Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2011) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Conference on Mobile Web Information Systems (MobiWIS)

# Efficient Querying of Distributed RDF Sources in Mobile Settings based on a Source Index Model

Elien Paret[a,*], William Van Woensel[a], Sven Casteleyn[b], Beat Signer[a], Olga De Troyer[a]

[a]Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
[b]Universidad Politécnica de Valencia, Camino de Vera S/N, 46007, Valencia, Spain

**Abstract**

Recent technological developments in the area of mobile devices in combination with the ubiquity of mobile connectivity have turned handheld devices into rich mobile web clients. On the other hand, we witness an increased number of web resources that are no longer only available in HTML but also in RDF format. In order to access relevant information in a mobile setting, we often want to query a subset of this large amount of heterogeneous RDF resources based on various context factors. We present a semantic technology-based client-side solution for efficient querying of large sets of distributed RDF sources without query endpoints. Our solution continuously extracts metadata from distributed RDF sources and manages this metadata in a local Source Index Model (SIM). When a new query has to be processed, the SIM is consulted to identify, assemble and query only potentially relevant resources. We do not plan to replace existing mobile query engines, but rather build on them to efficiently and transparently query large sets of distributed online RDF sources. The evaluation of our enhanced query handler reveals significant improvements in the overall query execution time based on the proposed SIM.

Keywords: RDF indexing; querying distributed RDF; mobile querying; source index model; SPARQL;

## 1. Introduction

Over the past few years, we have witnessed a tremendous evolution in multifunctional mobile devices offering advanced hardware features (e.g. GPS or accelerometer sensors) and common software applications (e.g. personal organizers, web browsers or email clients). The increased availability of wireless networks in combination with affordable flat-rate and high-speed transmission rates for mobile phone networks resulted in a permanent mobile accessibility of online resources. Furthermore, mobile application developers now have the possibility to fully exploit the rich features offered by state-of-the-art mobile devices based on various sophisticated software development kits that are available for different mobile phone platforms.

On the other hand, there has been a major effort to make existing and new data sources semantically available and interoperable, for example, via the Linked Data[a] initiative. The increased use of RDF annotations (e.g. RDFa)

* Corresponding author. Tel.: +32-2-629-1103; fax: +32-2-629-3525.
*E-mail address*: Elien.Paret@vub.ac.be.
[a] http://linkeddata.org

on websites significantly contributed to the wide availability of distributed and linked heterogeneous RDF, RDFS and OWL resources. According to the Yahoo! BOSS API[b], there are currently close to 955 million websites making use of RDFa. In various mobile application settings, the management, access and integration of local and remote semantic data is considered of paramount importance. Examples include context-aware semantic service discovery [1], semantic mobile augmented reality [2], mobile semantic personalization [3] and semantic context-aware systems [4]. Existing mobile applications seeking to exploit available online RDF data mostly rely on SPARQL query endpoints. However, in practice only major data providers offer such query endpoints and they do not provide a solution for application developers who would like to exploit the vast amount of small and distributed online RDF(S)/OWL sources. Recent mobile query engines, such as androjena[c] or RDF On the Go [5], minimize the need to outsource queries to external query endpoints by supporting the local querying of RDF(S)/OWL. These engines are in an early development stage, and experiments show that they are not optimized to scale with the size of the dataset. Moreover, they do not support the transparent querying and management of online RDF data sets.

We present a solution to efficiently query large amounts of small and potentially linked online RDF(S)/OWL sources in mobile settings. The main idea is to provide a client-side query service which extracts metadata from RDF(S)/OWL sources and manages it in a local Source Index Model (SIM). The metadata consists of information about the predicates found in each source together with their subject and object types. At query execution time, our query handler uses the SIM to determine and filter potentially relevant sources to be queried via an existing mobile query engine. We do not plan to replace existing mobile query engines, but rather build on them to efficiently and transparently query large sets of distributed online RDF sources by automatically downloading and managing the relevant resources. In contrast to most existing solutions, we do not rely on query endpoints for integrating and querying distributed RDF sources. By ensuring that we query only relevant resources, we were able to significantly improve the overall query execution time. We also address computational and memory limitations of mobile devices by keeping the index quickly updatable and smaller than a full text index.

Note that resources in a mobile application setting are often dynamically discovered and queried based on various context factors (e.g. information about services in a user's vicinity). This implies that mobile applications will continuously deal with references to new resources that have to be added to the index of the underlying query service on-the-fly. Last but not least, to prevent a loss in query performance, we have to ensure that a mobile query index fits into the mobile device's main memory and pays attention to other computational limitations.

We start with a discussion of related work in Sect. 2 and then present the indexing of distributed online RDF resources via a SIM in Sect. 3. Section 4 provides a detailed description of the querying process based on the Source Index Model. In Sect. 5, we report on a series of experiments that have been performed to demonstrate the feasibility and performance of our approach. The results of the conducted experiments are discussed in Sect. 6, and concluding remarks are given in Sect. 7.

## 2. Related Work

Our solution uses a metadata index for the efficient querying of distributed RDF data sources by filtering non-relevant sources. Indexes are also used in other contexts for optimizing data access. In the field of query distribution, a metadata index is used to divide a query into subqueries and optimize the query distribution plan. For example, Quilitz et. al [6] use a service description which defines the capabilities and contents of query endpoints. It contains information about predicates, statistical information, such as the amount of triples using a specific predicate, and details obtained from object analysis (e.g. all objects of a predicate starting with the letters A to D). Specific characteristics about the data as well as the data provider, for example join predicate selectiveness and the data production rate by the data provider, are used in [7] to divide a query into subqueries or so-called source queries. In [8], full-text indexes are applied to determine which peers contain a particular subject, predicate and object in Distributed Hash Table-based RDF. Additionally, statistical information, such as the frequency of triple parts in the dataset, is kept in order to optimize the query distribution process. Source-index hierarchies are employed in [9] to identify query endpoints that are relevant for a given query. They enable the identification of query endpoints that

[b] http://developer.yahoo.com/search/boss/structureddata.html
[c] http://code.google.com/p/androjena/

can handle combinations of query triple patterns (or "paths"), and aim to reduce the number of joins on the individual results. We share the common goal of determining query-relevant data sources with all these approaches. However, since our data sources are not accessible via a query endpoint, we cannot distribute the query execution.

In the context of query optimization, [10] uses a full-text triple index with fully indexed subjects, predicates and objects to enable fast retrieval of quads (subject, predicate, object and context) by trading index-space for retrieval time. In contrast, we try to minimize the index size since our solution is intended to be used in mobile settings and the index should ideally be kept in memory. A full-text index called HexaStore [11] is used on an iPad/iPhone in [12] to store RDF information from distributed locations in order to enable fast access to the RDF data, which needs to be loaded into the RDF store in advance. However, full-text indexes are storage intensive and computationally expensive with high update and insertion costs. They are thus not applicable in our setting, where new RDF data needs to be added frequently, and the index has to be small and updatable on-the-fly.

Other approaches depend on a central query service acting as a mediator for a set of data sources [13-15]. Such a service receives queries, distributes them across the data sources registered with the service, and returns an integrated result. This solution reduces the load on the mobile device, but it is less scalable, and requires each data source to be registered with the central service. In recent years, there have been efforts for storing and querying semantic web data on mobile devices. In RDF On the Go [5] and androjena, the Jena framework and ARQ query engine have been adapted for mobile devices and the Android platform in particular. Our approach uses androjena to locally access, manipulate and query RDF data. To the best of our knowledge, no other approaches allow to transparently manage and query large amounts of small and distributed online RDF data sources on a mobile device.

## 3. Source Index Model

As stated earlier, our efficient query service for distributed RDF sources is based on a Source Index Model for filtering and assembling relevant sources at query execution time. In this section, we describe the indexing of online RDF sources and the construction of the SIM that is going to be used by the query handler to optimize the query execution as described in Sect. 4.

The first step of our approach consists of extracting metadata from existing RDF data sources and storing it in the SIM as outlined in Fig. 1a. The application communicates the references of online RDF sources to be considered to the Source Manager (1). The Source Manager then extracts the necessary metadata (2) from these sources. The extracted metadata consists of the predicates together with their subject and object types. Once the metadata has been retrieved, it is stored in the SIM (3). Furthermore, the downloaded source is forwarded to the cache (4), where it is either stored or ignored based on the caching strategy. Note that in a mobile environment the Source Manager continuously receives new source references from the application when new data sources are discovered. This implies that the continuous indexing process should be efficient and impose minimal processing overhead.

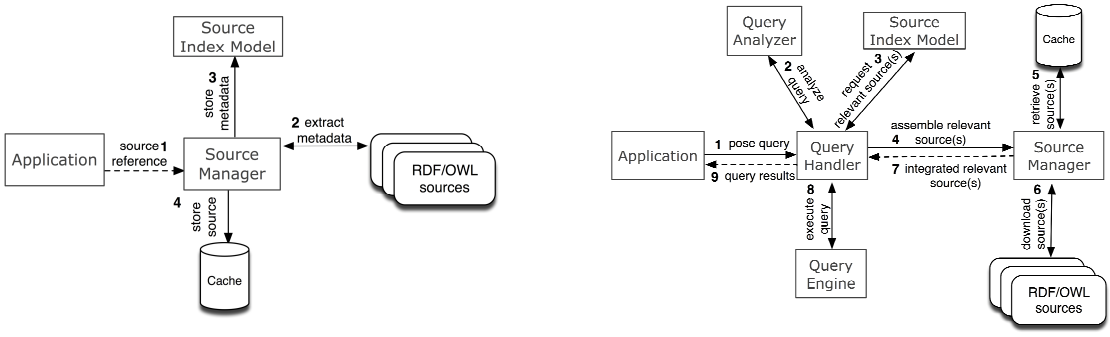


**Fig. 1.** (a) Indexing online RDF sources (b) Resolving a query

The idea of minimizing the amount of sources to be queried by using a source index is also found in other approaches, which focus on optimizing the distribution of queries over query endpoints as mentioned in Sect. 2. However, in our specific setting, we aim to minimize the required storage space and computational effort to maintain the SIM, while optimizing the filtering of potentially relevant sources. Since RDF is a predicate-based

formalism, we base our approach on the presence of predicates in particular data sources similar to [6, 9]. Furthermore, we exploit the semantic information embedded in RDF files, namely the type of the predicate subjects and objects. Both RDF sources and SPARQL queries often use subject and object information to describe triples in more detail and to restrict triple patterns. For instance, in most of our sample queries that were used in the validation scenario, the subject types of the triple patterns need to be restricted, since mobile applications are only looking for information on certain physical entities (e.g. restaurants) in a user's vicinity. At the same time, RDF sources typically specify resource type information for the contained resources. Based on these observations, we consider three variants of the SIM: the first variant (SIM1) stores the found predicates only. The second variant (SIM2) stores the predicates together with their subject types and the third variant (SIM3) manages the found predicates together with their subject and object types. Note that although the two latter variants store additional metadata, they still support queries on sources with unspecified subject and/or object types. Since each variant stores additional metadata, the required storage space and complexity to create and query the SIM also increases. In return, we expect improved filtering of relevant sources which should result in a better overall query performance for richer SIM variants.

After introducing the metadata that is stored in the SIM, we now explain how to extract this information from online RDF sources. The Source Manager performs this extraction each time it receives a new source reference. As we focus on RDF documents without query endpoints, the source is downloaded and queried locally to extract the necessary metadata. Depending on the used SIM variant, we need to extract predicates (SIM1), predicates and their subject types (SIM2), or predicates and their subject and object types (SIM3). Since the three extraction queries are rather similar, we only show the most complex one in Listing 1.

```
1 SELECT DISTINCT ?stype ?p ?otype
2 WHERE {
3   ?s ?p ?o .
4   OPTIONAL {?s rdf:type ?stype} .
5   OPTIONAL {?o rdf:type ?otype} . }
```

Listing 1. Extracting predicates with their subject and object types

The query considers all triples (line 3) with optionally specified subject types (line 4) or object types (line 5). By making the latter conditions optional, we ensure that predicates that do not have any specified types are still stored in the SIM. Listing 1 finally returns all occurring predicates along with their subject and object types (line 1). Note that since the size of the extracted metadata is much smaller than the source itself, our query service is able to index many more sources than it can locally store.

## 4. Querying Distributed RDF Sources

Once the Source Manager has indexed a number of online RDF sources, the application can execute queries over the combined data set of indexed sources as illustrated in Fig. 1b. The central component is the Query Handler, which receives incoming queries from the application (1), and handles the query by delegating specific tasks to specialized components. First, the query is passed to the Query Analyzer (2) which analyzes the query and extracts relevant metadata. The extracted query metadata corresponds to the metadata that is managed by the SIM, namely query predicates (SIM1) together with their subject (SIM2) and/or object types (SIM3). Based on this metadata, the Query Handler requests the references of potentially relevant sources from the Source Index Model (3). Note that this identification is done for each triple pattern; sources that contain predicates (and subject/object types) referenced in only one or several query triple patterns are also included in the final dataset. This ensures that queries that are not solvable by any single data source can potentially still be solved by a combination of data sources. Subsequently, the Query Handler requests the relevant sources from the Source Manager (4) which assembles them by either retrieving them from the cache (5) or downloading them if they are no longer in the cache (6). The filtered sources are integrated and returned to the Query Handler (7). Finally, the Query Handler uses the Query Engine to execute the query over the integrated dataset of relevant sources (8) and returns the result to the application (9).

In the following, we describe the query analysis for the SIM3 variant in more detail. In a SPARQL query, triple patterns occurring in the `WHERE` clause are used in the graph pattern matching process to restrict the query results.

Therefore, we need to examine this clause in order to identify potentially relevant sources. The `WHERE` clause can contain regular triple patterns, `FILTER` clauses, `OPTIONAL` clauses and `UNION` clauses. In the case of a regular triple pattern, the used predicates together with their subject and object types are extracted. `FILTER` clauses are scanned for functions that specify a predicate, subject type or object type (e.g. `sameTerm()` function) and these predicates and types are extracted as well. `OPTIONAL` clauses specify triple patterns for which query results optionally contain result values. We also need to consider the triple patterns from `OPTIONAL` clauses, otherwise we might not include query results for these triple patterns, while they would have been included if the query had been executed over the entire dataset. Extracting metadata from a `UNION` clause consists of recursively extracting the metadata of its subparts.

## 5. Experiments

The experiments to validate our approach have been designed to compare the three different SIM variants in terms of the computational overhead and storage space as well as the overall query execution time. We also compare our results with the case where no SIM is used, which corresponds to the "native" query performance of the underlying mobile query engine over the complete dataset.

Our solution has been evaluated in the application field of context- and environment-aware mobile applications using the SCOUT framework [16]. SCOUT provides a conceptual and integrated view of a user's physical environment (Environment Model) by combining information from a variety of online RDF(S)/OWL sources describing environmental entities (i.e. people, places and things) that are gradually discovered. In order to provide efficient query access to the Environment Model, the SCOUT framework makes use of our SIM-based query service. Furthermore, it acts as an application that continuously provides new source references to the Source Manager. SCOUT and our SIM-based query service are both based on Android OS 2.2. The androjena library is used to access, manipulate and query RDF data on the mobile device and persistent hash tables have been used to implement the SIMs. Note that since we only wanted to evaluate the effect of different SIM variants, we used a cache-all strategy to avoid any cache interference.

The experiments were performed on a Sony Ericsson Xperia X10 with 567 MB memory and a 1 GHz processor. The online data sources that have been discovered and downloaded to create the SIM were distributed across three different web servers with different response times. Their average size was 3.7KB (with a minimum size of 120 bytes and a maximum size of 70KB).

In order to measure the necessary time and memory space to construct and maintain the three SIM variants, we populated each of the three SIM variants with extracted metadata from 50, 100, 250 and 500 sources. By gradually increasing the amount of sources, we were able to investigate the scalability of our solution. The RDF datasets containing information about physical entities in a mobile user's environment were automatically generated by using random resource types and properties from selected ontologies (e.g. geo, travel or SUMO vocabularies). In order to reflect a real-world situation, from these ontologies different properties and types reflecting the same concepts (e.g. absolute coordinates) were randomly used in combination with types and properties from proprietary ontologies.

```
SELECT ?bLabel ?floorNr ?roomNr
WHERE {
 ?building rdfs:label ?bLabel .
 ?building region:containsFloor ?floor .
 ?floor region:floorNr ?floorNr .
 ?floor region:containsRoom ?room .
 ?room region:roomNr ?roomNr .
 ?room region:housesPerson
<http://wise.vub.ac.be/members/william/> .}
```

```
SELECT ?restaurant ?label ?latLong
WHERE{
 ?restaurant rdf:type resto:Restaurant .
 ?restaurant dcmi:title ?label .
 ?restaurant geo:lat_long ?latLong .
 ?restaurant resto:typeOfCuisine <http://gaia.fdi.ucm.es/
ontologies/restaurant.owl#ItalianCuisine> .}
```

**Listing 2.** (a) Validation query Q1. (b) Validation query Q2

We extracted four types of queries from existing mobile SCOUT applications [3, 16] to evaluate the performance of our solution. Let us consider that a user is working with a number of mobile applications for different types of information requests resulting in SPARQL queries executed over the SCOUT Environment Model. A user visiting the first author's university requests information about the location of William Van Woensel's office via the SCOUT EmployeeFinder application, which leads to the SPARQL query Q1 shown in Listing 2a. After the meeting

with William, the user decides to have lunch and searches for nearby restaurants suiting his taste via the PlaceFinder application (query Q2 shown in Listing 2b) which finally plots the resulting information on a map. After lunch, the user decides to do some sightseeing in Brussels and again consults the PlaceFinder application to find different points-of-interest (query Q3 shown in Listing 3a). Finally, while walking through the city, the user consults the ProductFinder application to find some information about computer stores as reflected in query Q4 in Listing 3b.

```
SELECT ?poi ?label ?latitude ?longitude
WHERE {
 ?poi rdf:type region:PointOfInterest .
 ?poi rdfs:label ?label .
 ?poi perv-space:latitude ?latitude .
 ?poi perv-space:longitude ?longitude .}
```

```
SELECT ?shopLabel ?productLabel ?productPhoto
WHERE {
 ?shop rdf:type sumo:RetailStore .
 ?shop rdfs:label ?shopLabel .
 ?shop region:sells ?product .
 ?product rdf:type
<http://www.ontologyportal.org/SUMO.owl#ComputerResource> .
 ?product dcmi:title ?productLabel .
 OPTIONAL {
  ?product dcmi:description ?productPhoto .
  ?productPhoto rdf:type dcmi:StillImage . }}
```

**Listing 3.** (a) Validation query Q3 (b) Validation query Q4.

We first consider the overhead in size and time for creating and maintaining the SIM. Table 1a shows the total size in bytes of the different SIM variants for each dataset (50, 100, 250, 500 sources). We also calculated how the size of the different SIMs compares to the total size of the indexed sources. For the first SIM variant SIM1, the size amounts to 3,9% of the total size of the indexed sources. This grows to 6,9% for SIM2 and amounts to 13,5% for SIM3. The computational overhead to create the SIM includes the average time to download the source and the average time to extract and add the metadata to the SIM as highlighted in Table 1b.

Table 1. (a) Size overhead (in bytes) for storing the SIM

| # sources | SIM1 | SIM2 | SIM3 |
|---|---|---|---|
| 50 | 8 984 | 18 955 | 34 026 |
| 100 | 16 959 | 31 727 | 61 000 |
| 250 | 43 779 | 71 801 | 155 511 |
| 500 | 86 046 | 126 736 | 246 753 |

(b) Computational overhead (in ms) for creating the SIM

| | SIM1 | SIM2 | SIM3 |
|---|---|---|---|
| extract + add | 17,37 | 38,22 | 61,41 |
| download | 785,00 | 818,00 | 766,00 |
| total | 802,37 | 856,22 | 827,41 |

Table 2a shows the overall query resolution time, including the query analysis time, the SIM access time, the data collection time and the query execution time. The results are clustered according to the dataset size and each cluster shows the overall query resolution time in milliseconds for the different queries and SIM variants. Note that all execution times were obtained by performing 20 runs of each experiment and averaging the results. Last but not least, Table 2b shows the number of sources that were identified as relevant for each query and SIM variant.

## 6. Discussion

A first observation that we make based on the results of Sect. 5 is that the overhead for creating any SIM variant is relatively low and a one-time cost. The overhead increases with the complexity of the SIM, but even for the most complex SIM3 variant, the overhead per source is marginal (61,41ms on average) compared to the download times (average 790ms) or query execution times. The sources have to be downloaded before metadata can be extracted, but they have to be downloaded in any case (also if no SIM is used) in order to execute queries locally. The computationally non-intensive SIM maintenance task runs in an independent background thread and is non-disruptive to other tasks running on the mobile device.

A second observation concerns the size of the SIM (see Table 1a). Even for the smallest SIM, there is some overhead, which increases with the complexity of the SIM. Compared with the original sources, the average overhead is 3,9% for SIM1, 6,9% for SIM2 and 13,5% for SIM3. For comparison reasons, we constructed a full text hash table-based index for found RDF nodes per subject, predicate and object, which results in 70% space overhead.

Table 2. (a) Query resolution time (in ms) per SIM variant

| # sources | | SIM1 | SIM2 | SIM3 | no SIM |
|---|---|---|---|---|---|
| 50 | Q1 | 3 122 | 3 634 | 3 662 | 3 372 |
| | Q2 | 2 487 | 509 | 462 | 3 302 |
| | Q3 | 3 022 | 113 | 112 | 3 301 |
| | Q4 | 3 926 | 288 | 242 | 3 306 |
| 100 | Q1 | 6 320 | 8 664 | 8 754 | 6 717 |
| | Q2 | 7 620 | 769 | 718 | 6 686 |
| | Q3 | 7 189 | 344 | 407 | 6 697 |
| | Q4 | 10 054 | 573 | 461 | 6 692 |
| 250 | Q1 | 15 303 | 13 874 | 14 013 | 17 672 |
| | Q2 | 11 333 | 2 190 | 2 161 | 17 682 |
| | Q3 | 13 766 | 3 211 | 3 236 | 17 672 |
| | Q4 | 15 957 | 2 858 | 2 637 | 17 593 |
| 500 | Q1 | 19 976 | 20 377 | 20 465 | 25 575 |
| | Q2 | 15 183 | 3 347 | 3 376 | 26 190 |
| | Q3 | 20 985 | 5 418 | 5 325 | 26 610 |
| | Q4 | 24 648 | 4 915 | 4 503 | 26 022 |

(b) Number of relevant identified sources per SIM variant

| # sources | | SIM1 | SIM2 | SIM3 |
|---|---|---|---|---|
| 50 | Q1 | 28 | 28 | 28 |
| | Q2 | 24 | 8 | 8 |
| | Q3 | 31 | 8 | 8 |
| | Q4 | 41 | 8 | 8 |
| 100 | Q1 | 48 | 47 | 48 |
| | Q2 | 57 | 17 | 17 |
| | Q3 | 58 | 17 | 17 |
| | Q4 | 80 | 16 | 15 |
| 250 | Q1 | 122 | 122 | 122 |
| | Q2 | 144 | 40 | 40 |
| | Q3 | 143 | 45 | 45 |
| | Q4 | 206 | 52 | 46 |
| 500 | Q1 | 234 | 234 | 234 |
| | Q2 | 229 | 59 | 59 |
| | Q3 | 269 | 101 | 101 |
| | Q4 | 362 | 105 | 96 |

It is evident from Table 2a that the biggest performance gain is for SIM2 and SIM3 for queries Q2, Q3 and Q4, independent from the amount of sources. This was to be expected, since these are the queries and SIM variants that contain most semantic information. For all three SIM variants however, query Q1 reports a smaller performance gain, and in some cases even a performance loss. On the other hand, the performance gain for SIM1 for queries Q2, Q3 and Q4 is much lower than for SIM2 and SIM3 and only gets slightly better than in the case where no SIM is used for larger datasets (>250).

The reason for the poor performance gain of SIM1 lies in the small amount of source metadata, which leads to only a limited source selectivity. The limited source selectivity and filtering of relatively small sources (average size of 3,7 KB) does not compensate the overhead for accessing the SIM. Since query Q1 does not specify the metadata contained in SIM2 and SIM3 (i.e. subject or subject/object types), SIM2 and SIM3 have a similar performance as SIM1 for query Q1. Additionally, for few sources (<=100), the small number of eliminated sources cannot compensate the overhead of accessing SIM2 and SIM3, resulting in a performance loss. It does not seem to be beneficial to store additional metadata for queries that provide little selectivity.

One of the most obvious observations is the significant performance gain for queries Q2, Q3 and Q4 when using SIM2 or SIM3. As mentioned before, this can be explained by the fact that these three queries specify rich semantic information. Queries Q2, Q3 and Q4 contain subject type restrictions for each triple pattern, whereas Q4 also includes information on the object types. As a result, SIM2 and SIM3 perform similar for queries Q2 and Q3, since the extra object type metadata present in SIM3 is not usable and thus does not provide any extra source selectivity. However, in query Q4, the available object type information causes some extra selectivity resulting in a slight performance gain.

One would expect a much better performance of SIM3 for query Q4 based on the given object type restriction. However, the restriction is only given for one triple pattern (and one optional pattern). As relevant sources are identified per triple pattern, this implies that the object type restriction is only considered when handling the triple pattern in question. This leads to a small difference in selectivity between SIM2 and SIM3 for this query (see Table 2b). However, the impact on the total execution time is minimal with higher costs for maintaining the richer SIM. It can further be noted that the total execution time based on SIM1 is always worse for query Q4 than for queries Q2 and Q3. This is because query Q4 contains almost double the amount of triple patterns than queries Q2 and Q3 and more potentially relevant sources will be returned by the SIM1.

We can thus conclude that overall, SIM2 seems to perform best: in most cases it provides a significant query performance improvement over SIM1 at little extra cost. Furthermore, often it performs as good as SIM3, but at lower costs. However, the gain in execution time also depends on the number of sources and the query selectiveness. The particular use of our query service, the source content and the nature of the intended queries might influence the selection of a specific SIM. In our experiments, we assumed that all sources are stored on the client device (cache-

all strategy). However, in case there is not enough space to cache all sources, an approach without a SIM model will no longer work since all resources have to be present to execute a query. On the other hand, our SIM-based solution will still work, since we can selectively download the required resources based on the SIM metadata. Note that we do currently not check the freshness of data in the Source Index Model but plan to do this in the near future based on HTTP cache-control header fields.

## 7. Conclusion

We have presented a client-side query service for the efficient querying of distributed RDF sources in mobile settings. The resulting enhanced query service runs on top of arbitrary existing mobile query engines. Our solution is based on a continuously updated Source Index Model, which is consulted at query execution time to significantly reduce the number of data sources to be queried. When designing our Source Index Model, we had to find the right balance between the amount of stored index data and the corresponding maintenance overhead on the one hand, and the number of potentially filtered irrelevant data sources on the other hand. Our validation experiments highlight that we can achieve a significant speedup in querying distributed RDF sources, and further confirm that our solution scales well with a growing set of data sources. In the future we plan to incorporate strategies to ensure data freshness, and devise and experiment with different caching strategies.

## Acknowledgement

Sven Casteleyn is supported by an EC Marie Curie Intra-European Fellowship (IEF) for Career Development, FP7-PEOPLE-2009-IEF, N° 254383. Elien Paret is funded by the FWO project DB-QueriDO, N° G.0212.08N.

## References

[1] P. Bellavista, A. Corradi, R. Montanari, A. Toninelli, Context-Aware Semantic Discovery for Next Generation Mobile Systems, IEEE Communications Magazine 44 (2006), pp. 62-71.
[2] V. Reynolds, M. Hausenblas, A. Polleres, Exploiting Linked Open Data for Mobile Augmented Reality, Proceedings of W3C Workshop: Augmented Reality On the Web, Barcelona, Spain, 2010.
[3] S. Casteleyn, W.V. Woensel, O.D. Troyer, Assisting Mobile Web Users: Client-Side Injection of Context-Sensitive Cues into Websites, Proceedings of 12th Int. Conference on Information Integration and Web-based Applications & Services, Paris, France, 2010, pp. 441-448
[4] J. Euzenat, J. Pierson, F. Ramparany, Dynamic Context Management for Pervasive Applications, Knowledge Eng. 23 (2008), pp. 21-49
[5] D. Le-phuoc, J.X. Parreira, V. Reynolds, RDF On the Go: An RDF Storage and Query Processor for Mobile Devices, Proceedings of 9th International Semantic Web Conference (ISWC2010), Shanghai, China, 2010.
[6] B. Quilitz, U. Leser, Querying Distributed RDF Data Sources with SPARQL, Proceedings of 5th European Semantic Web Conference, Tenerife, Spain, 2008, pp. 524-538.
[7] S. Lynden, I. Kojima, A. Matono, Y. Tanimura, Adaptive Integration of Distributed Semantic Web Data, Databases in Networked Information Systems 5999, 2010, pp. 174-193.
[8] Z. Kaoudi, K. Kyzirakos, M. Koubarakis, SPARQL Query Optimization on Top of DHTs, Proceedings of 9th International Semantic Web Conference (ISWC2010), Shanghai, China, 2010, pp. 418-435.
[9] H., Stuckenschmidt, R., Vdovjak, G.J. Houben, J. Broekstra, Towards Distributed Processing of RDF Path Queries, International Journal of Web Engineering and Technology 2(2/3) (2005), pp. 207-230.
[10] A. Harth, S. Decker, Optimized Index Structures for Querying RDF from the Web, Proceedings of Third Latin American Web Congress (LA-WEB'2005), Buenos Aires, Argentina, 2005, pp. 71-80.
[11] C. Weiss, A. Bernstein, Hexastore: Sextuple Indexing for Semantic Web Data Management, Proceedings of VLDB Endowment 1(1), 2008, pp. 1008-1019.
[12] C. Weiss, A. Bernstein, S. Boccuzzo, i-MoCo: Mobile Conference Guide – Storing and Querying huge amounts of Semantic Web data on the iPhone / iPod Touch, Proceedings of 7th International Semantic Web Conference, Karlsruhe, Germany, 2008.
[13]G. Judd, P. Steenkiste, Providing contextual information to pervasive computing applications, Proceedings of First IEEE International Conference on Pervasive Computing and Communications, 2003, pp. 133-142.
[14] W. Xue, H. Pung, P.P. Palmes, T. Gu, Schema matching for context-aware computing, Proceedings of 10th International Conference on Ubiquitous Computing (UbiComp 2008), COEX, Seoul, South Korea, 2008, pp. 292-301.
[15] G.D. Solomou, D.A. Koutsomitropoulos, S.T. Papatheodorou, Semantics-Aware Querying of Web-Distributed RDF(S) Repositories, Proceedings of 1st Workshop on Semantic Interoperability in the European Digital Library, Tenerife, Spain, 2008.
[16] W.V. Woensel, S. Casteleyn, O.D. Troyer, A Framework for Decentralized, Context-Aware Mobile Applications Using Semantic Web technology, Proceedings of On the Move to Meaningful Internet Systems: OTM 2009 Workshops, Vilamoura, Portugal, 2009, pp. 88-97.