From Keywords to Queries: Discovering the User's Intended Meaning

Carlos Bobed, Raquel Trillo, Eduardo Mena, and Sergio Ilarri

IIS Department
University of Zaragoza
50018 Zaragoza, Spain,
{cbobed,raqueltl,emena,silarri}@unizar.es

Abstract. Regarding web searches, users have become used to keyword-based search interfaces due to their ease of use. However, this implies a semantic gap between the user's information need and the input of search engines, as keywords are a simplification of the real user query. Thus, the same set of keywords can be used to search different information. Besides, retrieval approaches based only on syntactic matches with user keywords are not accurate enough when users look for information not so popular on the Web. So, there is a growing interest in developing semantic search engines that overcome these limitations.
In this paper, we focus on the front-end of semantic search systems and propose an approach to translate a list of user keywords into an unambiguous query, expressed in a formal language, that represents the *exact* semantics intended by the user. We aim at not sacrificing any possible interpretation while avoiding generating semantically equivalent queries. To do so, we apply several semantic techniques that consider the properties of the operators and the semantics behind the keywords. Moreover, our approach also allows us to present the queries to the user in a compact representation. Experimental results show the feasibility of our approach and its effectiveness in facilitating the users to express their intended query.

1 Introduction

The Web has made a huge and ever-growing amount of information available to its users. To handle and take advantage of this information, users have found in web search engines their best allies. Most search engines have a keyword-based interface to allow users to express their information needs, as this is an easy way for users to define their searches. However, this implies that search engines have to work with an input that is a simplification of the user's information need. Besides, most search engines follow an approach based mainly on syntactic matching techniques, without taking into account the semantics of the user keywords. Thus, when users are not looking for information very popular on the Web, they usually need to browse and check many hits [18].

Under these circumstances, Semantic Search [7] can help to overcome some current problems of the searching process and provide users with more precise

results. In any search system, and particularly in a semantic search engine, the main steps performed are: query construction, data retrieval, and presentation of results [1]. In this paper we focus on the first step, as capturing what users have in mind when they write keywords is a key issue to retrieve what they are looking for. Although keyword-based interfaces create a semantic gap, we focus on this type of systems due to their massive adoption [9]. Moreover, it is not clear that current natural language (NL) interfaces perform better than keyword-based interfaces in the context of Web search, as systems based on NL interfaces require performing more complex tasks, as indicated in [19].

Several approaches (e.g., [15, 18]) advocate finding out first the intended meaning of each keyword among the different possible interpretations. For instance, the keyword "book" could mean "a kind of publication" or "to reserve a hotel room". These approaches consult a pool of ontologies (which offer a formal, explicit specification of a shared conceptualization [5]) and use disambiguation techniques [13] to discover the intended meaning of each user keyword. So, plain keywords can be mapped to ontological terms (concepts, roles, or instances).

Identifying the meaning of each input keyword is a step forward. However, several queries might be behind a set of keywords, even when their semantics have been properly established. For example, given the keywords "fish" and "person" meaning "a creature that lives and can breathe in water" and "a human being", respectively, the user might be asking for information about either biologists, fishermen, or even other possible interpretations based on those meanings. Therefore, it is interesting to find queries that represent the possible semantics intended by the user, to allow her/him choose the appropriate one and perform a more precise search [17]. Moreover, the queries generated should be expressed in a formal language to avoid the ambiguity of NL and express the user information need in a more precise way. The formal queries generated can be used for different purposes, such as semantic information retrieval or any other task that needs to find out the intended meaning behind a list of user keywords.

In this paper, we present a system that translates a set of keywords with well defined semantics (*semantic keywords*, they are mapped to ontology concepts, roles or instances) into a set of formal queries (expressed in different selectable query languages) which attempt to define the user's information need. Any query that the user may have in mind will be generated as a candidate query as long as an expressive enough output query language has been selected. Our system applies several semantic techniques to reduce the number of possible queries while avoiding the generation of inconsistent and semantically equivalent ones. Finally, it uses a compact representation of the queries to present them to the user. Experimental results show the feasibility of our approach and its effectiveness in reducing the number of queries generated, achieving the goal of not overwhelming users with a huge list of queries.

The rest of this paper is as follows. Firstly, in Section 2, we explain our approach to query generation. The semantic techniques used to reduce the number of generated queries are explained in Section 3. Then, the way in which our system obtains compact representations of the final queries is shown in Section 4.

Experimental results that prove the interest of our proposal are presented in Section 5. We discuss some related work in Section 6. Finally, the conclusions and future work are drawn in Section 7.

2 Semantic Query Generation

In this section, we present our approach to query generation [3]. Our system takes as input a list of keywords with well-defined semantics (*semantic keywords*), chosen from the set of terms of one or several ontologies or mapped automatically to ontology terms by using the semantic techniques described in [18]. Then, considering the semantics of the output query language selected, it outputs a list of possible queries that could comprise the user's information need. The main steps are shown in Figure 1. Part of the generation process is performed by considering only the type of each keyword (concept, role, or instance). This allows the system to cache results and improve its performance in future iterations.

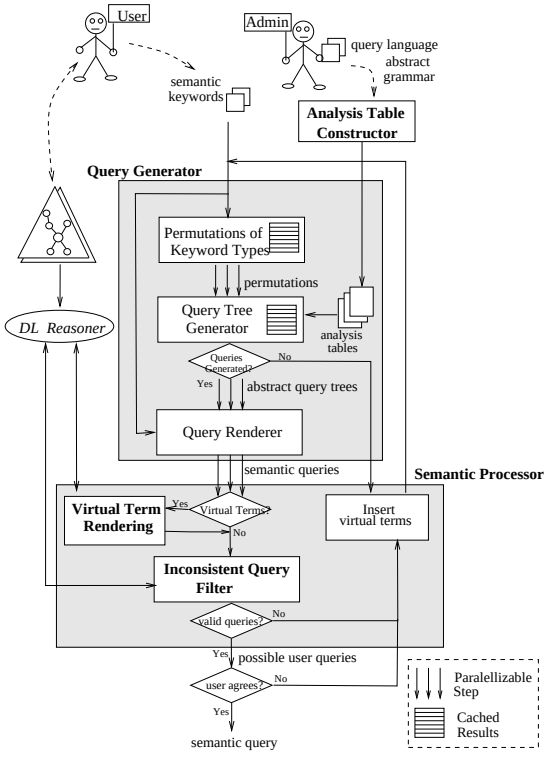


Fig. 1. From semantic keywords to formal queries.

To illustrate the different steps along the paper, let us assume that a user enters "person fish" to find information about people devoured by fishes. They

are mapped to the homonym terms in the ontology Animals[1]. Simplified versions of the language BACK [14] and the language DIG [16] will be used to show the flexibility of the approach[2]. The following steps are executed:

- *Permutations of keyword types*: To decouple the generation process of any specific language, its first stage is syntax-based. So, the system firstly obtains all the possible permutations of the types of terms (concepts, roles, and instances) corresponding to the input keywords, to discover any syntactically possible query in latter steps. In this case, *person* and *fish* are concepts, so the output of this step would be <C,C>, as no more permutations are possible.
- *Generation of abstract query trees*: For each permutation obtained in the previous step, the system generates all syntactically possible combinations according to the syntax of the selected output query language. We call these combinations *abstract queries* because they have *gaps* that will be filled later with specific concepts, roles, or instances. These abstract queries are represented as trees, where the nodes are operators and the leaves are *typed gaps* (concept, role, or instance gaps). Following the previous example, with the input <C,C> and simplified BACK as output language, And(C,C) would be built as an abstract query.
  In this process, the system uses bottom-up parsing techniques and analysis tables where the semantics of the selected output query language, but not its syntactic sugar, is stored. These tables are built only once, using a special annotated grammar, when a new output query language is made available to the system (*Analysis Table Construction* step). The semantics of the operators are considered to avoid generating equivalent queries (see Section 3.1).
- *Query rendering*: For each abstract query tree generated, the gaps in the leaves are filled with the user keywords matching the corresponding gap type. Then, the query trees are rendered into a full query in the selected target language. Several properties of the operators are also considered in this step to avoid generating equivalent queries (see Section 3.1). The result of this step for the running example would be And(Person, Fish).
- *Inconsistent query filtering*: The result of the previous steps is a set of syntactically correct queries. However, some of these queries might not be semantically correct according to the used ontologies. So, the system filters out the inconsistent queries with the help of a Description Logics (DL) reasoner [2] compatible with the chosen output query language. In our example, And(Person, Fish) would be removed in this step as it is classified as being inconsistent (*person* and *fish* are defined as disjoint classes).
  The performance of this step is greatly boosted by the fact that the set of generated queries forms a *conservative extension* [4] of the original ontology. Once an ontology has been classified, this property makes it possible to

[1] `http://www.cs.man.ac.uk/~rector/tutorials/Biomedical-Tutorial/Tutorial-Ontologies/Animals/Animals-tutorial-complete.owl`

[2] The simplified version of BACK consists of four operators (*And*, *Some*, *All*, *Fill*). Simplified DIG is equivalent to simplified BACK plus the *Or* operator.

evaluate the satisfiability of the queries in polynomial time, as each query does not assert new knowledge into that ontology.
- *Insert virtual terms*: When no query is generated or satisfies the user, the system is able to consider new ontology terms. This is performed by adding *virtual terms* (*VTs*) to the original list of user keywords. VTs represent possible terms that the user may have omitted as part of her/his query. Then, the previous steps are executed again. In the example, the new inputs considered would be "person fish concept" and "person fish role". This allows the system to build, among others, And(Person (Some (role, Fish)).
- *Virtual term rendering*: If a built query includes VTs, this step replaces them by appropriate terms extracted from the background knowledge ontologies considered, in order to build a possible query. To build only relevant queries, the system narrows the set of candidate terms by using the ontology modularization techniques described in [10] (see Section 3.2). So, in the example, the previous enriched abstract query is rendered into And(Person (Some (is_eaten_by, Fish)), which was the initial user's intended query.

Note that a query generation that involves filling a semantic gap will generate many queries. Even when the meaning of each keyword has been perfectly established (and thus, its polysemy avoided), the number of possible interpretations will grow as the number of user keywords increases and the output query language allows to combine them in more ways. Besides, we do not want to limit our approach to provide the most popular results (syntactic search engines do that already), so we aim at not missing any possible interpretation (according to the accessible knowledge). We are aware that this may lead to the generation of a high number of queries (as there are many possible interpretations), and therefore in the rest of the paper we propose different semantic techniques to deal with this problem.

3 Avoiding the Generation of Useless Queries

As stated previously, trying to guess the user's information need is a hard task due to the high number of possible interpretations. In our example, for the input "person fish" and one extra VT, there exist 780 syntactically possible queries for simplified BACK and 2832 for simplified DIG. So, it is critical to reduce the number of generated queries. Apart from the number of input keywords, there are two main elements that lead to this high number of possible queries: the expressivity of the output query language, and the semantic enrichment step with VTs performed to fill the semantic gap.

On the one hand, expressive query languages are very valuable, as they are more likely to be able to represent the user's intended query. However, the higher the number of operators, the higher the number of possible queries (the operands can be combined in more different ways). On the other hand, adding new terms to the user's input can help to discover the intention of the user. However, this will also increase the number of possible queries, mainly for two reasons: 1) new

possible interpretations appear, and 2) there may be a high number of candidates to replace a given VT (some of them probably irrelevant).

Along this section, we show how the system deals with these two issues. Then, in Section 4, we present a semantic technique that is applied to further simplify the output of the query generation.

3.1 Considering the Expressivity of the Query Language

For expressivity of a query language we understand its set of operators and the possible ways in which they can be combined to form a proper query. The more operators the language has and the more ways to combine them exist, the more queries will be possible. For example, if you add the *Or* operator to a language that had the *And* operator, the number of possible queries for a user input considering both operators will be larger than the double. There is apparently no way to reduce this number because the different options express different queries. However, we can avoid building equivalent queries along the generation process by considering the semantic properties of the operators.

With each new output query language that is made available to the system, a special grammar is provided as stated in Section 2. In this grammar, the properties of each operator are asserted. In particular:

- *associativity*: It is used by the *Query Tree Generator* to avoid, for example, building *and(and(c1, c2),c3)* if it has already generated *and(c1,and(c2,c3))*.
- *involution*: It is used by the *Query Tree Generator*, for example, to avoid building $not(not(c))$, which is equal to c.
- *symmetry*: It is used by the *Query Renderer* to avoid, for example, building *and(c2,c1)* if it has already generated *and(c1,c2)*.

Apart from those well-known properties, we consider two other properties of the operators: *restrictiveness* and *inclusiveness*. They are defined as follows:

Definition 1. *A binary operator* op *is* restrictive *if* $\exists f : K \rightarrow C,\ K = \{R, C\} \mid f(x) \sqsubseteq y \Rightarrow op(x, y) \equiv op(x, f(x))$.

Definition 2. *A binary operator* op *is* inclusive *if* $\exists f : K \rightarrow C,\ K = \{R, C\} \mid f(x) \sqsupseteq y \Rightarrow op(x, y) \equiv op(x, f(x))$.

From these definitions, and according to the semantics of the operators in BACK and DIG, it directly follows that the *And*, *Some* and *All* operators are restrictive and *Or* is inclusive. These properties are used in the *Virtual Term Rendering* step to avoid substituting the VTs by terms that would result in equivalent queries. For example, if we have $and(c1, concept)$, all the candidates that subsume c1 can be avoided as any of them *and* $c1$ would result in $c1$.

Following with the running example in Section 2, let us suppose that a user enters "person fish" to find information about people devoured by fishes. Adding one VT to that input, the system generated 780 queries using simplified BACK and 2832 queries using simplified DIG, which are reduced to 72 queries

(90, 77% reduction) and 364 (87, 15% reduction), respectively, by considering the semantics of the operators. In both cases the intended query *person and some(is_eaten_by, fish))* was among the final results.

3.2 Reducing the Number of Candidate Terms for Query Enrichment

Usually, users simplify the expression of their information needs when they write keyword queries, which contributes to the semantic gap. Thus, a user may omit terms that form part of the actual information need. To deal with the problem of possible information loss, some VTs can be added to the input, in order to generate the possible queries as if these VTs were proper terms introduced by the user. Then, the system performs a substitution of those VTs with actual terms extracted from the ontologies considered.

Following this idea, one can think about using each term of the same type as the inserted virtual one. For example, if the VT was a concept, the system could substitute it with all the concepts of the input ontologies (and generate one query for each different candidate concept). However, as the number of queries with VTs could also be high, considering all the terms of the input ontologies to render each VT for each query is too expensive.

In order to reduce the number of candidates for rendering a VT while avoiding losing any possible related term, we apply the modularization and re-using techniques explained in [10]. More specifically, the system uses ProSÉ, which, given a set of terms of an ontology (*signature*), allows to extract different *modules* that can be used for different purposes. Our system uses the user input terms as signature and extracts a module such that the same information can be inferred from the extracted module as from the whole ontology, as shown in [10]. This allows the discovery process to focus only on what is related to the input terms.

After applying the modularization techniques for the user query "person fish", the system generates 32 queries using BACK (15 after filtering, 98.07% less than the 780 original possible ones) and 148 queries using DIG (73 after filtering, 97.42% less than the 2832 original possible ones). The intended query is still among the final results, as the system does not miss any possible interpretation.

4 Extraction of Relevant Query Patterns

Besides reducing the number of generated queries, the way in which they are presented to the user also makes a difference. Users' attention is a capital resource and they can get easily tired if they are forced to browse over too many options to select their intended query.

In this stage of the search process, recall seems to be crucial as only one interpretation will fit the user's information need. Ranking the generated queries according to their probability of reflecting the user's interest can be an approach to minimize the interaction with the user. However, it is not clear how to identify the query that a specific user had in mind when writing a set of keywords, even

though the meanings of the individual keywords in the input have been identified previously. For example, with the input "person fish" the user might be looking for information about people devoured by fishes, but also about people who work with fishes (e.g., biologists, fishermen, ...), among other interpretations. Besides, a statistics-based approach may be not suitable for ranking queries, as it would hide possible interpretations that are no popular in the community. Approaches based on semantic and graph distances would also hide possible meanings.

Due to these reasons, we advocate a different and orthogonal approach to ranking. Our approach tries to minimize the amount of information that will be presented to the user by identifying query patterns, and could be combined with any query ranking approach if it is available. The semantic technique that we propose takes advantage of the syntactic similarity of the generated queries and of the ontological classification of the terms that compose the queries. A small example is shown in Table 1, where we can see that several queries may have a similar syntactic structure.

Table 1. Queries and patterns generated for "person bus" using BACK.

Queries	**Patterns**
all(drives, person) and bus	all(Role, Concept) and Concept
all(drives, bus) and person	
...	
some(drives, person) and bus	some(Role, Concept) and Concept
some(drives, bus) and person	
...	
all(drives, bus and person)	all(Role, Concept and Concept)
...	
some(drives, bus and person)	some(Role, Concept and Concept)
...	

Thus, the system analyzes the structure of the queries to extract common query patterns which lead to a compact representation of the queries. This is especially useful when the system tries to find out the user's intention by adding VTs. A query pattern shows an expression with the VTs not substituted (i.e., with *gaps*) and the system maintains a list of potential candidates for each gap.

At this point, a new challenge arises: how can the candidates for a gap be organized to facilitate their selection by the user? We advocate the use of a DL reasoner to show the candidates and allow users to navigate through their taxonomy. The interface shows a list of candidate terms and three buttons for each gap in each pattern (see Fig. 2):

- *Fix*: performs the substitution with the candidate term selected.
- *Subsumers*: enables the user to generalize the candidate term selected.
- *Subsumees*: enables the user to refine the candidate term selected.

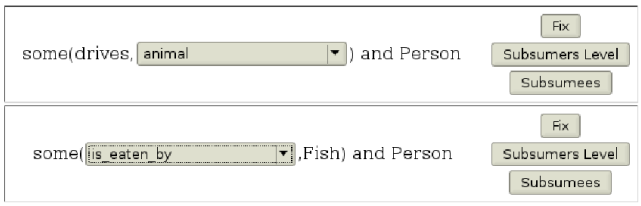


Fig. 2. Example of query patterns for "person drives" and "person fish".

This allows to show a high number of queries in a really compact way and provide a navigation in a top-down style through the terms of the ontology. Thus, the more general terms are initially presented to the user, who can then move through the taxonomy by accessing each time a direct subsumers/subsumees level. Users are allowed to select only terms that are relevant for the corresponding gap, i.e., their selections will never lead to an inconsistent query.

Thus, for example, for the input "person fish", the system is able to show the 15 final queries obtained using BACK under 7 patterns and the 73 obtained using DIG under 20 patterns. Moreover, this representation allows the system to establish an upper bound on the number of user clicks needed to select their query, which is equal to the depth of the taxonomy of the ontology multiplied by the number of substitutions to be performed (number of gaps).

5 Experimental Results

We developed a prototype to evaluate our approach in a detailed and systematic way. It was developed in Java 1.6 and using Pellet[3] 1.5 as background DL reasoner, and was executed on a Sunfire X2200 (2 x AMD Dual Core 2600 MHz, 8GB RAM). As knowledge bases, we selected two well-known ontologies: *People+Pets*[4] and *Koala*[5], which are two popular ontologies of similar size to those used in benchmarks such as the OAEI[6]. Due to space limitations, we only show the experimental results obtained with simplified BACK as output query language because most search approaches are based only on conjunctive queries. Nevertheless, we have also performed the experiments with simplified DIG, and we obtained similar conclusions (e.g., a similar percentage of query reduction).

For the experiments, we considered different sample sets of input terms and measured average values grouped by the number of terms in the set. It is important to emphasize that the terms in each set were not selected randomly but based on actual queries proposed by students of different degrees with skills in Computer Science. The sets were chosen according to the following distribution:

[3] `http://clarkparsia.com/pellet`
[4] `http://www.cs.man.ac.uk/~horrocks/ISWC2003/Tutorial/people+pets.owl.rdf`
[5] `http://protege.stanford.edu/plugins/owl/owl-library/koala.owl`
[6] `http://oaei.ontologymatching.org/`

10 sets with a single term (5 roles and 5 concepts), 15 sets with two terms (5 with 2 roles, 5 with 2 concepts, and 5 with 1 concept and 1 role), 20 sets with three terms (5 with 2 concepts and 1 role, 5 with 1 concept and 2 roles, 5 with 3 concepts, and 5 with 3 roles) and, following the same idea, 25 sets with four terms and 30 sets with five terms. Notice that, even though our approach can effectively deal with instances as well, we do not consider sets with instances because the selected ontologies do not have instances (as it happens frequently [20]). We set the maximum number of keywords to 5, as the average number of keywords used in keyword-based search engines "is somewhere between 2 and 3" [12]. This way we can see how our system performs with inputs below and above this value.

Our experiments measured the effectiveness in reducing the amount of queries generated and the query pattern extraction, and the overall performance.

5.1 Reducing the Number of Queries Generated

Firstly, we analyze the impact of the techniques explained in Section 3. For each set of terms, we executed the prototype considering no VTs and one VT in the enrichment step. The results of the experiment are shown in Figs. 3.a and 3.b, where the Y-axis represents the number of generated queries in log scale and the X-axis is the number of user keywords. We observe in both cases (Figs. 3.a and 3.b) that the number of queries generated increases with the number of input terms, as the more operands there are the more queries can be built. Moreover, performing the semantic enrichment leads also to a significant increase in the number of queries because many new interpretations appear. However, the percentages of reduction are satisfactory as they vary from the 33% when using 1 keyword up to the 90% when using 5 keywords, and, what is more important, the user's intended query is always in the generated set.

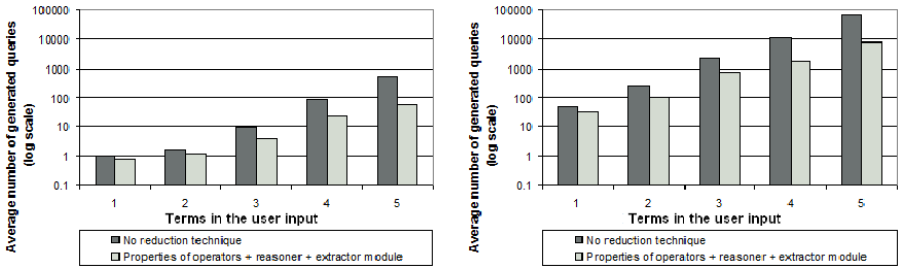


Fig. 3. Number of queries with: a) no VT, b) one VT.

5.2 Extracting Relevant Query Patterns

Having all the reduction techniques turned on, we now focus on the detection of query patterns (explained in Section 4). So, in this experiment we measure the number of query patterns that are extracted by the system to group the queries generated. Figure 4, where the Y-axis is in log scale, shows the benefits of providing query patterns to the user, instead of the plain list of queries generated. With few input terms and no VTs considered, the number of queries generated is quite low. However, as the number of input terms increases or when VTs are considered for query enrichment, the list of query patterns is very useful. Thus, it is about 50% smaller than the list of queries when no VTs are considered and about 92% when one extra VT is added. So, thanks to the use of the patterns identified with our semantic approach, the user can select her/his intended query much more easily than by browsing a plain list of possible choices.

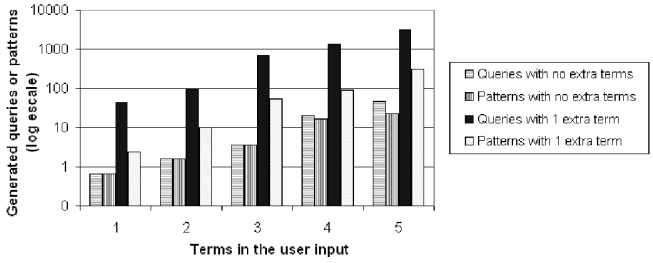


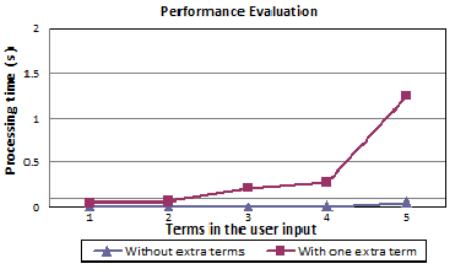
Fig. 4. Reduction obtained by extracting relevant query patterns.

5.3 Performance Evaluation

Finally, we focus on the performance of the system as a whole. The average times that the generation process takes are shown in Figure 5.a. They include the generation and the semantic filtering times. Being the low they are makes the system suitable to be a responsive front-end. Figure 5.b shows the maximum number of clicks required to reach any possible query allowed by the output query language considered in the experiments. With one VT, more effort may be needed as the number of possible queries grows considerably. However, as the user navigates through the taxonomy of candidate terms, s/he could consider different possibilities that s/he overlooked to fulfil her/his information needs.

6 Related Work

One of the first systems whose goal is building queries from keywords in the area of the Semantic Web was SemSearch [11]. In this system, a set of predefined templates is used to define the queries in the language SeRQL. However, not all



(a)

Maximum number of clicks

Keywords	Without extra terms	With one extra term
1	1	2.53
2	1	3.50
3	1	8.25
4	2	11.00
5	3	33.33

(b)

Fig. 5. Performance evaluation: processing time and maximum number of clicks.

the possible queries are considered in those templates, and therefore the system could fail in generating the user's intended query. Besides, it requires that at least one of the user keywords matches an ontology concept. On the contrary, our system considers all the possible queries syntactically correct in a query language selectable by the user and it works with any list of keywords.

Other relevant systems in the area of semantic search are [17] by Tran et al., Q2Semantic [8], SPARK [22], and QUICK [21]. These systems find all the paths that can be derived from a RDF graph, until a predefined depth, to generate the queries. However, these approaches can only generate conjunctive queries, whereas our system is able to generate any query that can be expressed in the selected output query language. Besides, they do not support reasoning capabilities for query reduction, as opposed to our system. In Q2Semantic, three possible ranking schemes are proposed, but they are too attached to their query generation model and are not applicable to our approach. Regarding these approaches, our approach works at a higher semantics level, as it exploits the background knowledge not only to build new queries but also to infer which ones are satisfiable and to avoid the generation of irrelevant ones.

There are also some works in the area of databases to provide a keyword-based interface for databases, i.e., translating a set of keywords into SQL queries. However, as emphasized in [6], most of these works rely only on extensional knowledge and neglect the intensional one (the structural knowledge). Besides, most of these approaches build only conjunctive queries and require the use of a specific language to express some constraints (e.g., *less than*). As opposed to these proposals, our approach considers both the intensional and the extensional knowledge, it is not tied to a specific query language, and it considers all the possible constraints of the selected query language.

Summing up, to our knowledge, our system is the only one with the goal of building formal queries from a set of keywords that uses a DL reasoner to infer new information and remove semantically inconsistent queries. Besides, no other approach is as independent as ours of the specific output query language. Finally, despite the potentially high number of queries, our system keeps it manageable by using several semantic techniques.

7 Conclusions and Future Work

In this paper, we have proposed an approach to discover the intended meaning behind a set of semantic keywords. To do so, our system translates the set of input keywords into unambiguous formal queries. Our approach is flexible enough to deal with any output query language (once the annotated grammar is provided), and so it can be used as a front-end for different semantic search engines. The proposed system has the following features:

- It exploits semantic properties of the operators (*associativity*, *involution*, *symmetry*, *restrictiveness*, and *inclusiveness*) of the output query language to avoid generating semantically equivalent queries.
- It filters inconsistent queries according to the used ontologies.
- It performs a semantic enrichment to fill the gap between the user keywords and the user's intended query. In this process, it uses modularization techniques to avoid generating irrelevant queries.
- It considers the semantics and structure of the generated queries to discover patterns that allow the user to locate her/his intended query iteratively with a small number of clicks.

Our experimental results show that, on average, our approach avoids generating an 80% of the possible queries without losing the user's intended meaning. Besides, the extraction of query patterns allows the user to reach any generated query with little effort, even if the user's query is not a popular interpretation.

As future work, we plan to integrate in a complete semantic search system, taking as starting point plain keywords [18]. We will also study ranking schemas and natural language translations for the query patterns in order to ease the query selection. Further experiments would be driven to evaluate the overall performance of our system and evaluate the user's satisfaction levels. Finally, we will move to the next stage of search by using the queries to access data once the exact meaning intended by the user is known.

Acknowledgments

This work has been supported by the CICYT project TIN2007-68091-C02-02.

References

1. *Proceedings of the Intl. Workshop on TRECVID Video Summarization (TVS'07), Germany*, 2007.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Pastel-Scheneider. *The Description Logic Handbook. Theory, Implementation and Applications*. Cambridge University Press, 2003.
3. C. Bobed, R. Trillo, E. Mena, and J. Bernad. Semantic discovery of the user intended query in a selectable target query language. In *7th Intl. Conference on Web Intelligence (WI'08), Australia*. IEEE, 2008.

4. B. Cuenca, I. Horrocks, Y. Kazakov, and U. Sattler. Modular reuse of ontologies: Theory and practice. *J. of Artificial Intelligence Research*, 31:273–318, 2008.
5. T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer Academic Publishers, 1993.
6. F. Guerra, S. Bergamaschi, M. Orsini, A. Sala, and C. Sartori. Keymantic: A keyword based search engine using structural knowledge. In *11th Intl. Conference on Enterprise Information Systems (ICEIS'09), Italy*. Springer, 2009.
7. R. V. Guha, R. McCool, and E. Miller. Semantic search. In *12th Intl. Conference on World Wide Web (WWW'03), Hungary*. ACM, 2003.
8. Q. L. Haofen Wang, Kang Zhang, D. T. Tran, and Y. Yu. Q2semantic: A lightweight keyword interface to semantic search. In *5th Intl. Semantic Web Conference (ESWC'08), Spain*. Springer, 2008.
9. L. Q. J. X. Yu and L. Chang. *Keyword Search in Databases*. Morgan & Claypool, 2010.
10. 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 *5th European Semantic Web Conference (ESWC'08), Spain*. Springer, 2008.
11. Y. Lei, V. S. Uren, and E. Motta. SemSearch: A search engine for the semantic web. In *15th Intl. Conference on Knowledge Engineering and Knowledge Management (EKAW'06), Czech Republic*. Springer, 2006.
12. C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
13. R. Navigli. Word sense disambiguation: A survey. *ACM Computing Surveys*, 41(2):1–69, 2009.
14. C. Peltason. The BACK system – an overview. *ACM SIGART Bulletin*, 2(3):114–119, June 1991.
15. W. Rungworawut and T. Senivongse. Using ontology search in the design of class diagram from business process model. In *Transactions on Engineering, Computing and Technology*, volume 12, March 2006.
16. P. C. Sean Bechhofer, Ralf Möller. The DIG description logic interface: DIG/1.1. In *Intl. Workshop on Description Logic (DL'03), Italy*, 2003.
17. D. T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *25th Intl. Conference on Data Engineering (ICDE'09), China*. IEEE, 2009.
18. R. Trillo, J. Gracia, M. Espinoza, and E. Mena. Discovering the semantics of user keywords. *Journal on Universal Computer Science. Special Issue: Ontologies and their Applications*, pages 1908–1935, November 2007.
19. Q. Wang, C. Nass, and J. Hu. Natural language query vs. keyword search: Effects of task complexity on search performance, participant perceptions, and preferences. In *Human-Computer Interaction (INTERACT'05), Italy*. Springer, 2005.
20. T. D. Wang, B. Parsia, and J. A. Hendler. A survey of the web ontology landscape. In *5th Intl. Semantic Web Conference (ISWC'06), USA*. Springer, 2006.
21. 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.
22. Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. SPARK: Adapting keyword query to semantic search. In *6th Intl. Semantic Web Conference (ISWC'07), South Korea*. Springer, 2007.