/owl-primer/`

narrower properties. So, depending on the modeling language, our prototype adopts one or another to build the constraints on the query.

In Figure 7, we can see a snapshot of our prototype, which is deployed at `http://sid.cps.unizar.es/HybridKeywordSearch`. It has been developed with Java 1.6, using HermiT 1.3.5[14] as DL reasoner, and Lucene 3.6.0[15]. The handling of ontologies and SKOS documents has been performed using OWL API v3[16] and SKOS API v3[17]. To form and pose the SPARQL queries, Jena 2.6.4[18] has been used. Finally, the DBpedia has evolved from version 3.6 to 3.8 during the development of the prototype (this affected mainly to the actual properties used in the DBpedia to describe several facts).

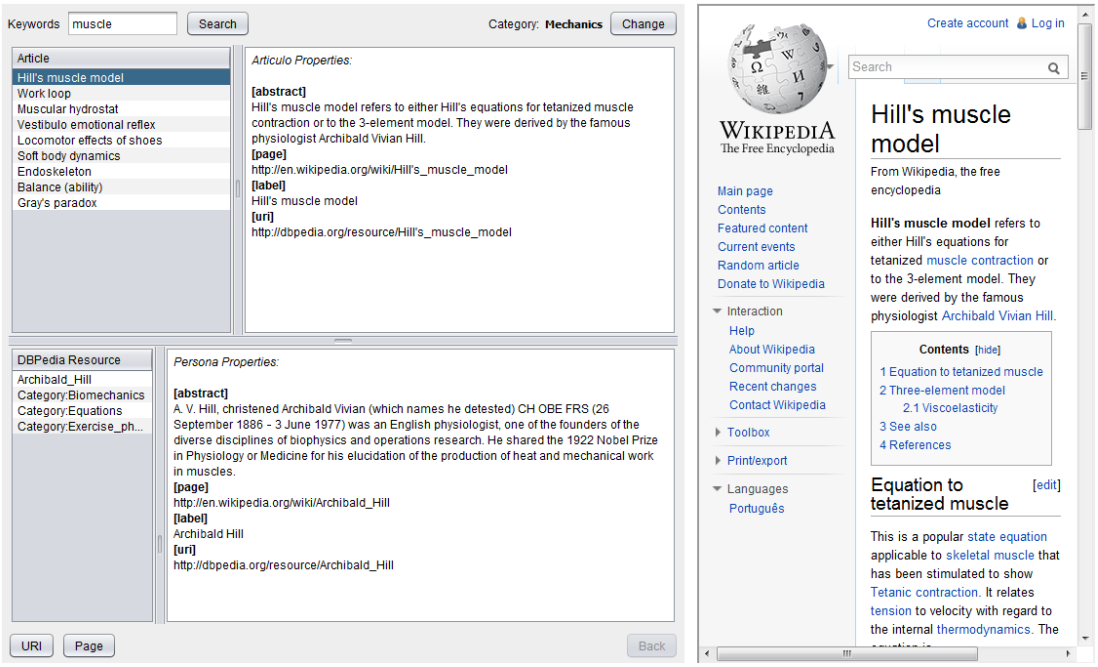

Figure 7: Snapshot of the developed prototype (showing the results for *'muscle'* in *'Mechanics'*).

In Table 1, the results retrieved for the input '*fish movement*' are shown (they are ranked downwards). We can see how all the results that retrieved by our system are related to the specified search domain '*Mechanics*'. However, when we perform the same search directly on the Wikipedia, we obtain results that would be out of our interests: the first two results are related to biology,

[14] `http://hermit-reasoner.com/`
[15] `http://lucene.apache.org/`
[16] `http://owlapi.sourceforge.net/index.html`
[17] `http://skosapi.sourceforge.net/index.html`
[18] `http://sourceforge.net/projects/jena/files/Jena/Jena-2.6.4/`

the third one is about a specific species of fish, the fourth is about a fishing artifact, the fifth about an island in Minnesota, and so on. While our system was able to focus on the domain we defined, Wikipedia always considers any domain in the search. Note that we have defined the search domain without modifying anything on the actual data (they are not under our control). Also note how the change of version in the DBpedia has led to different results in our system, as the values for the properties have changed. However, it still holds that all the results are within the search domain[19].

Our System		Wikipedia
DBpedia 3.7	DBpedia 3.8	
tripedalism	dynamical system	fish migration
fish locomotion	König's theorem (kinetics)	lateral line
role of skin in locomotion	Vestibulo-emotional reflex	Murray cod
aquatic locomotion	Journal of Applied Biomechanics	fishing lure
dynamical system	Long period ground motion	Bear Island
...	...	...

Table 1: Results retrieved for the input keywords '*fish movement*'.

Finally, following with this example, the user could select '*dynamical system*' out of the resources that the keyword search has retrieved. The results of the refinement will depend on how the Domain Ontology was defined. In this case, we define two properties for articles: *knownFor* and *subject*, that represent people related to the article and the categories where the article is included. Now, the refined results include the following people (in both 3.7 and 3.8 versions): Krystyna Kuperberg, Jean-Christophe Yoccoz, Yakov G. Sinai, Denis Blackmore, Bill Parry (mathematician), John Guckenheimer, ...; and categories: Systems, DynamicalSystems, and SystemsTheory. This refinement allows the user to navigate within the domain, finding resources that otherwise would have been hidden by more popular results.

6 Related Work

When it comes to accessing semantic data from a set of keywords, there are quite a lot different approaches, but most of them start with query building step which translates the input into an structured query [9, 16, 18, 8]. In [16], the input is matched to semantic entities by means of text indexes, and then a set of predefined templates is used to interpret the queries in the language SeRQL. However, in this system, the user has to be aware of the underlying data schema to be able to query as, at least, one of the keywords has to be matched to a class in the ontology that describes the underlying data. Moreover, the search

[19] A document with further examples as a proof of concept can be found in `http://sid.cps.unizar.es/HybridKeywordSearch-data/KES2012-KwdEvaluation.pdf` .

domain and the properties to be used cannot be adapted as it can be done in our system.

Other relevant systems in the area of semantic search are SemSearchPro [18], Q2Semantic [11], SPARK [21], and QUICK [20]. These systems find all the paths that can be derived from a RDF graph, until a predefined depth, to generate the queries. In [9] they propose a similar approach to keyword interpretation but they introduce the context of the user's search (the knowledge about previous queries) to focus the whole search process. However, all of these approaches assume that the data sources are under their control and can pre-calculate complex graphs to perform the query translation and, finally, access the data. As they work on the data level, they do not take into account the flexibility that using and adapting the describing ontology provides as we do. Moreover, the path generation they perform is quite similar to our refinement step, but, as our system works at ontological level, it performs semantic checkings that these approaches cannot.

A different approach to combine keyword-based search and semantic search is K-Search [5]. The main differences between their approach and ours are that they completely separate semantic searches from keyword ones, this is, they do not take into account the semantics of the domain to focus their keyword searches as our system does; and that they do not provide the possibility of defining external views via the domain definition and annotation as our approach does. Moreover, they rely again on an pre-indexing phase for the keyword search part of the system.

There are also some works in the area of databases to provide a keyword-based interface for databases, such as BANKS [1], DISCOVER [13] and DBXplore [2], which translate a set of keywords into SQL queries. However, as emphasized in [4], most of these works only rely on extensional knowledge obtained by applying IR-retrieval techniques, and so they do not consider the intensional knowledge (the structural knowledge). So, again, they have to have the data sources under their control, thus restricting the application in an open scenario such as the Linked Data one.

Finally, regarding keyword search on generic graphs, there are systems with different approaches such as BLINKS [12] or Yanii [19]. However, they also rely on an initial data indexing phase, which is not applicable when we do not have control over the dataset. Moreover, their approaches are focused by the structure of the data graph, and not by the defined semantics as ours is.

7 Conclusions and Future Work

In this paper, we have presented a system that enables a hybrid search approach based on keywords and guided by ontologies. Our system combines the ease of use of keyword search with the benefits of exploiting the structure of the underlying data in an efficient way. In particular, our system:

- Provides a keyword-based search guided and focused by the Domain Ontology, avoiding queries that would be too time-consuming. To do so, it

uses the information of the concept hierarchy of the search domain. This process is highly configurable, as the search can be restricted to a set of specified properties via annotations.

- Exploits the definition of the objects in the domain to suggest further close-related results, refining the search within the domain. This is done with the help of a DL reasoner, that performs all the semantic checkings needed to avoid inconsistent queries.

- Can be built on third parties' Linked Data repositories without overloading them, while allowing to provide the user with different domain views. Our system manages the ontologies as views on the underlying data, decoupling them and processing the results in more flexible ways.

As future work, we are planning to include crossed-domains searches, that is, to include inter-domain relationships in the search, while keeping the adopted hybrid strategy. We also want to perform tests with different kinds of final users to measure the semantic accuracy of our prototype and the overall performance of our approach.

Acknowledgments

This work has been supported by the CICYT projects TIN2010-21387-C02-02 and TIN2011-24660, and DGA-FSE. We want to thank Francisco J. Serón for his contributions during the design and development of the system.

References

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, and S. Sudarshan. BANKS: Browsing and keyword searching in relational databases. In *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB'02), China*, pages 1083–1086. Morgan Kauffman, 2002.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *Proc. of 18th Intl. Conf. on Data Engineering (ICDE'02), USA*, pages 5–15. IEEE, 2002.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Pastel-Scheneider. *The Description Logic Handbook. Theory, Implementation and Applications*. Cambridge University Press, 2003.

[4] S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo-Lado, and Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD'11), Greece*, pages 565–576. ACM, 2011.

[5] R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi, and D. Petrelli. Hybrid search: Effectively combining keywords and semantic searches. In *Proc. of 5th European Semantic Web Conference (ESWC'08), Spain*, pages 554–568. Springer, 2008.

[6] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[7] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154 – 165, 2009.

[8] C. Bobed, R. Trillo, E. Mena, and S. Ilarri. From keywords to queries: Discovering the user's intended meaning. In *Proc. of 11th Intl. Conf. on Web Information System Engineering (WISE'10), China*, pages 190–203. Springer, 2010.

[9] H. Fu and K. Anyanwu. Effectively interpreting keyword queries on rdf databases with a rear view. In *Proc. of 10th Intl. Semantic Web Conference (ISWC'11), Germany*, pages 193–208. Springer, 2011.

[10] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[11] Q. L. Haofen Wang, Kang Zhang, D. T. Tran, and Y. Yu. Q2semantic: A lightweight keyword interface to semantic search. In *Proc. of 5th European Semantic Web Conference (ESWC'08), Spain*, pages 584–598. Springer, 2008.

[12] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked keyword searches on graphs. In *Proc. of the ACM Intl. Conf. on Management of Data (SIGMOD'07), China*, pages 305–316. ACM, 2007.

[13] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB'02), China*, pages 670–681. Morgan Kauffman, 2002.

[14] E. Jimenez, B. Cuenca, U. Sattler, T. Schneider, and R. Berlanga. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In *Proc. of 5th European Semantic Web Conference (ESWC'08), Spain*, pages 185–199. Springer, 2008.

[15] E. Kaufmann and A. Bernstein. Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):377 – 393, 2010.

[16] Y. Lei, V. S. Uren, and E. Motta. SemSearch: A search engine for the Semantic Web. In *Proc. of 15th Intl. Conf. on Knowledge Engineering and Knowledge Management (EKAW'06), Czech Republic*, pages 238–245. Springer, 2006.

[17] N. Shadbolt, W. Hall, and T. Berners-Lee. The Semantic Web revisited. *Intelligent Systems, IEEE*, 21(3):96–101, 2006.

[18] T. Tran, D. M. Herzig, and G. Ladwig. SemSearchPro: Using semantics throughout the search process. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4):349 – 364, 2011.

[19] R. D. Virgilio, P. Cappellari, and M. Miscione. Cluster-based exploration for effective keyword search over semantic datasets. In *Proc. of Intl. Conf. on Conceptual Modeling (ER'09), Brazil*, pages 205–218. Springer, 2009.

[20] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl. From keywords to semantic queries – incremental query construction on the Semantic Web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):166–176, 2009.

[21] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. SPARK: Adapting keyword query to semantic search. In *Proc. of 6th Intl. Semantic Web Conference (ISWC'07), South Korea*, pages 687–700. Springer, 2007.