# Get the Google Feeling: Supporting Users in Finding Relevant Sources of Linked Open Data at Web-Scale
## (BTC Submission)

Thomas Gottron[1], Ansgar Scherp[1,2], Bastian Krayer[1], and Arne Peters[1]

[1] Institute for Web Science and Technologies, University of Koblenz-Landau, Germany
[2] Research Group on Data and Web Science, University of Mannheim, Germany
{gottron|scherp|bastiankrayer|arne}@uni-koblenz.de

**Abstract** Searching for Linked Open Data (LOD) has yet not reached the easiness and comfort we are accustomed with when using document search engines such as Google. To get closer to this "Google feeling" when searching for LOD, we have developed *LODatio* . Our system supports various kinds of queries on LOD such as searching for LOD sources containing specific types, properties, sets of types and properties, as well as their relations. The LOD sources are ranked along with the number of instances found that match the query. In addition, LODatio provides Google-style features such as *Did you mean?* and *Related queries* that allow the users for refining and broadening their queries.

## 1 Introduction

Semantic Web search engines such as Swoogle [2] or Sindice [7] and tools like Sig.ma [12] provide good services for finding and accessing semantic data on the web. Despite features such as filtering results by being a property or a class or considering the provenance of the triples, today's solutions lack what can be described as the "Google Feeling". Classical web search engines support the users in his information seeking task in various ways. They provide services which facilitate common search moves and tactics [1]. Such services, for instance, support the users in formulating more specific or more general queries to better describe their information need, refer to related information, or aid the users in making better relevance judgements, e. g., by providing snippets of the data sources together with the query results [11]. The importance of these services must not be undervalued. Research in the field of classical search engines indicates that users overcome weaknesses of a search system by adapting their information seeking behaviour [10]. Typical observation in these studies were a higher rate of queries per minute or more variations in query reformulation. Thus, it seems good advise to support users to perform these variations of their information seeking behaviour. In turn, this motivates the efforts of search engines to provide services in this field.

In this paper, we present the novel retrieval system LODatio. It provides for Google-style features to support users in satisfying their information need over a web-scale amount of Linked Open Data (LOD). Besides typical search engine functionalities such as providing a ranked list of relevant results, presenting result snippets, and estimating the size of the overall result set, we introduce some services which are new to the field of semantic retrieval. Inspired by the functions *Did you mean?* and *Related queries* by

web search engines, we have developed services which are capable of automatically suggesting more specific and more general queries that are close to the user's original information need. In the current version of LODatio we assume that the user specifies his information need in the form of a SPARQL query (we will extend the system in the future to keyword queries by using approaches as presented in [9]). By leveraging a schema-based index [5] over LOD, we are capable to provide a ranked list of data sources on the web capable of providing data that satisfies the query. In addition, we are capable to suggest generalized or more specific SPARQL queries. The central idea is to use the information in the schema index to modify the query patterns in a way that leads to modest extensions or restrictions of the result set.

Subsequently, we briefly present the schema index used for LOD retrieval in LODatio. Then, we present our core retrieval features used "under-the-hood", which also form the foundation of the Google-style services on top. Finally, we describe the prototype system and discuss briefly related work, before we conclude.

## 2 Leveraging a Schema Index for LOD Retrieval

The data structure we use in LODatio is a schema index over linked data [5]. The elements in this schema index correspond to RDF resources that satisfy certain types of SPARQL query patterns. The schema can precisely answer different kinds of queries: (i) resources of a given type or set of types, (ii) resources having certain properties, or even (iii) resources of certain type that are linked to other resources of a given type via a specific set of properties. While more complex queries cannot be answered exactly they can answered in an approximative way by combining the available patterns (i) to (iii). Looking up the information stored in the index allows for a fast retrieval of meta data of the RDF resources complying with the specified query patterns. In [5], this meta data provided solely information on *where* to find the RDF data, i. e., on which data sources on the LOD cloud. For LODatio, we have extended this meta data to implement the core services of ranking the results, estimating the size of the result set, and providing example snippets of the data. In addition to the URI of the 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). Figure 1 shows an extract of the extended schema index. The upper part represents the schema information and allows to map a SPARQL query based on its query patterns onto schema elements. Query patterns formulating restrictions on the (i) type of a resource are mapped onto so-called type clusters (TC). Query patterns formulating requirements with respect to the (ii) properties and the (iii) types of objects in this relation are mapped onto the equivalence class (EQC) elements that subdivide the TC. For instance, in Figure 1 the schema element EQC-8 represents resources of type Politician and Actor which are linked to a resource of type Model via a spouse property.

The lower part of Figure 1 shows the "payload" of the schema index, i. e., the information presented to the user when looking up resources with a SPARQL query. 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.

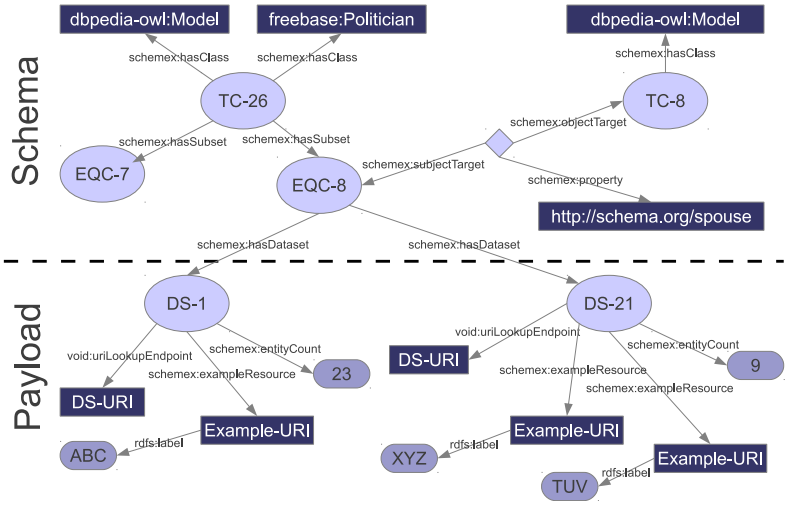


**Figure 1.** Information encoded in the extended schema index.

One particular feature of the schema index is that it can be computed very efficiently on commodity hardware using a single pass over a stream of RDF triples. Thus, it is worth mentioning that the extensions do not affect this property and the computation of this extended schema index can still be done efficiently.

## 3 Core Retrieval Features in LODatio

The extended schema index presented above allows for the implementation of core retrieval functions on the LOD cloud: ranking, estimation of the result set size, and the provision of result snippets. Common to all these functions is that they start from the initial SPARQL query specifying the user's information need. This query is then parsed and automatically transformed into a SPARQL query that matches our schema index. To this end, we analyse the patterns of the given query and construct a corresponding query pattern on the schema. The translated query allows for retrieving the relevant schema elements described above and access the meta information attached to them.

**Ranking** The computation of a result list, i. e., the list of data sources on the LOD cloud providing resources satisfying the given SPARQL query, can be realised with a basic schema index as in [5]. With the extended schema information introduced here, we can implement a naive ranking function. According to the classic Probability Ranking Principle [8], 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, the information need is specified in a sharp and precise way. Therefore, the question is not so much if a single triple is relevant to a SPARQL query, i. e., the resource suits the query patterns. This is already guaranteed by the schema which is built from the actual properties of the RDF triples which are indexed. In this context, we interpret the probability of relevance as

the probability that a data source contains resources the user is actually interested in. In conclusion, we formulate the naive assumption that a data source providing more resources is more likely to provide the ones of interest. Thus, we rank the data sources by decreasing number of complying resources provided. This information can directly be obtained from the extended schema index. The ranking also supports the user in judging the volume of data he can obtain from a data source to satisfy his information need.

**Estimation of the Result Set Size:** Similarly, we can quickly answer the question of how many resources in total satisfy the information need, i. e., 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. A very large result set might provide him with too much data, a small or even empty set with too little. Accordingly, he might wish to refine or generalize his information need. In addition to this, the result set estimation function plays an important role in the extended services described below.

**Result Snippets:** 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). This allows the user to get a first glimpse at the data he might find on the data source.

## 4 Google-style Features to Support Information Seeking on LOD

In order to provide for Google-style features to support the users in finding relevant data sources on the LOD cloud using LODatio, we make use of our schema index and the core functions presented above in an intriguing way. We briefly describe each feature and provide insights into the implementation and computational complexity for providing the results. We start from the initial SPARQL query, which originally specified the user's information need.

### 4.1 Extending the Result Set: Did You Mean?

If users are not successful in formulating their information need well, they end up with no or only few results. A typical tactic to overcome this problem is for the user to generalizes 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 queries—which are based on the original information need—that are more general. For each alternative query, we evaluate the size of its result set. The query that modifies the original query least and increases the result set by a significant amount is suggested to the user under the label *Did you mean?*

**Description:** 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 resulting resources in the most moderate way, i. e., adding the smallest number of resources, are suggested to the user.

**Complexity:** The complexity of this service depends on the number of candidate queries. For each candidate query, we need to estimate the size of the expected results set. Hence, for a query with $n$ query patterns we need to perform at most $n$ schema index lookups to retrieve the counts. The creation of candidates itself does not need to interact with the schema index, it operates solely on the original SPARQL query.

### 4.2 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 narrow down his information need. To support him in this task, we propose related queries which are more specific and reduce the number of data sources found.

**Description:** 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. 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. In addition, 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, 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.

**Complexity:** The computational complexity is little higher compared to the *Did you mean?* feature. First, we need to retrieve the schema information attached to the identified EQC elements. This corresponds to a single schema lookup for each relevant EQC to create the candidates. Second, we need to evaluate the result set size for each candidate, which adds an additional schema lookup. In conclusion, if we find $m$ relevant EQC, we have to perform at most $2 \cdot m$ lookups.

## 5 Demo System

We have implemented all of the above services and integrated them into the LODatio prototype. The system is live and publicly available (c. f. Fig. 2). To date, LODatio is running on a virtual machine on our cloud infrastructure with resources that correspond to 16 GB of RAM, a single core processor and 100GB of hard disk memory.

As data background, we use a schema index computed on the full BTC 2012 dataset. The schema has been computed using the streaming approach with default settings of

our schema extraction tools [5]. A real-time computation of the LODatio services can be achieved on a single core machine. To this end, the schema resulting from the BTC 2012 data set has been cleaned up such that all EQC elements with only a single resource have been removed.

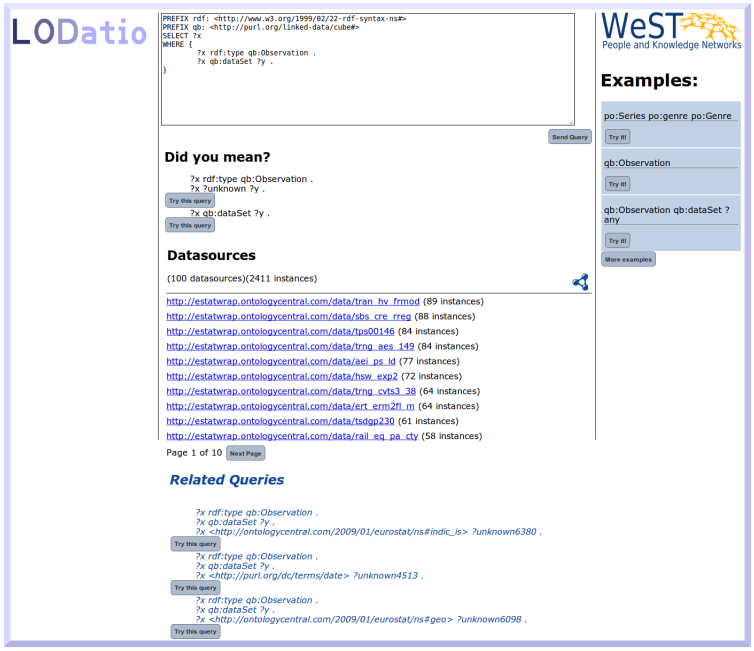


**Figure 2.** The LODatio prototype demonstrating all services

## 6 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 [4]. Different general purpose search engines for the Semantic Web have emerged such as Swoogle [2]. It contains more than 10,000 ontologies and allows for string-matching based search in ontologies and terms. There is also a more sophisticated ranking of the concepts and properties in Swoogle pursued by Ding et al. [3]. Other semantic search engines like SemSearch [6], Sig.ma [12], and Sindice [7] allow besides a keyword-based querying approach of traditional search also to enter 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 Google feeling 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.

## 7 Conclusion

We have presented LODatio, an advanced retrieval system for identifying relevant data sources on the LOD cloud. LODatio leverages a schema 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://btc2012.west.uni-koblenz.de:8080/`.

As next steps, we intend to develop a more sophisticated retrieval model to substitute the naive ranking approach. As long as the parameters and key indicators of such a model can be encoded into a schema index, the system will perform just as efficient. The generalization and specification of queries will be extended to a semantic operation mode in the sense that we will refine a query with, e. g., sub-types or sub-properties. Furthermore, we will extend the number of services to get even closer to a Google feeling. This includes the incorporation of techniques to derive a SPARQL query automatically from a keyword input as in [9]. Finally, we will evaluate the benefits on the user's side in extensive user studies. This shall reveal to which extent the services support an information seeking task a user is facing on the LOD cloud.

## References


1. Bates, M.J.: Where should the person stop and the information search interface start? Information Processing and Management 26(5), 575–591 (1990)
2. Ding, L., Finin, T.W., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: a search and metadata engine for the semantic web. In: CIKM. ACM (2004)
3. Ding, L., Pan, R., Finin, T., Joshi, A., Peng, Y., Kolari, P.: Finding and ranking knowledge on the semantic web. In: Proceedings of the 4th international conference on The Semantic Web. pp. 156–170. ISWC'05, Springer-Verlag, Berlin, Heidelberg (2005)
4. Guha, R.V., McCool, R., Miller, E.: Semantic search. In: WWW. pp. 700–709 (2003)
5. Konrath, M., Gottron, T., Staab, S., Scherp, A.: Schemex–efficient construction of a data catalogue by stream-based indexing of linked data. Web Semantics: Science, Services and Agents on the World Wide Web (2012)
6. Lei, Y., Uren, V.S., Motta, E.: Semsearch: A search engine for the semantic web. In: EKAW. pp. 238–245. Springer (2006)
7. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: a document-oriented lookup index for open linked data. IJMSO 3(1), 37–52 (2008)
8. Robertson, S.: The Probability Ranking Principle in IR. The Journal of documentation 33(4), 294–304 (1977)
9. Shekarpour, S., Auer, S., Ngomo, A.C.N., Gerber, D., Hellmann, S., Stadler, C.: Keyword-driven sparql query generation leveraging background knowledge. In: Web Intelligence. pp. 203–210 (2011)
10. Smith, C.L., Kantor, P.B.: User adaptation: good results from poor systems. In: SIGIR'08. pp. 147–154. ACM (2008)

11. Tombros, A., Sanderson, M.: Advantages of query biased summaries in information retrieval. In: SIGIR'98. pp. 2–10 (1998)
12. Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., Decker, S.: Sig.ma: Live views on the web of data. J. Web Sem. 8(4), 355–364 (2010)

## Addressing the Billion Triples Challenge Evaluation Requirements

### Minimal Requirements

**Web Scale:** LODatio makes use of a schema index that has been computed from the entire Billion Triples Challenge (BTC) 2012 Data Set. Compared to previous work, this schema is extended with additional information like the number of results to be found at some data source and example snippets.
**Realistic Web-quality Data:** Our systems works on a realistic data set of web-quality. It can cope in principle with any kind of RDF that is put into the schema extraction and representation process. Thus, it does not make any assumption about how many triples from how many data sources are provided, how well they are connected, and described using RDF types and properties.

### Additional Desirable Features

**More than Simply Store/Retrieve Large Numbers of Triples:** LODatio aims at being an application that can be applied by users to discover data sources on the LOD cloud in an easy and intuitive way. To this end, feature comparable to the textual Google search are provided like navigational pages of result sets and query suggestions for narrowing down (*Did you mean?*) and extending queries (*Related queries*). Manifold future extensions are described in the conclusions that will make LODatio even better.
**Scalable in Terms of Amount of Data:** As the schema extraction process can in principle process an arbitrary amount of linked data, LODatio will need to be able to deal with an arbitrary size of a schema. Currently, we rely on a simple single-core machine which is sufficient to host the schema of the full BTC 2012 data set. In need of hosting large schemata, it could easily be distributed by different means such as along the equivalence classes (EQCs) and the information attached to it like the type clusters (TCs) and distributed over a network of data providers.
**Scalable in Terms of Distributed Components:** As said above, LODatio can in principle run in a distributed fashion where different parts of the schema are distributed over several computing components. This includes a distribution of the service in a redundant fashion to implement load balancing over several machines.
**Use of Very Large, Mixed Quality Data:** The data provided by LODatio is the schema information extracted from the entire BTC 2012 data set. This data set is very large and contains data sources of different origin and mixed quality.
**Function in Real-time:** The LODatio can be run with the schema of the BTC 2012 data set on a single machine currently with 16 GB RAM and 100 GB of hard disk. The responsiveness of the system to queries lies within a couple of hundred milliseconds and a few seconds. To push towards a true real-time use of the system, we have optimized the schema and removed the EQCs that contain only one resource.