Semantics-Aware Querying of Web-Distributed RDF(S) Repositories

Georgia D. Solomou, Dimitrios A. Koutsomitropoulos, Theodore S. Papatheodorou

High Performance Systems Laboratory, School of Engineering University of Patras, Building B, 26500, Patras-Rio, Greece
{solomou, kotsomit, tsp}@hpclab.ceid.upatras.gr

Abstract. Because of the scattered nature of the Semantic Web, the existence of an integrated framework for storing, querying and managing distributed RDF data is of great importance. Already existing and widespread systems that share similar goals, like Jena and Sesame, do not appear to support distributed storage and retrieval of RDF data for the time being. In this paper we present a mechanism, based on Sesame that facilitates querying of distributed RDF repositories using the SPARQL protocol. This mechanism achieves to successfully retrieve requested RDF data; at the same time it aggregates web-distributed ontological knowledge in an attempt to exploit potential inferences, a fundamental aspect of the Semantic Web. We present its architecture and the method used for the analysis of SPARQL queries, trying to implement retrieval of RDF data in an optimized way.

Keywords: RDF, Semantic Web, Distributed Querying, SPARQL, Optimization, Reasoning

1 Introduction

Resource Description Framework (RDF) [10] and its related technologies appear in the core of the Semantic Web technologies stack. As a means to describe web resources in a consistent manner, RDF can be of great value in representing, unifying and possibly interpreting information hidden in disparate databases, information management systems and portals. This kind of description is for example greatly valuated in the manipulation and interoperability of metadata and information about resources stored and disseminated through digital libraries.

Since RDF descriptions have become commonplace in such scenarios, the need for their management, maintenance and exploitation has risen to spawn the development of integrated software systems known as RDF repositories. Such systems typically allow for the efficient administration of RDF data lifecycle, their preservation and most importantly, their querying in data-intensive contexts. Amongst the most prominent, Sesame [3] and Jena [11] appear dominant in production as well as research-driven applications.

Although the development of RDF repositories comes with the notion of coping with the scattered nature of web descriptions, it is inevitable that the simultaneous existence of multiple such repositories leads to the standard information integration problem; that is, the transparent treatment of distributed RDF data hidden now inside disparate repositories. This condition becomes even worse when reasoning about RDF information is to be considered, which is a rather complex process, since it may involve the combination of triples stored in arbitrary sources that ultimately form the distributed knowledge.

In this paper we therefore suggest a mechanism that achieves distributed querying of RDF(S) repositories, a feature not currently supported by any of the related systems. This method builds around a *mediate* repository that attempts to fetch and store from each remote system only the most relevant triples to each particular query.

In addition and when reasoning is to be taken into account, we show how, through a careful preprocessing of the query we may avoid fetching unnecessary triples that would not contribute to the distributed knowledge whatsoever. In this way we suggest a method towards the optimization of RDF distributed querying when reasoning is involved.

At the same time, when inferences are not required, possibly in need of rapid results to massive querying, this method exhibits desirable scaling behavior: These triples are fetched that are exactly the ones to participate in the query evaluation, a quite time-consuming process if conducted against each remote repository separately.

The rest of the paper is organized as follows: In section 2 we mention some of the most popular query languages for RDF as well as some existing frameworks supporting them. Section 3 deals with the mechanism that was developed for querying distributed RDF repositories, presenting its architecture and its particular techniques used for achieving its main purpose in an optimized way. In section 4 we detail the implementation process by showing some results, whereas in the last section we present the derived conclusions and possible future work

2 Background

The mechanism's development is based on a kind of analysis of the RDF query language SPARQL [14]. SPARQL is a W3C recommendation which has borrowed many elements from previous existing RDF languages. It fulfils all requirements stated in [6] as being necessary in order to query RDF data, like compositionality, schema awareness, optional path expressions and datatyping. The results of SPARQL queries can be result sets or RDF graphs.

Generally speaking, the most prominent query languages are those that were conceived as first generation tryouts of RDF querying, with little or no RDF-specific implementation and use experience to guide design, and based on an ever-changing set of syntactic and semantic specifications [8]. Many of these languages were created having in mind SQL, so they share many features in common. Some languages of this kind are RDQL[12], RQL[9] and of course SeRQL[5], the fundamental query language of the RDF framework Sesame. Because RDQL and RQL are first generation languages, they seem to be less powerful than, for example, their successor

SeRQL. As far as SeRQL and SPARQL are concerned, they appear to share many features in common. However, our preference to use SPARQL as a basic query language for our mechanism came from its substantially improved structure related to older languages of this kind, plus the fact that SPARQL is an official W3C recommendation.

SPARQL is fully supported by both Jena and Sesame. These open source systems offer full reasoning for the semantic extension of RDF, RDF Schema (RDFS) [2], and many different ways for storing RDF data. They evaluate queries against local or remote repositories, but they do this only for a single storage each time. They are not able to query distributed RDF data and so they lack the essential ability to infer new knowledge by combining distributed information.

Sesame forms the base of our mechanism, too. It is a mechanism that allows querying at the RDF Schema level. It makes use of a forward-chaining inference engine to compute and store the closure of its knowledge base [4]. Whenever a transaction adds data to the repository, the inferencing algorithm applies RDF Model Theory entailment rules in an optimized way, making use of the dependencies between them to eliminate most redundant inferencing steps. Furthermore, Sesame's modular architecture allows for an easy combination of different repositories, using a standardized repository access API called SAIL. The modularity of Sesame's architecture and its proven scalability are two of the most important features that encouraged our choice to use this particular system as our mechanism's basis.

Beyond the actual storage system that may be used in the backend, it seems that a number of applications that use multiple RDF sources in practice have been developed, such as the DOPE Project (Drug Ontology Project for Elsevier) [13] which accomplishes integration of information in a customized way. Moreover, Kowari[1] – another storage and retrieval system for RDF – appears to offer a kind of querying capabilities over multiple sources. Its basic drawback, though, is that it doesn't provide data independence, as the target repositories for such a distributed querying process must be explicitly defined, so it requires the users to know in advance where to get certain information.

Another related attempt is the development of a distributed storage and query infrastructure on top of Sesame, mentioned in [1]. This system extends Sesame in terms of a mediator component that provides centralized access to a collection of local and remote sources, but it doesn't offer inferencing capabilities, a requirement that constitutes the main objective of our mechanism.

3 A Technique for Distributed RDF Querying

Our query mechanism has two main features: the first one is that it queries distributed RDF repositories and retrieves data that may be combined in order to infer new RDF statements. The basic idea is to retrieve utilizable data from remote repositories and not just simple query results. Its second feature is that in order to achieve this data retrieval, our mechanism employs a preprocessing method that analyzes a query into smaller parts and retrieves statements according to this analysis.

[1] http://www.kowari.org

The preprocessing method exploits the basic structure of RDF data. RDF, in order to describe resources provides a simple tuple model, <*S*,*P*,*O*>. The interpretation of this *statement* is that subject *S* has property *P* with value *O*, where *S* and *P* are resource URIs and *O* is either a URI or a literal value. This simplified approach of knowledge expression, namely *triples*, is handled in an efficient way so as to achieve optimization of query processing of RDF data.

The architecture and the main processing phases of our mechanism are described in the sections that follow. We detail the query analysis method and the data retrieval method, as these phases play an essential role in the optimization of our query processing technique.

3.1 Architecture

The basic architecture of our SPARQL query mechanism is composed of a central storage system where all retrieved RDF triples are gathered and a central processing system which facilitates the communication with remote repositories (Fig. 1).

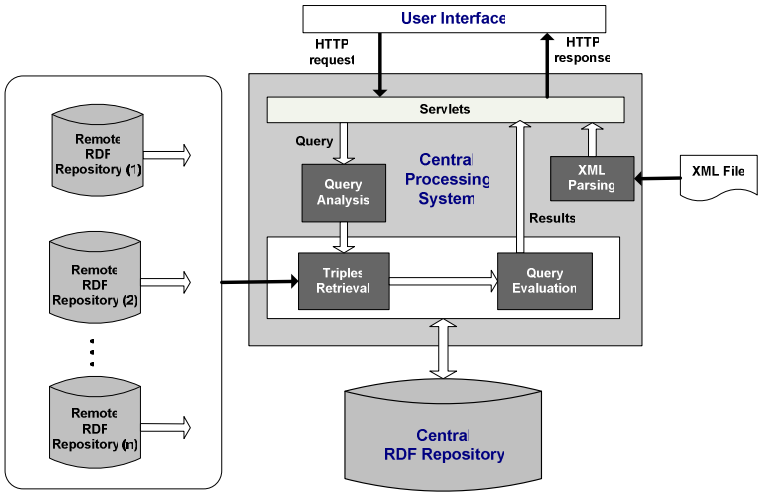

Fig. 1. Architecture of the SPARQL query mechanism for distributed RDF repositories. Central Processing System is divided into smaller, interconnected subsystems, each playing a fundamental role in the query evaluation process.

The processing system has, among others, the essential role of analyzing any query into smaller parts, as well as that of retrieving data from each remote repository, according to each resulting part. As a final step, it has to evaluate initial query against the central, local RDF repository.

The query is given as a string in a text box of a simple user interface and is passed to the central processing system through an HTTP connection. All necessary information, needed to establish connection to the available Sesame servers and RDF

repositories, is kept in a XML file, which is parsed by the corresponding mechanism in the central processing system.

3.2 Processing Phases

The basic idea of our mechanism is not just to fetch query results from each remote repository but to retrieve from them all the triples that take part in the formulation of final results. This kind of approach gives us the opportunity to combine distributed knowledge and obtain new inferred RDF statements.

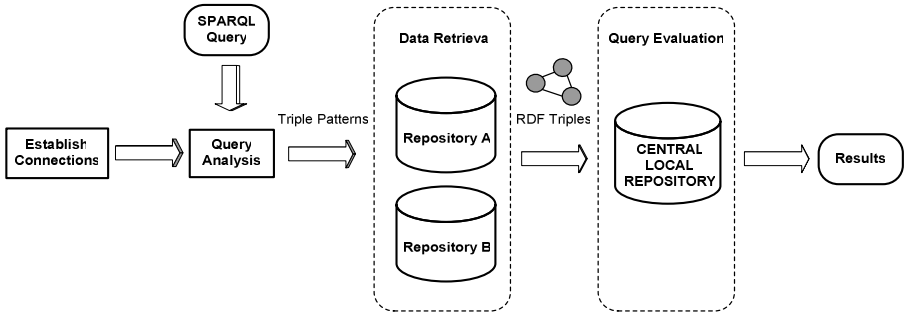

Fig. 2 Processing phases.

The first step is focused on establishing the connection with every available remote RDF repository (see Fig. 2). Afterwards, a query preprocessing is done, according to a method that involves a kind of syntactic analysis of query strings and their breaking into separate triple patterns and is detailed in section 3.3. Following this analysis, from each remote repository we retrieve RDF statements that adhere to these obtained triple patterns. All retrieved statements are placed in a central, local RDF repository, that also supports reasoning about RDFS. The central repository is emptied every time a new query evaluation is taking place. As soon as all RDF triples have been gathered from all available repositories, evaluation of the initial query follows. Evaluation is done against central repository's data, producing final results.

The motivation of our query analysis approach is better explained through the following example, shown in Fig. 3 and Fig. 4.

If we request all super-properties of <*subjectA*>, then, by evaluating such a query against each repository separately and by merging the results, we would get nothing more than single entities answering to the initial query, namely <*objectA1*>, <*objectA2*> and <*objectB4*> which cannot be further utilized (Fig. 3).

If we try, instead, to retrieve from each remote repository, not just the results, but statements that take part in the formulation of the results, the evaluation of the initial query against the central repository leads to the acquisition of new data, namely to new inferred information. Hence, in the following example, it is due to this analysis that we end up to obtain two extra statements, talking about super-properties of other entities. Furthermore, the central repository's capability to reason about RDF data leads to the proper combination of distributed data and to the disclosure of possible

"hidden" knowledge. As shown in Fig. 4, final answers to the example's initial query come from some extra information, apart from explicitly stored objects.

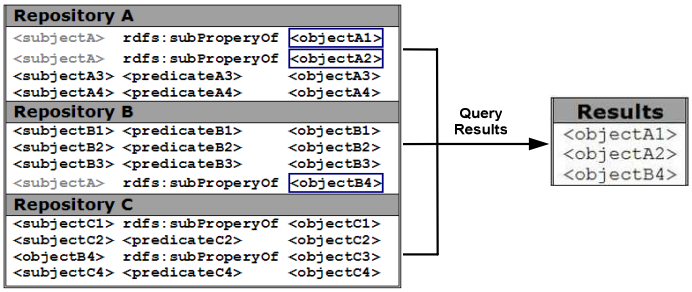

Fig. 3. Example of objects retrieved during a query evaluation against each of the three available distributed RDF repositories.

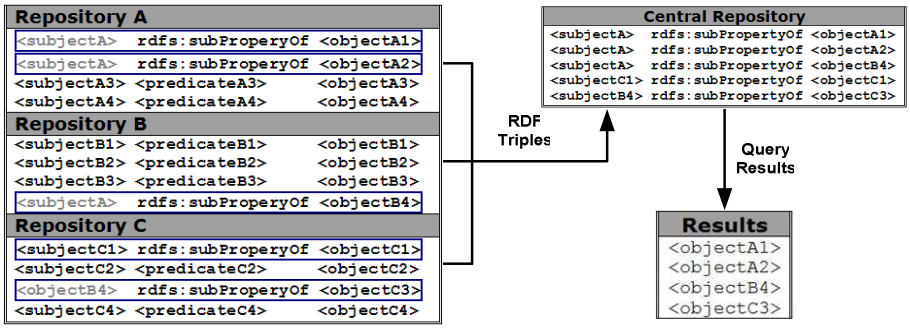

Fig. 4. Example of triples retrieved during a query evaluation against three distributed RDF repositories, using a central, local repository as an immediate store.

3.3 Query Analysis and Optimization

Query analysis is based on the idea that SPARQL queries are composed of one or more triple patterns, which create a graph pattern. A triple pattern is a RDF triple of the type <*S*, *P*, *O*> having the difference that some of its terms may have been replaced by variables. The objective of a query is to obtain results that match each triple pattern stated in its graph pattern. Hence, the main goal of the analysis is to recognize triple patterns found in a query's main body and retrieve RDF statements adhering to these patterns.

As shown in Table 1, where the structure of a SPARQL query is described, triple patterns are located in the fourth field of the query, namely in the WHERE clause. Therefore, analysis is focused in this particular field. It is also necessary to analyze the first field, the prologue, where all used base and prefix URIs are stated.

In the current phase of query analysis, we consider optional matches as obligatory, whereas we ignore any restriction imposed on retrieved data, stated by the keyword

FILTER. The first consideration has to do with our demand to avoid losing any possible utilizable information. As far as filter constraints are concerned, they mainly refer to the matching of data to certain numeric or string values: although these values would have restricted us to the retrieval of fewer RDF triples, the consideration of filter constraints can be rather time consuming, when applied to each repository separately. Finally, we don't take into account possible use of datasets, as this feature of SPARQL can't be utilized yet in a way that could benefit our mechanism.

Table 1. Structure of SPARQL query.

1. Prologue (*optional*)	`BASE <iri>`
	`PREFIX prefix: <iri> (repeatable)`
2. Query Result Forms (*required*)	`SELECT (DISTINCT)` sequence of ?variable
	`SELECT (DISTINCT) *`
	`CONSTRUCT {graph pattern}`
	`ASK`
	`DESCRIBE` sequence of ?variable or `<iri>`
	`DESCRIBE *`
3. **Query Dataset Sources** (*optional*)	Add triples to the background graph (repeatable): `FROM <iri>`
	Add a named graph (repeatable): `FROM NAMED <iri>`
4. Graph pattern (*optional, required for ASK*)	`WHERE {graph pattern [FILTER expression]}`
5. Query results ordering (*optional*)	`ORDER BY …`
6. Query results selection (*optional*)	`LIMIT n, OFFSET m`

By treating each query as a simple string, our method starts its analysis by searching the prologue field in order to find used URI bases and prefixes, which are necessary for later steps. We continue by analyzing the WHERE clause so as to find the stated triple patterns. Triple patterns are written as a whitespace-separated list of subject, predicate and object and the process of breaking them into smaller pieces is relatively complicated and based on the specification of SPARQL. The detailed description of this process is beyond the scope of this paper.

Following some simple rules and restrictions we reach to a point where each triple pattern found in the query's graph pattern has been isolated and seen as an individual triple with a subject, a predicate and an object. The RDF statement retrieval is done according to these obtained triple patterns.

3. 4 Data Retrieval

As mentioned above, a triple pattern is just a RDF triple (<*S*,*P*,*O*>) where some of its terms may have been replaced by variables. Another option for these terms is to be blank nodes or either to have a certain value (e.g. a URI or a literal). The data

retrieval method depends on whether or not inferred statements are requested to be included in the results, as well as on the type (variable, blank node, URI or literal) of the terms found in the triple patterns.

When concrete values (URI or literal) are used for triple terms, and provided that no inferencing is required, then we retrieve RDF statements that match these given values. When, instead, a triple term is represented by a variable or a blank node, we translate it as having to retrieve triples with any value at this particular place.

When inferencing constitutes the main objective of a query, data retrieval is done in a slightly different way. In this case, for each term found in a triple pattern of the query, we initially define its role (subject, predicate or object) and then we seek statements matching its value for this particular role. We also retrieve inferred statements related to this term. That means that we retrieve data in a way that ignores combinations of a certain subject-predicate-object triple. Instead, each term is treated as a separate entity and the retrieval process adheres only to this term's restrictions. Obviously, terms represented by variables or blank nodes are excluded, as they are seen as entities which can take "any value".

Table 2. Triples stored in each repository.

Sesame Server 1
Repository A
ex:Animal rdf:type rdfs:Class ex:Bear rdfs:subClassOf ex:Mammal
Repository B
ex:BrownBear rdfs:subClassOf ex:Bear
Sesame Server 2
Repository C
ex:Mammal rdfs:subClassOf ex:Animal ex:Fish rdfs:subClassOf ex:Animal ex:Bird rdfs:subClassOf ex:Animal
Repository D
ex:PolarBear rdfs:subClassOf ex:Bear
Sesame Server 3
Repository E
ex:Salmon rdfs:subClassOf ex:Fish ex:PolarBear ex:colour "white" ex:BrownBear ex:colour "brown" ex:PolarBear ex:eat ex:Seal ex:BrownBear ex:eat ex:Salmon ex:BrownBear ex:eat ex:Honey ex:PolarBear ex:weight "600"^^xsd:integer ex:BrownBear ex:weight "250"^^xsd:integer

Use Cases and Results

Based on Sesame's available API for storing RDF data and querying local and remote RDF repositories, our SPARQL query mechanism exploits all Sesame features related to reasoning about RDFS data. In this section we show a use case which demonstrates how this mechanism extends RDFS inferencing capabilities in distributed repositories.

We tested our mechanism using five different RDF repositories, located in three Sesame servers. Each repository's content was carefully chosen so that, if seen in combination with the stored triples in the other repositories, new RDF statements could be inferred. The distribution of RDF data in the available repositories is shown in Table 2. In this table, RDF data are represented as triples in their short form, using prefixes defined as follows:

1. prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2. prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3. prefix ex: <http://anemos.sesameweb.org/ex#>
4. prefix xsd: <http://www.w3.org/2001/XMLSchema#>

A more detailed look in the content of available repositories shows that a combination of their data implies a simple hierarchy of *Animal* class, as depicted in Fig. 5.

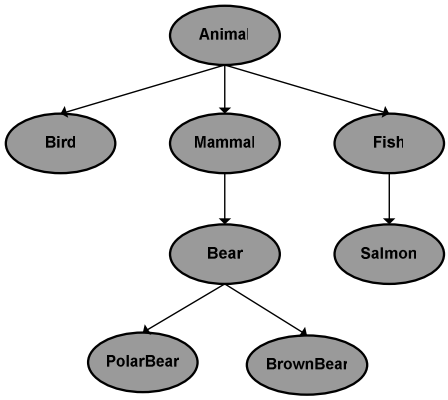

Fig. 5. Hierarchy of the Animal Class

To show our mechanism's special features in combining distributed knowledge, we evaluated a query for the simple case where no inferencing is required. We did the same when inferred statements is asked to be included in the formulation of the final results. In the next paragraphs we comment on the results.

Our first example query requests the projection of all possible superclasses of *PolarBear* class and is structured as follows:

```
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex:<http://example.org/ex#>
SELECT DISTINCT ?class
WHERE {ex:PolarBear rdfs:subClassOf ?class }
```

In the first case, where inferred statements are excluded from the query evaluation, the only answer we take is that *PolarBear* is subclass of *Bear*.Taking a detailed look at repositories' contents in Table 2, we indeed see that the only information about *PolarBear* superclasses is given in Repository D. The latter contains only one statement which corresponds to the necessary answer. No other repository contains explicit information about superclasses of *PolarBear*.

As a second step, we evaluate the previous query but this time we request the participation of inferred statements in the results. The answers we take contain some extra objects due to the effects of the reasoning process. Therefore, beyond the obvious result that *Bear* is superclass of *PolarBear*, we get the extra information that *PolarBear* is also subclass of *Mammal* and *Animal*, as shown in the Table 3. This is because in Repository A there is a triple stating that *Bear* is subclass of *Mammal*, whereas in Repository C we find the information that *Mammal* is subclass of *Animal* class. The combination of these three triples explains the final answers.

The fact that the results also contain the information that *PolarBear* is subclass of *Resource* and of itself comes from RDFS rules, stating that every class is subclass of itself and of *Resource* class.

Table 3. Results of the first query, including inferred statements.

class
http://example.org/ex#Bear
http://example.org/ex#Mammal
http://example.org/ex#Animal
http://example.org/ex#PolarBear
http://www.w3.org/2000/01/rdf-schema#Resource

A more complicated example, asks for the projection of the animal class, its corresponding weight and nominated food class as well as of all possible superclasses of the latter, restricting the results to those animals weighting less than a certain number. This query can be stated as follows:

```
PREFIX ex:<http://anemos.sesameweb.org/ex#>
SELECT DISTINCT ?animal ?weight ?food ?subClass
WHERE {   ?animal ex:eat ?food .
          ?animal ex:weight ?weight .
          OPTIONAL {?food rdfs:subClassOf ?subClass}.
          FILTER (?weight < 500)  }
```

Results, in the first case, when no inferencing is asked, correspond to the repositories explicit information, as shown in Table 4 (triple terms are depicted in their short form).

In the second case where inferred statements are taking part in the query process, results differ, as they contain new knowledge about superclasses of *Salmon* class, coming from the combination of data located in the other repositories (Table 5). Hence, new results inform us that *Salmon* is also subclass of *Animal* class, based on information coming from Repository C: *Fish*, an explicitly stated superclass of *Salmon*, is also subclass of *Animal*.

In this example, it is important to mention that results are indeed correct *and* complete, as optional matches and restrictions on numeric values were taken into account, omitting information about *PolarBear* class that has a weight value that is more than the requested one. These limitation conditions, as described earlier, are omitted during the phase of query analysis and data retrieval, but are always included in the last phase of query evaluation against the central repository. Therefore, our

choice to exclude them from the preprocessing phase in order to avoid extra overhead seems to have no negative effect in the final results.

Table 4. Results of the second query, where inferred statements are excluded from the results.

class	Weight	food	subClass
ex:BrownBear	"250"xsd:integer	ex:Salmon	ex:Fish
ex:BrownBear	"250"xsd:integer	ex:Honey	-no binding-

Table 5. Results of the second query, where inferred statements are included in the results

class	Weight	food	subClass
ex:BrownBear	"250"xsd:integer	ex:Salmon	ex:Fish
ex:BrownBear	"250"xsd:integer	ex:Salmon	rdfs:Resource
ex:BrownBear	"250"xsd:integer	ex:Salmon	ex:Salmon
ex:BrownBear	"250"xsd:integer	ex:Salmon	ex:Animal
ex:BrownBear	"250"xsd:integer	ex:Honey	-no binding-

5 Conclusions and Future Work

RDF distributed querying appears to be of crucial importance, having in mind the scattered nature of web resources descriptions. To this end, we have shown how distributed query answering can be achieved, in multiple RDF repositories. We have done so following a standards based approach, namely adopting the recent W3C standard SPARQL as querying protocol.

In addition, care has been shown on how to treat queries that involve reasoning as well. In this case, we have suggested a concrete preprocessing technique that amounts to a careful lexical analysis of the query and aims ultimately at the optimization of the response by amortizing related overheads, such as fetching and reasoning times.

As a proof of concept, a web-based querying mechanism has been developed, building upon and extending Sesame. Taking advantage of this service, we have presented a series of results that clearly demonstrate the potential of our method. At the same time this mechanism can be seen as enabling for richer interfaces that can be used in querying and harvesting metadata from distributed repositories and digital libraries. In particular, by gathering, combining and querying distributed information in a transparent way, it contributes in communicative interoperability of such digital stores. Furthermore, possible applied inferencing, leads to implied associations and information, thus gaining in the field of semantic interoperability.

Potential improvements to this approach may involve: first, the in-depth exploration of the various overheads occurring during the querying process, in an attempt to devise an upper bound on the optimization that is possible to be achieved; second the implementation and support for more expressive Semantic Web languages, namely OWL, a situation where reasoning-based query analysis will be even more subtle and demanding.

References

1 Adamku, G., Stuckenschmidt, H.,: Implementation and Evaluation of a Distributed RDF Storage and Retrieval System. In: Proceedings of of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (2005)

2 Brickley, D., Guha, R.V., (eds): RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, http://www.w3.org/TR/rdf-schema/

3 Broekstra, J., Kampman, A., Harmelen, van F. : Sesame: A generic architecture for storing and querying rdf and rdf schema. In: The Semantic Web - ISWC 2002, volume 2342 of LNCS, pages 54–68. Springer (2002)

4 Broekstra, J., Kampman, A.: Inferencing and truth maintenance in RDF Schema: exploring a naive practical approach. In: Workshop on Practical and Scalable Semantic Systems (PSSS) 2003, Second International Semantic Web Conference (ISWC), Sanibel Island, Florida, USA (2003)

5 Broekstra, J., Kampman, A.: SeRQL: An RDF Query and Transformation Language. In: the Proceedings of the Third International Semantic Web Conference (2004)

6 Broekstra, J., Kampman, A.: Serql: A second generation RDF query language, Technical report (2003)

7 Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K.: Jena: implementing the semantic web recommendations. In: Proceedings of the 13th international World Wide Web Conference on Alternate Track Papers & Posters (New York, NY, USA, 2004). WWW Alt. '04. ACM Press, New York, NY, pp. 74-83.

8 Haase P., Broekstra, J., Eberhart, A., Volz, R.: A Comparison of RDF Query Languages. In: Proceedings of the Third International Semantic Web Conference (2004)

9 Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Schol, M.: RQL: A Declarative Query Language for RDF. In: Proceedings of the Eleventh International World Wide Web Conference (WWW'02), Honolulu, Hawaii, USA (2002)

10 Klyne, G., Carroll, J. J., (eds): Resource Description Framework (RDF):Concepts and Abstract Syntax. W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

11 McBride, B.: Jena: Implementing the RDF model and syntax specification. In S. Decker et al., editors, Second International Workshop on the Semantic Web, Hong Kong (2001)

12 Seaborne A.: RDQL - a query language for RDF, W3C member submission (2004), http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

13 Stuckenschmidt, H., Waard, A. de, Bhogal, R., Fluit, C., Kampman, A., Buel, J. van, Mulligen, E. van, Broekstra, J., Crowlesmith, I., Harmelen, F. van, Scerri, T.: Exploring large document repositories with rdf technology - the dope project. IEEE Intelligent Systems (2004)

14 SPARQL query language for RDF. W3C Recommendation, http://www.w3.org/TR/rdf-sparql-query/