An Intelligent SPARQL Query Builder for Exploration of Various Life-science Databases

Atsuko Yamaguchi[1], Kouji Kozaki[2], Kai Lenz[3], Hongyan Wu[1], and Norio Kobayashi[3]

[1] Database Center for Life Science (DBCLS), Research Organization of Information and Systems, 178-4-4 Wakashiba, Kashiwa, Chiba, 277-0871 Japan
{atsuko,wu}@dbcls.rois.ac.jp

[2] The Institute of Scientific and Industrial Research (ISIR), Osaka University, 8-1 Mihogaoka, Ibaraki, Osaka, 567-0047 Japan
kozaki@ei.sanken.osaka-u.ac.jp

[3] Advanced Center for Computing and Communication (ACCC), RIKEN, 2-1 Hirosawa, Wako, Saitama, 351-0198 Japan
{kai.lenz, norio.kobayashi}@riken.jp

Abstract. Database integration of a wide variety of life-science data is an important issue for comprehensive data analysis. Since Semantic Web technologies, such as Resource Description Framework (RDF), are expected to provide efficient data integration technologies, many life-science databases are published in RDF with SPARQL Protocol and RDF Query Language (SPARQL) endpoints as search application programming interfaces on the web. However, although SPARQL supports very useful functions for exploring and integrating various datasets, many biologists find SPARQL difficult to use. To overcome this problem, we propose an intelligent SPARQL query builder that aids users with no knowledge on SPARQL in building queries. This paper discusses the methods used by this tool, its system design and implementation. The tool assists users in generating queries for cross-database annotations based on RDF and enhances the value of life-science data by exploring and integrating these data.

Keywords: semantic web, SPARQL, intelligent query generation, database integration, life-science databases

1 Introduction

The high-throughput measurement apparatuses recently developed for life sciences have generated enormous and various data. Such data are stored and published in databases across all around the world. To enhance the value of such databases and acquire innovative knowledge from the extracted information, these databases must be integrated. The interoperability of heterogeneous, widely distributed biological databases has been recently improved by Semantic Web technologies. Currently, many important life-science databases provide their data in Resource Description Framework (RDF) model. RDF is a standard Semantic Web data model that is used as one of the key technologies for Linked Data. For example, UniProt [1], the largest protein database, has employed an RDF data model since 2008 to handle the many interlinks to various existing databases. Around the same time, the Bio2RDF project [2] published Linked Data originally generated from numerous major biological databases in RDF. In October 2013, an RDF platform [3] was made publicly available by the European Bioinformatics Institute (EBI). Through this platform, users can access the RDF data of six EBI databases. Data are retrieved by SPARQL Protocol and RDF Query Language (SPARQL) endpoints, a type of web application programming interface.

Under these circumstances, if life-science researchers are to easily access RDF data in Linked Data, their requirements should be accommodated by SPARQL. Since a SPARQL query construct is intractable to biologists who are unfamiliar with programming languages and RDF data schema, these researchers require informatics supports for constructing SPARQL queries. Corresponding SPARQL queries for standard requirements may be prepared in advance. Indeed, many SPARQL endpoints provide typical example queries on RDF datasets. However, because the interests of biological researchers are wide-ranging, they are not easily covered by SPARQL queries constructed in advance.

In this study, we present an intelligent tool named SPARQL Builder that assists biological researchers in building SPARQL queries. This prototype version assumes that target users have no knowledge of RDF and SPARQL, a situation that is typical of biological researchers. Peculiar to life sciences, comprehensive datasets are frequently viewed, edited and analyzed in table form, where a column of the table is associated with an RDF class as a set of instances. Therefore, our initial prototype system is designed to be compatible with TogoTable [4], an RDF-based cross-database annotation system. TogoTable implements a function that retrieves annotations from many RDF databases, which accesses up-to-date user-specified SPARQL endpoints and their corresponding SPARQL queries. Our SPARQL Buidler was designed for a SPARQL query construction tool for TogoTable before being trialed on various cases.

2 Related work

Many SPARQL based tools and semantic search methods have been proposed to date. The most popular method for instance searching is faceted search. For example, Ferré *et al.* proposed query-based faceted search (QFS) as a navigational support tool for faceted searching by logical information system query language (LISQL) [5]. QFS employs a SPARQL endpoint, enabling searching of large datasets [6].

Ferré *et al.* also developed a web based tool named Sparklis[1], which supports complex queries and exploratory searching for SPARQL endpoints. The

[1] http://www.irisa.fr/LIS/ferre/sparklis/osparklis.html

tool presents users with lists of classes in its target endpoints and allows users to make queries through faceted based graphical user interfaces (GUIs). Although the tool presents queries in a logical language format, interactive GUIs for building SPARQL queries are provided by other systems such as NITELIGHT [7], iSPARQL[2] and RDF-GL [8]. These systems build queries through users' interactive selections of the candidates of possible query specification options.

Popov proposed an exploratory search called Multi-Pivot [9]. This search method extracts concepts and relationships from the ontologies of interest to the user. The extracts are visualized and used for semantic searches among instances (data) associated with ontology terms. Kozaki *et al.* [10] also proposed a user-guided divergent ontology exploration tool. Multi-Pivot and Kozaki *et al.*'s tool are good examples of semantic searching for instances based on ontologies as conceptual structures.

Following Popov's approach, our proposed SPARQL builder is designed for quick discovery of possible paths between instances of selected classes. Although some systems are designed to accelerate RDF data retrieval [11, 12], the proposed system accelerates only the computation of possible paths among preprocessed metadata.

Currently, users can build SPARQL queries for life-science databases using GUI-based support tools, such as facet based searching. For example, BioSPARQL [13] supports paths among instances in selected classes while limiting the target to local data.

3 SPARQL Builder

SPARQL Builder is an intelligent tool by which users with no knowledge of SPARQL can generate SPARQL queries and retrieve results satisfying their requirement. In this section, we discuss a prototype version of SPARQL Builder that collaborates with TogoTable and explain the core software components of the tool, including the controller and crawler modules.

3.1 System requirement

Building a SPARQL query in TogoTable TogoTable is a web application enabling biological researchers to upload their data in a table form and add annotations obtained from SPARQL endpoints. More precisely, when a user selects one column of an uploaded table, TogoTable displays the candidate databases containing the SPARQL endpoint of the annotation search and the candidate annotation types. If a user selects one database and one annotation type in a system-prepared SPARQL query, TogoTable obtains annotations of that type from the selected endpoint and adds these annotations as a column to the user's original table. Presently, approximately 50 queries are inbuilt to TogoTable for access to the major life-science databases; however, these queries are insufficient

[2] http://oat.openlinksw.com/isparql/index.html

to satisfy the diverse interests of biological researchers. They also exclude newly published dateabases.

To cover the SPARQL endpoints of interest to diverse biologists, TogoTable has a function that sets SPARQL queries for arbitrary SPARQL endpoints. Our first goal is to support users in constructing SPARQL queries using TogoTable.

Limitation of SPARQL queries As discussed above, SPARQL implemented in TogoTable obtains an annotation for each element in the user's table. Because SPARQL itself handles lists of instances, the query outputs annotations corresponding to a list of all data elements in a user-specified input column. Since each column of a TogoTable table is an extension of instances (data) corresponding to a class, a user's specification of input and output columns corresponds to specifying input and output classes defined in a dataset.

Therefore, we assume that SPARQL queries in a TogoTable are used to search instances in an output class given some instances in an input class. Note that the instances of input and output classes may be related in multiple ways. Since classes may not be directly related, our system should display all possible relationships between the instances of input and output classes. The relational expressions between two classes are detailed in Subsection 3.3.

3.2 System overview

Figure 1 shows an overview of our system's architecture. The primary components are five modules: GUI, path finder, query constructor, crawler and controller modules.

The GUI module The SPARQL endpoint and the input and output classes are specified in a text box in the GUI module. A panel displays a rooted tree generated from the possible relationships between the input and output classes, and the system-generated SPARQL query is displayed in another text box.

The path finder module This module includes a function that computes the possible relationships between input and output classes.

The query constructor module This module includes the SPARQL query generator, which generates SPARQL queries based on the user-specified relationship between input and output classes.

The crawler module This module accelerates the path finder module by extracting metadata from the SPARQL endpoints in advance, as described in Subsection 3.4.

The controller module This module is the core module of the system that manages and integrates the activities of the other modules.

Our sytem proceeds through the following steps: 1) the user selects a SPARQL endpoint from the endpoint list containing the URLs of SPARQL endpoints preprocessed by the crawler module. If the desired SPARQL endpoint is not in the list, the user can provide the URL of the SPARQL endpoint or update the

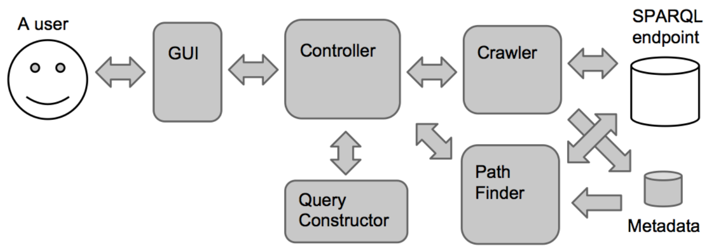


Fig. 1. Overview of the proposed SPARQL query builder.

SPARQL endpoint list at the starting process of the crawler module. 2) After specifying the SPARQL endpoint, the user is presented with a list of classes contained in the dataset of the SPARQL endpoint. From the input and output classes specified by the user, the system then finds the relationships among the start and end classes. 3) Using metadata extracted by the crawler module, the path finder module dynamically constructs a tree whose root and leaves correspond to the start and end classes, respectively. The resulting tree is displayed to the user. 4) When the user selects one leaf of the tree, representing a single relationship, the SPARQL query is constructed and displayed to the user. 5) Finally, the generated query is executed.

3.3 Generating SPARQL by browsing a class graph

To enumerate the relationships between input and output classes, we introduce a *class graph* whose nodes and edges correspond to classes and the class–class relations with predicates, respectively. Given an RDF dataset R, we denote by C the set of all classes in R. A class graph $G_R = (V, E, c, p)$ of R is a directed labeled multigraph defined as follows: V is a $|C|$-sized set of nodes and c is a one-to-one mapping from V to a set of URLs of C. E is a multiset of directed edges between the nodes of V, and p maps E to a set of URLs of predicates in R. To construct E and p from R, we add to E a directed edge e_{pred} from node n_d to n_r, where $c(n_d) = class_d$ and $c(n_r) = class_r$, and define $p(e_{pred}) = pred$ if $pred$ satisfies either of the following two conditions: (1) both the triples "*pred* rdfs:domain $class_d$" and "*pred* rdfs:range $class_r$" exist in R for some classes $class_d$ and $class_r$; (2) there exist three triples "*sub pred ob*", "*sub* rdf:type $class_d$", "*ob* rdf:type $class_r$" in R, where *sub* and *ob* are resources and $class_d$ and $class_r$ are classes.

Given a class graph G_R, we define a *class path* p from a start class *start* to an end class *end* by a sequence $(n_1, e_1, n_2, e_2, \ldots, n_m)$, where the nodes n_i and edges e_i of G_R satisfy the following conditions: (1) $c(n_1) = start$, $c(n_m) = end$, (2) $c(n_i) \neq end$ for any $i \neq m$, (3) e_i is a directed edge from n_i to n_{i+1} or from n_{i+1} to n_i, (4) if $c(n_i) = c(n_{i+2})$, $e_i \neq e_{i+1}$. An edge e_i directed from n_i to n_{i+1} or from n_{i+1} to n_i is called *forward* or *reverse* directed, respectively. Note that a class path corresponds to a SPARQL query to obtain the instances of an end class from the instances of a start class by relating a sequence of predicates $p(e_i)$. By searching the possible class paths from the start class to the end class, we can obtain the candidates of a SPARQL query that match the user's purpose.

We now explain how a SPARQL query is constructed from a class path $(n_1, e_1, n_2, e_2, \ldots, n_m)$. Basically, because a class path indicates a relationship between a start and an end classes, the WHERE clause of the SPARQL query should include "$?s_i\ p(e_i)\ ?o_i$" or "$?o_i\ p(e_i)\ ?s_i$" if the direction of e_i is forward or reverse, respectively. In addition, because s_i and o_i should be restricted to instances of classes $c(n_i)$ and $c(n_{i+1})$, the WHERE clause should also include two triples "$?s_i$ rdf:type $c(n_i)$" and "$?o_i$ rdf:type $c(n_{n+1})$" for any i. Therefore, for a class path $(n_1, e_1, n_2, e_2, \ldots, n_m)$, the following SPARQL query is constructed.

```
SELECT ?$r_m$ WHERE {
    ?$r_1$ $p(e_1)$ ?$r_2$. (or ?$r_2$ $p(e_1)$ ?$r_1$.)
    ?$r_2$ $p(e_2)$ ?$r_3$. (or ?$r_3$ $p(e_2)$ ?$r_2$.)
    ...
    ?$r_{m-1}$ $p(e_{m-1})$ ?$r_m$. (or ?$r_m$ $p(e_{m-1})$ ?$r_m$.)
    ?$r_1$ rdf:type $c(n_1)$.
    ?$r_2$ rdf:type $c(n_2)$.
    ...
    ?$r_m$ rdf:type $c(n_m)$
}
```

3.4 Preprocessing method for metadata acquisition

If the path finder module sends a SPARQL query to obtain the adjacency classes in a class graph each time the user searches a path through the GUI module of SPARQL Builder, the search time may become unacceptably long.

Therefore, we implemented the crawler module, which extracts metadata from the major SPARQL endpoints in advance. The crawler module gathers the necessary metadata to construct a class graph, namely the classes, properties, property domains and ranges, by executing low-load SPARQL queries even if the SPARQL endpoint contains large amounts of data.

Property schema generated by the crawler module To construct a class graph from an RDF dataset, we should extract the relationship between classes c_s and c_o with property p appearing as triples "$s\ p\ o$" in the RDF dataset, where s and o are instances of c_s and c_o, as explained in Subsection 3.3. We now define the *property schema* of an RDF dataset R by a quintuple $(p, c_s, c_o, I_s^p, I_o^p)$, where p is the property, c_s and c_o are classes, and sets I_s^p and I_o^p are instances s in c_s and instances o in c_o, respectively, existing as triples "$s\ p\ o$" in R. Note that each property schema corresponds to one edge of a class graph. Ideally, the property schema should be derived from explicit information written into the RDF dataset using rdfs:domain and rdfs:range. In practice, however, the property domains and ranges in RDF datasets may not be defined. Moreover, even when the domain d and range r are defined for a property p, erroneous triples “$s\ p\ o$” may exist for which the s and o classes written to the RDF dataset using rdf:type do not specify d and r.

Therefore, for a property p, there may exist multiple pairs of classes. For each property p, the crawler module infers the domain and the range of p from all triples in an RDF dataset R as follows:

1. Collect a quadruple (c_s, c_o, s, o) of classes c_s and c_o and resources s and o such that three triples “$s\ p\ o$”, “s rdf:type c_s” and “o rdf:type c_o” reside in R, and neither c_s nor c_o are defined as classes in RDF Schema 1.1 [3], such as rdfs:Resource and rdfs:Class. Then, for each pair (c_s, c_o) of classes containing (c_s, c_o, s, o) for some s and o in the collected quadruple, a property schema $(p, c_s, c_o, I_s^p, I_o^p)$ is generated by computing $I_s^p = \{s \mid (c_s, c_o, s, o) \text{ for some } o\}$ and $I_o^p = \{o \mid (c_s, c_o, s, o) \text{ for some } s\}$ for classes c_s and c_o.
2. Collect the domain classes c_d of p explicitly described as “p rdfs:domain c_d” triples and store them in a set C_d of classes. Similarly, collect the range classes c_r of p explicitly described as “p rdfs:range c_r” triples and store them in a set C_r of classes. Then, for every pair (c_d, c_r) of classes $c_d \in C_d$ and $c_r \in C_r$ generate a property schema $(p, c_d, c_r, I_d^p, I_r^p)$, where $I_d^p = \{s \mid \text{a triple “} s\ p\ o \text{” in } R \text{ for some } o\}$ and $I_r^p = \{o \mid \text{a triple “} s\ p\ o \text{” in } R \text{ for some } s\}$.

If the domain or range of p of an “$s\ p\ o$” triple is undefined in R and if the classes of s or o are not described using rdf:type, that triple is disregarded when generating the property schema, and data related to the triple cannot be retrieved by our system-generated SPARQL queries. We refer to such a triple as a *junk triple*. In addition, out system disregards classes c for which no property schema $(p, c_s, c_o, I_s^p, I_o^p)$ exists with $c_s = c$ nor $c_o = c$. Such a class is called a *junk class*.

3.5 Statistical indicators of SPARQL endpoints

The coverage of the RDF data searchable by a SPARQL Builder query depends on the comprehensiveness of the data schema decelerated in an RDF dataset. Especially, in life sciences, the comprehensiveness and numbers of entities, such as genes and proteins, can be an crucially important for data analysis. Therefore, we introduce statistical indicators called *data schema categories*, comprising *property*, *class* and *endpoint categories*, by which users can intuitively understand the statistics of their data.

Property category Let T_p be a set of triples with predicate p, where p is a property outside of rdf:type, rdfs:subClassOf, or any other properties defined in RDF Schema 1.1. The *property category* of p can be defined using T_p as follows:

property category 1 (complete): For all triples $t \in T_p$, the classes of both subject and object of t are explicitly declared using rdf:type. Moreover, both the domain and range classes of p are explicitly declared using the properties rdfs:domain and rdfs:range. Then, the property category of p is 1.

[3] http://www.w3.org/TR/rdf-schema/

property category 2 (complete by inference): When the property category of p is not 1 for all $t \in T_p$, if each subject or object class of t is defined using rdf:type or can be inferred as show below, the property category of p is 2.

- When the subject s of t is not defined using rdf:type, a domain class c_s of the predicate of t can be defined using rdfs:domain, c_s is decided as the class of s.
- When the object o of t is not defined using rdf:type, a range class c_o of the predicate of t can be defined using rdfs:range, c_o is decided as the class of o.

property category 3 (partial): When the property category of p is not 1 or 2, if there exists a triple $t \in T_p$ such that each subject or object class of t is defined using rdf:type or can be inferred as shown in category 2, the property category of p is 3.

property category 4 (none): When the property category of p is not 1, 2, or 3, it is set to 4.

Note that if a property p is category 4, all the triples in T_p are junk triples.

Class category For an RDF dataset R, the *class category* of R is an index specifying the coverage of classes in R that are not junk classes. The class category is defined as follows:

class category 1 (complete): If R contains no junk classes, it is category 1.

class category 2 (partial): When the category of R is not 1, but R contains at least one non-junk class, it is category 2.

class category 3 (none): R contains only junk classes, it is category 3.

Endpoint category For an RDF dataset R, the *endpoint category* of R is an index specifying the coverage of non-junk properties and classes. Endpoint category is defined as follows:

endpoint category 1 (complete): If R satisfies the following two conditions, its endpoint category is 1.

- Every property p in R not defined in the RDF schema is either category 1 or 2.
- The class category of R is 1.

endpoint category 3 (none): If the class category of R is 3, the endpoint category of R is 3.

endpoint category 2 (partial): If the endpoint category of R is neither 1 nor 3, it is set to 2.

Resultant RDF file As discussed earlier, the crawler obtains data schema and statistics by accessing a SPARQL endpoint. These data are stored as RDF files in our SPARQL Builder server and allow the path finder module to rapidly find relationships among classes. The RDF files are written in standardized vocabularies including SPARQL 1.1 Service Description [4] and VoID [5], with our original vocabularies to include classes describing the relationships among subject–object classes extracted from triples.

4 Results

4.1 Statistical analysis of data preprocessed by the crawler module

To evaluate the performance of the crawler module of SPARQL Builder, we selected five self-maintained large-scale databases accessed by the most cutting-edge biological researches as SPARQL endpoints; namely, Expression Atlas [6], BioModels [7], BioSamples [8], ChEMBL [9] and Reactome [10]. As described in Subsection 3.4, the property schemata of SPARQL endpoints are computed by the crawler module. As shown in Table 1, the crawler module successfully obtained the property schemata of the five SPARQL endpoints valid as of July 2014.

Because all five of the SPARQL endpoints are classified as endpoint category 2, part of the RDF data provided in the endpoints can be retrieved by the inbuilt SPARQL queries. For a SPARQL endpoint in endpoint category 2, the RDF data coverage can be evaluated in more detail by the resultant property categories.

The numbers of the properties in property category 2, in which complete class–class relationships are inferred from subject–object classes of triples, are worthy of special mention. The inferred class–class relationships cannot be dynamically retrieved from the SPARQL Builder GUI within a practical length of time, since all triples of the corresponding properties must be analyzed by executing SPARQL queries. As shown in the "property categories" column of Table 1, the crawler successfully inferred the class–class relationships for 55% of the properties (378 out of 693 properties) obtained from the five endpoints. These inferred relationships account for 68% of all triples (488,539,717 out of 710,506,582 triples) in the five RDF databases. Each property requires 48 seconds of runtime, which is unacceptably long for an interactive system.

4.2 Application example

As mentioned above, the SPARQL Builder was originally designed to obtain new annotations from SPARQL endpoints via TogoTable. Therefore, we connected

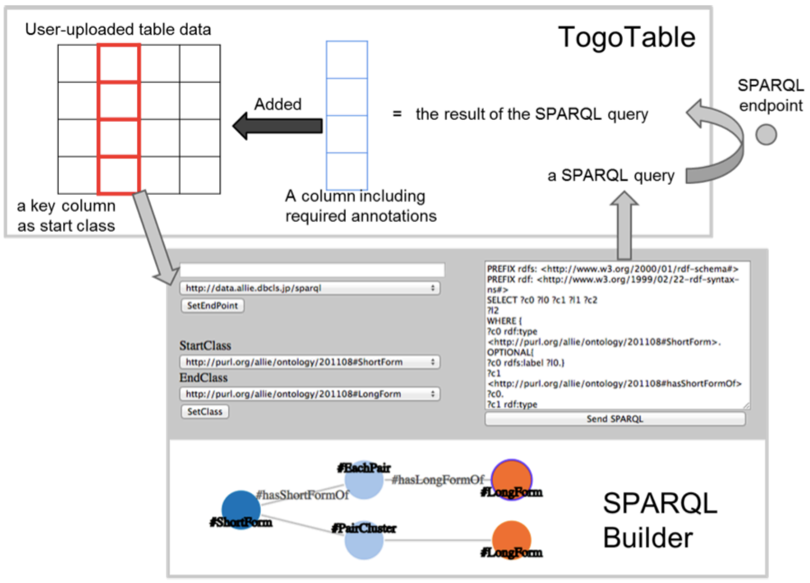
[4] http://www.w3.org/TR/sparql11-service-description/
[5] http://www.w3.org/TR/void/
[6] http://www.ebi.ac.uk/rdf/services/atlas/sparql
[7] http://www.ebi.ac.uk/rdf/services/biomodels/sparql
[8] http://www.ebi.ac.uk/rdf/services/biosamples/sparql
[9] http://www.ebi.ac.uk/rdf/services/chembl/sparql
[10] http://www.ebi.ac.uk/rdf/services/reactome/sparql

SPARQL endpoint	#access	#class and datatype	property categories #property and triple	endpoint category	execution time [s]
Atlas	18,975	1,834 7	[1] 2 (157,636/157,636) [2] 145 (297,243,685/297,243,685) [3] 14 (1,229,512/1,246,354) [4] 9 (0/4,618)	2	22,930
BioModels	2,596	32 5	[1] 44 (12,723,813/12,723,813) [2] 15 (1,065,313/1,065,313) [3] 6 (2,598/351,850) [4] 40 (0/1,470,635)	2	2,916
BioSamples	5,611	575 6	[1] 6 (4,799,275/4,799,275) [2] 121 (73,153,500/73,153,500) [3] 25 (91,319/91,966) [4] 19 (0/7,699)	2	11,433
ChEMBL	3,816	126 7	[1] 57 (157,767,439/157,767,439) [2] 76 (112,819,134/112,819,134) [3] 11 (13,949,839/13,963,278) [4] 40 (0/23,615,337)	2	19,919
Reactome	1,186	46 4	[1] 33 (5,766,953/5,766,953) [2] 21 (4,258,085/4,258,085) [3] 0 (0/0) [4] 9 (0/12)	2	1,259

Table 1. Statistics of EBI SPARQL endpoints analyzed by a crawler. #access is the number of querying accesses to the endpoints; #class and datatype denote the numbers of classes and datatypes, respectively. Property categories show the numbers of properties and triples for each property category in the notation [c] n (s/t), c is the property category, n is the number of properties in the category, t is the number of triples in the category, and s is the number of non-junk triples in the category.

SPARQL Builder and TogoTable, and evaluated whether users could extract their desired annotations from arbitrary RDF databases.

Figure 2 shows how the SPARQL Builder is connected to TogoTale. The TogoTable contains the user-uploaded table data, a start class corresponding to a key column of the table data, and the URL of the SPARQL endpoint of a database containing the user's desired annotations. All this information is sent to the SPARQL builder. Based on the start class and URL of the SPARQL endpoint, the SPARQL Builder displays candidate end classes. If the user selects an end class, SPARQL Builder shows the possible paths from the start class to the end class. If the user selects a path, a SPARQL query is generated and sent to TogoTable. TogoTable then adds a new column containing the desired annotations returned by the query. SPARQL Builder will be supported in the next release of TogoTable.


User-uploaded table data
TogoTable
Added
= the result of the SPARQL query
SPARQL endpoint
a SPARQL query
a key column as start class
A column including required annotations
http://data.allie.dbcls.jp/sparql
SetEndPoint
StartClass
http://purl.org/allie/ontology/201108#ShortForm
EndClass
http://purl.org/allie/ontology/201108#LongForm
SetClass
Send SPARQL
#EachPair
#hasLongFormOf
#LongForm
#hasShortFormOf
#ShortForm
#PairCluster
#LongForm
SPARQL Builder

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#>
SELECT ?c0 ?l0 ?c1 ?l1 ?c2
?l2
WHERE {
?c0 rdf:type
<http://purl.org/allie/ontology/201108#ShortForm>.
OPTIONAL{
?c0 rdfs:label ?l0.}
?c1
<http://purl.org/allie/ontology/201108#hasShortFormOf>
?c0.
?c1 rdf:type
```

Fig. 2. Connection between TogoTable and SPARQL Builder.

5 Discussion and Conclusion

Using the SPARQL Builder a user can discover a sequentially connected triple path to an arbitrary SPARQL endpoint. An intuitive GUI specifies the class–class relationships as data schema of the path. To obtain these data schema, users access the corresponding SPARQL endpoint directly via the GUI or by a crawler that obtains comprehensive data schema in advance and stores the results in an RDF file. Although the RDF file is designed to rapidly find the user's data schema, it also includes the essences of the data semantics as a profile of the corresponding SPARQL endpoint. Life-science data are especially large-scale and comprehensive and are stored in diverse classes such as genes and phenotypes. The crawler result files will assist biologists to understand data semantics as a general data retrieval tool.

The SPARQL endpoints of EBI and similar sources have begun to replicate manually drawn data schema, enabling users to intutively program their SPARQL queries. However, we observed that the output data schema generated by the crawler module is comprehensive but cannot emphasize important structures. The search result of the queries generated by the SPARQL builder is also imperfect, as the query may yield no solutions. This occurs because our tool investigates only the relationships between classes connected by a property. Such class–class relationships cannot prove the existence of a sequentially connected instance (triple path). To ensure that the generated queries always discover solutions, the crawler needs to traverse all possible triple paths of two or more steps, which is not practically feasible for arbitrary SPARQL endpoints.

Our method provides the best compromise between intelligent exploration of a SPARQL endpoint and the convenience of generating SPARQL queries.

In future, we hope that our tool will support queries for discovering class–class relationships, such as ontology terms. We also hope to support additional structures such as blank nodes, class inferences and the OWL vocabulary.

Acknowledgments. This work was supported by JSPS KAKENHI Grant Number 25280081 and the National Bioscience Database Center (NBDC) of the Japan Science and Technology Agency (JST).

References

1. The UniProt Consortium: Reorganizing the protein space at the Universal Protein Resource (UniProt). Nucl. Acids Res. 40(D1), D71–D75 (2012)
2. Belleau, F., Nolin, M. A., Tourigny, N., Rigault, P., Morissette J. Bio2RDF: towards a mashup to build bioinformatics knowledge systems. J. Biomed. Inform. 41(5), 706–716 (2008)
3. Jupp, S., Malone, J., Bolleman, J., Brandizi, M., Davies, M., Garcia, L., Gaulton A., Gehant, S., Laibe, C., Redaschi, N., Wimalaratne, S. M., Martin, M., Le Novére, N., Parkinson, H., Birney, E., Jenkinson, A. M.: The EBI RDF platform: linked open data for the life sciences. Bioinformatics 30(9), 1338–1339 (2014)
4. Kawano, S., Watanabe, T., Mizuguchi, S., Araki, N., Katayama, T., Yamaguchi, A.: TogoTable: cross-database annotation system using the Resource Description Framework (RDF) data model. Nucl. Acids Res. 42(W1), W442–W448 (2014)
5. Ferré, S., Hermann, A.: Reconciling faceted search and query languages for the semantic web. IJMSO 7(1), 37–54 (2012)
6. Guyonvarch, J., Ferré S.: Scalewelis: a scalable query-based faceted search elena work. Multilingual Question Answering over Linked Data (QALD-3), Valencia, Spain
7. Russell, A., Smart, P. R., Braines, D., Shadbolt, N. R.: NITELIGHT: a graphical tool for semantic query construction. Semantic Web User Interaction Workshop (SWUI 2008), Florence, Italy
8. Hogenboom, F., Milea, V., Frasincar, F., Kaymak, U.: RDF-GL: a SPARQL-based graphical query language for RDF. In Chbeir, R., Badr, Y., Abraham, A., Hassanien, A-E. (eds.) Emergent Web Intelligence: Advanced Information Retrieval, 87–116, Springer, London (2010)
9. Popov, I. O., Schraefel, M.C. Hall, W., Shadbolt, N.: Connecting the dots: a multi-pivot approach to data exploration. International Semantic Web Conference (ISWC 2011), LNCS7031, 553–568 (2011)
10. Kozaki, K., Hirota, T., Mizoguchi R.: Understanding an ontology through divergent exploration. Extended Semantic Web Conference (ESWC2011), 305–320, Heraklion, Greece
11. Li, F., Le, W., Duan, S., Kementsietsidis, A.: Scalable keyword search on large RDF data. IEEE Transactions on Knowledge and Data Engineering, doi:10.1109/TKDE.2014.2302294 (2014)
12. Tran, T., Ladwig, G., Rudolph, S.: Managing structured and semistructured RDF data using structure indexes. IEEE Transactions on Knowledge and Data Engineering, 25(9), 2076–2089 (2013)
13. Kobayashi, N., Toyoda, T.: BioSPARQL: ontology-based smart building of SPARQL queries for biological linked open data. SWAT4LS, 47–49, London, UK