Procesamiento del Lenguaje Natural, Revista nº 61, septiembre de 2018, pp. 39-46 recibido 04-04-2018 revisado 30-04-2018 aceptado 04-05-2018

Getting answers from semantic repositories: a keywords-based approach

Obteniendo respuestas de repositorios semánticos usando palabras clave

Francisco Abad Navarro[1], **Jesualdo Tomás Fernández Breis**[1]
[1] Departamento de Informática y Sistemas, Universidad de Murcia, IMIB-Arrixaca, CP 30100 Spain
email:francisco.abad@um.es, jfernand@um.es

Abstract: The Web of Data proposes to publish and connect data by applying the semantic web technologies for the representation of knowledge and data and the definition of queries. The success of the Web of Data requires that both humans and machines are able to extract information from such semantic repositories. For this purpose, query interfaces for humans must make the interaction with the repository as transparent as possible. In this work, we present a generic method for querying semantic repositories based on processing and recognizing the keywords input by the users as entities of the ontology used in the description of the data. A SPARQL query is automatically derived from the query graph extracted from the list of keywords. We also describe the application of the method to three different semantic repositories from different domains.
Keywords: Semantic web, question answering, ontology, sparql, rdf, automatic query construction, semantic exploitation

Resumen: La Web de Datos propone publicar y conectar los datos utilizando las tecnologías de la web semántica para la representación del conocimiento, de los datos y la especificación de consultas. El éxito de la Web de Datos requiere que tanto los humanos como las máquinas sean capaces de obtener información en estos repositorios semánticos. Para ello, los interfaces de consulta para humanos deben hacer lo más transparente posible el proceso de interacción con el repositorio. En este trabajo presentamos un método genérico para la consulta de repositorios semánticos basado en el reconocimiento de las keywords introducidas por los usuarios como entidades de la ontología utilizada para la descripción de los datos. Del procesamiento del grafo de consulta generado se deriva automáticamente la consulta SPARQL que se ejecuta contra el repositorio. Describimos el uso del método con tres repositorios de distintos dominios.
Palabras clave: Web semántica, pregunta-respuesta, ontología, sparql, rdf, construcción automática de consultas, explotación semántica

1 Introduction

The Semantic Web (Berners-Lee, Hendler, and Lassila, 2001) is a set of technologies whose principal goal is to describe the data on the web in such way that a machine can read and process it easily. Ontologies (Bechhofer, 2009) play a main role as part of this set of technologies. They describe how the data is organized through concepts, their properties and relations with each others expressed by logical axioms. The other great pillar of the Semantic Web is RDF (Resource Description Framework) (Klyne and Carroll, 2006). RDF is used to represent the data as triples formed by subject, predicate and object, where subjects are concrete instances of concepts from the ontology, predicates are relations or properties described in the ontology and objects could be another instance of a concept or a primitive value. All the data in a RDF repository could be represented as a graph, where nodes are instances or primitive values and edges are relations or properties connecting the nodes.

ISSN 1135-5948. DOI 10.26342/2018-61-4

The development of Web of Data is the major objective of the Semantic Web initiative called Linked Open Data (LOD) (Bizer et al., 2008). There, data is ideally shared using formats like RDF, and the meaning of the entities is provided by an ontology. The LOD has penetrated in many domains such as biology (Consortium, 2016) or music (Swartz, 2002). The data stored in these repositories can be navigated through a web browser or queried using SPARQL. The exploitation of such repositories by non-semantic web experts is hampered by the need of knowing such language, so there is a need for making semantic web technologies as transparent as possible for human users interested in exploiting semantic repositories.

In the last years there have been different approaches related to controlled or natural language interfaces that automatically generate SPARQL queries, such as autoSPARQL (Lehmann and Bühmann, 2011), FREyA (Damljanovic, Agatonovic, and Cunningham, 2011) or OWLPath (Valencia-García et al., 2011), but they do not emulate the keywords-based search users are familiar with.

In this paper we propose a method which processes the keywords input by the users, generates the tree that interprets the query and automatically designs and executes SPARQL queries. The application of the method to three freely available, not developed by us, SPARQL endpoints in Spanish and English language is also reported in this paper.

2 Method

In this section we describe our method, which can be applied to any RDF dataset whose vocabulary is provided by an ontology and which offers a SPARQL endpoint for querying. Our query model assumes that the input consists of keywords in natural language, which have to be processed, transformed into semantic entities and then used for the creation of SPARQL queries.

The method consists of the following modules: text normalizer, index builder, named entity recognizer, tree generator and SPARQL query generator. These modules are summarized in Figure 1 and detailed next:

2.1 Text normalizer

We use a language dependent text normalization in order to transform natural language text into a canonical form, so different forms of the same word are translated into a unique representation. Currently Spanish and English are the languages supported. This process is performed in two steps:

1. Preprocessing: the input is converted to lower case and common words and special characters as *, + or & are removed.
2. Normalization: the preprocessed text is the input for a Stanford NLP pipeline (Manning et al., 2014), which provides the following annotation tools, which are configured depending on the language used: token annotator, sentence annotator and part of speech annotator. Also a custom stem annotator was implemented through Snowball stemmer API [1] and appended to the Stanford pipeline.

At the end of the normalization step, the detected stems are concatenated with a blank space between them. Table 1 shows several examples of the text normalization.

Language	**Original Text**	**Normalized Text**
Spanish	Empresas SL	empres
Spanish	Pirineos (los)	pirine
Spanish	teléfono	telefon
English	Companies	compani
English	United States (the)	unit state
English	telephone	telephon

Table 1: Examples of text normalization y Spanish and English language

2.2 Index builder

We use a text index for the recognition of the named entities in the input text. This index is built by extracting the labels of the ontology classes and properties, as well as of the individuals stored in the dataset. The index contains the following fields:

- URI. The uniform resource identifier of an element. For example `<http://opendata.caceres.es/recurso/cultura-ocio/museos/Museo/10-museo-de-armas>`.

[1] `http://snowballstem.org/`

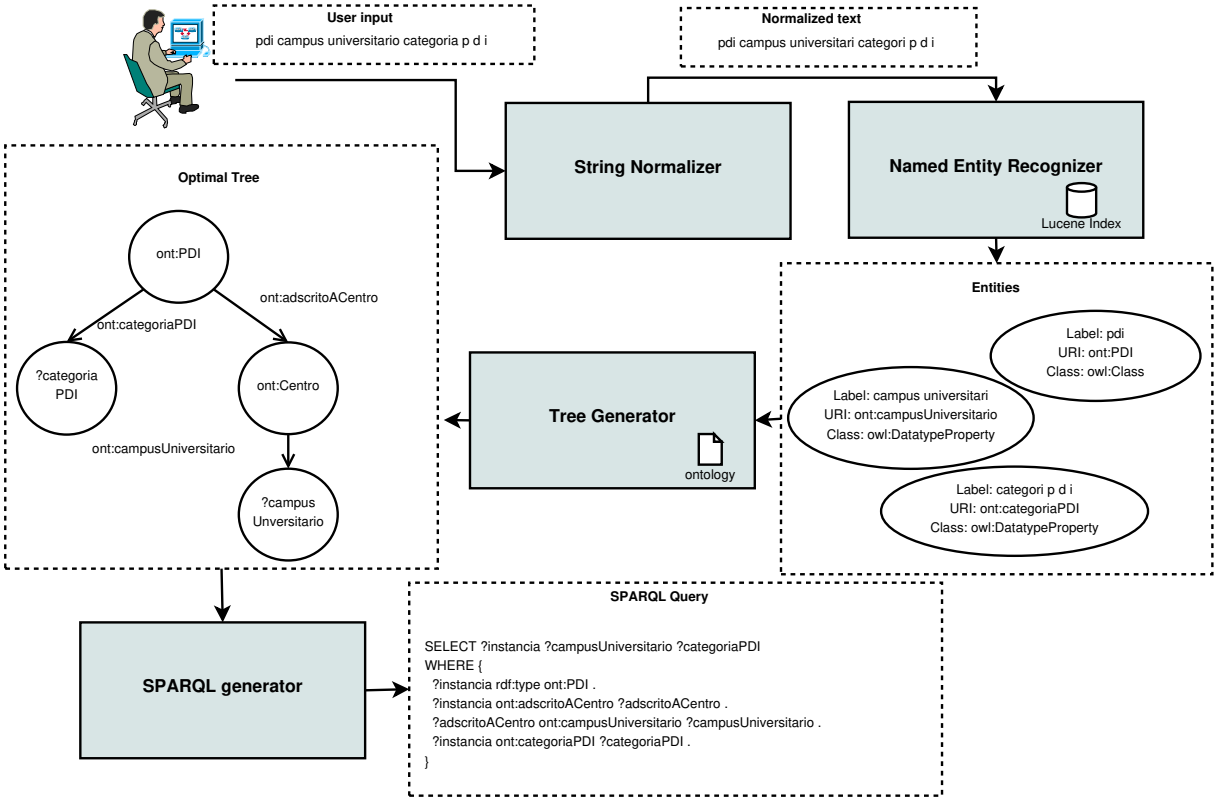

Figure 1: Method pipeline example

- Local name. The last part of the URI. For example `10-museo-de-armas` in the previous URI.
- Class. The class of the ontology that the element belongs to. In this case, `10-museo-de-armas` belongs to `<http://opendata.caceres.es/def/ontomunicipio#Museo>`. If the element is not an individual, this field is set as `owl:Class` for classes, `owl:DatatypeProperty` for properties, `owl:ObjectProperty` for relations.
- Type. The type of element, that could include "INSTANCE", if the element is an individual or "CLASS", "PROPERTY" or "RELATION" if the element is an ontology class. In this case the type of `10-museo-de-armas` would be "INSTANCE".
- Label. The original label associated with the element. For example "Museo de Armas". The value of this field is obtained from the `rdfs:label` annotations, filtering by the language used.
- Preprocessed label. The label after the preprocessing described in Section 2.1. For example "museo de armas".
- Normalized Label. The label after the normalization described in Section 2.1. For example "muse de armas".

While classes, relations and properties are extracted from the ontology owl file directly through `rdfs:label` annotations, a query is performed for each class to retrieve its individuals and their labels, also filtered by language. It is possible to specify properties that could play a textual label role, for instance `foaf:name`. For example, when indexing individuals that belong to a hypothetical class `ont:Person` from a Spanish RDF repository that uses `foaf:name` for representing the name of a person, the following query would be performed (prefix definition is omitted):

```
SELECT DISTINCT ?uri ?class ?label
WHERE{
  VALUES ?class { ont:Person } .
  ?uri a ?class .
  {
    ?uri rdfs:label ?label .
    FILTER (lang(?label) = 'es') .
  }
  UNION {
    ?uri foaf:name ?label .
```

```
    FILTER (lang(?label) = 'es') .
  }
}
```

Finally, the result of the query has enough information to add the individuals of the class `ont:Person` to the index.

2.3 Named entity recognizer

The named entity recognizer is implemented through a language dependent Stanford pipeline. This pipeline has the same annotators than the pipeline used for text normalization with a new custom annotator at the end, whose objective is to perform the entity recognition.

The implementation of this custom annotator is based on the standard named entity recognizer `TokensRegexNERAnnotator` from Stanford NLP. The original one uses a gazetteer file that contains, for each line, a regular expression associated with a type. Our customization consists in using the index instead of the original gazetteer file. This annotator is configured to use the index fields `normalized label`, `URI` and `class` as textual label representing an element, its identifier and its type, respectively.

By using this pipeline, the input text is segmented in named entities. Each named entity may match more than one entity in the index so each one will have a list of matched entities. For instance, if an ontology contains the classes `ont:CountryCapital`, referring the capital of a country, and `ont:Capital`, referring the money used in business, both annotated with the `rdfs:label` "capital", the entity recognizer would return both classes for the input text "capital".

Table 2 shows the named entities together with their related entities (URI and class) found in the Spanish version of DBpedia for the input "ocupación personas España".

Named entity	**URI**	**Class**
"ocupación"	ont:occupation	owl:ObjectProperty
"personas"	ont:person ont:Person	owl:ObjectProperty owl:Class
"España"	res:España res:España res:España	ont:Country ont:PopulatedPlace ont:Place

Table 2: Example of named entity recognized in the input "ocupación personas España" using the Spanish DBpedia

2.4 Tree generator

The next step is to try to connect all the entities extracted in the previous steps by using the ontology. For this purpose a tree that describes the query is built and the goal of this step is to obtain the tree with the minimum height that connects all the entities through the relations in the ontology, checking the domain and the range of these relations.

Firstly, the system tries to remove redundant information in order to get more precise results and to reduce the size of the problem. In this process, for each named entity detected, we check the hierarchy of ontology classes, properties and relations associated with it. The most specific classes are maintained while the general classes are removed. For example, in DBpedia, the resource `res:Napoleon` belongs to the classes `ont:Royalty`, `ont:Person` and `ont:Agent`. In this example, the method would keep `ont:Royalty` since it is the most concrete one. With this action, the system is able to find more specific trees, which are translated into more precise queries. Contrariwise, the method selects the most general property or relations when a hierarchy of properties or relations is detected. For example, if a hypothetical ontology has the properties `personalPhoneNumber` and `workPhoneNumber`, which are sub-properties of `phoneNumber`, the method will maintain `phoneNumber`. We expect that a query performed against a SPARQL endpoint by using a top-level relation or property returns, at least, the union of the results that have all its sub-properties as predicate. That is, a query about `phoneNumber` should include `personalPhoneNumber` and `workPhoneNumber`.

As a named entity can be finally associated with more than one entity, a backtracking process is performed to iterate over all possible combinations. Each one is evaluated by calculating the height of the different trees, which result from computing the shortest path between each entity (playing the role of root element) to the others (acting as leaf nodes) by using breadth first algorithm. Finally, the combination whose evaluation results in a tree with the minimum height is selected.

This process is performed twice using two different strategies to compute the shortest path between the ontology elements. On one hand, paths between root and leaf elements are computed by using the exact classes defined in the range and the domain of the ontology relations.On the other hand, ontology classes that are compatible with the domain and range are taken into account in order to expand the tree during their construction. The compatible classes of a class are defined as the union of the super and subclasses of that class. Finally, if only one strategy has found a tree, that tree is selected. If both strategies have found a tree, that whose height is lower is selected. At equal height, the tree found by the most conservative strategy is chosen. If none of the strategies finds a solution, then the elements are not connected and the method will not be able to generate a SPARQL query.

Following the example of DBpedia commented in Section 2.3, Figure 2 shows the tree obtained from the named entities described in Table 2.

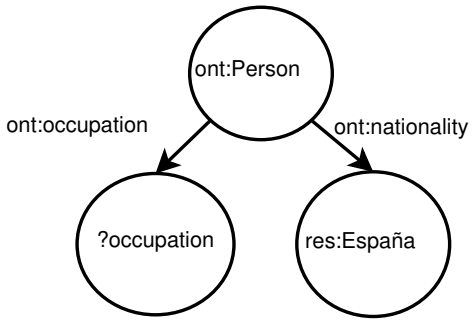

Figure 2: Example of tree construction by using the input "ocupación personas España" with the Spanish DBpedia

2.5 SPARQL query generator

If only one named entity is detected in the input, the method generates a query depending on the type of the first entity associated with the named entity. If the entity is a property or a relation, the SPARQL query retrieves all RDF subjects and objects connected by that property or relation. For instance, if only `ont:birthYear` property is detected, the following query is generated:

```
SELECT ?instance ?object
WHERE {
  ?instance ont:birthYear ?object .
}
```

If the entity is a class, the SPARQL query retrieves the individuals that belong to that class. For instance, if only `ont:Writer` class is recognized in the user input, the following query is generated:

```
SELECT ?instance
WHERE {
  ?instance rdf:type ont:Writer .
}
```

If the entity is an individual, firstly a SPARQL query is executed to retrieve all the RDF predicates that link the individual to other entities or primitive types. Then, the returned predicates are used for generating the final query. For example, if the individual `res:Napoleon` is detected in the input, the following query will extract the predicates that are linked to the individual:

```
SELECT DISTINCT ?predicate WHERE {
res:Napoleon ?predicate ?value .
}
```

Then, the final query is built using the predicates. For example, if the returned predicates are `rdf:type`,`ont:deathPlace` and `ont:deathDate`, the following query will be generated:

```
SELECT ?instance
       ?type
       ?deathPlace
       ?deathDate
WHERE {
  VALUES ?instance { res:Napoleon } .
  ?instance rdf:type ?type .
  ?instance ont:deathPlace ?deathPlace .
  ?instance ont:deathDate ?deathDate .
}
```

In the case of more than one named entity detected in the input, the SPARQL query generator transforms the tree of entities into a SPARQL query by applying the following heuristics:

1. The root element of the tree is the element from which the user wants to retrieve information. This element could be a concrete individual or a class. In the latter case the method would return a list of the individuals that belong to that class.
2. The leaf nodes of the tree could be classes, relations or properties in the ontology. In this case they act as the information that the user wants to retrieve

from the root element. On the other hand, a leaf node could be a concrete individual, so playing the role of a filter, imposing that the root element had to be related with the individual.

This kind of trees fits very well to SPARQL queries. Root and leaf elements of the tree are associated with variables that appear in `SELECT` clause. If the root element is a concrete individual, its URI identifier will be stored in the associated variable through a `VALUES` expression. Otherwise, if the root element is a class, the associated variable will contain all individuals that belong to the class through an statement of type `?var rdf:type classURI`. Then, this root variable is connected to the leaf variables according the paths in the tree. Finally, if a leaf element is an individual, their associated variable is set with its URI through a `VALUES` clause, acting as filter. Otherwise, the leaf variable will contain all elements related with the root variable through the relations found in the tree.

According to the example described in Section 2.4, whose tree is shown in Figure 2, the root element is the class `ont:Person` and the leaf nodes are the relation `ont:occupation` and the individual `res:España`, which is reached through the relation `ont:nationality`. Therefore, based on the heuristics, this tree indicates that the user wants to retrieve a list of persons from Spain together with their occupations. The final SPARQL query built from this tree is the following (prefix definition is omitted):

```
SELECT ?instance ?occupation ?Country
WHERE {
?instance rdf:type ont:Person .
?instance ont:occupation ?occupation .
?instance ont:nationality res:España .
VALUES ?Country { res:España } .
}
```

3 Results

A web application (available at `http://sele.inf.um.es/sesssamo-demo/`) has been developed implementing the described method. Three repositories have been configured: the University of Extremadura, the city of Cáceres and both the English and Spanish versions of DBpedia. Next, we describe some examples of queries executed against each repository. All the queries generated a tree of height 2 except one, which generated a query of height 3. The time to build these queries was 435.5 ms on average, with a median of 297 ms. These tests were performed on a laptop with 12 GB of RAM and an Intel Core i5-3337u processor.

3.1 University of Extremadura

This university publishes RDF data in Spanish through a SPARQL endpoint (`http://opendata.unex.es/sparql`). The data is organized according to ontouniversidad ontology (`http://opendata.unex.es/def/ontouniversidad.html`)

The following list shows some examples of queries:

- **directores departamento**. List of departments together with their heads.
- **asignaturas isabel cuadrado gordillo**. The subjects from which Isabel Cuadrado Gordillo is teacher of.
- **publicacion isabel cuadrado gordillo**. List of publications of Isabel Cuadrado Gordillo.
- **asignaturas pdi**. List of professors together with their subjects.
- **pdi campus universitario**. Returns a list of researchers together with the university campus that they belong to. This query generates a tree of height 3.

3.2 City of Cáceres

The city of Cáceres has a SPARQL endpoint (`http://opendata.caceres.es/sparql`) in which RDF data in Spanish are published according to ontomunicipio ontology (`http://opendata.caceres.es/def/ontomunicipio.html`).

The following queries were tested:

- **cofradias nombre asociacion numero miembros**. List of brotherhoods together with their official name and number of members.
- **monumentos enlace s i g**. List of monuments with their Geographic Information System link.
- **barrios centro**. List of neighborhoods that belong to the center district.
- **hoteles via**. List of hotels and the street name where they are situated.

- **cofradias procesiones**. List of brotherhoods together with the processions they organize.

3.3 DBpedia

DBpedia is an encyclopedic knowledge base whose data is extracted from Wikipedia through automatic processes (Auer et al., 2007). Different SPARQL endpoints are available depending on the language, but all conform to an unique ontology, available at `http://downloads.dbpedia.org/2016-04/dbpedia_2016-04.owl`.

Spanish labels were added to the ontology automatically via Google Translator API in those cases where it exists an English but not a Spanish label. Moreover, only elements from the ontology were added to the index. The named entity recognizer uses the index for identifying these elements (concepts, relations and properties) and the DBpedia Spotlight API (Mendes et al., 2011) for detecting concrete individuals.

Some examples of queries using Spanish and English versions of DBpedia are shown below.

3.3.1 Spanish version

- **progenitores Juan Carlos I**. Parents of Juan Carlos I of Spain (Juan de Borbón, María de las Mercedes de Borbón-Dos Sicilias).
- **ocupación personas España**. List of Spanish persons together with their occupation.
- **superficie Atlanta**. The area of Atlanta (3.412E8).
- **esposa John Lennon**. The spouse of John Lennon (Yoko Ono).

3.3.2 English version

- **The Beatles former band members**. The name of the persons who formed The Beatles (George Harrison, John Lennon, Ringo Starr, Paul McCartney).
- **Einstein spouse**. The spouse of Albert Einstein (Mileva Marić, Elsa Löwenthal).
- **Stephen Hawking doctoral advisor**. The advisor of Stephen Hawking's PhD thesis (Dennis William Sciama).
- **mean temperature k Mars**. The mean temperature of Mars in Kelvin degrees (210.0, 210.15).

4 Discussion

In this paper we have presented a method whose goal is to facilitate human users the exploitation of semantic repositories, among whose main applications we can identify semantic search and question answering systems. State of the art solutions in this field usually require a complete, well written question in natural language. This permits a more exhaustive analysis of the query, and shows good results, but this is not the usual interaction way between users and search engines. Our work assumes that we should use the same type of interaction when exploiting semantic repositories, and that this is an essential feature for the success of semantic web technologies.

Our work is in line with the method shown in (Tran et al., 2007), which proposes a searcher based on keywords. This method also uses trees to describe queries. As in our work, and based on the locality principle, the tree with the lowest height is consider more likely to contain the answer the user wants to know. Nonetheless, in this work only one entity is associated with each keyword before computing the tree. This may cause problems if a keyword is ambiguous and the selected entity is not what the user had in mind. Our method takes this into account and, as previously mentioned in Section 2.4, it uses all possible entities matching a keyword in order to compute the tree. Finally, based on locality, the correct entity will be selected. Our method renovates the technological approach developed by our research group in (Valencia-García et al., 2011), since now the query design is not driven by the ontology but by the user input.

One limitation of our current implementation is due to ambiguity in cases where two concepts are linked by more than one relation. That situation permits to generate different trees with the same height, and we are currently executing only one of them. For example, in DBpedia, the input "writers Spain" shows no results because the system links `ont:writer` and `res:Spain` through `ont:livingPlace` instead of `ont:nationality`. As future work we will explore the execution of both queries

or offering the users the two interpretations of the queries and asking them to select the desired one. This way of solving ambiguity has already been used (Damljanovic, Agatonovic, and Cunningham, 2011; Lehmann and Bühmann, 2011). Another limitation is the use of synonyms that are not indexed. We plan to enrich the index by using tools like WordNet (Miller, 1995) to provide these synonyms. Once these limitations have been overcome we hope to obtain good results by using the Question Answering over Linked Data (QALD) evaluation system (Lopez et al., 2013).

Acknowledgements

This work has been funded by the Spanish Ministry of Economy, Industry and Competitiveness, the European Regional Development Fund (ERDF) Programme and the Fundación Séneca through grants TIN2014-53749-C2-2-R and 19371/PI/14.

References

Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, pages 722–735.

Bechhofer, S. 2009. Owl: Web ontology language. In *Encyclopedia of database systems*. Springer, pages 2008–2009.

Berners-Lee, T., J. Hendler, and O. Lassila. 2001. The semantic web. *Scientific american*, 284(5):34–43.

Bizer, C., T. Heath, K. Idehen, and T. Berners-Lee. 2008. Linked data on the web (ldow2008). In *Proceedings of the 17th international conference on World Wide Web*, pages 1265–1266. ACM.

Consortium, U. 2016. Uniprot: the universal protein knowledgebase. *Nucleic acids research*, 45(D1):D158–D169.

Damljanovic, D., M. Agatonovic, and H. Cunningham. 2011. Freya: An interactive way of querying linked data using natural language. In *Extended Semantic Web Conference*, pages 125–138. Springer.

Klyne, G. and J. J. Carroll. 2006. Resource description framework (RDF): Concepts and abstract syntax. Technical report, W3C.

Lehmann, J. and L. Bühmann. 2011. Autosparql: Let users query your knowledge base. In *Extended Semantic Web Conference*, pages 63–79. Springer.

Lopez, V., C. Unger, P. Cimiano, and E. Motta. 2013. Evaluating question answering over linked data. *Web Semantics Science Services And Agents On The World Wide Web*, 21:3–13.

Manning, C., M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

Mendes, P. N., M. Jakob, A. García-Silva, and C. Bizer. 2011. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*, pages 1–8. ACM.

Miller, G. A. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Swartz, A. 2002. Musicbrainz: A semantic web service. *IEEE Intelligent Systems*, 17(1):76–77.

Tran, T., P. Cimiano, S. Rudolph, and R. Studer. 2007. Ontology-based interpretation of keywords for semantic search. In *The Semantic Web*. Springer, pages 523–536.

Valencia-García, R., F. García-Sánchez, D. Castellanos-Nieves, et al. 2011. Owlpath: An owl ontology-guided query editor. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(1):121–136.