Supporting Semantic Web Search and Structured Queries on Mobile Devices

Andrea Dessi
University of Cagliari
09124 - Cagliari, Italy
andrea.dessi@unica.it

Andrea Maxia
University of Cagliari
09124 - Cagliari, Italy
andre.maxia@gmail.com

Maurizio Atzori
University of Cagliari
09124 - Cagliari, Italy
atzori@unica.it

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

ABSTRACT

There has been much recent interest in user-friendly interfaces that support queries and searching the Semantic Web, without requiring knowledge of SPARQL and the internal structure used by DBpedia or other knowledge bases. Although powerful, the existing proposals assume the use of desktop computers featuring rather large displays and pointing devices such as a mouse or trackpad. In this paper we tackle the problem of querying and searching the Semantic Web from mobile devices, by taking full advantage of their small touch-enabled screens. We focus on a user-friendly interface that can be used from smartphones, mini tablets, smart watches and possibly other wearable computers such as Google Glass. Indeed existing approaches become much less usable and effective on mobile since these support and require different modalities of user interaction. Our proposal is based on an adaptation of the recently proposed concept of SBE query system, developing a novel mobile interface that allows both browsing and querying the Semantic Web without using SPARQL nor knowledge of the underlying ontology/schema of the supporting knowledge base. To demonstrate the properties of the proposed interface we have developed *QPedia*[1], a mobile app that allows to take full advantages of DBpedia through our mobile-enabled user-friendly interface.

Keywords

Semantic Web Search, Mobile User Interface, Human Computer Interaction

[1]available at `http://dmi.unica.it/qpedia/`

SSW'13, August 30, 2013, Riva del Garda, Italy.

1. INTRODUCTION

The advent of smartphones and thus mobile computing confirm that the future of the Web is to create more transparency and simplicity, to allow an easy use though there exist problems such as low interoperability with the devices, small screens and more. In parallel, the recent evolution of the web, namely the Semantic Web, is growing rapidly, and contains a large amount of data and knowledge. The challenge thus will be to join Semantic Web technology and the mobile world to provide new additional supports for knowledge-based, location and context-aware information.

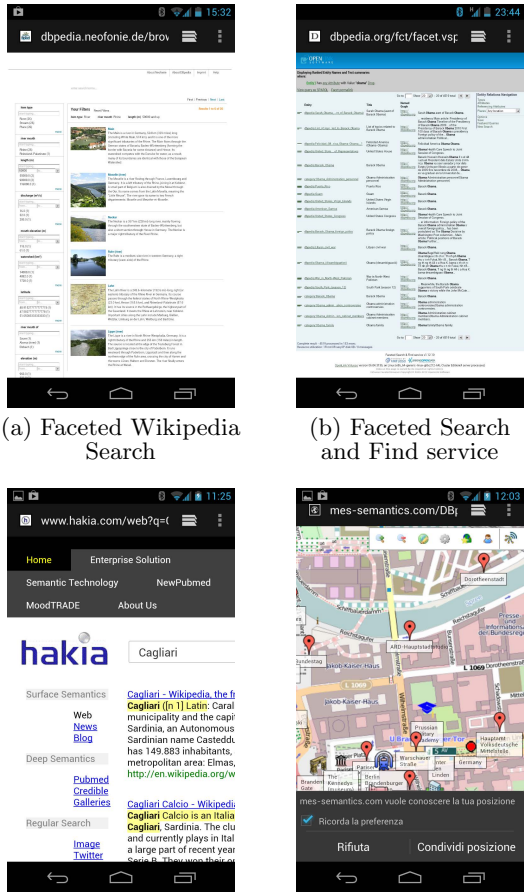
An excellent testing ground is DBpedia [2], a very well-known Semantic Web data source which provides information that could be useful for knowledge exploration, backed by the Virtuoso triplestore[2]. DBpedia is available through W3C standards for the Semantic Web and it stores its data as Resource Description Framework Schema (RDF/S) triples [6]. The DBpedia dataset has been extracted from Wikipedia and currently has more than 3.77 million things with 400 million facts. It also features labels and short abstracts in 15 different languages, 588,000 links to images and 3,150,000 links to external web pages.

There has been a number of useful web interfaces to navigate and query DBpedia and they are discussed in Section 4. Unfortunately they are based on interfaces that require a standard monitor and mouse, handling specific user events such as the `mouseover` event. Fig. 1 shows how four existing interfaces are rendered on a recent smartphone screen, drastically reducing the usability on such devices.

We propose a novel cross-platform system called *QPedia* which supports querying SPARQL endpoints dynamically without previous knowledge of web semantics from a mobile device. We try to address for the first time the problem of accessing and querying semantic web data coming from any endpoint (not necessarily DBpedia, and no assumptions on the schema or the content), using the search by example approach in [1] and adapting it to mobile devices.

Our work is therefore motivated [7] by the need for an easier way of using semantic web resources, such as DBpedia, for casual users accessing from a mobile device, therefore with a small screen, no proper pointing device and without knowledge of the ontology behind the Semantic Web. Our proposal and its related prototype *QPedia* described next

[2]`http://virtuoso.openlinksw.com/`



(a) Faceted Wikipedia Search

(b) Faceted Search and Find service

(c) Hakia

(d) DBpedia Mobile

Figure 1: Existing approaches rendered on a recent mobile browser

introduce a novel approach to display, query and interact with the Semantic Web from the mobile using well-known gestures, voice recognition, a simple way of introducing constraints and enabling location-based queries based on the user position.

2. THE QPEDIA APP

QPedia allows users to show DBpedia facts and search among them in an intuitive way from smartphones and other mobile devices. Searches can be done by providing keywords, values or ranges for properties (either through a keyboard or by voice), and/or location constraints, optionally based on the user location (through GPS if available).

The way constraints can be provided by the user leverages the achievements of the Search By Example approach in [1], where the constraint is associated to a specific RDF property without requiring the user to know the name of the property (e.g., `dbpedia-owl:birthplace` or `dbpedia:placeofbirth`).

QPedia can be accessed by a mobile phone's web browser, using as a development framework *jQuery Mobile* [4] which is compatible with all mobile browsers. The application can be used on any smartphone operating system and desktop, with an interface able to adapt to any resolution and method of interaction.

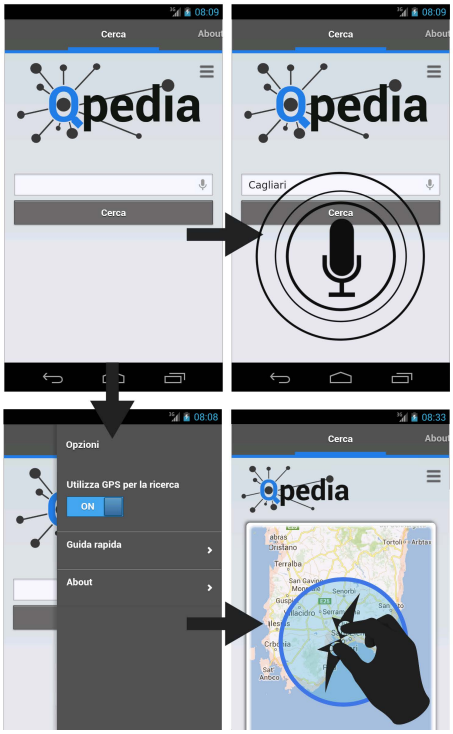


Figure 2: User interaction QPedia

QPedia's initial view contains a free text search and a search button. The way it works is very simple and intuitive: when the user enters some search terms (also by voice through speech recognition), and press enter or the search button, then the application will try to match those terms against DBpedia entities. In case the provided terms are too short or anyway no result is found, a dialog box will pop up warning the user inviting her to change the keywords. Otherwise, the matching results are shown.

Before starting a search, the user can flick (i.e., swiping with the finger) the page on the left, showing an map area that indicates the user position and allowing to select a location range constraint about the entities looked for. The map view can be unzoomed (and therefore the location range will update) through pinch-to-zoom, i.e., by touching screen's surface with two fingers bringing them closer together, or zoomed if moved them apart, in order to respectively increase or decrease the location range constraint. Location constraint can be easily switched off by a slide button. Figure 2 shows the various QPedia interface views under a recent Android web browser.

After launching the search, a new view will appear showing matched entities in DBpedia. By clicking on a result, its corresponding entity will be chosen, and its infobox (as in Wikipedia) will be shown. This is done to introduce entity data to the user in a familiar way. The user can then choose, flicking the view on the left or on the right, between the current infobox view, the advanced search view and the map view.

The advanced search view shows all the properties of the current resource, using an expandable listview instead of the infobox. In this view, by a long press on a property, it's possible to introduce a new constraint on the selected property.

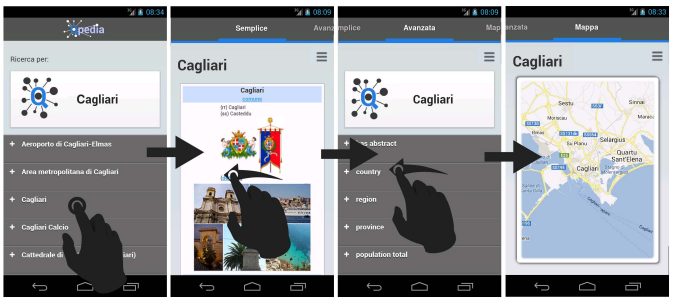
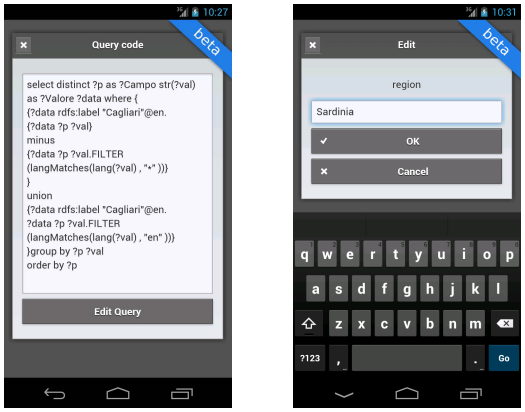


Figure 3: Searching and querying in QPedia

By further flicking, the map view is shown. If spatial information is available (such as latitude and longitude), the map will be centered on that point, also allowing to input a location constraint. Interactions among views are shown in Fig. 3.

By pressing the search button, it will be started a background SPARQL query generated by QPedia. Fig. 4 shows the raw SPARQL query, available to experienced users, for the corresponding query *"all Sardinian cities with population between* 15,000 *and* 25,000 *inhabitants"*. In order to write such query the user will just specify the constraints for each property of interest, by and editing dialog, such as changing `Sardinia` for the property "region" or the range `15000<> 25000` for the property "population total"; after pressing the search button in the action bar dialog, QPedia will list all Sardinian cities with a population total between 15,000 and 25,000.



(a) Background SPARQL Query

(b) Property Change

Figure 4: Search By Example in QPedia

3. IMPLEMENTATION

So far we described the user interface of QPedia. In order to achieve such user-friendly experience, QPedia is made of a number of modules that we review in the following.

The UI Module. This module is responsible for showing the user interface described in the previous Section. In order to be portable and available on the majority of the devices, this part has been developed using HTML and JavaScript, therefore accessible through any mobile browser. Most of the interface has been developed with *jQuery Mobile* [4], compatible with almost all browsers in use. QPedia should therefore be available on any smartphone operating system and desktop, with an interface self adapting to any resolution and method of interaction. The use of jQuery Mobile allows skins to be personalized depending on the device/OS used, enhancing the UX.

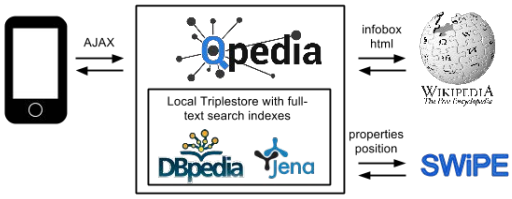


Figure 5: The QPedia System

The UI module is also in charge of communicating with the QPedia backend server through AJAX calls (see Fig. 5), sending user inputs and obtaining the elements to be shown in the interface. In particular, constraints provided by the user in the query are sent to the server, which in turn will answer with the query results. Results are graphically elaborated by the UI module before showing them to the user.

The Query Manager. This module is responsible to generate the SPARQL queries to be sent to the triplestore described next. The Query Manager is in charge of interpreting the conditions entered by the user through the user interface. These conditions can correspond to a text-based keyword search on some RDF properties (e.g., `dbpedia-owl:abstract`), a constraint on a location property (e.g., `geo:long` and `geo:lat`) of RDF entities expressed using the map view of QPedia UI, or a constraint on other RDF properties (currently any available in DBpedia).

Other than a query translation function, the query manager also provides alias for the items shown by the interface, i.e., instead of showing raw RDF attributes, the QM sends more explicative strings such as the ones obtained querying the `rdfs:label` property of the entity at hand.

Most of the features and solutions regarding this module are part of the Search by Example approach used in SWiPE [1]. One big difference in QPedia is that in SWiPE the user inputs the constraints within the HTML of the original infobox, which is a non trivial problem to solve given the fact that there is no markup to recognize the property position within the HTML, and RDF values do not necessarily match strings in the infobox. In our case, QPedia shows a structured list of properties that allows the user to input a constraint, therefore it is straight-forward to know which property the user was meaning to edit. On the other side, the properties should be shown where the user will expect to be, that is, in the same order as shown in the original infobox (which is one flick away from the advanced search view).

Finding the order in which properties appears within the infobox HTML is a new non-trivial problem to solve. Fortunately we can leverage the tools developed in [1] to find property positions and therefore, by inspecting the `top` CSS attribute, we can easily recover how properties are vertically sorted in the infobox HTML.

Triplestore / Execution Manager. This module is responsible for executing the SPARQL query and returning the results to the users. In order to ensure fast response and execution times we have experimented with alternative query execution engines. In particular, the Virtuoso system used in DBpedia proved too slow on some keyword-based queries where multiple attributes where involved. Further, the online service freely provided by DBpedia (either standard and "live" endpoints) showed low service availability when accessed programmatically. To solve theses performance problems we have developed a version of QPedia backend server that uses full-text Lucene indexes on a local server, based on a modified version of the *Apache Jena* triplestore. QPedia also features a mechanism that dynamically tries different endpoints whenever a service availability issue might occur.

Native Android Client. Generally a web application has limits and missing features, avoidable only through a native client. The restrictions primarily affect some performances, for example, using for a long time a web application the browser cache can get saturated and the UX will decrease. Other opportunities coming from a native app are the social aspects, integration with other mobile apps, sharing customized searches or new features like bookmarking favorited searches. Therefore we also developed an Android application based on a simple *webview*, optimizing performance and implementing such extra features. The main screen is a *Fragment Activity*[3] with a *PagerAdapter* that contains two sections, module search and favorites list, where it's possible to save each favorited search on smartphone physical memory. To overcome possible cache problems, every search runs on a different *webview* using a new javascript interface. This solution, after a number of tests has shown a significant performance improvement..

4. RELATED WORK

The existing proposals, such as SWiPE [1], Faceted Wikipedia Search [5] (Fig. 1a) and Virtuoso Faceted Web Search (Fig. 1b), allow users to ask complex queries only with a desktop user interface. In more detail SWiPE [1] generates automatically semantic queries for DBpedia using the Search by Example approach, helping people who do not have knowledge about SPARQL to pose their desired query. The system provides an interface like Wikipedia which has, on the infobox, editable fields to input the query. The user can choose which fields to modify in order to start a new query using shown information about the underlying related DBpedia page.

Another example of semantic web search engine is Hakia (see Fig. 1c), that brings relevant results based on concept match rather than keyword match or popularity ranking.

A few others try to address the problem of making web semantic data useful in a mobile context, such as DBpedia Mobile [3] (Fig. 1d), that provides a map view annotated with DBpedia entities and information from other knowledge bases. This application, based on geographic location, generates a map that contains information of the surrounding locations contained in the DBpedia dataset. It works on desktop browsers, while for mobile devices, the application is optimized for QVGA display (320x240 pixels) therefore not specifically focused on current devices (featuring full HD displays). Other than being designed for low-resolution screens, DBpedia mobile is a system that tackles only a specific search need, by focusing on locations. Therefore it is not addressed to the general problem of accessing and querying large datastore of (possibly) unknown domains.

[3] `http://developer.android.com/reference/android/support/v4/app/FragmentActivity.html`

5. CONCLUSION AND FUTURE WORK

This paper has presented a new graphical user interface which combines the advantages of both mobile devices and Semantic Web. We described in details how our application functions with different search modalities. Our proposal and its related prototype *QPedia* introduce a novel approach to display, query and interact with the Semantic Web from the mobile using well-known gestures, voice recognition, a simple way of introducing constraints and enabling location-based queries based on the user position. Future work will be devoted to extend our application with new features, such as *graph search* through constraints on multiple infoboxes, query composition and query templates.

Acknowledgements. Work funded in part by RAS Project CRP-17615 *DENIS: Dataspaces Enhancing Next Internet in Sardinia*, and by NSF under grant *IIS 1118107*.

6. REFERENCES

[1] M. Atzori and C. Zaniolo. Swipe: searching wikipedia by example. In A. Mille, F. L. Gandon, J. Misselis, M. Rabinovich, and S. Staab, editors, *WWW (Companion Volume)*, pages 309–312. ACM, 2012.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In K. Aberer, K.-S. Choi, N. F. Noy, D. Allemang, K.-I. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *ISWC/ASWC*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007.

[3] C. Becker and C. Bizer. Dbpedia mobile: A location-enabled linked data browser. In C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee, editors, *LDOW*, volume 369 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[4] M. Firtman. *jQuery Mobile - Up and Running: Using HTML5 to Design Web Apps for Tablets and Smartphones*. O'Reilly, 2012.

[5] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted wikipedia search. In W. Abramowicz and R. Tolksdorf, editors, *BIS*, volume 47 of *Lecture Notes in Business Information Processing*, pages 1–11. Springer, 2010.

[6] B. McBride. The resource description framework (rdf) and its vocabulary description language rdfs. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 51–66. Springer, 2004.

[7] M. L. Wilson, B. Kules, M. C. Schraefel, and B. Shneiderman. From keyword search to exploration: Designing future search interfaces for the web. *Foundations and Trends in Web Science*, 2(1):1–97, 2010.