

Integrating RDF Querying Capabilities into a Distributed Search Infrastructure

Elena Demidova and Wolfgang Nejdl

L3S Research Center / University of Hannover
Deutscher Pavillon, Expo Plaza 1
30539 Hannover, Germany
{demidova,nejdl}@l3s.de

Abstract. The Semantic Web is inherently distributed, and covers both metadata and full-text information. Semantic search therefore can profit a lot from peer-to-peer infrastructures as well as from powerful metadata search functionalities based on full-text search technologies. In this paper we focus on an approach extending an existing P2P search infrastructure with RDF querying capabilities, which both exploits efficient indexing structures conventionally employed in a search engine, and provides metadata search based on the SPARQL query language. We will discuss indexing strategy and querying steps, and show that our approach provides an efficient solution for metadata querying on the Web.

1. Introduction

In order to allow for semantic information retrieval a search engine needs to combine full-text, metadata and distributed search capabilities. Although full-text search alone can not completely satisfy information needs of the user, it certainly provides an important component of a semantic search engine, since a lot of resources on the Web are not supplied with enough metadata information to retrieve them.

Metadata, on the other hand, provide important additional information about resources. This information can, for instance, allow disambiguation of search terms. It also promises additional result information not available otherwise. A typical full-text search engine provides its search results only as a list of document references. The user then has to scan this list and extract any information of interest manually. On the contrary, semantic search aims to provide meaningful and specific information, and thus can profit from a logic programming based view where the whole Web plays the role of a knowledge base and variable bindings obtained from the relevant documents should be returned to the user.

Furthermore, a decentralised environment like the Semantic Web incorporates many kinds of resources like web pages, databases and e-document servers. Associated RDF-encoded metadata are stored in different serialisation formats, in files as well as in relational or XML databases. In order to satisfy information needs of Semantic Web agents, a search infrastructure needs to provide a uniform interface for retrieval of resources stored in all these formats. In this paper we propose a

solution for semantic search based on a decentralised peer-to-peer search engine initially designed for full-text retrieval. In this context the peers are responsible for crawling different information providers and for creating a common homogeneous index structure.

The rest of this paper is organised as follows: Section 2 discusses our distributed semantic search scenario and provides relevant background information about the SPARQL query language and the YaCy P2P search engine used for our infrastructure. Based on these technologies, in section 3 we propose a new approach for RDF query execution in a distributed environment. Section 4 discusses our RDF indexing approach, suitable for integration into a full-text search engine. The implementation section 5 provides a brief overview of the technical details of the integration of these RDF query capabilities into the YaCy P2P search engine. Section 6 discusses experimental results, related work is presented in section 7. Finally, section 8 provides conclusions and a discussion of future research.

2. Motivation and Background

2.1. Scenario

In [1] Tim Berners-Lee and his co-authors described a scenario where Semantic Web agents set up doctor appointments. We will build on a slightly modified version of their scenario to motivate the approach presented in this paper.

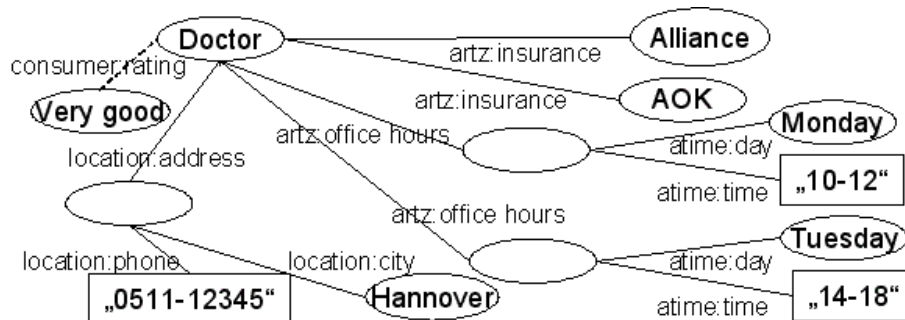


Fig. 1. Scenario

Assume that Bob needs to find office hours for a Tuesday appointment for his mother, Alice, at a suitable doctor in Hannover who accepts Alice's medical insurance plan, from Allianz Insurance. Such information is typically not stored within the web pages themselves, but can be provided as RDF-encoded metadata. We therefore can assume that the doctor's webpages export the data about location, office hours and supported insurances in such format. A possible RDF-encoding of this data is illustrated in the Fig. 1. The required information can then be obtained through the following query:

User Query:

```

SELECT ?id ?time
WHERE
  {?id artz:insurance <Alliance>}           --check if insurance is accepted
  {?id location:address ?address}           --retrieve address blank node
  {?address location:city <Hannover>}       --check city
  {?id artz:officehours ?officehours}       --get an office hours blank node
  {?officehours atime:day <Tuesday>}        --check the week day of the node
  {?officehours atime:time ?time}          --retrieve the time

```

The dashed line in the graph hints at the fact that ranking information is not stored within the same RDF graph, but is available from a different information provider. This information can additionally be queried after an appropriate doctor has been selected by Bob's agent. The corresponding query is:

```

SELECT ?rating
WHERE {<id> consumer:rating ?rating}       --select rating information

```

Clearly, the search task described above can not be solved by a full-text search engine. In order to provide the correct query answer, context information provided by the metadata structure needs to be preserved. Even if a conventional search engine would obtain both time values "10-12" and "14-18" contained in the data, their connection to the weekdays would be difficult to reconstruct.

So in order to provide correct and complete search results, a search engine needs not only to identify relevant RDF documents on the network but also extract all required information out of these documents. Full-text search engines provide a well-developed technology for identification of the relevant resources that should also be used by a semantic search engine. On the other hand, RDF query languages provide appropriate means to extract all required information from metadata contained in these documents. The following two sections discuss the SPARQL query language and the YaCy P2P search engine in more detail as both are relevant for our solution.

2.2. The SPARQL Query Language

An RDF capable search engine should enable structured RDF query formulation. The W3C DAWG - RDF Data Access Working Group¹ formulated requirements on an RDF query language in "RDF Data Access Use Cases and Requirements" [5]. On the basis of these requirements, the SPARQL [6] – query language for RDF is currently being developed by the DAWG working group, to create a standardized way for RDF resource discovery and extraction. SPARQL provides a human readable interface for RDF data retrieval and allows expression of RDF graph patterns. It facilitates, among other things, extraction of the graph information in form of node values. Alternatively, SPARQL supports creation of new RDF graphs based on the

¹ <http://www.w3.org/2001/sw/DataAccess/>

information contained in the queried graphs. The SPARQL syntax was inspired both by SQL and logic programming languages like Prolog. Queries contain variables which are represented by question marks. The triples included in the query are connected to a graph pattern by means of repeated node values.

Although the work on SPARQL is not yet finished, SPARQL implementations are already available for some existing RDF frameworks.

2.3. The YaCy Peer-to-Peer Search Engine

YaCy [4] is a distributed peer-to-peer open source search engine. One important characteristic feature of peer-to-peer architectures is equality of rights among the machines / peers connected to the network. Peers can be server and client at the same time. The main advantage of P2P search compared with centralised search is the ability of distributed processing. This includes all typical activities of a search engine like crawling, indexing and answering of user queries. Every peer can choose web content to be gathered. The resulting index structure is distributed over the P2P network. At query time, search results obtained from peers are combined to provide a homogeneous information view. P2P technology helps in keeping the content up to date and in protecting users against censorship and profiling.

YaCy uses a reverse word index (RWI) structure to store its index information. RWI associates each word extracted from the indexed documents with a list of URL references of the documents where this word occurs. At the end of every crawling procedure the index is distributed over the peers participating in the YaCy P2P network. Only index entries in the form of URLs are stored, no other caching is performed. YaCy implements its index structure as a distributed hash table (DHT): a hash function is applied to every index entry, the entry is then distributed to the appropriate peer. The resulting index contains no information about the origin of the keywords stored in it. Moreover, shared filters offer customized protection against undesirable content.

3. Distributed Query Evaluation for RDF Metadata

Let us now describe our query evaluation procedure based on the YaCy infrastructure. We want to use YaCy's full-text capabilities to index all relevant information including metadata, as this allows us to use the efficient index structures of YaCy. However, access through indexes just covers part of the query functionality we need (basically supporting the selection operator, but only partially the join operator). Our solution therefore builds upon a two step query processing approach. In the first step we use the YaCy indexing infrastructure to retrieve the relevant documents/data, in the second step we execute a SPARQL query on the retrieved data. The next section discusses this procedure in more detail.

SPARQL query execution is a graph pattern matching process between the query pattern and an RDF graph. RDF-encoded metadata files are available within as well as outside of the P2P network in various serialisation formats. As graph pattern

matching itself is time and resource consuming, we need to pre-select those RDF documents that can be relevant to a particular query.

In the second query execution step the pre-selected RDF resources are gathered from the network and analysed in order to answer the user query. Whereas the index structure is stored on the YaCy peers, the RDF resources themselves can be stored anywhere on the Internet and have to be retrieved and parsed. The final query answer includes the values contained in the queried RDF graphs in the form of variable bindings and is provided as an XML document using the SPARQL Query Results XML Format [5]. In the following both steps are discussed in more detail.

Step 1: Selection of RDF Resources

In order to identify RDF resources that are relevant for the user query, index information is extracted from the query graph. This information includes URI references and RDF literals contained in the graph as well as their combinations, which we will discuss in the query pattern section. Extracted terms are normalised and encoded in order to become comparable with the RDF index entries. The full-text search engine matches these search expressions within its full-text index. The result of this step is provided as a set of URL references where the full-text search terms occur. Algorithm of the 1st step in pseudo code:

```
Set FTQuery=SPARQLQuery.toFullText(); --Extract full-text query terms
Set URLSet= FTQuery.execute();          --Execute full-text query
```

The function SPARQLQuery.toFullText() maps a SPARQL user query to its full-text representation, providing a list of metadata / keyword patterns. In our scenario this produces the following mapping:

FTQuery=

```
“artz:insurance<Alliance>”,“location:address”,“location:city<Hannover>”,
“artz:officehours”, “atime:day <Tuesday>”, “atime:time”.
```

The function FTQuery.execute() evaluates the FTQuery using the RWI-based retrieval capabilities of the search engine and returns a set of relevant URL references.

Step 2: Execution of the SPARQL Query

In the second step the documents selected in the first step are downloaded and merged to build a graph. The original RDF query is executed on this graph and bindings for the variables contained in the query are provided.

Algorithm of the 2nd step in pseudo code:

```

RDF Graph g;
for each URL in URLSet do           -- go through the URL set
    {RDFFile f=URL.download()      -- download the RDF document
      g.add(f);                     -- add the document to the graph
    }
Result=g.matchPattern(Query)        -- match query pattern in the graph

```

Alternatively, query matching can be performed separately for each document.

Our current YaCy-RDF prototype implementation provides a SPARQL query interface based on ARQ² – an open source SPARQL processor for Jena. ARQ returns variable bindings in SPARQL Query Results XML Format. This format specifies an XML file that contains a header section where variable names are listed and a result section where variable bindings are provided. In the following, we present an example of the resulting XML structure for the data described in our scenario.

Query:

```
SELECT ?time ?phone WHERE ...
```

Query Result:

```

<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  ... head ...
  <results ordered="false" distinct="false">
    <result>
      <binding name="time">"14-18"</binding>
      <binding name="phone">"0511-12345"</binding>
    </result>
  </results>
</sparql>

```

The resulting XML file can be displayed directly by the browser as an XML tree. Another possibility is to use an XSL transformation.

4. Metadata Indexing in Peers

Every peer in the YaCy P2P network can choose documents to be indexed. Indexing means extraction of the relevant information subset. We will first describe the general YaCy index structure, and then consider RDF specific queries. Based on these query structures we discuss our approach for RDF specific index entry formulation in a full-text search engine.

² <http://jena.sourceforge.net/ARQ/>

4.1. The YaCy Index

The following table presents the YaCy RWI index structure. Through the indexing procedure, each word w extracted from the indexed document is associated with a set of URL references of the documents where it occurs.

<i>Word</i>	<i>URL</i>
W_1	$URL_1, \dots, \dots, URL_m$
...	...
...	...
w_n	$URL_1, \dots, \dots, URL_m$

Table 1. Reverse Word Index (RWI)

The next section discusses the RDF specific index entry formulation based on RDF query patterns.

4.2. RDF Query Patterns

Information contained in the index of a search engine should contain only the relevant subset of data. In order to extract this subset and create an efficient index structure, we discuss the set of typical RDF query patterns in the next paragraphs.

RDF documents consist of RDF statements. An RDF statement $\{S P O\}$ is a triple that contains subject, predicate and object. The predicate P describes the relationship between the subject S and the object O of the statement. This relationship is uniquely identified using a URI and can be a part of an XML namespace or an ontology and carries valuable semantic information. Common RDF query patterns are described in terms of relationships.

SP-Query

In this case the user wants to find entities which have a particular relation to a given resource. An example of such a query is

$$\{<URI\ of\ the\ resource>\ arzt:insurance\ ?o\}$$

An appropriate natural language interpretation of the query is “Which insurances are accepted by the doctor?” This kind of query is called a SP-Query, since the user provides both subject and predicate of the RDF-statement.

S-Query

The user wants to know everything about a particular resource. This is called S-Query, only information about the subject of the statement is provided.

$$\{<URI\ of\ the\ resource>\ ?p\ ?o\}$$

Other common RDF query patterns we consider are O-Query, PO-Query and SO-Query. These query patterns cover most of the relationship patterns within RDF triples. RDF triples described in this way can be arbitrarily combined to build complex graph patterns. For instance, the query:

```

SELECT ?time
WHERE
  {<id> arzt:officehours ?officehours }
  {?officehours atime:day <Monday> }
  {?officehours atime:time ?time }

```

can be represented as a combination of SP-, PO- and P-Query patterns. The model can be extended in order to integrate other kinds of simple relationships, as well complex ones, that turn out to be commonly used, based on query statistics.

4.3. Adding RDF Query Patterns to the Search Engine Index

RDF indexing is performed in three steps. At first RDF statements (triples) are extracted from the graph using the Jena framework [6]. In the next step, the triple type is determined. The type of an {SPO} triple is represented as a bit mask containing '1' in case the corresponding term of the triple is a URI reference or an RDF literal and '0' if it is a blank node or a variable. The following table presents these assignments:

Triple name	Triple type
O	{? ? O}
P	{? P ?}
PO	{? P O}
S	{S ? ?}
SO	{S ? O}
SP	{S P ?}
All	{S P O}
No	{? ? ?}

Table 2. RDF triple types

After the type of the triple has been determined, index entries are calculated:

```

switch(triple type)
{
  case SP:
    result.add({SP});
    if(query) return;
  case PO:
    result.add({PO});
    if(query) return;
  case SO:
    result.add({SO});
    if(query) return;
}

```

```

case All: //document indexing case only
    if(query) break;
    result.add({SP}, {PO}, {SO});
}result.add({S}, {P}, {O});

```

The algorithm differentiates between query and document cases, since in the case of the query it is enough to find one appropriate index entry. In the document indexing case, all available entries are added to the index. The resulting index structure contains the following entries:

<i>Word</i>	<i>URL</i>
{S}	URL ₁ , ..., ..., URL _m
{O}	...
{P}	...
{SP}	...
{PO}	...
{SO}	URL ₁ , ..., ..., URL _m

Table 3. RDF index structure

5. Implementation

We have integrated the RDF querying capabilities as described in this paper into the YaCy P2P search engine. In order to deal with RDF-encoded metadata, the YaCy architecture was generalised and specific RDF classes were added. In order to allow for efficient handling of RDF resources, a Jena [6] module was integrated into YaCy. Jena is an open source framework for Semantic Web applications developed by HP Labs. It provides a useful environment for RDF, RDF(S) and OWL. SPARQL query execution is provided by means of integration of the ARQ module, which is a query engine for Jena that supports the SPARQL Query language.

Our YaCy RDF prototype implementation can be downloaded from the URL <http://www.l3s.de/~demidova/yacyrdf/YaCyRDF.zip>

6. Evaluation and Experimental Results

We evaluated our approach in a small recall and precision measurement. A set of 50 RDF documents was analysed using the ARQ module for the relevance to the given query set and integrated into the index structure of the search engine. The document set was selected using the SWOOGLE³ search engine, using the search term “title”. The query set included all query patterns discussed in the previous section as well as their conjunctive combinations. For recall maximisation the following document selection criteria were applied:

³ <http://swoogle.umbc.edu/>

- All URL references were accessible at the time point of indexing.
- The document used for tests include only well-formed RDF documents.
- Redirecting URL references were not allowed. This restriction was required so that we could verify results in our YaCy-RDF implementation.
- MIME type restrictions: due to the current YaCy architecture a document is directed to the corresponding parser class according to its MIME type. Thus only RDF documents of the content types application/rdf+xml, application/xml, text/xml are parsed properly by our extended YaCy search engine:

Under these conditions, our YaCy-RDF prototype retrieved all relevant documents contained in the sets for all query patterns.

6.1. Precision of full-text document selection

Precision was measured for the first query execution step. We analyzed the set of the documents selected by the search engine using the ARQ module for the relevance to the user query.

Query type	YaCy total (Y)	ARQ precision analyse (PA)	Precision $P=PA/Y$
S	32	2	0,0625
P	38	37	0,97
O	30	30	1
SP	2	2	1
PO	32	32	1
SO	1	1	1
QComplex (1)	25	22	0,88

Table 4. Precision calculation

where:

- Query type – type of the query pattern as described in the indexing section.
- QComplex(1) - a simple, conjunctive graph pattern.
- YaCy total Y – number of URL references found by the YaCy-RDF prototype.
- ARQ precision analyse PA – number of URL references within the YaCy result Y confirmed by the ARQ query engine.

All full-text search terms are contained in all documents found by our YaCy-RDF prototype in all test cases. However, not all of them subsequently matched the graph patterns specified by the SPARQL queries. In general, precision results obtained in the full-text step differ depending on the query pattern. Combined query patterns like SP-, PO- and SO-Queries show high precision already in the document selection step, the same applies to a P- and O- query patterns. The lowest precision value of 0,0625 was obtained by singleton subject query triples (S-Query).

The more precise query patterns specify the document they intend to match, the more precise document selection can be performed. Combined query patterns like SP-, PO- and SO-Queries were able to produce exact document matches in our test scenario. The same applies to P-Queries since predicates describe relations within the RDF statements and URI references used for predicates in general do not appear in the role of a subject or an object of another RDF statement. URI references used as subjects and objects of the RDF statements are interchangeable, which leads to the loss of precision. Nevertheless, O-Queries deliver more precise results because of usage of RDF labels / strings in the test query set, which do not appear in the subject position.

Final responsibility for search precision as a whole is of course assigned to the second step of the query execution, thus the resulting search precision is satisfactory. Nevertheless, precision in the first query execution step is important in order to reduce the amount of documents that have to be processed by the RDF engine.

7. Related Work

Edutella [7] is a P2P networking infrastructure based on RDF. Edutella offers services for working with RDF metadata like query and annotation services. Although Edutella provides an RDF-based metadata infrastructure for P2P applications, it does not allow search for arbitrary textual information. The work described in this paper is focused on creation of a search engine that facilitates search for both, text and metadata information.

A few RDF search engines like QuizRDF [8] and Swoogle [9] already exist. Swoogle is a search engine for RDF and OWL documents encoded in XML or N3. It provides web service interfaces to support software agents and a web interface, which allows search for Semantic Web documents. However, Swoogle is not intended to answer more complex queries users with variable bindings and does not provide fulltext search.

8. Conclusions and Future Work

The YaCy-RDF prototype implementation described in this paper is a search engine for both textual and metadata information. RDF search functionality was integrated with the YaCy search engine, but is independent of specific implementation details of the YaCy architecture. Our RDF indexing approach can therefore be re-used for other full-text search engines as well. The two step query execution procedure allows for separation of responsibilities for recall and precision characteristics of the search engine.

Peer-to-peer networks offer additional advantages for metadata retrieval. Among them are facilities for distribution of document indexing and query processing. Our current YaCy-RDF prototype does not yet implement query distribution in the second query execution step. Documents identified within the index structure by their URL references are centrally processed by the requesting peer. However, if the peer

software is installed on an e-document server where the documents are physically located, it might be more efficient to delegate query execution to the hosting peer. In other scenarios it is more beneficial to merge query processing in order to combine results obtained from the RDF graphs that are stored in different locations. It will be interesting to investigate these scenarios in more detail and extend our current implementation with additional distributed query processing functionalities..

Although the development of SPARQL is not finished, our ARQ-based RDF query processing module will evolve together with SPARQL query language, and will facilitate long-term availability of the created extension.

During the work on our YaCy-RDF prototype, we were able to identify additional features important for efficient RDF retrieval. First, our current YaCy-RDF implementation supports only conjunctive queries. If user queries require merging of metadata stored in different locations, disjunctive query functionality is required in the first step in our query execution procedure. Another important aspect is the formulation of the SPARQL query, which requires knowledge of the underlying graph structure. Easier interfaces and schema-agnostic querying possibilities (for example based on malleable schemas, see [10]) would certainly be beneficial in this context.

9. References

- [1] Berners-Lee, T., Hendler, J.A., Lassila, O.; The Semantic Web; Scientific American, 284(5):34--43, May 2001.
- [2] Clark, K. G.; RDF Data Access Use Cases and Requirements; W3C Working Draft 25 March 2005; <http://www.w3.org/TR/rdf-dawg-uc/>
- [3] Prud'hommeaux, E., Seaborne, A.; SPARQL Query Language for RDF; W3C Working Draft 23 November 2005; <http://www.w3.org/TR/rdf-sparql-query/>
- [4] Christen, M.; Nicht-Monopolisierbarkeit. Vortrag zur Peer-to-Peer Suchmaschine YaCy; <http://www.yacy.net/yacy/material/YaCy-nichtMonopolisierbar.pdf>
- [5] Beckett, D.; SPARQL Query Results XML Format; W3C Working Draft 1 August 2005; <http://www.w3.org/TR/rdf-sparql-XMLres/>
- [6] Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.; Jena: Implementing the Semantic Web Recommendations; HP Labs 2003 Technical Reports; <http://www.hpl.hp.com/techreports/2003/HPL-2003-146.pdf>
- [7] Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmer, M., Risch, T.; EDUTELLA: A P2P Networking Infrastructure Based on RDF; Proc. of the 11th international conference on World Wide Web; Honolulu, Hawaii, USA; 2002.
- [8] Davies, J., Weeks, R.; QuizRDF: Search Technology for the Semantic Web; Proc. of the 37th Hawaii International Conference on System Sciences - 2004; http://www.cs.rutgers.edu/~shklar/www11/final_submissions/paper6.pdf
- [9] Ding, L., Finin, Joshi, A., Peng, Y., Cost, R.S., Sachs, J., Pan, R., Reddivari, P., Doshi, V.; Swoogle: A Semantic Web Search and Metadata Engine; Proc. of Thirteenth ACM Conference on Information and Knowledge Management (CIKM'04), Washington DC, November 2004; http://ebiquity.umbc.edu/_file_directory_/papers/115.pdf
- [10] Zhou, X., Nejdl, W.; Desktop Search with Malleable Schemas; Technical Report, University of Hannover, April 2006.