SWiPE: Searching Wikipedia by Example

Maurizio Atzori *
Math/CS Department
University of Cagliari
09124 - Cagliari, Italy
atzori@unica.it

Carlo Zaniolo †
Computer Science Department
University of California
Los Angeles, CA 90095, USA
zaniolo@cs.ucla.edu

ABSTRACT

A novel method is demonstrated that allows semantic and well-structured knowledge bases (such as DBpedia) to be easily queried directly from Wikipedia's pages. Using *Swipe*, naive users with no knowledge of RDF triples and SPARQL can easily query DBpedia with powerful questions such as: "Who are the U.S. presidents who took office when they were 55-year old or younger, during the last 60 years", or "Find the town in California with less than 10 thousand people". This is accomplished by a novel *Search by Example* (SBE) approach where a user can enter the query conditions directly on the Infobox of a Wikipedia page. In fact, *Swipe* activates various fields of Wikipedia to allow users to enter query conditions, and then uses these conditions to generate equivalent SPARQL queries and execute them on DBpedia. Finally, *Swipe* returns the query results in a form that is conducive to query refinements and further explorations. *Swipe*'s SBE approach makes semi-structured documents queryable in an intuitive and user-friendly way and, through Wikipedia, delivers the benefits of querying and exploring large knowledge bases to all Web users.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms

Algorithms, Design, Experimentation, Human Factors.

Keywords

Structured query interface, visual query language, semi-structured data querying

*Work founded in part by RAS Project CRP-17615 *DENIS: Dataspaces Enhancing Next Internet in Sardinia*.

†Work performed in part under the aegis of the Visiting Professor Program at the University of Cagliari.

WWW 2012 Companion, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1230-1/12/04.

1. INTRODUCTION

There has been much recent interest in querying the massive knowledge bases, such as DBpedia [1] and Yago [8], that were harvested from Wikipedia and other Web sources. Indeed, DBpedia contains more than one billion pieces of information organized as RDF triples. Using systems such as Virtuoso this public-domain database can be searched using powerful SPARQL [11, 13] queries. The realization that these new capabilities can deliver major benefits to users and applications has recently stimulated much research interest [12, 6, 7]. Indeed this will take us from Web-search engines and their keyword-oriented searches toward the much more powerful query capabilities of database management systems, whereby the following queries will be supported:

Q1 Who are the politicians that studied in the same university as Nicolas Sarkozy? Among them, who belongs to a political party that is different from his?

Q2 Which Italian singers/songwriters belong to the same genre as Madonna's music genres?

Q3 What is the average population of California cities with less than 10 thousand people, and what is the largest of those cities and its population?

The excitement of having these powerful queries available on such a large knowledge base is moderated by the realization that there is no simple user-friendly way to pose such queries: the power of DBpedia and Virtuoso is only available to those who, besides SPARQL, know the internal DBpedia names of entities and attributes (i.e., names like: `foaf:givenName` and `dbpprop:populationTotal`).

This situation has motivated many interesting approaches to improve the access to DBpedia for casual Web users, including the two discussed next.

Exploratory Browsing. This approach allows users to navigate through the triplets of the SPARQL graph by starting from an entity (node) and then, by clicking on a property (edge), move to another related entity[1]. Although the user does not need to know the exact names of properties in advance, this approach can only be used effectively for exploring the graph in the vicinity of the original entity. No close integration with Wikipedia is provided in this approach.

Faceted Search. An interesting user interface that supports a top-down search on DBpedia triplet graph is proposed in [6]. This approach deserves many praises insofar as

[1]Available at `http://dbpedia.org/fct`

(a) a Wiki page as example

(b) mouse move on field "Birth name"

(c) user types, introducing a constraint

(d) other constraints, then click "Swipe" button

Figure 1: The swipe interface. In this example, the user is asking for all "musician" people named "Mike" that doesn't play "Pop-rock".

it seeks to provide a user-friendly interactive way to query DBpedia[2] for users who are unfamiliar with either the internal representations used by DBpedia, or SPARQL. On the other hand, this approach only supports queries that can be expressed via a cascade of filters and, even for those, the query formulation process can be laborious requiring several iterations when many properties are present. Unfortunately, this is often the case, and the problem is compounded by the fact that property names often leave room for ambiguity.

For instance, a user searching for "cities in California" will have to start by supplying an item-type (e.g., "City"). This reduces the search space to cities, whereby the user can now specify a number of other type-depending filters, such as a latitude or `is-city-of`. If the user specifies `is-city-of = "California"`, the search returns no results, since this property in DBpedia only applies to rivers on which the city is a riparian settlement (e.g., the Hudson River, for New York) rather than the U.S. State to which the city belongs. Finally, queries requiring complex boolean expressions or aggregates are not supported in this approach. For instance, there is no simple way to express queries Q2 and Q3 above.

2. SWIPE

Our *Swipe* system[3] seeks to maximize both ease of use and query power by letting users work directly on the Wikipedia pages displayed in their browsers; there users specify their queries by marking up the information-box shown in the page–Fig. 1(b). The approach is inspired by the very successful Query By Example (QBE) interface of relational query languages and, along with ease-of-use offers significant

[2]Available at `http://dbpedia.neofonie.de/`

[3]*Swipe* is an acronym for *Searching WIkiPedia by Example.*

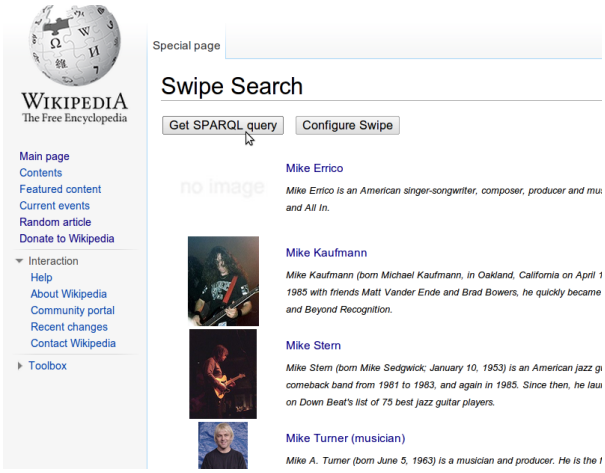

Figure 2: Results shown by the SBE query in Fig 1.

power and flexibility, whereby all the previously mentioned queries are easily expressed.

The *Swipe* system enables users to perform a *Search by Example* (SBE) on DBpedia by the following three steps:

1. A user starts by loading an "example" page in the browser, i.e., a page that represents the general kind of entities the user is interested in. For instance, users interested in cities might retrieve the Wikipedia page for "Boston" but any large city would do. This initial step provides the starting point for the actual query.

2. The example page looks like the original Wikipedia page. However, the Infobox of the page is now an active form that can be used to enter the search conditions defining the query.

3. The user specifies the query by typing into selected value fields in the Infobox, and then issues the query by a simple click. In response to that, *Swipe* returns a description of the Wikipedia pages that satisfy the query conditions entered by the user.

An example of the above interaction is the sequence shown in Fig. 1, where our user wants to find musicians whose birth name is "Mike" and "Pop-rock" is not among their genres. Once the mouse is over a particular field, the box changes to yellow denoting that it is ready to accept the input condition. Thus, our user can move the pointer to the box next to "Birth name" and type `Mike`, and then move it to the box next to occupation and enter `musician`. Finally, the user can type `"un(pop-rock)"`in order to exclude `pop-rock` from the acceptable music genres. Having thus specified conditions on three property values (the specification order is immaterial) the user can hit the *Swipe* button, whereby *Swipe* generates the SPARQL query that searches DBpedia using these conditions. The query results are then returned to the user as shown in Fig. 2 (this default output format is easily modified via the control buttons provided by *Swipe*).

2.1 The Query language

When the user moves the mouse over a field, this becomes editable whereby the user can enter various constraints, including complex conditions (via regular expressions) and

simple ones described next. To enter a simple condition the user can type *@const*, where *const* is either a number or a character string that does not contain spaces, commas, parentheses or other separators, and @ is one of the comparison operators: `>`, `<`, `<>` or `=`. Also `=` can be omitted and instead of `=`*const* the user can simply enter *const*. These conditions are then applied to the list of values in the Infobox, where spaces, commas and parentheses are treated as separators between successive list items. Therefore, the search for "Mike" in "Birth name" succeeds whenever Mike appears as the first-name, middle-name, or last-name. Likewise, typing `pop-rock` in the genres field returns musicians who have "pop-rock" among their genres. To select musicians whose genre list does not contain pop-rock, the user will write `un(pop-rock)` (whereas, `<>` `pop-rock` selects musician whose list contains some genre different from `pop-rock`).

Besides the negation operator `un( )` just discussed, *Swipe* supports conjunction and disjunction operators. Finally a variable, denoted by a starting `?`, can be used to denote the value of a field and relate it to the values of other fields, scalar functions, or aggregate functions [19]. The syntactic conventions used here are very similar to those of QBE.

Thus, the following queries can be easily expressed on the Robbie William page in Fig. 1(b):

List of all people named "Robert".
Set the "Birth name" field to `Robert`.

Find people whose name begin with "F".
Set the "Birth name" field to `>=F,<G`.

Find people that can play at least 4 instruments.
Set the "Instruments" field to `count()>=4`.

Find people whose number of played instruments is less than the number of her occupations.
Set the "Instruments" field to `?instr`.
Set the "Occupations" field to `count(?instr) < count()`.

Through its knowledge of the property names used by DBpedia, *Swipe* generates the SPARQL query that implement the SBE conditions entered by the user. The user can also see the actual query generated by *Swipe* (by clicking on the SPARQL button in Fig. 2) and then modify and resubmit the query. This is a very useful feature for more expert users who, having now seen the internal names of the attributes involved, can exploit the full power of SPARQL (e.g., to write queries that are not yet supported in the current implementation of *Swipe*).

3. THE SYSTEM

Swipe is a middleware system that implements the SBE paradigm on Wikipedia by (1) intercepting the browser's requests for Wikipedia pages to augment them with in-page UI scripts, and (2) answering structured queries posed by the user through the SBE interface. As shown in Fig. 3, these two functions are implemented by the User Interface module (UI) and the Query Manager module (QM).

3.1 The User Interface

The user loads Wikipedia pages on her browser through the *Swipe* Web Server that adds query-enabling code to the original page. This UI module acts as a middleware layer between the browser and Wikipedia Web servers, intercepting the user's request and returning a modified page (as shown in Fig. 3, left side). The *Swipe* pages look exactly as the original Wikipedia pages, except that fields in their Infobox are now active, inasmuch as they respond to the mouse being moved over them (as per the JavaScript `onMouseOver`)[4]. Thus, the UI module provides an Augmented Browsing experience to Wikipedia *Swipe* users, as shown in Fig. 1. The JavaScript introduced into the Wikipedia page by the UI module will be called SWipe Action Code (SWAC for short).

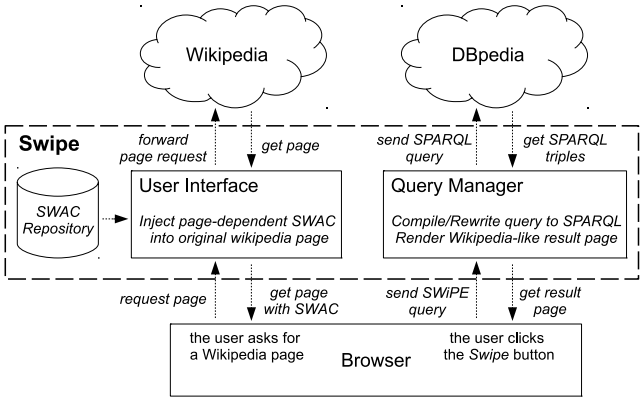

Figure 3: The architecture of *Swipe*. It is implemented as a middleware between the user's browser and both Wikipedia and DBpedia servers.

The information sent to the *Swipe* Query Manager includes the value(s) specified by the user, i.e., "Mike" for the example at hand, along with an internal identifier of the label next to it (i.e., "Birth name", which is specified in the Infobox template as `birth_name`). Information about internal field ID (`birth_name`), values ("Mike"), and field position (on the right of the "Birth name" label) are page-dependent, and not easily inferable from the html tree; however this information is needed to assure that the SBE interface reacts on user events and send the correct query to the QM module. Therefore, the SWAC scripts must include this page-dependent information. Thus, the UI module retrieve this information from the SWAC repository (leftmost part in Fig. 3) which is created by *Swipe* to store and manage the SWAC scripts.

3.2 The Query Manager

The conditions entered by the user are sent to the QM as pairs *(field-ID, condition)*; then, the QM converts field names used in the Infobox templates to the equivalent DBpedia property names and generate the SPARQL query. This conversion is far from trivial since the mapping from external names used in the Infobox to the internal ones used in DBpedia is not the same for all pages. Fortunately, this problem has already been addressed and solved by DBpedia, which has now derived a standard property ID for the labels of most of the Infoboxes in Wikipedia. Whenever needed, a single field query may be rewritten to a complex query making use of multiple fields in DBpedia. These rewriting rules

[4]Alternatively the user can register with Wikipedia as a *Swipe* user. Then, the Wikipedia server enhances all his/her pages with the *Swipe* skin, which rather than appearance-changing code contains query-enabling code.

can be customized by *Swipe* administrators[5]. A simplified version of the SPARQL query generated by the *Swipe* QM for the example in Fig. 1, i.e., "all musician named Mike that doesn't play pop-rock", is as follows:

```
SELECT DISTINCT ?res ?image ?comment
WHERE {
   ?res dbpprop:occupation ?occupation .
   FILTER (REGEX(STR(?occupation),  "musician", "i") )
   ?res dbpprop:birthName ?name.
   FILTER (REGEX(STR(?name), "mike", "i") )
   ?res dbpprop:genre ?genre.
   FILTER (! REGEX(STR(?genre), "pop-rock", "i") )

   OPTIONAL
   {    ?res foaf:page ?page.
        ?res dbpedia-owl:thumbnail ?image .
        ?res rdfs:comment ?comment.
        FILTER (LANG(?comment) = "en")
        ?res rdfs:label ?label.
        FILTER (LANG(?label) = "en")  }
} ORDER BY ?res LIMIT 100
```

The optimized SPARQL query is finally sent to DBpedia, which returns back a list of results. The QM module then further processes this list, providing a dynamic generated html page with associated images and clickable results in a Wikipedia-looking layout. An example of the result page is shown in Fig. 2. This page also provides buttons that can be used by the user to further manipulate the result list, e.g., by introducing aggregates.

4. RELATED WORK

In addition to Exploratory Browsing and Faceted Search [6] discussed in the Introduction, several other approaches were proposed to improve the Web users' searching experience [17, 10, 3, 2, 14]. A recent survey [15] provides a good introduction of the topic. For instance, several approach, including [18, 4], focused on improving the keyword-based search using contextual information and summarization.

While projects such as CSEwiki [5] focus on customizing the Google search engine for Wikipedia pages, other projects have focused on the new opportunities provided by the knowledge bases derivable from it. For instance, the work in [16] describes an interface for semantically annotating contents to help semantic Web-engines indexing. In [9], the authors propose an iterative querying (IQ) approach to search DBpedia and other knowledge bases via SPARQL queries. But while IQ simplifies some complex queries, it requires exact knowledge of the names of entities and properties used in the RDF graph, besides expertise in SPARQL.

5. CONCLUSION

While the *Swipe* prototype is undergoing continuous extensions and improvements, it has already achieved two important research objectives by demonstrating (i) the naturalness of its SBE approach and the ease with which new users can enter simple queries through its interface, and (ii) the high level of expressive power that are available to more experienced users via more advanced queries.

The basic features and capabilities of *Swipe* are easily learned from examples of popular Wikipedia pages describing common entities, such as cities, colleges, and movies. These will allow a user to ask queries on topics of common interest; other people watching our user can easily join in and help in the formulation of the queries. After a query is launched and the results are returned to the screen, users will also be able to see the equivalent SPARQL queries used by *Swipe* to search DBpedia, along with a description of how such queries were generated and the overall system works.

Besides being ease to learn and use, *Swipe* is also capable to support complex queries, such as those requiring aggregates, that are difficult, if not impossible, to express in other systems. More information about these and *Swipe*, including a longer technical report and a video preview is available from [19].

The effectiveness of the SBE query paradigm that *Swipe* has demonstrated for Wikipedia and DBpedia is not restricted to that environment. Indeed, generalizing this approach to other application areas represents an important objective for our future research.

[5]This personalization layer of *Swipe* QM is similar to user-defined mappings introduced in recent version of DBpedia.

6. ACKNOWLEDGMENTS

Thanks are due to Hamid Mousavi for his comments and to Kai Zeng for his implementation of DBpedia on Trinity. This work was supported in part by the National Science Foundation under Grant No. IIS 1118107.

7. REFERENCES

[1] S. Auer et al. Dbpedia: A nucleus for a Web of open data. In *ISWC/ASWC*, LNCS 4825, pages 722–735, 2007.

[2] Z. Bar-Yossef and N. Kraus. Context-sensitive query auto-completion. In *WWW '11*, pages 107–116, 2011.

[3] P. Ferragina and A. Gulli. A personalized search engine based on Web-snippet hierarchical clustering. In *WWW '05*, pages 801–810. ACM, 2005.

[4] L. Finkelstein et al. Placing search in context: the concept revisited. *ACM Trans. Inf. Syst.*, 20(1):116–131, 2002.

[5] Google. *A contextual search experience for Wikipedia (blog page)*, 2011. http://googlecustomsearch.blogspot.com/2009/10/contextual-search-experience-for.html.

[6] R. Hahn et al. Faceted wikipedia search. In *BIS*, volume 47 of *LNCS*, pages 1–11. Springer, 2010.

[7] A. Hartl, K. A. Weiand, and F. Bry. viskqwl, a visual renderer for a semantic Web query language. In *WWW '10*, pages 1253–1256.

[8] J. Hoffart et al. Yago2: exploring and querying world knowledge in time, space, context, and many languages. In *WWW (Companion Volume)*, pages 229–232. ACM, 2011.

[9] Y. Mass et al. Iq: The case for iterative querying for knowledge. In *CIDR*, pages 38–44, 2011.

[10] P. Papadakos et al. Exploratory Web searching with dynamic taxonomies and results clustering. In *ECDL*, *LNCS* volume 5714, pages 106–118. Springer, 2009.

[11] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *CoRR*, abs/cs/0605124, 2006.

[12] J. R. Pérez-Agüera et al. Inex+dbpedia: a corpus for semantic search evaluation. In *WWW '10*, pages 1161–62.

[13] E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. *W3C working draft*, 2008.

[14] A. Termehchy and M. Winslett. Keyword search over key-value stores. In *WWW '10*, pages 1193–1194.

[15] M. L. Wilson et al. From keyword search to exploration: Designing future search interfaces for the Web. *Foundations and Trends in Web Science*, 2(1):1–97, 2010.

[16] G. Wu et al. Falconer: once sioc meets semantic search engine. In *WWW '10*, pages 1317–1320, 2010. ACM.

[17] S. Xu, T. Jin, and F. C. M. Lau. A new visual search interface for Web browsing. In *WSDM '09*, pages 152–161.

[18] Yahoo! Y!q contextual search tool (2005). http://searchenginewatch.com/article/2066478/Yahoo-Offers-New-YQ-Contextual-Search-Tool.

[19] M. Atzori, C. Zaniolo. The SWiPE System: Searching Wikipedia By Example. UCLA/CS Tech. Rep. (2011). Additional preview resources about *Swipe* available at: http://swipe.i-mozart.com/www2012demo/