LODatio: Using a Schema-level Index to Support Users in Finding Relevant Sources of Linked Data

Thomas Gottron[1], Ansgar Scherp[1,2], Bastian Krayer[1], Arne Peters[1]
[1] Institute for Web Science and Technologies, University of Koblenz-Landau, Germany
[2] Data and Web Science, University of Mannheim, Germany
{gottron|scherp|bastiankrayer|arne}@uni-koblenz.de

ABSTRACT

The Linked Open Data (LOD) cloud provides a vast amount of heterogeneous data, distributed over numerous data sources. This makes it difficult to find those data sources in the cloud which are relevant for a given information need. Existing search engines for the Semantic Web focus on instance-oriented information needs, i. e., searching for specific RDF instances or literals and exploring the search results. However, they do not address the question of finding linked data sources relevant to a schema-oriented information need, i. e., queries based on triple patterns relating to a specific combination of RDF types and/or properties. In this paper, we present the semantic search system LODatio leveraging a schema-level index for finding sources of Linked Data relevant to a schema-oriented information need. Beyond its capability to retrieve relevant data sources, LODatio actively supports the user in his schema-oriented search tasks. To this end, it provides ranked result lists of relevant data sources together with example snippets and an estimation of the result set size. Furthermore, LODatio provides support for novel features in semantic search such as recommending alternative queries in order to refine or broaden the result set.

1. INTRODUCTION

The Linked Open Data (LOD) cloud provides structured data that is distributed over numerous, independent data sources. Just like the classical web of documents, there is no central instance of control which maintains a catalogue of the available information or keeps track of the schema underlying the data. This motivates the need for search engines to enable users to find the data they are interested in. Search engines such as Sindice [8], Sig.ma [14], and Swoogle [3] provide good services for finding semantic data on the web. Given a keyword-based query, they show a list of relevant resources such as instances but also properties and classes. In this way they allow users to satisfy instance-oriented information needs, i.e. information needs addressing a specific resource or resources with a specific label. These search engines find their limitations when the information need refers to a collection of resources described by a more general query pattern. Such information needs are typically schema-oriented, i. e., they address resources of a specific type or resources with specific properties (without constraints regarding the values of these properties). For instance, a user seeking for *actors* might be interested in data sources in the LOD cloud which provide information about many individual actors rather than references to RDFS classes for actors or a single instances with a literal property containing the string “actor”. Also, a user seeking for *married* might be looking for data sources providing information about people that are married rather than looking up specific persons matching this criterion or finding a vocabulary defining the poperty *married*. Such schema-oriented information needs are common and well known in information retrieval. They form open information needs where the user wants to know all items matching a certain criterion [10, 2]. The expected response is a list of items; translated to the case of the LOD cloud this corresponds to a list of data sources providing these items. The examples above describe users looking for data sources providing actors or people and their relationships. Such knowledge is also of relevance for software engineers who want to develop applications making use of specific types of Linked Data [12] or Linked Data engineers who publishing data on the cloud and wanting to interlink it with other data sources already published as Linked Data [11]. Summarized, in order to answer these questions, we need to capture the knowledge of *where* on the LOD cloud *which* kind of data is located. Providing this knowledge through a search system would allow for answering schema-oriented information needs which so far are not supported by existing semantic search engines. Such a search system should provide the following types of support for the user:

(A) Find relevant data sources based on a schema-oriented query specified by the user.
(B) Provide a ranking of the relevant data sources. The sorting criterion should reflect the ability of the data sources to satisfy the user’s information need. Given the potentially high number of relevant data sources, such a ranking is essential.
(C) Provide meta information about the data sources that allow for a quick judgement on whether or not the data source really is relevant. In the context of document retrieval such meta information about the results have been demonstrated to lead to an improved efficiency on the user side [13].
(D) Help the user to overcome difficulties in the information seeking process, e. g., when the query results in too few or too many data sources. This user support should suggest typical search tactics to circumvent the problems and implement the appropriate services [1].

In this paper, we present the novel semantic search system LODatio which aims at addressing these requirements. It provides features to actively support users in finding relevant sources of Linked Data in the cloud. The features comprise ranked retrieval of data sources relevant for a given schematic-oriented query, providing figures about the amount of resources to be found there as well as example data to help a user judge the extent and usefulness of the data to expect. Furthermore, LODatio supports the user in reformulating queries in order to widen or narrow down the result set. In the current version of LODatio, the user specifies his information need in the form of a SPARQL query. This is motivated by the ability of SPARQL to precisely capture a schema-oriented information need. To respond to this query, the online prototype of LODatio makes use of a schema-level index that has been computed from the entire Billion Triples Challenge (BTC) data set from 2012. However, our index can in principle be filled with any kind of additional RDF data from the LOD cloud.

The remainder of the paper is organized as follows: We start with an overview of the index structure used in the backend of our LOD retrieval system LODatio. In Section 3, we present the system's features and explain how they have been implemented. Finally, we discuss related work, before we conclude the paper.

2. A SCHEMA-LEVEL INDEX

As basis for LODatio we make use of a schema-level index called SchemEX [6, 4]. A SchemEX index consists of two parts: (a) the schema which serves for the look-up functionality and (b) the references to meta data which corresponds to the payload in the index. The elements in the schema represent sets of RDF resources that satisfy schema-oriented SPARQL query patterns. The payload, instead, can be used to store meta information about these resources in the index, e.g. where such resources can be found or how many resources on the LOD comply with this pattern. Figure 1 shows an illustrative extract of the index we use. The upper part represents the schema information. The type cluster (TC) elements in the schema describe resources of a given set of types. For example, TC-26 in Figure 1 corresponds to resources which are both Actor and Politician. The equivalence class (EQC) elements subdivide the TC and capture information about properties and links between the TC. The schema element EQC-8, for instance, represents resources of RDF type Politician and Actor which are linked to resources of type Model via a spouse property. Looking up the information stored in the index allows for a fast retrieval of the payload attached to the RDF resources complying with the specified query patterns. The payload is directly attached to the EQC elements as they are the most fine-grained and selective elements in the schema. For LODatio, we store specific meta data about the relevant data sources in the payload part of the index as shown in the lower part of Figure 1. The payload is key to the implemented user-centred services for supporting information seeking tasks on the LOD cloud. In addition to the URIs of the relevant data source, we store for each data source also the number of complying RDF resources, up to three example URIs of such resources, and literals used as their labels (as far as available). In the illustrated example, there are two entries complying with the schema information attached to EQC-8. For both entries, a data source (DS-URI) is given that contains RDF resources satisfying the query as well as the number of compliant resources and one or more example URIs and their labels that can be found there. The equivalence of schema elements with specific query patterns allows for an automatic translation of SPARQL queries to match the elements in the schema. For this translation, a SPARQL query is parsed and for each pattern of the query we construct a corresponding graph pattern on the schema index. Those query patterns which refer to schematic information (i. e., RDF types and properties) can be directly mapped to the respective schema elements. This process is entirely transparent and has the advantage that the end users do not need any knowledge

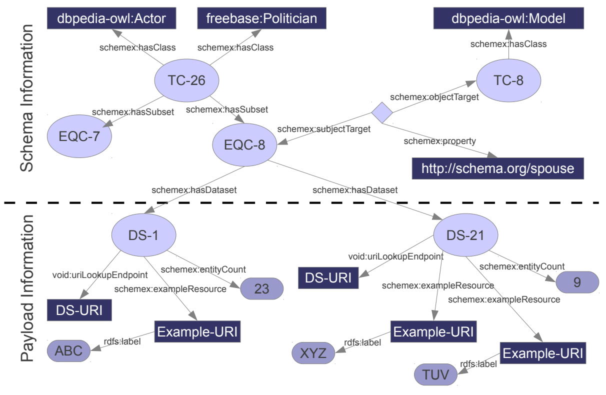


Figure 1: Schema-level index with the schema information (top) and payload information (bottom).

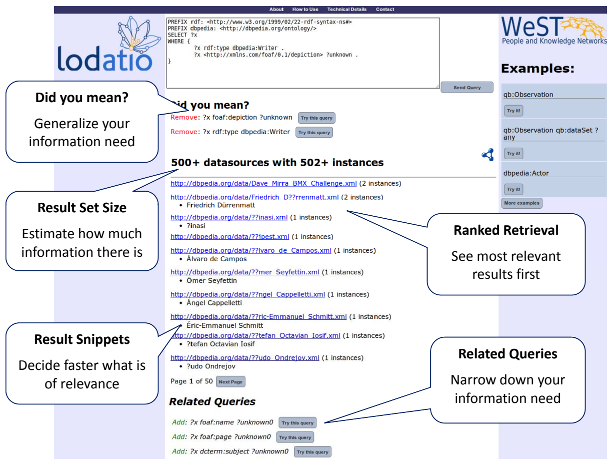
about the implementation of the schema structure. Given the focus of our index on the schema-level, we ignore all instance-level query patterns. The instance-level information is not stored in the index. Such information needs are not in the focus of LODatio. However, in principle the payload part is extensible to directly provide also instance level data.

3. THE LODatio SYSTEM

The aim of LODatio is to support the users in finding data sources on the LOD cloud relevant to schema-oriented information needs. We strive to provide support for all four aspects (A) to (D) mentioned in the introduction. LODatio achieves this aim by providing features directly targeted at alleviating the users' burden when searching for data on the LOD cloud. The features comprise ranked retrieval of data sources relevant for a given schematic data pattern, providing figures about the amount of resources to be found there, example data (called *snippets*) to help a user judge the extent and usefulness of the data to expect, and support for reformulating queries in order to widen or narrow down the result set. A screenshot of the demo system is shown in Figure 2 demonstrating all features of LODatio. Fast response times when querying this index are essential for the retrieval functionality as well as the other features implemented in LODatio. This requires an efficient implementation of the index structures. For LODatio we opted for a hybrid setup where we divide schema-index and payload over two architectures. We will present this hybrid knowledge base in Section 3.1. Subsequently, we briefly describe the support features and provide insights into their implementation. We show how we use our schema-level index for implementing the feature, and provide information on the computational effort.

3.1 Hybrid Knowledge Base for the Index

The schema-level index described in Section 2 is a graph structure as illustrated in Figure 1. As such, our index itself is represented in RDF using our own vocabulary (defined in the namespace http://schemex.west.uni-koblenz.de/). In addition, we use the property void:uriLookupEndpoint to denote the URI of a particular data source. The property rdfs:label is used to attach example resource labels found in the data sources to the payload. In general, it is possible to provide both the schema information and the payload information through a single triple store. However, this variant of implementing our index has shown drawbacks in


About
How to Use
Technical Details
Contact
lodatio
WeST
People and Knowledge Networks
Send Query
Try this query
Try it!
More examples
Next Page
Did you mean?
Generalize your information need
Result Set Size
Estimate how much information there is
Ranked Retrieval
See most relevant results first
Result Snippets
Decide faster what is of relevance
Related Queries
Narrow down your information need

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
SELECT ?x
WHERE {
        ?x rdf:type dbpedia:Writer .
        ?x <http://xmlns.com/foaf/0.1/depiction> ?unknown .
}
```

Examples:

qb:Observation

qb:Observation qb:dataSet ? any

dbpedia:Actor

Did you mean?

Remove: ?x foaf:depiction ?unknown

Remove: ?x rdf:type dbpedia:Writer

500+ datasources with 502+ instances

http://dbpedia.org/data/Dave_Mirra_BMX_Challenge.xml (2 instances)

http://dbpedia.org/data/Friedrich_D??rrenmatt.xml (2 instances)
- Friedrich Dürrenmatt

http://dbpedia.org/data/??inasi.xml (1 instances)
- ?inasi

http://dbpedia.org/data/??jpest.xml (1 instances)

http://dbpedia.org/data/??lvaro_de_Campos.xml (1 instances)
- Álvaro de Campos

http://dbpedia.org/data/??mer_Seyfettin.xml (1 instances)
- Ömer Seyfettin

http://dbpedia.org/data/??ngel_Cappelletti.xml (1 instances)
- Ángel Cappelletti

http://dbpedia.org/data/??ric-Emmanuel_Schmitt.xml (1 instances)
- Éric-Emmanuel Schmitt

http://dbpedia.org/data/??tefan_Octavian_Iosif.xml (1 instances)
- ?tefan Octavian Iosif

http://dbpedia.org/data/??udo_Ondrejov.xml (1 instances)
- ?udo Ondrejov

Page 1 of 50

Related Queries

Add: ?x foaf:name ?unknown0

Add: ?x foaf:page ?unknown0

Add: ?x dcterm:subject ?unknown0

Figure 2: The LODatio prototype demonstrating all features.

terms of performance when retrieving data sources for very general schema-oriented queries or queries which affect many different data sources (e.g., all data sources containing resources of type foaf:Person). Therefore, we use a hybrid architecture for the index. While the upper part of the schema information is provided through a triple store, the lower part of the payload information is implemented using a relational database. The unique identifiers of the EQC elements serves as primary key for the meta data stored in the relational database. This allows for an efficient and simple conversion of SPARQL queries to match the schema level and for fast lookup of the payload information.

3.2 User-support Features in LODatio

When using LODatio, the users always start by issuing an initial SPARQL query, which specifies their schema-oriented information need. Accordingly, all the user support features (ranking, result set size estimation, result snippets, and query generalization as well as specialization) are based on such an initial query.

Ranking of Data Sources. At the heart of LODatio—as in any retrieval system—is the ranked retrieval of relevant data sources matching the user's query. The computation of a result set, i. e., the set of relevant data sources on the LOD cloud providing resources satisfying the given SPARQL query, is provided by the process described above. Based on the payload information attached to data sources, we can implement a naive ranking function, which is described below. According to the classic Probability Ranking Principle [9], an optimal ranking of documents (in our case data sources containing RDF triples) is achieved when ranking the documents in decreasing probability of being relevant. Given that we are dealing with SPARQL queries as initial input to LODatio, the information need is specified in a precise way by a set of schema-oriented query patterns. Due to the nature of SPARQL queries, the query patterns either do or do not match the schema information and return relevant data sources. Thus, this binary relevance to the SPARQL query itself is not suitable for providing a ranking of the sources. Rather, we interpret the probability of relevance as the probability that a data source contains resources the user is actually interested in. With other words, we formulate the naive assumption that a data source providing more matching resources is more likely to provide the ones that are of interest. Thus, we rank the data sources by decreasing number of complying resources provided in the sources. This information can directly be obtained from the extended schema index as shown by the relations schemex:entityCount depicted in the payload of Figure 1. The ranking also supports the user in judging the volume of data he can obtain from a data source to satisfy his information need. This might play an important role on his next step of actually considering one or the other data source for obtaining data.

Result Set Size Estimation. Another central feature is the estimation of the overall result set size in terms of the number of instances matching the user's query. In LODatio, the estimation of the result set size can be obtained in a relatively simple and efficient way, as described below. Similar to the ranked retrieval of data sources, we can quickly answer the question of how many resources in total satisfy the user's information need, i. e., match the SPARQL query. By using the result set obtained for the ranked list above and simply aggregating the resource count information over all data sources, we can estimate the total number of complying resources available on the indexed fragment of the LOD cloud. This allows the user to estimate how successful his query is globally. In addition to this, the result set estimation function plays an important role in the query modification services described below.

Result Snippets. Traditional document retrieval systems like today's web search engines extend the items in the result list with query-related additional information, e.g. text passages containing the query terms. These snippets of additional information quite often provides the users with insights to support his subjective relevance decision on a result item [13]. In LODatio we include such result snippets as well. They consist of example instances found in the different sources of LOD. Along with the entries in the result list, we can provide the user with more information about the data sources. In addition to providing the endpoint URI of the data source, we can show him one or more relevant resources at this data source as well as a human readable label of this resource (if available). We therefore use the attached example resources and their label information as result snippets (shown by the rdfs:label properties in Figure 1). This allows the user to get a first glimpse at the data he might find on the data source. In turn, this might support him in his decision if the data source actually provides the kind of data he is interested in.

Extending the Result Set: Did You Mean? If users are not successful in formulating their information need well, they might end up with only few results. A typical tactic of the user to overcome this problem is to generalize his query [1]. Therefore, we support the user with a suggestion for a generalized query which actually provides more results. To this end, we iteratively generalize the SPARQL query by either removing or relaxing some of its query patterns. In this way, we obtain candidates for alternative, more general queries which are based on the original information need. For each alternative query, we evaluate the size of its result set. The top-k queries that modify the original query least and increase the result set by a significant amount are computed as a *Did you mean?* service. The process of removing or replacing query patterns follows a heuristic sequence of which patterns are addressed first. In a first round, we look for resources bound to more than one type. For these resources, we remove each individual type binding to create a candidate query. In a second round, we look for resources with multiple property bindings between them and also iteratively remove each property. As third step, we remove type bindings that uniquely specify a resource and finally generalize property constraints to arbitrary, i. e., unbound properties. This leads to a set of candidate queries that has a size directly depending on the number of query patterns in the original query. For each candidate query, we subsequently apply the above function for estimating the result set size. The candidates which extend the number of results in the most moderate way, i. e., adding the smallest number of resources, are suggested to the user.

Narrowing the Result Set: Related Queries. If the result set is too large, a user will have difficulties to browse the entire set and might want to specify his information need to narrow down the results. To support him in this task, we propose related queries which are more specific and reduce the number of relevant data sources without leading to an empty result set. For the related queries, the candidate queries are created by retrieving further information from the schema. Given that the schema index lookup typically covers several EQC elements, we can re-use this information for creating more specific queries. To this end, we consider the schema level information in the index that is attached to the retrieved EQC elements such as additional properties or RDF types. This information is used to extend the original SPARQL query by adding further constraints. This operation is monotonic, i. e., all previous constraints remain satisfied. With this approach we add only constrains which do not lead to an empty result set—otherwise they would not appear in the schema index. To avoid an over-specification of the related query, i. e., adding too many constrains at the same time that would reduce the result set size too quickly, we consider only those EQC elements which add exactly one additional constraint to the SPARQL query. Again, we evaluate the result set size associated with the candidate queries. Subsequently, we rank the candidates by decreasing size of the result set to suggest more specific queries starting with those that most moderately reduce the result set size.

4. RELATED WORK

Besides traditional textual search, the area of semantic search, i. e., the application of Semantic Web technologies for search has gained a lot of popularity in the last years [5]. Different general purpose search engines for the Semantic Web have emerged such as Swoogle [3]. It contains more than 10,000 ontologies and allows for string-matching based search in ontologies and terms. Other semantic search engines like SemSearch [7], Sig.ma [14], and Sindice [8] support the keyword-based querying approach of traditional search and entering an URI to start browsing. Subsequently, more sophisticated features are provided such as rating the trustworthiness of sources or removing specific sources from the results. Different user interface approaches are applied in these existing semantic search systems that range from simple string-matching search in Swoogle up to provenance-based search in Sig.ma. An approach like ours that imitates the services of classical Web search engines in order to make it easier and more intuitive to browse through relevant data sources of LOD and refining or extending the results has not been pursued so far.

5. CONCLUSION

We have presented LODatio, an advanced retrieval system for identifying data sources on the LOD cloud relevant to schema-oriented information needs. LODatio leverages a schema-level index to respond to a user's information need specified via a SPARQL query. The system provides core retrieval functionality such as ranked result lists, result snippets, and estimates of the total result set size. Furthermore, it supports two typical tactics of users when seeking information: generalizing and specifying the information need. The system is available as a live prototype at `http://lodatio.west.uni-koblenz.de:8080/`.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 257859, ROBUST.

6. REFERENCES

[1] Marcia J. Bates. Where should the person stop and the information search interface start? *Information Processing and Management*, 26(5):575–591, 1990.

[2] W. S. Cooper. A definition of relevance for information retrieval. *Information Storage and Retrieval*, 7(1):19 – 37, 1971.

[3] Li Ding, Timothy W. Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM*. ACM, 2004.

[4] Thomas Gottron and Rene Pickhardt. A detailed analysis of the quality of stream-based schema construction on linked open data. In *CSWS'12: Proceedings of the Chinese Semantic Web Symposium*, 2012.

[5] Ramanathan V. Guha, Rob McCool, and Eric Miller. Semantic search. In *WWW*, pages 700–709, 2003.

[6] Mathias Konrath, Thomas Gottron, Steffen Staab, and Ansgar Scherp. SchemEX - efficient construction of a data catalogue by stream-based indexing of linked data. *J. Web Sem.*, 16:52–58, 2012.

[7] Yuangui Lei, Victoria S. Uren, and Enrico Motta. Semsearch: A search engine for the semantic web. In *EKAW*, pages 238–245. Springer, 2006.

[8] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *IJMSO*, 3(1):37–52, 2008.

[9] Stephen Robertson. The Probability Ranking Principle in IR. *The Journal of documentation*, 33(4):294–304, 1977.

[10] Daniel E. Rose and Danny Levinson. Understanding user goals in web search. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 13–19, New York, NY, USA, 2004. ACM.

[11] Johann Schaible, Thomas Gottron, Stefan Scheglmann, and Ansgar Scherp. LOVER: Support for modeling data using linked open vocabularies. In *LWDM'13: 3rd International Workshop on Linked Web Data Management*, 2013. to appear.

[12] Stefan Scheglmann, Gerd Gröner, Steffen Staab, and Ralf Lämmel. Incompleteness-aware programming with rdf data. In Evelyne Viegas, Karin Breitman, and Judith Bishop, editors, *Proceedings of the 2013 Workshop on Data Driven Functional Programming, DDFP 2013, Rome, Italy, January 22, 2013*, pages 11–14. ACM, 2013.

[13] Anastasios Tombros and Mark Sanderson. Advantages of query biased summaries in information retrieval. In *SIGIR'98*, pages 2–10, 1998.

[14] Giovanni Tummarello, Richard Cyganiak, Michele Catasta, Szymon Danielczyk, Renaud Delbru, and Stefan Decker. Sig.ma: Live views on the web of data. *J. Web Sem.*, 8(4):355–364, 2010.