DISCLAIMER: version before final editing.

Full reference:
Van Woensel, W., Casteleyn, S., Paret, E., De Troyer, O.: "Mobile Querying of Online Semantic Web Data for Context-Aware Applications", IEEE Internet Computing Special Issue (Semantics in Location-Based Services), Vol. 15, N° 6, pp. 32-39, Eds. Sergio Ilarri, Arantza Illarramendi, Eduardo Mena, Amit Sheth, ISBN-ISSN: 1089-7801

Mobile Querying of Online Semantic Web Data for Context-Aware Applications

William Van Woensel[1], Sven Casteleyn[2], Elien Paret[1], Olga De Troyer[1]

[1]Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
{Elien.Paret, William.Van.Woensel, Olga.DeTroyer}@vub.ac.be
[2] Universitat Politècnica de València, Camino de Vera S/N, 46007, Valencia, Spain
Sven.Casteleyn@upv.es

Keywords: 8.III.III.II, IV [**Information Technology and Systems**]: Information Storage and Retrieving – *Information Search and Retrieval*, Information filtering, Metadata; 10.IX.I, II [**Computer Applications**]: Mobile applications – Location-dependent and sensitive

Abstract

Mobile devices are becoming increasingly multi-functional and personal, providing mobile applications with the necessary user information (e.g., preferences, personal calendar) to achieve personalization. At the same time, detection technologies (e.g., Radio Frequency IDentification, or RFID) allow mobile devices to detect nearby physical entities, and thus map the user's environment. By exploiting existing online data sources about these detected entities, mobile applications can further improve personalization by including knowledge on the mobile user's physical environment. Semantic Web sources are useful in this respect, as they are machine-readable and facilitate integration with other sources. SCOUT, developed by the authors, is a mobile application framework that supports the linking of physical entities to online semantic sources, and provides applications with an integrated, query-able view on these sources and the user's environment. In order to efficiently access this large set of distributed online semantic sources, a tailored data management approach has been developed.

1. Introduction

Modern mobile devices are fully-fledged web clients and personal information managers, capable of running applications previously only reserved for desktops. By exploiting the rich personal information captured by these devices (e.g., user preferences, social network data), the provided content and functionality can be personalized to the particular user. However, in a mobile setting, a user's needs also depend on his environment, not just on his personal profile. Detection and sensing technologies such as QR (Quick Response) codes or RFID allow mobile devices to detect tagged physical entities (i.e., people, places, things) in their surroundings, while online services such as LinkedGeoData [url-1] can be used to obtain entities near the user's current GPS location. Often, information on these physical entities is already available on the Web: for instance, in webpages or in Semantic Web sources. If a reference to this information can be extracted from the detected entity (e.g., by decoding the URL from a QR code), extensive knowledge on the user's environment can be obtained, allowing mobile applications to tailor functionality and content to the user's full context. For instance, around lunchtime, they can draw the user's attention to nearby restaurants serving his favorite cuisine, and provide walking

distance. Online Semantic Web data is especially useful, as it encodes rich semantic information about any digital or physical resource and is machine-readable. There already exist a large number of online RDF sources. The W3C Linking Open Data project [url-2] currently lists 216 datasets with sizes ranging from 1000 triples to over a billion. Increasingly, websites are also semantically annotated, using annotation languages such as microformats and RDFa. According to the Yahoo! BOSS API [url-3], around 955 million websites are currently annotated using RDFa. Such websites serve as semantic sources in their own right, as RDF data can be directly extracted from their annotations. As another advantage, Semantic Web technology supports interoperability between sources, allowing data to be easily integrated.

We present SCOUT [1], a mobile application framework that supports a range of methods to connect nearby physical entities to associated online semantic sources, and provides a data service that transparently handles the management, integration, and querying of this data. By offering this integrated, query-able view on the user's surroundings, mobile application developers are empowered to personalize and contextualize content and functionality. In order to efficiently manage all this information in a volatile environment and on limited devices, a tailored data management approach has been developed, exploiting the capabilities of Semantic Web technology. SCOUT is lightweight and scalable, and does not require proprietary servers or middleware; instead, it uses the Web itself as information system to obtain useful environmental information.

In the next section, we present the architecture of SCOUT. Subsequently, we present our data management approach, centered on a Semantic Web-based indexing strategy tailored to mobile environments, and provide an experimental validation. We then present related work, and end with conclusions and future work.

2. Architecture

The SCOUT framework consists of a number of distinct layers, separating the different concerns present in a location-based, context-aware system. In doing so, SCOUT decouples the technologies and application logic used in these layers, allowing them to contain interchangeable components. Below, we discuss the SCOUT layers bottom-up, and illustrate their features using the COIN [2] application.

2.1. Detection Layer

The Detection Layer is responsible for detecting physical entities in the user's surroundings, and extracting references to their online semantic descriptions (e.g., an RDF description). For this purpose, we utilize existing detection techniques; e.g., a camera to extract the URL encoded in the entity's QR code, or an RFID reader to read a URL from an RFID-tagged entity. Existing third-party services can also be used; for instance, the LinkedGeoData Semantic Web service [url-1] finds physical entities in a radius around the user's location (obtained via GPS), together with references to their semantic descriptions. SCOUT allows transparently switching between detection techniques, or using several in parallel. Currently, QR, RFID, Bluetooth and the LinkedGeoData service are supported. For example, in the COIN application, tourist attractions and other points-of interest are detected using QR codes and the LinkedGeoData service. The extracted URLs along with relevant detection data (e.g., approximate entity coordinates, detection distance) are passed to the next layer.

2.2. Location Management Layer

The Location Management Layer interprets the raw information received from the Detection Layer. It determines whether detected entities are nearby the user or other detected entities, and when they are no longer nearby. As the definition of "nearness" can differ per application (i.e., different distances can be considered nearby), application-specific proximity criterions are supported. For example, COIN uses a 100m radius as its proximity criterion (meant for pedestrian use). SCOUT employs so-called "nearness" and "remoteness" strategies that exploit detection information to determine whether an entity is nearby (or no longer nearby) the user or another entity, according to the monitored proximity criterions. In case of COIN, detection techniques with limited range (i.e., QR readers) use a nearness strategy that directly infers nearness to the user when the

entity is detected. For LinkedGeoData, latitude and longitude are compared with the user's (GPS) position. Similarly, nearness between two detected physical entities is determined by comparing the approximate positions at which the entities were detected. The remoteness strategy consists of comparing the user's current position with the detected entity's (exact or inferred) position at set time intervals, to determine when the distance exceeds the used proximity criterion. SCOUT is pre-configured with proximity strategies for each supported detection technique, but applications can also deploy custom strategies. This layer notifies the Environment Layer of "nearness" and "remoteness" events, along with the employed criterion, the entities' (approximate) locations and references to their online data sources.

2.3. Environment Layer

The Environment Layer provides mobile applications with an integrated view on the user, his environment and the physical entities in it, called the Environment Model. This layer maintains two local data models that provide the necessary information for this integrated view. In the ***User Model***, the user's characteristics, preferences and device information are stored using ontologies such as CC/PP [url-4] or FOAF [url-5], along with personal information obtained from other applications the user is willing to share (e.g., personal agenda). As such, this model allows personalizing content and functionality to the user. Positional information on the user's environment is encoded in the ***Proximity Model***. It keeps time-stamped positional relations between the user and the physical entities, together with references to the entities' associated data sources. A positional relation represents the fact that an entity is, or has been, nearby the user or another entity. This model underlines the location-based nature of SCOUT, as it keeps relative positional information in an abstract, high-level format. The Proximity Model Management component keeps this model up-to-date based on nearness and remoteness notifications from the Location Management Layer. Additionally, this component provides applications with their own view on the Proximity Model, corresponding to their specified proximity criterion. The ***Environment Model*** encompasses the User and Proximity Model, and extends them with information obtained from the physical entities' online semantic sources.

Applications query the Environment Model using the *Query Service*. For instance, the COIN application may issue the following SPARQL query to find restaurants in the user's vicinity, which are close to a metro station leading back to the user's hotel and serve the user's favorite cuisine (namespaces omitted for brevity):

```
SELECT ?restaurant ?rLat ?rLong ?station1 ?cuisine
WHERE
{
 ?user rdf:type um:User ;
     um:stayingAt ?hotel .
 ?hotel rdf:type space:Hotel .

 ?user prox:currentlyNearby ?restaurant .
 ?restaurant rdf:type resto:Restaurant ;
    geo:lat ?rLat ;
    geo:long ?rLong .

 ?restaurant prox:isNearby ?station1 .
 ?station1 rdf:type space:Tube_Station .
 ?route space:connects ?station1 .
 ?route space:connects ?station2 .
 ?station2 prox:isNearby ?hotel .

  ?user um:prefersCuisine ?cuisine .
  ?restaurant resto:typeOfCuisine ?cuisine .
}
```

Query 1. Query retrieving nearby interesting restaurants.

This query first employs the User Model to retrieve the user's hotel (`um:stayingAt`), and obtains entities currently nearby by utilizing the Proximity Model (`prox:currentlyNearby`). Subsequently, it checks whether these are restaurants (`resto:Restaurant`), and searches for entities nearby these restaurants (`prox:isNearby`) that are metro stations (`space:Tube_Station`) with a stop (`?route space:connects`) nearby the user's hotel,

utilizing the Proximity Model and the entities' online sources. Finally, it checks whether these restaurants serve a cuisine (`resto:typeOfCuisine`) that is one of the user's favorites (`um:prefersCuisine`). This example illustrates one of the strengths of using Semantic Web technology; it allows integrating information from different heterogeneous data sources, by relying on the re-use of well-known ontologies and unique resource identity via URIs. This integration enables SCOUT to resolve complex context-sensitive queries that cannot be solved by any single source. COIN subsequently communicates the results to a browser plugin that injects the data on-the-fly in existing webpages (e.g., restaurant listings). The latter is outside the scope of this article; we refer to [2]. Both SCOUT and COIN are based on Android OS 2.2.

3. Data Management

In SCOUT, a major challenge is realizing efficient access to the online data associated with physical entities. Firstly, there are various types of online semantic sources available: online RDF files, semantically annotated webpages, and datasets accessible via query endpoints. The latter are accessible by issuing queries, while obtaining information from RDF files and websites requires downloading them. Most related approaches focus on optimizing access to distributed query endpoints (see related work); we focus on the huge dataset contained in the other two types of sources, which has specific challenges for efficient access.

A first problem in a mobile setting, where the user detects numerous physical entities over time, is that the downloaded data may exceed the allocated storage space. Additionally, our experiments show that querying large RDF datasets without optimization yields unacceptable response times for real-time applications (see section 4). To deal with these issues, we use a semantics-based indexing mechanism to identify sources relevant to a given query, combined with a caching mechanism to store downloaded sources (see fig. 1).

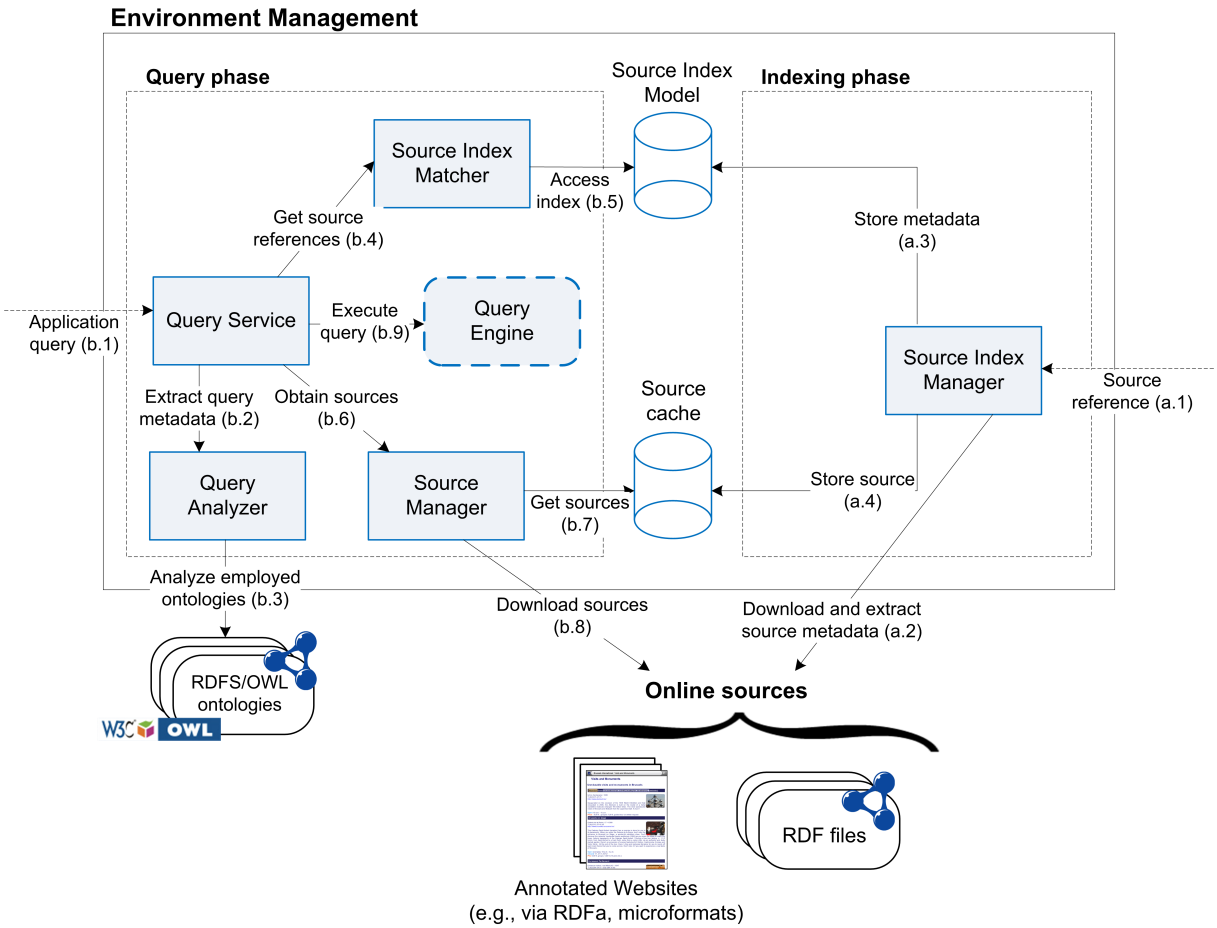

Fig. 1. Detailed Environment Layer.

3.1 Indexing online semantic sources

To identify query-relevant sources, we keep an index called the *Source Index Model* (SIM), which contains metadata about encountered sources. The SIM is maintained in the background during the *indexing phase*, which is triggered whenever a physical entity becomes nearby and its online reference is passed to the Environment Layer. Firstly, the obtained reference is communicated to the *Source Index Manager* (fig.1-a.1), which downloads the source, extracts the metadata (a.2), and stores it in the *Source Index Model* (a.3). For semantically annotated websites, an existing tool (currently, a ported version of java-rdfa [url-6]) is used to extract RDF triples from semantic annotations. Afterwards, the RDF source is passed to the *Source Cache*, where it is stored for later use (a.4). The Source Cache employs a Least-Recently-Used replacement function to locally store frequently required sources.

Since sources must be indexed on-the-fly on devices with limited capabilities, the indexing phase should consume minimal resources, while still enabling the fine-grained identification of relevant sources (selectivity). To achieve this, we extract and keep semantic metadata that is quickly obtainable and cheap to store, while still guaranteeing high selectivity. We first observe that in order to navigate RDF graphs, most SPARQL queries specify concrete predicates (i.e., RDF graph edges), with subjects or objects given as variables. Secondly, as mobile applications mostly consider *certain* entities in the user's surroundings (e.g., shops, restaurants), their queries often restrict the types of these variables. At the data side, RDF sources usually utilize ontologies with rich type hierarchies (e.g., FOAF, DCMI) to specify fine-grained types for contained resources. In this respect, we assume that sources are self-describing, i.e., specify the types of their resources. By keeping source predicates and their subject and object types in the SIM, and comparing this information to query predicates and type restrictions, our validation shows we can rule out a significant amount of sources irrelevant to posed queries, while still minimizing the indexing overhead. In our implementation, the SIM is realized as in-memory multi-level index using hashtables (comparable to [3]), where each level corresponds to an indexed metadata part. Given a predicate key in the first index, a second index with subject type keys is returned. The final index maps object type keys to lists of source URIs containing the given combination of predicate, subject type and object type. We employ dictionary encoding to reduce SIM size, by mapping source URIs and term namespaces to unique identifiers.

3.2 Query resolving using the Source Index Model

In the second part of our approach, the *query phase*, application queries are analyzed to determine all query-relevant sources, and the queries are resolved. The central component in this process is the *Query Service*, which receives incoming application queries (fig.1-b.1). The query is first passed to the *Query Analyzer* (b.2), which extracts predicates and type restrictions (i.e., query metadata). This component also exploits ontological knowledge on RDF terms, to increase the amount of extracted query metadata (b.3) (see below). The Query Service then communicates the query metadata to the *Source Index Matcher* (b.4), which accesses the SIM (b.5) to identify relevant sources. Subsequently, the *Source Manager* is contacted for these sources (b.6), which are fetched from the cache (b.7) or downloaded if no longer available locally (b.8). Finally, the Query Service employs an existing *Query Engine* (in our case, androjena [url-7]) to execute the query over the combined set of sources (b.9). When integrating the data, different URIs or vocabulary terms for the same resources or concepts may pose problems. As no fully automatic approaches to tackle these issues exist, we currently rely on 1) interlinks between resources and vocabulary terms (e.g., via owl:sameAs) that allow us to deduce equivalence automatically, and 2) explicitly specified alternatives in posed queries. Below, we discuss the most important components: the Query Analyzer and the Source Index Matcher.

The Query Analyzer extracts the query metadata, consisting of predicates and type restrictions. For regular triple patterns in the WHERE, OPTIONAL and UNION clauses, specified predicates and type restrictions of their subject and object (if any) are extracted. FILTER clauses are scanned for functions that specify a predicate, subject or object type (using sameTerm()). Furthermore, we exploit the domain knowledge captured in OWL ontologies to infer additional query metadata. In particular, we infer the type of a variable used as subject or object of a certain predicate, based on its domain or range specification in its associated ontology.

The Source Index Matcher uses the extracted query metadata to contact the SIM. For each of the query triple patterns, sources are identified that contain triples with the specified predicate and subject / object types. By performing this matching on the level of triple patterns, sources relevant to only one or several triple patterns are still included, allowing us to solve queries that require data from multiple sources.

The indexing and querying phase are implemented as separate threads, allowing queries to be processed while new sources are still being indexed in the background (the querying thread is prioritized).

4. Experimental validation

To validate our indexing strategy, we ran a series of experiments on a Samsung Galaxy S with a 1 GHz processor and 512MB RAM, using three different SIM variants: SIM1, which keeps found predicates, SIM2, which keeps predicates and subject types, and SIM3, keeping predicates, subject and object types. The online set of data sources was distributed across three webservers, and had a total size of 117MB (with an average file size of 14.4KB). These sources were extracted from the LinkedGeoData, DBPedia [url-8], Geonames [url-9] and Semantic Web Dog Food [url-10] datasets, and were complemented with product data obtained from a Berlin SPARQL Benchmark dataset [url-11]. The complete dataset, as well as the queries used for validation, can be found at [url-12].

In the evaluation of the indexing phase, each source in the dataset was encountered and indexed sequentially. The SIM keeps all encountered sources; an option to remove older index entries was not used in the experiments. The indexing phase has an average execution time per source of 74ms for SIM1, 719ms for SIM2 and 1398ms for SIM3, corresponding to insertion rates of 13.5, 1.4 and 0.72 sources per second, respectively (excluding download times). These execution times are reasonable, compared to the average source download time (2066ms). Compared to the total size of the indexed sources, SIM average sizes amount to 0.08% for SIM1, 2.50% for SIM2, and 3.03% for SIM3. Fig. 2 shows the detailed performance of the indexing phase.

To evaluate the query phase, we used four queries of varying complexity, typically posed by mobile applications on top of SCOUT. In the first query, subject and object types are specified, while the others only specify subject types. To avoid cache interference, we used a cache-all approach where all sources are kept locally after downloading and indexing. In fig. 2, the detailed query resolution performance is shown for each SIM variant and for an increasing amount of encountered sources (including the case where no SIM is used, i.e., native query engine performance over all assembled sources). Note that no performance measurements are available for “no SIM” and SIM1 for more than 400 and 550 sources (marked as a cross in fig. 2), as the set of collected sources became too large, leading to out-of-memory exceptions. In our experiments, the indexing and query phases were executed separately, and no other tasks were simultaneously executed to avoid any interference.

As observed in fig. 2, the data collection step, comprising the fetching of relevant source data from persistent cache and loading it in a repository, forms a performance bottleneck. This mainly results from the required read-time for large amounts of source data. This leads to a significant performance gain for the SIMs, as they are able to rule out large amounts of sources. This gain increases as more semantic information is exploited in the subsequent SIM versions, while the overhead of indexing remains minimal. In our experiments, the average amount of sources determined to be query-relevant is 26.1% for SIM1, 3.5% for SIM2 and 1.5% for SIM3. We also observe that the performance gain for SIM3 over SIM2 is relatively small. This is because typically, the type of subject variables is more often restricted than object variables (this is also the case in our validation queries), leading to object type data in SIM3 being underused.

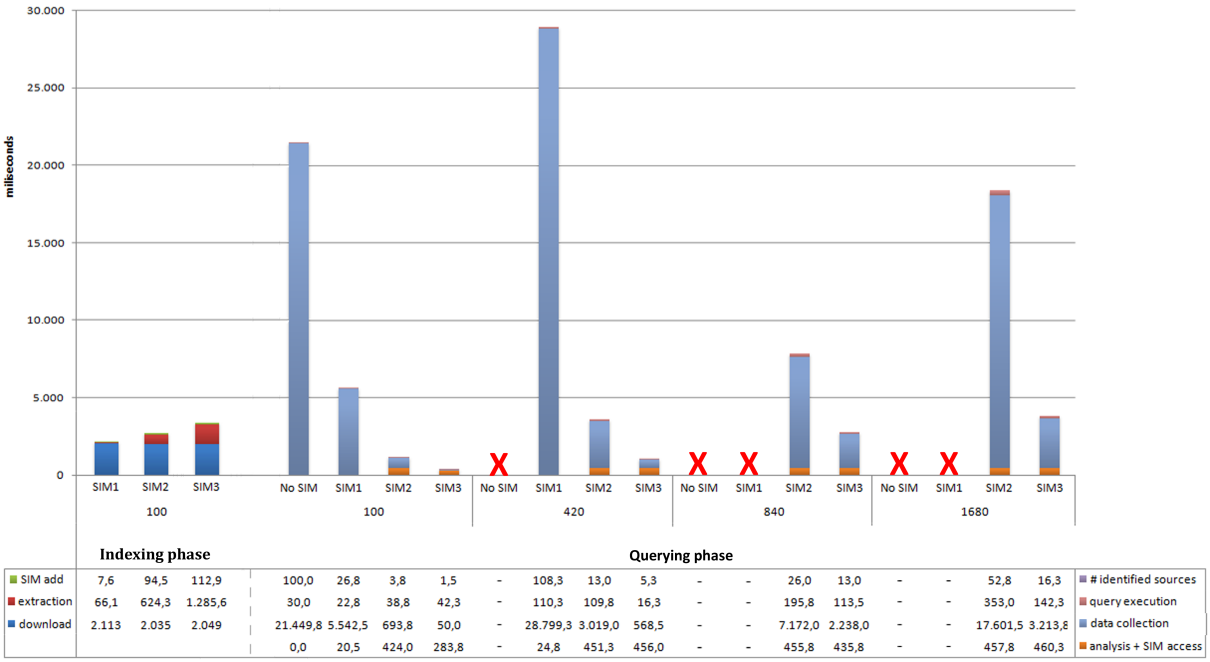

Fig. 2. Validation results.

5. Related work

Many context-aware middleware approaches rely on context providers to obtain information on the user's environment, which extract data from internal (e.g., sensors) or external (e.g., traffic service) sources (e.g., [4-5]). Such solutions are useful for achieving efficient access to dynamic data, and often require query stream processing techniques [6] to handle high-volume datastreams and support queries with lively updated results. In SCOUT however, data acquisition and query time are typically distinct, fewer data is handled over a larger amount of time, and queries are performed on a snapshot of the currently available data. Similar to SCOUT, some of these approaches focus on integrating context sources to provide a unified view on the user's environment [4]. However, in contrast to SCOUT, this integration is mostly achieved via a single centralized service, which has a higher participation threshold for source providers and is less scalable and flexible.

Data indices, such as the proposed SIMs, are also used in other fields to optimize data access. In RDF stores, indices allow the quick retrieval of relevant RDF triples, given the concrete values (e.g., resource URIs) provided in queries. For instance, androjena keeps separate indices for subjects, predicates and objects, while systems such as HexaStore [7] index each possible query access pattern to further optimize query execution. In HexaStore, this leads to six indices with high update and insertion costs, and a (worst-case) five-fold increase in storage space. In contrast, our largest SIM requires only around 3% of the RDF data size. In the MQuery system [8], issues related to efficient RDF access in mobile environments are tackled. This system supports four types of queries, allowing users to find related content and navigate between related nodes (e.g., from a photo to its photographer). Two indices are used to optimize query performance, namely a text index (inverted index) and a graph index (comprising node adjacency lists), while an index compression method is proposed to reduce the index sizes. However, even after compression, each individual index takes up more space than the dataset size. Moreover, our approach supports any type of information request encoded as a SPARQL query.

In the field of query distribution, a query is divided into subqueries on particular data sources. To achieve this, indices are kept describing the contents and capabilities of the different query endpoints. For instance, in [9], properties found in a dataset are extracted and stored, which is equivalent to SIM1. In semwiq [10], lists of classes and properties are kept; this leads to information loss compared to SIM2 and SIM3, as it is no longer known which classes occur in the domain or range of the found properties. DARQ [11] relies on manually specified service descriptions, listing found properties together with value constraints on their subjects and objects. In contrast, we focus on metadata that is more efficiently extractable (i.e., types).

6. Conclusion

This paper presents a framework that supports the development of location-based, context-aware mobile applications. The framework provides various techniques for detecting nearby physical entities and extracting their associated online semantic information. Based on this online data, an integrated and query-able view on the user's surroundings is provided. To efficiently access and manage this view, we developed a tailored data management approach that filters online data sources relevant for a query, based on semantic information from the sources and ontological domain knowledge. According to our experimental validation, query performance is significantly improved, making management and local querying of online RDF sources feasible in a real-world, mobile environment.

Future work consists of investigating how semantic data can be further utilized, for example by using OWL's reasoning capabilities to infer additional knowledge. We are also studying automatic semantic query enrichment (e.g., using Wordnet [url-13] synonyms) to mitigate losing relevant results due to different vocabularies. For the same purpose, we plan to investigate existing solutions like Silk [url-14] to setup and use online services to automatically link together disjoint datasets. Furthermore, since SCOUT applications typically pose location-related queries, focusing on these specific semantic relations should imply performance gains. Other interesting research avenues include studying how to ensure index freshness without negatively impacting performance, and swapping parts of the index to persistent storage to increase the potential index size.

Acknowledgement

Sven Casteleyn is supported by an EC Marie Curie grant, FP7- PEOPLE-2009-IEF, N° 254383.

References

[1] Van Woensel, W., Casteleyn, S., De Troyer, O: A Framework for Decentralized, Context-Aware Mobile Applications Using Semantic Web Technology, OTM Workshops, 2009, 88-97.
[2] Casteleyn, S., Van Woensel, W., De Troyer, O: Assisting Mobile Web Users: Client-Side Injection of Context-Sensitive Cues into Websites, International Conference on Information Integration and Web-based Applications & Services, 2010, 441-448.
[3] Weiss, C., Bernstein, A., Boccuzzo, S., i-MoCo: Mobile Conference Guide - Storing and querying huge amounts of Semantic Web data on the iPhone/iPod Touch, 7th International Semantic Web Conference, 2008.
[4] Judd, G., Steenkiste, P: Providing contextual information to pervasive computing applications, First IEEE International Conference on Pervasive Computing and Communication, 2003, 133-142.
[5] Euzenat, J., Pierson, J., Ramparany, F. Dynamic context management for pervasive applications. *Knowledge Engineering* 23, 1, 2008, 21-49.
[6] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.A: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World, Fifth Biennial Conference on Innovative Data Systems Research, 2003.
[7] Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endow.* 1, 1, 2008, 1008-1019.
[8] Zhang, Y., Zhang, N., Tang, J., Rao, J., Tang, W.: MQuery: Fast Graph Query via Semantic Indexing for Mobile Context, International Conference on Web Intelligence and Intelligent Agent Technology, Vol. 1, 2010.
[9] Lynden, S.J, Kojima, I., Matono, A., Tanimura, Y: Adaptive Integration of Distributed Semantic Web Data, Databases in Networked Information Systems, 2010, 174-193.
[10] Langegger, A., Wöß, W., Blöchl, M.: A Semantic Web Middleware for Virtual Data Integration on the Web, 5th European Semantic Web Conference, 2009, 493-507.
[11] Quilitz, B., Leser, U: Querying Distributed RDF Datasources with SPARQL, 5th European Semantic Web Conference, 2008, 524-538.

URL's

[url-1] http://linkedgeodata.org/
[url-2] http://ckan.net/group/lodcloud
[url-3] http://developer.yahoo.com/search/boss/structureddata.html
[url-4] http://www.w3.org/Mobile/CCPP/
[url-5] http://xmlns.com/foaf/spec/
[url-6] https://github.com/shellac/java-rdfa
[url-7] http://code.google.com/p/androjena/
[url-8] http://dbpedia.org
[url-9] http://www.geonames.org
[url-10] http://data.semanticweb.org
[url-11] http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/
[url-12] http://wise.vub.ac.be/SCOUT/IEEE-Internet-Computing-Nov2011/
[url-13] http://www.w3.org/2006/03/wn/wn20/
[url-14] http://www4.wiwiss.fu-berlin.de/bizer/silk/

Biographies

William Van Woensel is a Ph.D. student at the Web & Information Systems Engineering (WISE) lab of the Vrije Universiteit Brussel, where he obtained his Master degree in 2007. His current research interests concern distributed querying, Semantic and Mobile Web. http://wise.vub.ac.be/members/william.

Sven Casteleyn holds a Marie Curie IEF fellowship at the ProS Research Center, Polytechnic University of Valencia. He previously was a researcher at the Vrije Universiteit Brussel, from which he holds a Ph.D. (2005) and Master degree (1999). He published extensively on various aspects of Web and Mobile applications, and co-authored the book Engineering Web Applications (Springer, 2009). http://wise.vub.ac.be/members/sven.

Elien Paret is a Ph.D. student at the WISE lab of the Vrije Universiteit Brussel where she obtained her Master degree in 2010. Her current research work is on distributed querying and adaptive systems. http://wise.vub.ac.be/members/elien.

Olga De Troyer is full professor in Computer Science at the Vrije Universiteit Brussel since 1998. She is co-director of the WISE research lab. Previously, she worked in industry, the University of Hasselt (Belgium), and Tilburg University (The Netherlands) where she obtained her Ph.D. degree (1993). Her research background includes databases, Web systems, Semantic Web, and Virtual Reality. http://wise.vub.ac.be/members/olga.

Contact Information

William Van Woensel
mailing address: WISE lab, Department of Computer Science, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Elsene, Belgium
phone: +32 479 78 20 97
email: William.Van.Woensel@vub.ac.be

Sven Casteleyn
mailing address: ProS Center - Building 1F, Universidad Politécnica de Valencia, Camino de Vera S/N, Valencia 46022, Spain
phone: +34 963 87 35 76
email: Sven.Casteleyn@upv.es

Elien Paret

mailing address: WISE lab, Department of Computer Science, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Elsene, Belgium
phone: +32 2 629 11 03
email: Elien.Paret@vub.ac.be

Olga De Troyer
mailing address: WISE lab, Department of Computer Science, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Elsene, Belgium
phone: +32 2 629 35 04
email: Olga.DeTroyer@vub.ac.be