







A UML-Style Visual Query Environment Over DBPedia

Kārlis Čerāns^(✉) , Lelde Lāce , Mikus Grasmanis , and Jūlija Ovčinnikova 

Institute of Mathematics and Computer Science, University of Latvia, Riga, Latvia
{karlis.cerans, lelde.lace, mikus.grasmanis,
julija.ovcinnikova}@lumii.lv

Abstract. We describe and demonstrate a prototype of a UML-style visual query environment over DBPedia that allows query seeding with any class or property present in the data endpoint and provides for context-sensitive query growing based on class-to-property and property-to-property mappings. To handle mappings that connect more than 480 thousand classes and more than 50 thousand properties, a hybrid approach of mapping pre-computation and storage is proposed, where the property information for “large” classes is stored in a database, while for “small” classes and for individuals the matching property information is retrieved from the data endpoint on-the-fly. The created schema information is used to back the query seeding and growing in the ViziQuer tool. The schema server and the schema database contents can be re-used also in other applications that require DBPedia class and property linking information.

Keywords: DBPedia · SPARQL · Visual queries · ViziQuer · RDF data schema

1 Introduction

DBPedia [1, 2] is one of the central Linked Data resources and is of fundamental importance to the entire Linked Data ecosystem. DBPedia extracts structured information from Wikipedia¹-the most popular collaboratively maintained encyclopedia on the web. A public DBPedia SPARQL endpoint², representing its “core” data, is a large and heterogeneous resource with over 480 thousand classes and over 50 thousand properties, making it difficult to find and extract the relevant information. The existing means for DBPedia data querying and exploration involve textual SPARQL query formulation and some research prototypes that offer assisted query composition options, as e.g., RDF Explorer [3], that do not reach the ability to use effectively the actual DBPedia schema information to support the query creation by end-users.

There is a DBPedia ontology that consists of 769 classes and 1431 properties (as of July 2021); it can be fully or partially loaded into generic query environments, as SPARKLIS [4] (based on natural language snippets), or Optique VQs [5, 6] or ViziQuer

¹ <https://www.wikipedia.org/>.

² <http://dbpedia.org/sparql>.

[7] (based on visual diagrammatic query presentation). The DBPedia ontology alone would, however, be rather insufficient in supporting the query building process, as it covers just a tiny fraction of actual DBPedia data classes and there are quite prominent classes and properties in the data set (e.g., the class *foaf:Document*, or any class from *yago:* namespace, or the property *foaf:name*) that are not present in the ontology.

We describe here services for the DBPedia data retrieval query composition assistance, running in real time, based on the full DBPedia data schema involving all its classes, all properties, and their relations (e.g., what properties are relevant for instances of what classes; both class-to-property and property-to-property relevance connections are considered). We apply the developed services to seeding and growing visual queries within the visual ViziQuer environment (cf. [7, 8]), however, the services can be made available also for schema-based query code completion in different environments, including the ones for textual SPARQL query composition, as e.g., YASGUI [9].

Due to the size of the data endpoint we pre-compute the class-to-property and property-to-property relevance mappings, using then the stored information to support the query creation. We limit pre-computation of the class-to-property mapping just for sufficiently large classes as most classes would have way less instances than the connected properties (for smaller classes the on-the-fly completion approach is used).

The principal novelty of the paper is:

- A *method for auto-completing queries*, based on the class-to-property and property-to-property connections, working over the actual DBPedia data schema in real time, and
- A *visual query environment* for exploration and querying of a very large and heterogeneous dataset, as DBPedia is.

The papers' supporting material including a live server environment for visual queries over DBPedia can be accessed from its support site <http://viziquer.lumii.lv/dss/>.

In what follows, Sect. 2 outlines the query completion task. The query completion solution architecture is described in Sect. 3. Section 4 describes the DBPedia schema extraction process to build up the data schema necessary for query completion. The visual query creation is described in Sect. 5. Section 6 concludes the paper.

2 Query Completion Task

A *diagrammatic presentation* of a query over RDF data is typically based on nodes and edges, where a node corresponds to a query variable or a resource (or a literal) and an edge, labelled by a property path, describes a link between the nodes. A *UML-style query diagram* (as in ViziQuer [7], Optique VQs [5] or LinDA [10]) would also provide an option (in some notations, a request) to specify the class information for a variable or a resource represented by the node. Furthermore, some links of the abstract query graph can be presented in the UML-style query notation as node attributes.

The presence of a class information for a variable or a resource in a query, facilitated by the UML style query presentation, could facilitate the query readability. Still, this would not preclude queries that have nodes with empty class specification (cf. [8]).

Figure 1 shows example visual queries corresponding to some of QALD-4 tasks^{3,4}, suitable for execution over DBpedia SPARQL endpoint, in the ViziQuer notation (cf. [8] and [11] for the notation and tool explanation).

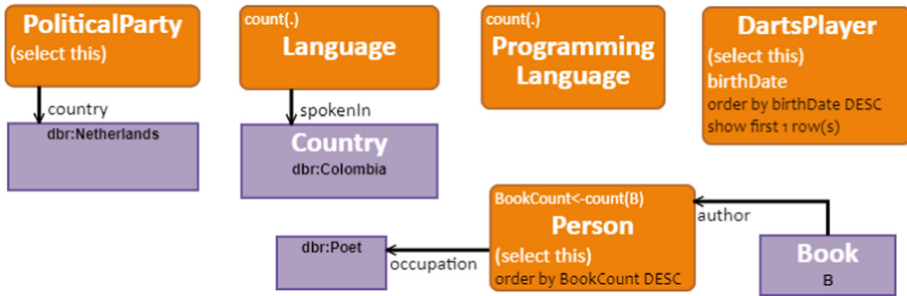


Fig. 1. Example visual queries. Each query is a connected graph with a main query node (orange round rectangle) and possibly linked connection classes. Each node corresponds to a variable (usually left implicit) or a resource and an optional class name. There can be selection and aggregation attributes in a node. The edges correspond to properties (paths) linking the node variables/resources. [8] Also describes more advanced query constructs.

From the auto-completion viewpoint a query can be viewed as a graph with nodes allowing entity specifications in the positions of classes and individuals and edges able to hold property names⁵.

The process of the visual query creation by an end-user starts with query initialization or *query seeding* and is followed by query expanding, or *query growing*⁶. Within each of these stages the query environment is expected to assist the end-user by offering the names from the entity vocabulary (involving classes, properties, possibly also individuals) that would make sense in the query position to be filled.

The simplest or *context-free* approach for the entity name suggestion would provide the entities for positions in a query just by their type—a class, a data property, or an object property (or an individual). This approach can provide reasonable results, if the user is ready to type in textual fragments of the entity name. The “most typical” names that can be offered to the user without any name fragment typing still can be significantly dependent on the context information where the entity is to be placed.

Another approach, followed e.g., by SPARKLIS [4] or RDF Explorer [3] would be presenting only those extensions of a query that would lead to a *query with non-empty solutions* (if taken together with the already existing query part). In the case of a large data endpoint, as DBpedia is, this would not be feasible, as even the simple queries to the endpoint asking for all properties that are available for instances of a large class typically do time-out or have running times not suitable for on-the-fly execution.

³ <http://qald.aksw.org/index.php?x=task1&q=4>.

⁴ cf. also <http://www.irisa.fr/LIS/ferre/sparklis/examples.html>.

⁵ Even if the query has a more complicated structure, the completion suggestions are computed on the basis of the described simple node-edge model.

⁶ The same applies also to query building in other (e.g.textual) notations.

We propose to use an in-between path by suggesting to the end-user the entity names that are *compatible with some local fragment* of the existing query (these are the entity names that make sense in their immediate context). We shall follow a *complete* approach in a sense that all names leading to an existing solution of the extended query need to be included into the suggestion set (possibly after the name fragment entry), however the names not leading to a solution can sometimes be admitted, as well.

In a schema-based query environment the main context element for a property name suggestion would be a class name, however, suggestion of a class name in the context of a property and suggestion of a connected property in the context of an existing property would be important to support the property-centered modeling style, and to enable efficient auto-completion within a textual property path expression entry⁷ (after a property name within an expression, only its “follower” properties are to be suggested, along with inverses of those properties whose triples can have common object with the last property from the already entered part of the property path).

3 Query Completion Principles

In what follows, we describe the principles of the query completion that can be shown to efficiently serve both the query seeding and context-aware query growing tasks for a SPARQL endpoint, as DBPedia core, with more than 480 thousand classes and more than 50 thousand properties, offering the text-search, filtering and prioritization options over the target linked entity sets. The query completion method has been implemented within a data shape server⁸ (also called schema server), featuring the example environments over the DBPedia core and other data sets.

3.1 Entity Mapping Types

The query completion on the data schema level is based on class-to-property and property-to-property relations, observing separately the outgoing and incoming properties for a class⁹, and “following”, “common subject” and “common object” modes for the property-property relations. The relations shall be navigable in both directions, so:

- The class-to-property (outgoing) relation can be used to compute the outgoing properties for a class, and source classes for a property,
- The class-to-property (incoming) relation can be used to compute the incoming properties for a class, and target classes for a property,
- The “following” property-property relation can be used for computing “followers” and “precursors” of a property.

⁷ For DBPedia core the direct property-property relation is much smaller than the property-property relation derived from the property-class-property mappings. For endpoints with less subclassing and the class structure more fully representing the property availability, the property-property mapping derived from the property-class-property relation may be sufficient.

⁸ <https://github.com/LUMII-Syslab/data-shape-server>.

⁹ A property p is outgoing (resp., incoming) for a class c , if there is a c instance that is subject (resp., object) for some triple having p as its property.

For each of the mappings it is important to have the list of suggested entities ordered so that the “most relevant” entities can be suggested first. To implement a context-aware relevance measure, we compute the triple pattern counts for each pair in the class-to-property and property-to-property relations; for the class-to-property (outgoing) relation also the counts of “data triple” patterns and “object triple” patterns are computed separately. An entity X is higher in the list of entities corresponding to Y , if the triple pattern count for the pair (X, Y) is higher¹⁰.

For query fragments involving an individual, the means shall be available for retrieving all classes the individual belongs to, all properties for which the individual is the subject (the properties “outgoing” from an individual) and for which the individual is the object (the properties “incoming” into the individual). We expect that the data SPARQL endpoint shall be able to answer queries of this type efficiently.

A further query completion task is to compute the individuals belonging to a class or available in the context of a given property (the class-to-individual, property-to-individual (subject) and property-to-individual (object) mappings). Since these mappings may return large sets of results for an argument class or property (e.g., around 1.7 million instances of *dbo:Person* class in DBPedia core), a text search with entity name fragment within the results is necessary. Such a search can be reasonably run over the SPARQL endpoint for classes with less than 100000 instances. For larger classes the suggested approach in query creation would be to start by filling the individual position first (using some index for the individual lookup as e.g., DBPedia Lookup¹¹).

The solution that we propose can also provide linked entity (property, class, individual) suggestion from several initial entities; this is achieved (logically) by computing the linked entity lists independently for each initial entity and then intersecting¹².

3.2 Partial Class-to-Property Mapping Storage

The modern database technologies would allow storing and serving to the query environment the full class-to-property and property-to-property relations¹³. Still, this may be considered not effective for a heterogeneous data endpoint, as DBPedia is, where for about 95% of classes the number of class instances is lower than the number of properties that characterize these instances. Out of 483 748 classes in the DBPedia core there have been 93 321 classes (around 19%) with just a single instance; in this case only a single link from the class to an instance is available in data. To record the relation of such a singleton class to the properties, all properties that the class instance exhibits, would need to be recorded. Since an instance may belong to several classes, such full storage of the class-to-property mapping is considered superfluous.

¹⁰ In the case of a heterogeneous endpoint, as DBPedia core is, the computation of local frequency of target instances in a context can give substantially different results from looking at the global “size” of the target entity.

¹¹ <https://lookup.dbpedia.org/>

¹² if a class c corresponds to both a property p and a property q , it is going to be suggested in a context of both p and q , although there may be no instance of c with values for both p and q .

¹³ There are about 35 million rows in the class-to-property (outgoing) relation in DBPedia core; the class-to-property (incoming) relation is much smaller.

Therefore, we propose to pre-compute and store the class-to-property relation just *for a fraction of classes* (we call them “large” classes), and to rely on the information retrieval from the data endpoint itself, if the class size falls below a certain threshold¹⁴ (regarding the property-property relation, our current proposal is to store it in full).

The partial storing of the class-to-property relation does not impede the possibility to compute the linked property lists for a given class, since for the classes that are not “large”, these lists can be efficiently served by the data SPARQL endpoint¹⁵.

The property-to-class direction of such a “partially stored” class-to-property relation becomes trickier, as, given a property, only the large classes are those that can be directly retrieved from the data schema. In order not to lose any relevant class name suggestions, we assign (and pre-compute) to any “small” class its representing superclass from the “large” classes set (we take the smallest of the large superclasses for the class). There turn out to be 154 small classes without a large superclass in the DBPedia endpoint (in accordance with the identified superclass information); the property links are to be pre-computed for these classes, to achieve complete class name suggestion lists.

The effect of suggested extra small classes in the context of a property can be analyzed. We note that in the DBPedia core out of top 5000 largest properties just 50 would have more small classes than the large ones within the source top 30 class UI window; in the case of target classes the number would be 190; so, the potentially non-exact class name suggestions are not going to have a major impact on the user interface (lowering the large class threshold would lower also the extra suggestion ratio even further).

3.3 Schema Server Implementation and Experiments

The schema server is implemented as REST API, responding to GET inquiries for (i) the list of known ontologies, (ii) the list of namespaces, (iii) the list of classes (possibly with text filter) and (iv) the list of properties (possibly with text filter), and POST inquiries for computing a list of classes, properties, and individuals in a context. The POST inquiries can specify query limit, text filter, lists of allowed or excluded namespaces, result ordering expression and the data endpoint URL; Further on there is a query context element, involving a class name (except for class name completion), individual URI (except for individual completion) and two lists of properties—the incoming and the outgoing ones; in the case of property completion, the context information sets can be created for both their subject and object positions.

The parameters of the schema server operations allow tuning the entity suggestion list selection and presentation to the end user. They are used in the visual tool user interface customization, in applying specific namespace conditions, or featuring basic and Full lists of properties in a context, as illustrated in Sect. 5.

A preliminary check of the schema server efficiency has found that the operations for suggesting classes and properties in a context perform reasonably, as shown in Table 1. For each of the link computation positions at least 10 source instances that can be

¹⁴ Within our initial prototype version, the class-to-property mapping is pre-computed for top 3000 largest classes; these classes contain at least about 1000 instances each.

¹⁵ In the case of the DBPedia core endpoint the size of such a list for classes with less than 1000 instances typically do not exceed a few hundred.

expected to have the highest running times (e.g., the largest entities) are considered and the maximum of the found running times is listed.

The experiments with the schema server have been performed on a single-laptop (32 GB RAM) installation of the visual tool, with the PostgreSQL database over the local network and remote access to the public DBPedia endpoint¹⁶ as the data set; the query time is measured by the printouts from the schema server JavaScript code.

Table 1. Entity list suggestion timing estimates

	Time upper estimate
Top 30 classes (all classes, dbo: namespace only, all except yago:), with possible text filter	259 ms
Top 30 properties (all properties, object properties, data properties), with possible text filter	412 ms
c → p links (data and object out properties), from a large class	882 ms
c → p extended links (in/out object properties, with other end range/domain class, if available), from a large class	2141 ms
c → p links (data and object out properties), from a small class	2438 ms
c → p extended links (out and in object properties, with other end range/domain class, if available), from a small class	1148 ms
p → p links (data and object out properties), from incoming and outgoing properties	577 ms
p → p extended links (in/out object properties, with other end range/domain class, if available), from incoming and outgoing properties	2760 ms
p → c links, from an in and an out property (including the large classes only and both the large and small classes suggestion cases)	269 ms

We note that the queries for computing the entities in a multiple context, do not tend to blow up the execution time, if compared to the single-context inquiries.

4 Data Schema Retrieval

Some of the data endpoints may have an ontology that describes its data structure; however, it may well be the case that the ontology does not describe the actual data structure fully (e.g., including all classes, all properties and all their connections present in the data set)¹⁷, therefore we consider retrieving the data from the SPARQL endpoint itself¹⁸.

¹⁶ <http://dbpedia.org/sparql>.

¹⁷ The DBPedia ontology covers just a tiny fraction of the actual DBPedia core data structure.

¹⁸ The data owner or a person having access to the data dump can also have other options of producing the data schema.

The extraction of small and medium-sized schemas can be performed by methods described in e.g., [12] and [13]. We outline here retrieving the DBPedia schema.

The DBPedia core schema retrieval has been done from a local copy, installed from DBPedia Databus site¹⁹ (the copy of December 2020).

The basic data retrieval involves the following generic steps that can be followed on other endpoints, as well:

- 1) Retrieve all classes (entities that have some instance), together with their instance count²⁰.
- 2) Retrieve all properties, together with their triple count, their object triple count (triples, where the object is an URI) and the literal triple count.
- 3) For classes deemed to be “large”²¹, compute the sets of its incoming and outgoing properties, with respective triple counts, including also object triple count and literal triple count for outgoing properties. For the classes, where direct computation of properties does not give results (e.g. due to the query timeout), check the instance counts for all (*class,property*) pairs separately²².
- 4) Retrieve the property-property relations, recording the situations, when one property can follow the other (a), or both properties can have a common subject (b), or a common object (c), together with the triple pattern counts.
- 5) Pre-compute the property domain and range information, where possible (by checking, if the source/target class for a property with largest property triple count is its domain/range).

- 6) Create the list of namespaces and link the classes and properties to them.

The following additional schema enrichment and tuning operations are performed, using the specifics of the DBPedia endpoint organization.

- 7) Compute the display names for classes and properties to coincide with the entity local name, with some DBPedia-specific adjustments:
 - a. If the local name ends in a long number (as some yago: namespace classes do), replace the number part by ‘..’, followed by the last 2–4 digits of the number allowing to disambiguate the display names),
 - b. If the local name contains ‘/’, surround it by [[and]],
 - c. For the *wikidata*: namespace, fetch the class labels from wikidata²³ and use the labels (enclosed in [[and]]) as display names.
- 8) Note the sub-class relation²⁴ (to be used in the class tree presentation, and in determining the “representative” large classes for small classes).

¹⁹ <https://databus.dbpedia.org/dbpedia/collections/latest-core>.

²⁰ this requires setting up a local DBPedia instance to enable queries with 500K result set, split e.g., in chunks of 100K, we order the classes by their instance count descending.

²¹ currently, the 3000 largest classes; the class count, or size threshold is introduced by the user; the optimal level of the threshold can be discussed.

²² we did the detailed computations automatically for classes larger than 500K instances.

²³ <http://query.wikidata.org/>.

²⁴ From the explicitly stated ontology and the sub-class-of assertions in the main data graph.

- 9) Note the class equivalence relation, to allow the non-local classes to be “represented” by the local ones in the initial class list.
- 10) For each “small” class, compute its smallest “large” super-class (for use in the property-to-class mapping to suggest also the “small” class names). Perform the step (3) for “small” classes that do not have any “large” superclass.

The schema extraction process currently is semi-automated. It can be expected that after a full automation and some optimizations it would be able to complete within a couple of days. The process can be repeated for new DBPedia configurations and data releases. The database size on the PostgreSQL server (including the tables and indices) amounts to about 20 GB. The dump of the database for the currently analyzed DBPedia endpoint can be accessed from the paper’s supporting website.

5 Visual Query Creation

To enable the creation of visual queries over DBPedia (cf. Fig. 1 in Sect. 2), the ViziQuer tool [7] has been connected to the data schema server and enriched by new features involving: (i) new shape of the class tree, (ii) means for query seeding by properties and individuals, and (iii) search-boxes for names in attribute and link dialogues and for classes in the node property pane.

The implementation of the tool allows also for endpoint-specific extensions to customize the tool appearance while working on specific data endpoints.

The created ViziQuer/DSS tool can be accessed from the paper’s supporting website.

We briefly explain the visual environment elements that enable the schema-supported query creation experience, relying on the schema server API, (cf. Section 3).

For the query seeding there are tabs with class, property and individual selection, the class tab can show either the full list of classes, or the full list of classes without the dominating *yago:* namespace, or just the *dbo:* namespace classes (the top classes of the first two choices are in Fig. 2); the properties in their tab can be listed either in the basic (moving down the *dbp:* namespace properties and a few more “housekeeping” properties), or in the full mode (ordering just by the triple count descending). The property search can be restricted to either data or object properties only (a property of “dual nature” would be present in both lists). Both the class and property lists are efficiently searchable. There is also an option to obtain a list of subclasses for a class. Double click on an item in any of the tabs, initiates a new query from this element.

The main tools for query growing are the attribute and link addition dialogues, illustrated in Fig. 3, in the context of the *dbo:Language* class (cf. Figure 1); both basic and full lists of attributes and links are illustrated. In the link list the principal (range or domain) class is added, if available in the data schema for the property; the lists are efficiently searchable, as well.

If a query has been started by a property or an individual, there is an option to fill in the class name (in the element’s property pane to the right of the diagram) from the class name suggestions created in the context of the selected node and its environment in the diagram. Figure 4 illustrates the class name suggestion in the context of an outgoing property *dbo:spokenIn*.

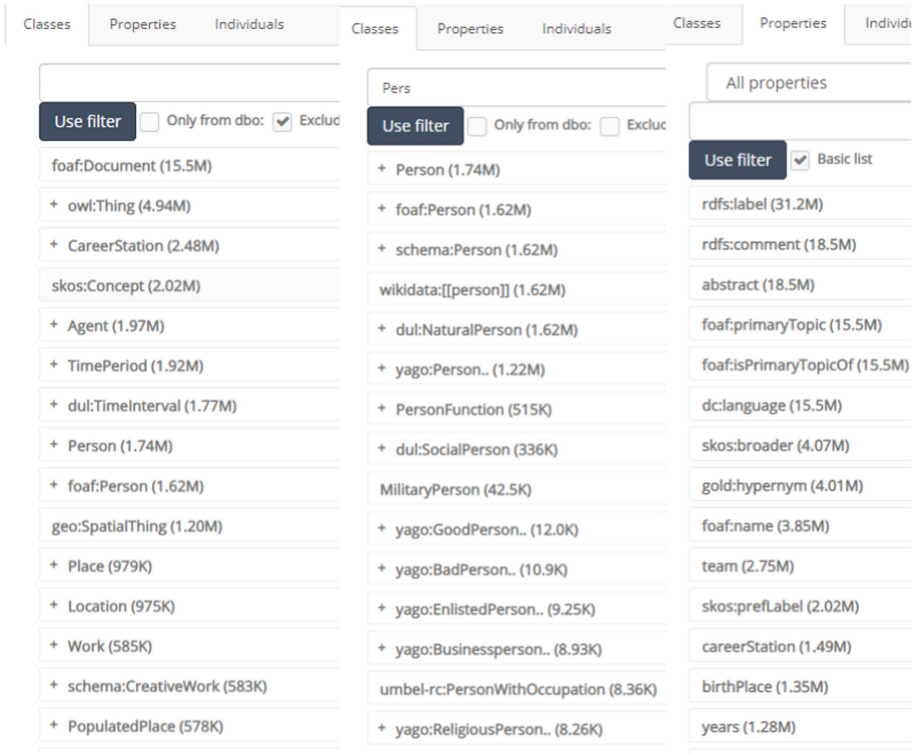


Fig. 2. Schema tree examples in the visual query tool: top classes except from *yago*: namespace, filtered classes, top properties

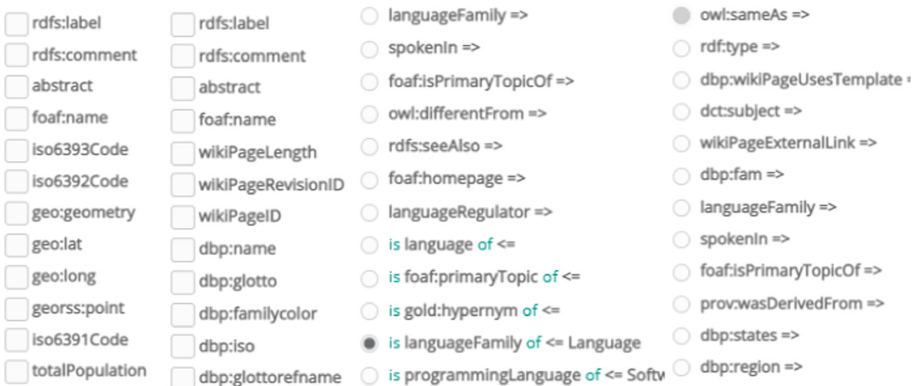


Fig. 3. Top attribute and link suggestions in the context of *dbo:Language* class and outgoing property *spokenIn*: top of basic and full attribute lists, top of basic and full link lists

The created visual environment can be used both for *Exploration* and *Querying* of the data endpoint (DBPedia).

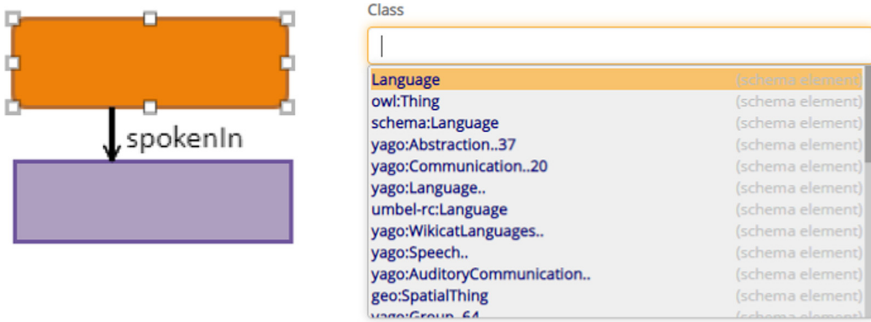


Fig. 4. Visual diagram after selection of *dbo:spokenIn* property from the initial property list, and following class name suggestion for its source class

The exploration would allow obtaining the overview of the classes and properties in the textual pane, together with their size, the subclass relation in the class tree is supported based on the subclass data retrieved from the data endpoint. The class and property lists can be filtered, so allowing to reach any of the 480 K classes and 50 K properties. For each class and property its surrounding context is available (starting from most important classes/properties), as well as the queries over the data can be made from any point reached during the exploration phase (the exploration can be used to determine the entities for further query seeding).

Within the data querying options, the environment provides the visual querying benefits (demonstrated e.g., in [5] and [11]) in the work with the data endpoint of principal importance and substantial size. The environment would allow creating all queries from e.g., the QALD-4 dataset, however, the end user experience with query creation would need to be evaluated within a future work.

6 Conclusions

We have described a method enabling auto-completion of queries based on actual class-to-property and property-to-property mappings for the DBPedia data endpoint with more than 480 thousand classes and more than 50 thousand properties by using hybrid method for accessing the stored data schema and the data endpoint itself.

The created data schema extraction process can be repeated over different versions of the DBPedia, as well as over other data endpoints, so creating query environments over the datasets that need to be explored or analyzed. The open-source code of the visual tool and the data schema server allows adding custom elements to the environment that are important for quality user interface creation over user-supplied data.

An interesting future task would be also moving the schema data (currently stored on PostgreSQL server) into an RDF triple store to enable easier sharing of endpoint data schemas as resources themselves and processing the schema data themselves by

means of visual queries and integrating them with other Linked Data resources. An issue to be addressed would be the efficiency of the schema-level queries over the data store, however, it can be conjectured that a reasonable efficiency could be achieved. The technical replacement of the PostgreSQL server by an RDF triple store (and generating SPARQL queries instead of SQL ones) is not expected to be a major challenge since the schema server architecture singles out the schema database querying module.

Acknowledgements. This work has been partially supported by a Latvian Science Council Grant lzp-2020/2-0188 “Visual Ontology-Based Queries”.

References

1. Bizer, C., et al.: “DBpedia-a crystallization point for the Web of Data” (PDF). *Web Semant. Sci. Services Agents World Wide Web* 7(3), 154–165 (2009)
2. Lehmann, J., et al.: DBpedia-a large-scale, multilingual knowledge base extracted from Wikipedia. *Semant. Web* 6(2), 167–195 (2015)
3. Vargas, H., Buil-Aranda, C., Hogan, A., López, C.: RDF Explorer: A Visual SPARQL Query Builder. In: Ghidini, C., et al. (eds.) *ISWC 2019. LNCS*, vol. 11778, pp. 647–663. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30793-6_37
4. Ferré, S.: Sparklis: an expressive query builder for SPARQL endpoints with guidance in natural language. *Semant. Web* 8, 405–418 (2017)
5. Soylu, A., Giese, M., Jimenez-Ruiz, E., Vega-Gorgojo, G., Horrocks, I.: Experiencing OptiqueVQS: a Multi-paradigm and ontology-based visual query system for end users. *Univ. Access Inf. Soc.* 15(1), 129–152 (2016)
6. Klungre, V.N., Soylu, A., Jimenez-Ruiz, E., Kharlamov, E., Giese, M.: Query extension suggestions for visual query systems through ontology projection and indexing. *N. Gener. Comput.* 37(4), 361–392 (2019). <https://doi.org/10.1007/s00354-019-00071-1>
7. Čerāns, K., et al.: ViziQuer: A Web-Based Tool for Visual Diagrammatic Queries Over RDF Data. In: Gangemi, A., et al. (eds.) *ESWC 2018. LNCS*, vol. 11155, pp. 158–163. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98192-5_30
8. Čerāns, K., et al.: Extended UML class diagram constructs for visual SPARQL queries in ViziQuer/web In *Voila!2017. CEUR Workshop Proceed.* 1947, 87–98 (2017)
9. YASGUI. <https://yasgui.triply.cc/>
10. Kapourani, B., Fotopoulou, E., Papaspyros, D., Zafeiropoulos, A., Mouzakitis, S., Koussouris, S.: Propelling SMEs Business Intelligence Through Linked Data Production and Consumption. In: Ciuciu, I., et al. (eds.) *OTM 2015. LNCS*, vol. 9416, pp. 107–116. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26138-6_14
11. Čerāns, K., et al.: ViziQuer: a Visual notation for RDF data analysis queries. In: Garoufallo, E., Sartori, F., Siatiri, R., Zervas, M. (eds.) *Metadata and Semantic Research. CCIS*, vol. 846. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14401-2_5
12. Dudáš, M., Svátek, V., Mynarz, J.: Dataset summary visualization with LODSight. In: *The 12th Extended Semantic Web Conference (ESWC2015)*
13. Čerāns, K., Ovčīņņikova, J., Bojārs, U., Grasmanis, M., Lāce, L., Romāne, A.: Schema-Backed Visual Queries over Europeana and Other Linked Data Resources. In: Verborgh, R., et al. (eds.) *ESWC 2021. LNCS*, vol. 12739, pp. 82–87. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-80418-3_15