User Queries for Semantic Search Engine

Arooj Fatima, Cristina Luca, George Wilson
Department of Computer Science
Anglia Ruskin University
Cambridge, UK
{arooj.fatima, cristina.luca, george.wilson}@anglia.ac.uk

Abstract**—The amount of linked open data stored on the internet is increasing hugely, yet it is difficult for non-expert users to access the semantic data without having knowledge of SPARQL language. Whilst the modern semantic search engines tackled the problem with standard user interfaces, these interfaces are not very usable for the users with no or little knowledge of semantic web technologies. One way to achieve a user friendly interface is the use of an effective tool to convert a natural language query into a standard SPARQL query. This paper introduces an efficient way of information retrieval from online repositories and a system to match user entered keywords with related ontologies to format a query. This paper proposes a solution based on a previously modelled framework and introduces algorithms for query tagging and keyword mapping in order to formulate SPARQL queries. Finally, the paper describes examples using prototype user interface.**

Keywords- Semantic web; Semantic search; Keywords tagging; user queries

I. Introduction

There is a large amount of data stored over the Internet and that information is only useful if it can be accessed in a meaningful way. To access data from the Internet we need a 'smart' search facility. Search engines are the tools to help users to find data from the huge repository of web pages. To extract data, most of the search engines use syntax-based search or full-text search methods.

Full-text searching is a technique whereby a computer program matches terms in a search query with terms embedded within individual documents in a database [1]. An important issue is that the full-text searching is subject to failure if a synonym of the searched word exists and is not matched [2].

The syntax-based search engines use various techniques to find results against an informal user query, such as full-text matching, keyword and meta tags, caching the more popular user queries. These search engines also use their own search algorithms based on their own database structure. To get web pages ranked by the search engines, website developers use a technique called Search Engine Optimisation (SEO). Keywords, meta-tags and micro-formats are the main tools used for a SEO. These techniques enhance the factor of user friendliness and increase the chances of more accurate results but these are not the ultimate solution. That is, data searched by a syntax-based search engine has many limitations, including high recall with low precision (e.g. thousands of results in response to one or few keywords), low or no recall (when there is no relevant results), and a high sensitivity of results to vocabulary [3]. A semantic web is the optimised solution to these challenges. The Semantic Web can be described as a web of documents linked in such a way so that the data becomes readable and understandable to machines in a meaningful way, hence allowing them to intelligently match related data [4]. Even if the semantic web is already used in practice, there is a need for a strong semantic search engine to utilise the full potential of that web. Semantic search engines such as Swoogle [5] (swoogle.umbc.edu) and Duckduckgo (duckduckgo.com/) have taken this approach but have a number of limitations especially regarding user experience.

In a conventional programming environment the developers write queries based on knowledge of the search criteria, data structure (e.g. relational database) and the query language. A truly optimised semantic search requires a search engine that can translate the user-queries into queries to be fed back into the query engine. This means that a query language alone will not benefit a search system but rather is highly dependent upon the base data structure it works upon and the top-level system interacting with it. This functionality must not be at the expense of the qualities of a conventional end-user interface and for example should include user-friendly features such as auto-suggest and auto-correct functionality [6] and an ontological classification of words, possibly by query-tagging [7].

The information stored within a semantic web has a different structure compared to that of the conventional web, and this means it necessarily needs different techniques to understand a user query and find results from online repositories. This paper investigates some concepts on how the semantic web might be queried in the context of semantic search engines. The proposed solution in this paper focuses on pure semantic queries and full-text pattern matching (for searches such as movies, books). We introduce the concept of a 'Bag of Keywords' where we save a Class, Property and

Instance (CPI) value for each keyword that helps the Query Optimiser to tag keywords. Tests have been made on DBpedia [8] online repositories using the SPARQL as an endpoint. The generic approach for the solution makes it flexible to work with various repositories and at a larger scale.

II. FEATURES OF AN EFFECTIVE SEARCH

Traditional search engines are continuously improving their techniques to enhance the user experience for searching the web. An effective search may have a number of features but from the point of view of an end user, an effective search experience includes:

- An easy and friendly user interface that provides certain features to get an understanding of user requirement for the data i.e. spell checking, grammar correction (if possible), auto-suggest [9].
- An efficient system that quickly finds results.
- A system that actually understands what the user is looking for.

III. THE CHALLENGE FOR A SEMANTIC SEARCH ENGINE

Although there are multiple challenges faced by a semantic search engine which include user experience, efficiency, page ranking, scalability, and cost effectiveness [10], the focus of this paper is to explore the challenge of understanding how a user query can be constructed and translated into a formal query that can be parsed by a semantic query engine.

A. *Understanding the User Query*

The end users not only expect high relevancy and efficiency but also require that the search interface understands what they are looking for, even if, incorrect grammatical syntax is present. The openness of the user query format makes this the most difficult challenge for a semantic search engine to understand the query meaning and fetch relevant results. If the user query is not understood by the ontological components of the semantic framework the results received are not relevant and/or accurate.

Semantic data is tagged with pre-defined meanings defined in vocabularies (ontologies) and this is why keywords entered (as a user query) need to be mapped with similar meanings as defined in the related ontology. The ontology can describe objects at a low level and can inference if an object belongs to a type or instance. End users normally search with specific keywords that are values for the properties of ontology classes e.g. Ford (is an instance of class Vehicle). Using CPI values and 'bag of keywords', a keyword like this can be mapped with the related properties and classes.

B. *Efficiency*

An efficient (i.e. fast in this context) search engine's performance depends upon the size of data to be matched, the request time to the web servers and the associated response time. For a semantic web query execution time also depends on factors such as delays caused by looking up Uniform Resource Locators [11], efficient support for indexing large-scale data [12], and dealing with query termination problem [13]. The proposed system looks into the possibilities to improve efficiency of the system.

IV. RELATED WORK

Swoogle [14] is a semantic web search engine that discovers and indexes information in semantic web documents. Swoogle provides a semantic web data access service, which helps human users and software systems to find relevant documents, terms and triples [5]. However, it does have limitations, including limited quality control (redundant ontologies), limited search (keywords based only) and no support for a formal query language like SPARQL [15]. Sindice [16] is another example of a popular semantic search engines, but that is not very usable for people with no understanding of semantic technologies.

Watson [17] uses keyword search features that make it similar in use with conventional web search systems. The Watson search tool matches a set of keywords entered by the user to the literals of entities saved in a semantic document and display-entities matching each keyword. The search tool can in principle find data based on type although in practice it is not easy to discover such data and results are not readable.

Swoogle, Sindice and Watson originate from a similar idea to formalize semantic data and make it available online, for users and applications to find and exploit. However, they mostly rely on extracting labels or comments from RDF documents and match the search keywords with saved data. However new mechanisms are required to enable applications to explore large scale semantic data [17].

V. PROPOSED SOLUTION

The authors have approached the problem using their previously modelled framework [9-10] described in Fig. 1.

A. *Preliminaries*

1) Ontologies: - In the context of a semantic web, the ontologies are the schematic vocabularies that describe how things for semantic data exist. Ontologies describe things using classes, instances and properties. For instance, Person is a class containing all properties a person can have i.e. name, height, gender, etc. An ontology classifies instances or individuals though it need not include individuals or instances explicitly. For an ontology where there are no instances defined explicitly, there is a need for a mechanism to relate instances to the ontology.

2) User Keywords: A user-entered query is informal and highly variant depending upon what and how that query is typed in. It is a major challenge for a search engine to understand a user query for various reasons (e.g. the user may type wrong spellings, different users use different arrangement for keywords). The conventional search engines use various techniques to resolve these issues e.g. auto-

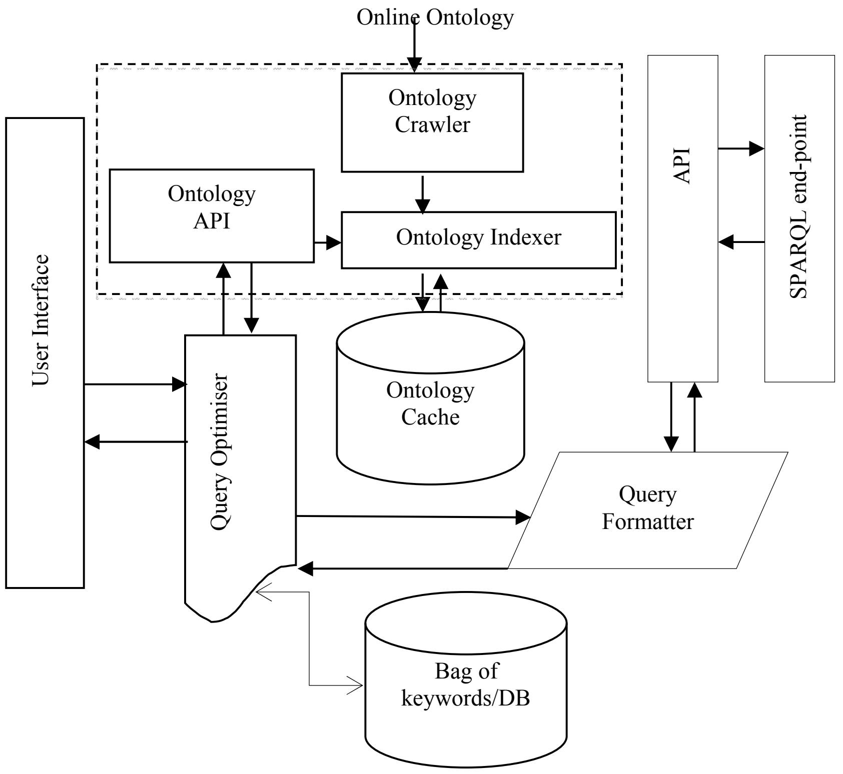

Fig. 1. Framework for Information Retrieval

correct and autosuggest are implemented, and finally selected keywords are matched against the indexed keywords, so that the linked URLs to the selection are listed in response. This approach works for a syntax based search (where relevant and irrelevant results are produced) but for a semantic search engine a different approach is required for keyword understanding and linking with URLs (actually URIs). The 'Autosuggest' technique is utilized depending if a keyword exists in more than one context e.g. apple {fruit, device}. Mostly users type short keywords for search queries e.g. *UK cities* => list of all cities in UK. But sometimes users are very specific about their search and type longer queries. This diversity of keywords/user-queries means that it is a considerable challenge for a semantic search engine to translate an informal user query to a formal query that can be parsed by the query engine i.e. SPARQL.

B. Definitions

1) Definition-1: The current authors define the user query entered by the user as a set of keywords K = {k1, k2… kn}. A user may enter a number of keywords that are passed to the query optimiser by the user interface. To process 'K', the query optimiser decides if it needs to evaluate keywords individually for inference, or the whole set refers to a single item e.g. 'The Dark Water' is a movie name and should be dealt as one keyword and mapped as a 'Film' or 'Movie' instead evaluating each keyword and then finding a relation among them.

2) Definition-2. The authors introduce the term 'Bag of keywords' which is a repository of saved terms (currently the dumps are limited to the English language) and sets an identifier value for each keyword that is called the CPI value. The term CPI has been derived from three main concepts in Ontology i.e. Class, Property and Instance. Each keyword may belong to a set of context (C, P, I) where 'C' denotes a class, 'P' denotes a property and 'I' denotes an Instance. The stored keywords can be used for auto-correct, autosuggest and tagging. Caching ontologies enhances efficiency of the system while finding relationship among keywords and ensures availability of the system if online repositories are not available.

C. Algorithms

This section describes algorithms for query processing. The first part of the section explains the procedure for keyword tagging (Algorithm-1) where user keywords are attached with CPI tags to be identified as classes, properties or instances. The second part of this section defines the process of query formatting (Algorithm-2).

TABLE I. ALGORITHM-1

```
Process: Keywords Tagging
1:   Require: user_query
2:   match ⟵full_text_match(user_query)
3:   if match found
4:       process results
5:       tags ⟵ get_tags()
6:   else
7:       words_0..n ⟵split(user_query)
8:       for each words w
9:           tags ⟵ get_tags(w)
10:      end for
11:  end if
12:  return tags
```

TABLE II. ALGORITHM-2

```
Process: Query Formatting
1:   Require: data , domain
2:   data_cpi ⟵ get_CPI_values(data)
3:   for each   data_cpi cpi   do
4:       if  cpi is a property then
5:           statements ⟵ property_statement(cpi)
6:       else
7:               if  cpi is a class  then
8:                   statements ⟵class_statement(cpi)
9:               end if
10:      else
11:              if  cpi is an instance  then
12:                  statements⟵instance_statement(cpi)
13:                  filters⟵getFilter(cpi)
14:              end if
15:      end if
16:  end_for
17:  getCPIrelations()
18:  prefixes ⟵ getPrefixes(domain)
19:  result ⟵createQuery()
20:  return result
```

1) *Explanation Algorithm-1:* This program gets a user search query as input and first evaluates if the user query needs to be matched as full text or parsed to find related tags for each keyword. If a full text match is found (line 2) the algorithm processes result to find property and concepts tags related to the text. The process also gets the domain (of ontology) against that and returns the result to the Query Optimiser. There may be more than one tag for one full-text keyword e.g. 'The Da Vinci Code' can have two tags {movie, book}. Multiple tags call the auto-suggest action at Query Optimiser level. If a full text match fails for the user query (line 6), the query is split into words and each word is evaluated to find its tags and domains of ontology. The domain of ontology helps the Query Formatter to get prefixes while creating a query.

2) *Explanation Algorithm-2:* This program gets an input from the Query Optimser containing keywords, tags and the domain. It generates queries per domain and that is how the user interface is able to display a list of results per domain (this addresses a redundant results issue). The program evaluates each element in data array and finds its CPI tag value (line 2) and decides the action accordingly. If the CPI tag suggest that the keyword is a property (line 4), it generates two statements for the query, namely (i) to attach property and (ii) to attach class. If the CPI tag suggests that the given keyword is a class (line 7), it adds a line to define the rdf type for the class. If the CPI tag suggest that the keyword is an instance (line 11) the algorithm generates the statement for related property, class and filters. The last step is to find CPI relations among extracted tag values. Finally the program generates prefixes for the given domain and a query is created from the prefixes and statements created during this program.

D. Implementation

1) *Bag of Keywords:* We have saved keywords in the database linked with their CPI value. The keywords that are classes get null values for property and instance. CPI value helps the program to decide whether a keyword is a property, class or an instance.

Table-III shows some example data from the bag of keywords. The CPI value for keyword 'country' suggests that it refers to a 'Class' i.e. country is a class. The CPI value for keyword 'pakistan' suggests that it is an instance of class 'Country' linked by property 'foaf:name'. Each keyword can be tagged with more than one existence that can be useful for autosuggest process.

2) *Query Optimiser:* Query Optimiser gets user input from the user interface and runs Algorithm-1 to tag keywords with their related ontology. Keywords with their CPI values are dumped in database (Bag of keywords).

3) *Query Formatter.* The query formatter receives tagged keywords from Query Optimiser and generates a query from the given information. The query is sent to SPARQL end point for validation and the results are sent back to the user interface.

4) *Test Base (Test Environment):* The proposed solution has been tested using prototype user interface. The tests have been carried out using DBpedia SPARQL endpoint (dbpedia.org/snorql).

VI. EXAMPLES

The following are some examples of queries generated by the Query Formatter for a given user input. For simplicity prefixes have not been added and results are summarized.

A. *Example-1*

User-Input: country
CPI value: Country(dbpedia-owl:Country, null, null)
Generated Query:

```
SELECT ?country
WHERE {
  ?country rdf:type dbpedia-owl:Country.
}
```

B. *Example-2*

User-Input: pakistan
CPI value: (dbpedia-owl:Country, foaf:name, null)
Generated Query:

```
SELECT ?country
WHERE {
  ?country rdf:type dbpedia-owl:Country.
  ?country foaf:name ?name
  FILTER regex(?name, "pakistan", "i")
}
```

C. *Example-3*

User-Input: karachi pakistan
CPI value:
For lahore: (dbpedia-owl:City, foaf:name, karachi)
For pakistan:(dbpedia-owl:Country, foaf:name, pakistan)
Generated Query:

```
SELECT *
WHERE {
   ?city  rdf:type dbpedia-owl:City.
   ?city foaf:name ?name.
   ?city  dbpedia-owl:country ?country.
   ?country foaf:name ?cname.
   Filter regex(?name, "karachi","i")
   Filter regex(?cname, "pakistan", "i")
}
```

D. *Example-4*

User-Input: cities in pakistan
CPI value:
For cities: (dbpedia-owl:City, null, null)
For pakistan:(dbpedia-owl:Country, foaf:name, pakistan)
Generated Query:

```
SELECT ?city
WHERE {
  ?city rdf:type dbpedia-owl:City.
  ?city dbpedia-owl:country ?country.
  ?country rdf:type dbpedia-owl:Country.
  ?country foaf:name ?cname
  Filter regex(?cname, "pakistan", "i")
}
```

TABLE 1II. BAG OF KEYWORDS

Keyword	**Tag (C, P, I)**
country	(dbpedia-owl:Country, null, null)
countrie	(dbpedia-owl:Country, null, null)
pakistan	(dbpedia-owl:Country,foaf:name,pakistan)

TABLE IV. RESULT EXAMPLE 1

country
http://dbpedia.org/resource/Afghanistan_
http://dbpedia.org/resource/Iran
http://dbpedia.org/resource/United_Arab_Emirates
http://dbpedia.org/resource/pakistan
…

TABLE V. RESULT EXAMPLE 2

country
http://dbpedia.org/resource/West_Pakistan
http://dbpedia.org/resource/East_Pakistan
http://dbpedia.org/resource/Pakistan
http://dbpedia.org/resource/Dominion_of_Pakistan
http://dbpedia.org/resource/Dominion_of_Pakistan
…

TABLE VI. RESULT EXAMPLE 3

city	**name**	**…**
http://dbpedia.org/resource/Karachi	“Karachi"@en	…

TABLE VII.RESULT EXAMPLE 4

city
http://dbpedia.org/resource/Rohri
http://dbpedia.org/resource/Wagan_City
http://dbpedia.org/resource/Muzaffargarh
http://dbpedia.org/resource/Sillanwali
http://dbpedia.org/resource/Islamabad
http://dbpedia.org/resource/Karachi
…

VII. POSSIBLE IMPLEMENTATIONS

The proposed framework can be used as a back bone of the search tool for any system using semantic web technologies i.e. linked open data and a SPARQL endpoint to access LOD data. Whilst being tested on single SPARQL endpoint, the current version is best suited for a single domain search though it has provision for federated SPARQL queries too.

The proposed framework contains a number of modules that may work independently with little configuration i.e. ontology processor, query optimizer, query formatter, API etc. This flexibility enables the framework to be used as a whole or as a part of other search tools.

VIII. CONCLUSION

Understanding user queries is one of the biggest challenges for a semantic search engine. Openness of user queries makes it harder for one algorithm to deal with all type of queries. The solution to this issue is to categorise the types of queries and handle them accordingly. The proposed solution in this paper focuses on pure semantic queries and full-text pattern matching (for search like movies, books). The authors introduce a 'Bag of Keywords' where they save CPI values for each keyword and this helps the Query Optimiser to tag keywords. The solution has been tested using open semantic data available at DBpedia SPARQL endpoint. The generic approach for the solution makes it flexible to work with various repositories at larger scale.

REFERENCES

[1] J. Beal, "Weaknesses of Full text search", The Journal of Academic Librarianship, vol. 34, Number 5, pp. 438-444, 2008.

[2] J. Beal, "Geographical research and the problem of variant place names in digitized books and other full-text resources" Library Collections, Acquisitions, and Technical Services, vol. 34, Issues 2–3, pp. 74-82, 2010.

[3] G. Antoniou, F. V. Harmelen, "A Semantic Web Primer", 2nd Edition. London: MIT Press, 2008.

[4] T. Berners-Lee, "Weaving the Web", pp. 2-5. The Original Design and Ultimate Destiny of the World Wide Web by its Inventor. Harper :San Francisco, 1999.

[5] T. W. Finin, L. Ding, R. Pan, A. Joshi, P. Kolari, A. Java and Y. Peng, "Swoogle: searching for knowledge on the semantic web", Proceedings of the National Conference on Artificial Intelligence (AAAI), pp.1682–1683, 2005.

[6] A. Latif, M.T. Afzal, P. Hoefler, A.U. Saeed, and K. Tochtermann, "Turning keywords into URIs: simplified user interfaces for exploring linked data", *Proc. of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human* (ICIS '09). ACM, New York, USA, pp. 76-81, 2009.

[7] M. Manshadi and X. Li, "Semantic tagging of web search queries", Proc. of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: vol. 2, pp. 861- 869, 2009.

[8] DBPedia, The DBPedia Ontology (2014) [Accessed Febraury 15, 2015]. Internet, 2014.

[9] A. Fatima, C. Luca and G. Wilson, "User Experience and Efficiency for Semantic Search Engine," IEEE OPTIM 22-24 May 2014 pp. 924-929.

[10] A. Fatima, C. Luca, and G. Wilson, A New Framework for a semantic search Engine, In: Proceeding of 16th International Conference UKSIM, Cambridge, 2014 pp. 446-451.

[11] O. Harting and F. Huber, "A main memory index structure to query linked data," In: Proc. 4th Linked Data On the Web (LDOW11), 2011.

[12] L. Chang, W. Haofen, Y. Yong, and X. Linhao, "Towards Efficient SPARQL Query Processing on RDF Data," In:Tsinghua Science & Technology, vol. 15, issue 6, pp. 613-622, 2010.

[13] O. Harting, and J. Freytag, "Foundations of traversal based query execution over linked data," In: *Proc. 23rd ACM conference on Hypertext and social media* (HT '12), pp. 43-52. ACM, New York, USA, 2012.

[14] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V.C. Doshi, J. Sachs, 2004. Swoogle: A Search and Metadata Engine for the Semantic Web. In: 13th ACM Conference on Information and Knowledge Management, Washington D.C.

[15] Motta, E., Sabou, M., 2006. Next Generation Semantic Web Applications. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 24–29. Springer, Heidelberg.

[16] G. Tummarello, R. Delbru, E. Oren,. Sindice.com: Weaving the open linked data. The Semantic Web, 552-565. Springer Berlin Heidelberg, 2007.

[17] M. d'Aquin, M. Sabou, E. Motta, S. Angeletou, L. Gridinoc, V. Lopez and F. Zablith, "What can be done with the Semantic Web? An Overview of Watson-based Applications," 5th Workshop on Semantic Web Applications and Perspectives, SWAP Rome, Italy, 2008.