LD-VOWL: Extracting and Visualizing Schema Information for Linked Data

Marc Weise[1], Steffen Lohmann[2], Florian Haag[1]

[1] Institute for Visualization and Interactive Systems (VIS), University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
[2] Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS), Schloss Birlinghoven, 53757 Sankt Augustin, Germany

Abstract. Users currently face the problem that schema information for Linked Data is often not available. If it is available, it tends to be incomplete or does not adequately represent the data. It can therefore be hard for users to get an impression of the data provided by some Linked Data source. In this paper, we introduce LD-VOWL, a web-based tool that extracts and visualizes schema information of Linked Data sources based on the VOWL notation. SPARQL queries are used to infer the schema information from the data of the source, which is then gradually added to an interactive VOWL graph visualization. We tested LD-VOWL on a number of Linked Data endpoints with promising results.

Keywords: Linked Data, Schema Extraction, Visualization, SPARQL, RDF, OWL.

1 Introduction

A huge amount of Linked Data has been published in recent years, and is ready for consumption [2,5]. A large portion of this data is available in RDF format and can be queried using the standardized query language SPARQL [4,5]. The data often does not follow a strict schema, but typically different ontologies and vocabularies are used to describe it in a flexible way. On the one hand, this flexibility is an important characteristic and benefit of Linked Data; on the other hand, it can make it difficult to get an idea of what data is actually provided by a SPARQL endpoint. Visualizations can help to get a better overview of the type and structure of the data and can serve as a useful starting point for further querying and analysis.

In this paper, we introduce LD-VOWL, a tool that extracts and visualizes schema information from Linked Data endpoints, based on a number of SPARQL queries. This schema information is then incrementally added to an interactive graph visualization, using a slightly adapted version of the Visual Notation for OWL Ontologies (VOWL) [13,14].

2 Related Work

There are surprisingly few works concerning the extraction and visualization of schema information from Linked Data. Presutti et al. describe an approach of extracting *core*

knowledge [16] from Linked Data by detecting knowledge patterns. Central types and properties are identified by their betweenness and number of instances. In contrast to our approach, Presutti et al. focus on the detection of patterns in the data but not on the extraction and visualization of schema information.

Peroni et al. developed an approach for the automatic identification of *key concepts* [15]. Different from our work, the approach runs on ontologies and not on Linked Data. They use a couple of metrics, such as the length of concept names and their centrality in the graph structure, to find natural categories in the dataset. The concepts are also weighted by their popularity, which is defined as the number of results found by a search engine.

Another related work is QueryVOWL [9], which is also based on the VOWL notation and enables users without prior knowledge about RDF and SPARQL to query Linked Data. A graph consisting of VOWL elements is gradually constructed by the user and mapped to SPARQL queries which are sent to an endpoint. However, in contrast to our approach, QueryVOWL does not provide an overview visualization of the dataset but assumes that the user has already an idea of the type and structure of the data and knows how to start the querying process—as it is also assumed by many related querying approaches, such as LodLive [7] or the RelFinder [10].

Other works are concerned with the recommendation of concepts based on Linked Data [8,17], or follow general approaches of applying *formal concept analysis* to the Semantic Web [11].

3 Extraction of Schema Information

The schema extraction in LD-VOWL uses a *class-centric perspective*, i.e., the classes are extracted first and define the view on the Linked Data source. The classes are then connected by properties and enriched by datatypes. A class-centric perspective is very common in ontology engineering and fits well with the node-link paradigm of the graph visualization that the VOWL notation is based upon [13].

The extraction is realized by dynamically generated SPARQL queries revealing the schema information from a given dataset based on a couple of assumptions. For these queries, we had to find a trade-off between the number of required requests and the complexity of the queries. Since the SPARQL endpoints of Linked Data sources can have strict limits in terms of execution time, the queries must not be too complex. At the same time, we were aiming for displaying parts of the retrieved schema information as soon as possible, hence short response times were important as well. In addition, short response times were important, as we were aiming for displaying parts of the retrieved schema information as soon as possible to the users to minimize waiting time. Therefore, our priority was on using simple SPARQL queries, while we were also interested in limiting the total number of requests.

The SPARQL queries are sent in a stepwise approach, based on a couple of assumptions that are detailed in the following:

1. **Extraction of classes with the most instances:** A generic SPARQL query asking for the n classes with the most instances is sent to the endpoint first (where n is a user-defined upper limit). Listing 1.1 (Appendix) shows this query for the default

limit of $n = 10$. The results of this query serve as a starting point for further extractions.

This approach is based on the assumption that a dataset is well represented by the classes having the most instances. On the other hand, these classes are often also the more generic ones. Therefore, we integrated three strategies to avoid a too generic visualization:

(a) All built-in classes and properties of RDF, RDFS, OWL and optionally SKOS are contained in a blacklist that is filtered by default.

(b) Users can customize this blacklist by adding or removing classes according to their needs. For instance, they can remove `owl:Thing` from the list to include it in the visualization or add `foaf:Agent` to filter it too.

(c) Users can increase the number n of retrieved classes if the n initially retrieved classes are too generic, by adapting the limit of classes accordingly.

2. **Detection of subclasses, equivalent and disjoint classes:** Based on the n extracted classes with most instances, further SPARQL queries are sent to the endpoint in order to detect classes that can be considered equivalent, subclasses or disjoint classes. This is done by a pairwise comparison of the numbers of shared instances for all n classes, using the following assumptions:

   (a) If the number of shared instances of two classes is equal to the number of instances of each individual class, the classes are assumed to be equivalent.

   (b) If the number of shared instances of two classes is equal to the number of instances of the class having fewer instances, the class with fewer instances is considered a proper subset of the other class, which indicates a subclass relation between the two classes.

   (c) If there are no common instances at all, the two classes are considered to be disjoint.

   All three assumptions are based entirely on the actual data retrieved from the endpoint. For instance, two classes might not be explicitly defined as disjoint; however, if they do not share any instances in the dataset, a disjoint relationship will be inferred following the above assumption. This informs users that any search for individuals in that dataset which belong to both classes will be in vain.

3. **Retrieval of object properties:** In the third step, properties between the instances of the classes are retrieved. As with the classes, we retrieve the most frequently used properties first, i.e., properties having the greatest number of subject individuals (see example in Listing 1.2, Appendix). This also includes property loops, i.e., properties where the subject and object individuals are from the same class.

   As there can be a huge amount of different properties between the instances of two classes, we retrieve the properties in an incremental manner. When using a single SPARQL query, the execution of the query could take a very long time, possibly too long for Linked Data endpoints that have a strict limit for the execution time. Therefore, we choose the following approach in LD-VOWL: Starting with a limit of l properties, this limit is doubled with each SPARQL query sent until all properties are retrieved.

4. **Retrieval of datatype properties:** In the fourth step, LD-VOWL retrieves datatypes linked with the instances of the extracted classes. This step can be performed either after the third step or in parallel to it. LD-VOWL executes it in parallel in order

to avoid the impression that there are no datatypes defined for the retrieved classes due to the delayed retrieval and visualization (remember that LD-VOWL follows a stepwise approach and visualizes the elements as soon as they are extracted).
For each class, LD-VOWL sends queries that retrieve up to m datatypes which are most often used with the instances of that class (Listing 1.3, Appendix). After the datatypes are retrieved, the properties that connect the instances of the classes with these datatypes are fetched in a second step (Listing 1.4, Appendix). The reason for this two-step approach is again the limited execution time of many SPARQL endpoints. In addition, it supports our goal of visualizing the extracted schema information as quickly as possible, even if it is still incomplete. This requires the use of placeholders as labels for the datatype properties in the visualization as long as the actual properties are unknown.

It must be noted that due to the pairwise retrieval in both step two and three of the extraction process, the number of SPARQL requests that need to be sent grows quadratically with the number of classes n retrieved in the first step (i.e., $N_{requests} \in \mathcal{O}(n^2)$). Thus, we recommend to select the number of classes n that are initially retrieved with care and in accordance to the endpoint performance (LD-VOWL currently uses $n = 10$ as default).

4 Visualization Based on VOWL

LD-VOWL uses VOWL [13,14] for the visualization of the extracted schema information. We had to make some minor modifications to VOWL in order to address the peculiarities arising when visualizing information extracted from Linked Data.

In accordance with VOWL, extracted classes are represented as circle nodes in a force-directed layout (see Figure 1). The radii of the circles refer to the number of instances of the classes. Extracted properties are shown as directed and labeled edges linking the nodes. Different from VOWL, multiple properties between instances of the same pair of classes are merged into one edge. The more different properties exist between the instances of a pair of classes, the broader the edge is drawn. If different properties are merged into one edge, the property which occurs most often is considered most important—analogous to the class extraction principle. Therefore, the label of this property is shown on the edge, with the number of properties that have been merged given in brackets. If we would not merge those properties into one edge, this could result in a large number of edges being displayed between two classes, which would quickly clutter the visualization. Datatypes are displayed as yellow rectangles with a black border, like it is specified by VOWL. Accordingly, datatype properties linking the class instances with the datatypes are shown as edges with a green label.

The ontology or vocabulary comprising most of the classes is set as the main namespace of the dataset. The recommended default color of VOWL (light blue) is used as the background color of all elements in this namespace. All other ontologies and vocabularies that are part of the extracted schema have a different background color and inverted font color (white) in accordance with the VOWL specification. The colors used to indicate and group these other namespaces range from dark blue to pink in order to make different namespaces easily distinguishable in the visualization.

5 Implementation and User Interface

LD-VOWL is a web application implemented in JavaScript that sends SPARQL queries via HTTP GET to extract the schema information[3], and uses web standards like HTML5, CSS and SVG to display the extracted information. Furthermore, it makes use of the visualization toolkit D3 [6] for computing and displaying the force-directed graph.[4] The user interface of LD-VOWL is inspired by WebVOWL [12] and consists of three views:

1. The *start view* allows the user to select a Linked Data source by either entering the URL of its SPARQL endpoint or selecting one from a predefined list.
2. The *main view* (see Figure 1) shows the visualization of the extracted schema information. It is complemented by a sidebar with controls and information details.
3. The *settings view* enables the user to adjust the extraction by editing the blacklist or the language of labels, among others.

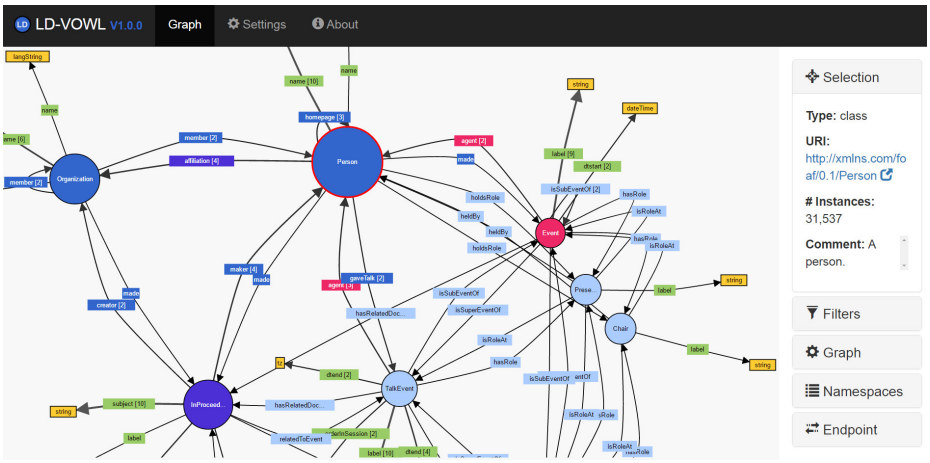

Fig. 1. LD-VOWL applied to the SPARQL endpoint of the Semantic Web Conference Corpus [3].

Users can zoom and pan to adjust the visible area and position of the graph that is shown in the main view. They can also modify the graph layout via drag and drop or by changing the average edge length. Furthermore, LD-VOWL provides options to filter parts of the extracted information in the visualization, such as datatypes, property loops, subclass relations, and disjoint classes. All nodes and edges in the graph visualization can be selected to see details on demand, for instance, the exact number of instances of a class or the list of all properties that connect two classes. URIs are displayed as hyperlinks, i.e., users can click on them to view further information (if available).

[3] Note that there are some endpoint requirements with regard to the supported SPARQL constructs, returned file format, handling of cross-origin requests, etc.

[4] A demo of LD-VOWL is available at: `http://ldvowl.visualdataweb.org`.

Finally, users can control the namespace classification by flagging namespaces as belonging to the main vocabulary or being marked as external. Users can also decide whether different colors should be used for the external namespaces or not.

6 Discussion

To unleash the full potential of Linked Data, it is important that users can get a quick overview of the type and structure of the data provided by a SPARQL endpoint. In this paper, we presented LD-VOWL, which allows to extract and visualize schema information from SPARQL endpoints. It uses a number of SPARQL queries that help to structure the data and reveal how it is described by ontologies and vocabularies, based on a set of assumptions. This schema information is then incrementally added to an interactive graph visualization using the VOWL notation.

We implemented LD-VOWL as a web application and tested it on several SPARQL endpoints. The results of these tests showed that LD-VOWL can create comprehensible overviews of the content and structures of datasets within a few seconds to minutes, depending on the performance of the endpoint (i.e., the used server, middleware, etc.), the extraction parameters selected in LD-VOWL (variables l, m, n, see Section 3) and the size of the dataset (which may affect the query execution time).

However, the results also show that the scalability of LD-VOWL is limited in several regards. As mentioned in Section 3, the number of SPARQL requests that need to be sent grows quadratically with the number of classes n that are initially retrieved. For this purpose, LD-VOWL retrieves only those classes that have the most instances (if not on the blacklist), which comes with benefits and limitations: On the one hand, LD-VOWL intends to provide an *overview visualization*, which implies that not *all* information is shown for datasets that contain a lot of classes and properties. On the other hand, it could be useful to explore certain regions of the overview visualization in more detail by 'expanding' parts of the graph and extracting further information for those parts on demand. Therefore, we could envision to extend LD-VOWL with such an exploration mode, or combine it with related visual querying approaches like QueryVOWL [9]. In general, LD-VOWL can be easily integrated with other tools, as it runs completely on the client side and only requests the server via SPARQL.

A direction for future research would be the extraction of further ontology concepts from the Linked Data sources, such as inverse properties or set operators, by developing corresponding assumptions and extraction patterns. These additional concepts could again be visualized with VOWL, which provides graphical representations for a large number of OWL language constructs [13,14]. LD-VOWL would also benefit from additional interactive features enabling the users to highlight, filter and collapse parts of the graph, as they are implemented in WebVOWL [12]. Such interactive features can improve the visual scalability and support the exploration and analysis of the extracted schema information.

However, the visual scalability of node-link diagrams can only be improved up to a certain extent with interactive features: Although a node-link diagram as used by VOWL is very suitable to depict the structure of some dataset, its visual scalability is inherently limited. The readability usually decreases with the number of nodes and

edges that are visualized. Therefore, another important direction of research is to investigate better means of visualizing a large amount of structured information—as it could potentially be extracted with approaches like LD-VOWL—in a more compact way.

References

1. DBpedia endpoint. http://dbpedia.org/sparql
2. Linked Data. http://linkeddata.org
3. Semantic Web Dog Food endpoint. http://data.semanticweb.org/sparql
4. SPARQL Endpoints Status. http://sparqles.okfn.org
5. Bizer, C., Heath, T., Berners-Lee, T.: Linked data – the story so far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
6. Bostock, M., Ogievetsky, V., Heer, J.: D^3 data-driven documents. IEEE Transactions on Visualization and Computer Graphics 17(12), 2301–2309 (2011)
7. Camarda, D.V., Mazzini, S., Antonuccio, A.: LodLive, exploring the web of data. In: 8th International Conference on Semantic Systems (I-SEMANTICS '12). pp. 197–200. ACM (2012)
8. Damljanovic, D., Stankovic, M., Laublet, P.: Linked data-based concept recommendation: Comparison of different methods in open innovation scenario. In: 9th Extended Semantic Web Conference (ESWC '12). LNCS, vol. 7295, pp. 24–38. Springer (2012)
9. Haag, F., Lohmann, S., Siek, S., Ertl, T.: QueryVOWL: A visual query notation for linked data. In: ESWC 2015 Satellite Events. LNCS, vol. 9341, pp. 387–402. Springer (2015)
10. Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., Stegemann, T.: RelFinder: Revealing relationships in RDF knowledge bases. In: 4th International Conference on Semantic and Digital Media Technologies (SAMT '09). LNCS, vol. 5887, pp. 182–187. Springer (2009)
11. Kirchberg, M., Leonardi, E., Tan, Y.S., Link, S., Ko, R.K.L., Lee, B.: Formal concept discovery in semantic web data. In: 10th International Conference on Formal Concept Analysis (ICFC '12). LNCS, vol. 7278, pp. 164–179. Springer (2012)
12. Lohmann, S., Link, V., Marbach, E., Negru, S.: WebVOWL: Web-based visualization of ontologies. In: EKAW 2014 Satellite Events. LNAI, vol. 8982, pp. 154–158. Springer (2015)
13. Lohmann, S., Negru, S., Haag, F., Ertl, T.: Visualizing ontologies with VOWL. Semantic Web 7(4), 399–419 (2016)
14. Negru, S., Lohmann, S., Haag, F.: VOWL: Visual notation for OWL ontologies. http://purl.org/vowl/ (2014)
15. Peroni, S., Motta, E., D'Aquin, M.: Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures. In: 3rd Asian Semantic Web Conference (ASWC '08). LNCS, vol. 5367, pp. 242–256. Springer (2008)
16. Presutti, V., Aroyo, L., Adamou, A., Schopman, B.A.C., Gangemi, A., Schreiber, G.: Extracting core knowledge from linked data. In: 2nd International Workshop on Consuming Linked Data (COLD '2011). CEUR Workshop Proceedings, vol. 782. CEUR-WS.org (2011)
17. Stankovic, M., Breitfuss, W., Laublet, P.: Linked-data based suggestion of relevant topics. In: 7th International Conference on Semantic Systems (I-SEMANTICS '11). pp. 49–55. ACM (2011)

A Examples of SPARQL Queries Used for the Schema Extraction

The following listings provide examples of SPARQL queries used by LD-VOWL to extract schema information from Linked Data sources, based on a couple of assumptions that are described in Section 3.

```
SELECT DISTINCT ?class (COUNT(?instance) AS ?instanceCount)
WHERE {
  ?instance a ?class .
}
GROUP BY ?class
ORDER BY DESC(?instanceCount)
LIMIT 10 OFFSET 0
```

Listing 1.1. SPARQL query retrieving the $n = 10$ classes having the most instances.

```
SELECT (COUNT(?originInstance) AS ?count) ?prop
WHERE {
  ?originInstance a <http://dbpedia.org/ontology/Agent> .
  ?targetInstance a <http://xmlns.com/foaf/0.1/Document> .
  ?originInstance ?prop ?targetInstance .
}
GROUP BY ?prop
ORDER BY DESC(?count)
LIMIT 10 OFFSET 0
```

Listing 1.2. SPARQL query retrieving the $l = 10$ most often used object properties connecting instances of the classes `Agent` and `Document` (run on the DBpedia endpoint [1]).

```
SELECT (COUNT(?val) AS ?valCount) ?valType
WHERE {
  ?instance a <http://dbpedia.org/ontology/Agent> .
  ?instance ?prop ?val .
  BIND(DATATYPE(?val) AS ?valType) .
}
GROUP BY ?valType
ORDER BY DESC(?valCount)
LIMIT 10
```

Listing 1.3. SPARQL query retrieving the $m = 10$ datatypes most often linked to the DBpedia class `Agent`.

```
SELECT DISTINCT ?prop
WHERE {
  ?instance a <http://dbpedia.org/ontology/Agent> .
  ?instance ?prop ?val .
  FILTER (
      DATATYPE(?val) = <http://www.w3.org/2001/XMLSchema#string>
  )
}
LIMIT 10 OFFSET 0
```

Listing 1.4. SPARQL query retrieving properties between instances of the DBpedia class `Agent` and the linked datatype `string`.