i-MoCo: Mobile Conference Guide – Storing and querying huge amounts of Semantic Web data on the iPhone/iPod Touch

Cathrin Weiss, Abraham Bernstein, Sandro Boccuzzo

University of Zurich
Department of Informatics
CH-8050 Zurich, Switzerland
{lastname}@ifi.uzh.ch

Abstract. Querying and storing huge amounts of Semantic Web data – this has usually required a lot of computational power. This is no longer true if one makes use of recent research outcomes like modern RDF indexing strategies.
We present a mobile conference guide application that combines several different RDF data sets to present interlinked information about publications, conferences, authors, locations, and others to the user. With our application we show that it is possible to store a big amount of indexed data on an iPhone/iPod Touch device. That querying is also efficient is demonstrated by creating the application's actual content out of real time queries on the data.

1 Motivation

A typical conference scenario: We attend a session and are really interested in a particular talk. We probably have never seen the speaker before. Thus, we open our laptop and enter the speaker's name into the preferred web search engine and collect the information we are interested in, like, whom she's collaborated with before, at which conferences she's already published, and so on. The search results may also contain a lot of irrelevant information, especially if the author's name is a common one. We have to refine our search with respect to what we want to know and continue.

Instead of collecting this information manually it would be desirable to have a system providing us with all conference-/publication-relevant information that gives us the possibility to browse within this particular topic area. Another advantage would be if the formerly described application ran on a mobile device such that a laptop is not needed at all. But preparing all semantically-interlinked information that can be useful for this particular type of application can sum up to big amounts of data. Querying big amounts of data and especially big Semantic Web graphs [1, 2, 5] is a computationally expensive task. Doing this in real-time usually requires high-end systems. Therefore our motivation for this project was it to show that with modern, sophisticated indexing and storing

techniques one does not need much computing power for dealing with lots of RDF data. We therefore implemented our i-MoCo mobile conference guide application which runs on the iPhone/iPod Touch and is based on the Hexastore indexing scheme presented in [4].

The remainder of this paper is structured as follows: Section 2 gives some background information about the Hexastore index and why it is suitable to be used on a mobile device. Section 3 describes the i-MoCo tool with the design decisions made, its architecture, and implementational details. In addition to that we present some "application-in-action" screenshots. Last but not least we conclude with a project summary in Section 4.

2 Theoretical Background

To understand why it is possible to store (and also query) so much of the given Semantic Web data on a device with limited capacities, we need to give a short introduction in our indexing and storing scheme.

In [4], we presented Hexastore, a sextuple indexing scheme for Semantic Web data which enables querying the data efficiently. A Semantic Web triple $< s, p, o >$ consisting of the graph nodes *subject* s, and *object* o, although o can also be a literal, and the graph edge p, the *predicate*. The nodes are identified by a URI. Hexastore maps URI or literal strings onto unique ids to limit the amount of storage needed. Furthermore, Hexastore recognizes that queries are constructed by a collection of graph patterns [3] which may (i) bind any of the three elements of the triples to a value, (ii) may use variables for any of the triple elements effectively resulting in a join with other graph pattern, and (iii) define any element with a wild card to be returned. Since any join order between triple patterns in a query is possible, Hexastore indexes the data in each possible triple permutation such that efficient joining over every element s, p, or o becomes possible. This results in six indices designated in the order in which the triple elements are indexed (e.g., SPO, OSP, etc.). This indexing scheme allows to retrieval all connected triple components within one index lookup.

Figure 1 demonstrates the structure of the indices. It shows that each index essentially consists of three different elements: a first-level `Type1` index, a second-level `Type2` index, and a third-level `Type3` ordered set, where the `TypeN`'s are one of the three triple elements $\{subject, predicate, object\}$. Given a `Type1` key a_i, the first-level index returns a second-level `Type2` index. Given a key b_j the `Type2` index returns a `Type3` ordered set, which lists all the matches to the query $< a_i, b_j, ? >$. Note that this structure has the advantage that every lookup is of amortized cost of order $O(1)$.

Since this indexing scheme still yields all information contained in the raw data, it suffices to store just the indices. For that we also provide a customized concise storage scheme. The combination of both enables highly efficient querying of huge data with a minimal number of disk accesses.

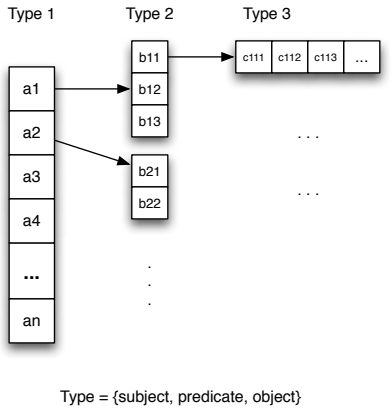

Fig. 1. Sketch of a Hexastore index

3 Tool Description

We wanted to design an application that is helpful during a conference visit. The idea behind is that every conference can provide customized data starting from which conference attendees can gather conference- and publication-related information.

The application we designed has the goal that starting from the publications published at the particular conference, the user can browse all kinds of related information:

1. Retrieve all authors related to the particular paper
2. Retrieve all other papers published by a particular author (using SwetoDBLP data)
3. Retrieve information possibly published on Wikipedia about a particular author (using DBPedia data)
4. Retrieve information about a location (using Geonames data)
5. Finding a location on the map (using the iPhone/iPod Touch Google Maps application)
6. Retrieve the PDF file of a particular publication (using Citeseer data)

3.1 Design

Since with a mobile device we are quite limited in the size of the presentation screen, design decisions have to be made with respect to:

- What information is crucial about the presented data?
- Which relations must necessarily be presented together?
- Short version of information versus long version (i.e., paper title versus paper represented with all related information)

– How much information should be shown at all and how?

With the iPhone/iPod Touch we are bound to particular UI components that can be used. How we actually implemented the application will be described in Section 3.3.

3.2 Architecture

The application consists of several layers. The UI layer, representing the actual results, a query layer, which maps the UI's requests onto the underlying index, and the index layer itself which contains the data. The architecture is depicted in Figure 2. First of all we need to create the Hexastore indices out of the RDF- (or in general: Semantic Web-) data. These indices are the application's lowest layer and are implicitly stored. On top of the index there is a query layer that communicates with the actual iPhone UI. Thus, the content within the application is dynamically created on demand by posing a request to the query layer which then fetches the requested information from the index layer.

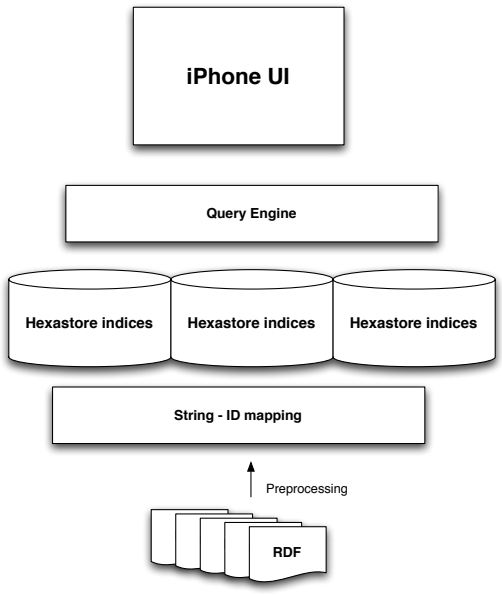

Fig. 2. The Application Architecture

3.3 Implementation

We implemented the application in *Objective C* with the *Apple iPhone SDK*[1] for the UI and programming logics. The index/database layer was implemented in *C* and could therefore easily be integrated within the Objective C Framework. For a minor part of the string-id-mapping we used sqlite databases which are supported by the iPhone SDK.

For the UI we mostly used TableViews within TouchLists, as well as (in case of only one option) simple buttons and included Images. For retrieval of PDFs or actual Wikipedia entries, we did URL calls in Mobile Safari. Geodata is shown within the integrated Google Maps application or – in case this application is *not* integrated – in Mobile Safari as well.

3.4 Data amount stored

Given our indexing and storing method, a full-indexing – full means with all possible indices and all string mapping information, which we usually do not need completely – requires approximately 100MB per million triples. This depends on the nature of the underlying data but that was the average space requirement for the data sets we used. Therefore, we can store *and* query about $250-300$ million fully indexed triples on an 32GB iPod Touch . If we don't store all indices, the whole billion is possible.

Efficient querying means that the construction of the application parts runs in real time. The necessary information is retrieved quickly and the user does not realize any lags.

3.5 Result

The application "in-action" is presented in Figures 3 to 8. In Figure 3 we see the application start: a list of all publications at the actual conference. By clicking on a particular publication we get a detailed description of the paper, like shown in Figure 4. There we have the paper title again, all authors, possible links to download the PDF file of the paper, and the conference at which it was published. If the particular fields contain information, they are clickable. If we have a link to a PDF file, we can download it as shown in Figure 5.

Continuing with a click on an author, we are forwarded to the author information table, here shown in Figure 6. This part contains – if available – a photo, the name and two links exchanging by clicking on the photo : (i) a link to all publications of the author, (ii) a link to a possible wikipedia entry – if available. We have (a) implicit Wikipedia data that can be requested from the data stored if no web access is available or (b) online access like shown in Figure 7.

Since conferences as well as people can be related to locations (conference location or birth location), we can upon this information retrieve a particular location in Google Maps via the location name or Geonames data (c.f. Figure 8).

[1] http://developer.apple.com/iphone/

4 Summary

We decided to use the iPhone/iPod Touch as our underlying platform, because applications need to fulfill some requirements to succeed on such a device, i.e. they need to be (i) memory efficient, (ii) storage efficient, and (iii) fast.

With our i-MoCo mobile conference guide we have demonstrated that it is possible to store and query a big amount of Semantic Web data on a mobile device like the iPhone/iPod Touch. Thus, we provide a convenient tool to browse quickly through conference and publication contents. i-MoCo combines so far data from DBLP, DBPedia, Citeseer, and Geonames. More data sets can easily be added if additional features require that.

We think that i-MoCo is a demonstration of, first, the improvement on Semantic Web data management achieved by modern methods, and second, the scalability of the indexing and storing techniques used.

References

1. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *ISWC*, pages 54–68, 2002.
2. R. V. Guha. rdfDB : An RDF database. `http://www.guha.com/rdfdb/`.
3. M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 595–604, New York, NY, USA, 2008. ACM.
4. C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple Indexing for Semantic Web Data Management. In *Vol. 1 of JDMR (formely Proc. VLDB) 2008*, Auckland, New Zealand, 2008.
5. K. Wilkinson, C. Sayers, H. A. Kuno, and D. Reynolds. Efficient RDF Storage and Retrieval in Jena2. In *SWDB*, pages 131–150, 2003.

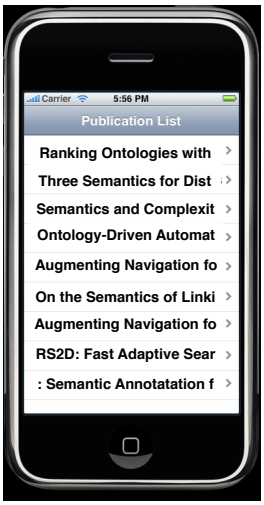

Fig. 3. Application start (all publications of particular conference)

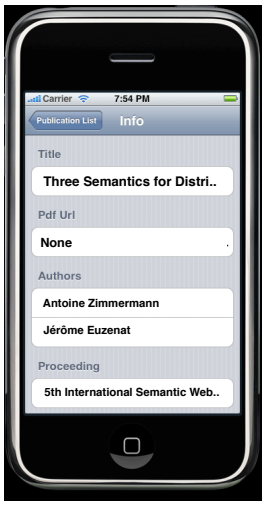

Fig. 4. Extended information about publication

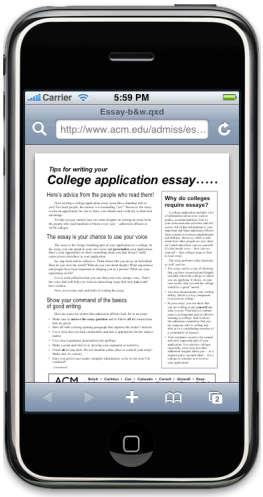

Fig. 5. Find paper as PDF

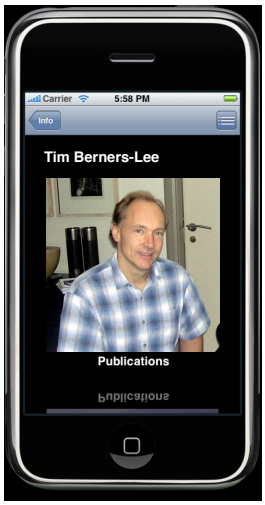

Fig. 6. Information about an author

Fig. 7. Retrieve Wikipedia information

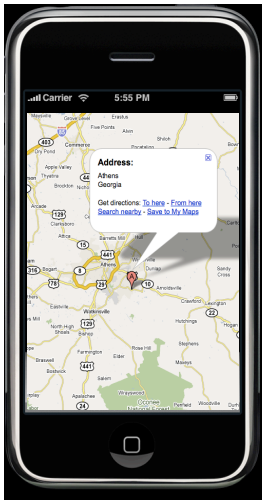

Fig. 8. Show a location in Google Maps