Assisted Composition of Linked Data Queries

Imen Sarray[a] and Aziz Salah[b]
Computer Science Department, Université du Québec à Montréal, 201 President Kennedy Avenue, Montreal, Canada

Keywords: Linked Data, RDF, SPARQL Queries, Semantic Web, Schema Construction, Resource Clustering, Query GUI Tool.

Abstract: Much research has been undertaken to facilitate the construction of SPARQL queries, while other research has attempted to facilitate the construction of the RDF dataset schema to understand the structure of RDF datasets. However, there is no effective approach that brings together these two complementary objectives. This work is an effort in this direction. We propose an approach that allows assisted SPARQL query composition. Linked data interrogation is not only difficult because it requires mastering a query language such as SPARQL, but mainly because RDF datasets do not have an explicit schema as what you can expect in relational databases. This paper provides two complimentary solutions: synthesis of an interrogation-oriented schema and a form-based RDF Query construction tool, name EXPLO-RDF.

1 INTRODUCTION

An increasing number of RDF datasets is available on the Web for users and their applications. A key challenge for the users to reuse these data is in exploring, querying and understanding the large and unfamiliar RDF sources.

Today, the SPARQL query language is almost the de facto tool for RDF data queries and exploration. However, the formulation of SPARQL queries is a complex task. Indeed, the user should know the syntax of SPARQL and requires technical knowledge and some understanding of RDF, RDFS and URIs, among others. Moreover, RDF data is not only intended for the Semantic Web community, but also for non-expert users and experts of other fields who are not necessarily familiar with the different technologies used. Subsequently, it becomes very difficult for these users to query and explore an RDF dataset with SPARQL.

Before an RDF dataset can be reused and in order to write a SPARQL query, the user must understand the data and must have information about the RDF dataset schema to locate the relevant information for their specific needs and to determine whether such data can be easily reused.

Users currently face the problem that schema information for RDF data is often not available or even missing. Even when it is available, it tends to be incomplete or does not adequately represent the RDF data because the latter does not have to conform to a constraining schema. It can therefore be hard for users to obtain the big picture when handling a large and complex RDF dataset.

The lack of schema can limit the interrogation of RDF linked data: for example, writing a query without knowing of the existing classes and their properties (known as predicates) is not straightforward. In this case, the user must first submit multiple queries and manually browse the results in order to collect all the relevant classes and properties, which will be used to formulate the main query that will provide the final result.

In recent years, some work has been done to provide the user with exploration approaches of RDF schema. Thus, the user will build a global view of the RDF source and can select relevant classes and their properties. The user must use a query language like SPARQL or data exploration tools to explore and get more details about the data.

There is also work to explore directly RDF data sources. These works help users to query and understand RDF data without needing to know the

[a] https://orcid.org/2222-3333-4444-5555
[b] https://orcid.org/0000-1111-2222-3333

Sarray, I. and Salah, A.
Assisted Composition of Linked Data Queries.
DOI: 10.5220/0008169601850194
In *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2019)*, pages 185-194
ISBN: 978-989-758-382-7

query language. To query a dataset, the user needs a minimum of knowledge about the data structure. However, this information is often not available.

To the best of our knowledge, no approach allows the exploration of the schema on the one hand and data on the other, although these two types of exploration are complementary and depend on each other. In this context, we propose EXPLO-RDF, a tool for assisted composition of SPARQL queries. This tool offers two complementary approaches: schema construction and form-based query composition. The two approaches together allow a user to understand the content of an RDF source as quickly as possible and to better fulfil their needs of RDF dataset interrogation.

The reminder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes how to extract and visualize schema information from RDF dataset, based on a number of SPARQL queries. The visualisation will use UML class diagram extended with some statistics to better understand the profile of the RDF dataset. Section 4 describes our approach for form-based query composition featuring keyword search and completion suggestions, a simple and intuitive way to RDF dataset interrogation. Section 5 presents and reports our experimental results.

2 PRELIMINARIES AND RELATED WORK

RDF [1] (Ressource Description Framework) is a standard data model that represents the spinal cord of the semantic web. An RDF dataset is composed of triples in the form (*subject*, *predicate*, *object*), representing statements, example (<Bill> <name> ”Bill Gates”.) The *predicate* is a property representing a relationship between the *subject* (a resource) and the *object* (either a resource or a literal). Resources and *predicates* are identified with URIs (Uniform Resource Identifier). An RDF dataset is a directed graph where *subjects* and *objects* are the nodes, connected with directed edges representing the *predicates*.

RDF is very flexible as it accepts any triple respecting its syntax. RDFS which stands for RDF Schema is an extension of RDF that allows the definition the vocabulary to be used in an RDF dataset. RDFS defines `rdfs:Class`, rdfs:range etc, but are not used in most published RDF datasets. An important feature in RDF is declaring the type of resources (eg. <Bill> rdf:type <Person>.) which allows to easily construction the schema of the RDF dataset. This schema is similar to the schema of relational databases. Resources for which type declarations are missing are untyped resources. Inferring the schema of an RDF dataset with untyped resources is a challenging task. The schema is very important for RDF dataset query because it provides a summary of the RDF dataset that reveals it structure.

SPARQL[2] represents for RDF datasets what SQL is for relational databases. A SPARQL query is mainly composed of triple patterns in which the three components *subject*, *predicate* and *object* are either variables or closed constants (i.e. URIs or literals). The SPARQL engine executes the query by matching its triple patterns with the triples of the RDF graph. Each match can be part of the query response.

Related work to RDF dataset querying concerns two tasks: schema exploration and construction, and RDF dataset interrogation. We also discuss their limitations.

2.1 RDF Schema Construction

Visualization of the RDF dataset schema can help get a better overview of the data structure and may be a useful starting point for queries and further analysis. There are two main methods for constructing the schema of an RDF dataset: graph-based model, and UML class diagram based.

2.1.1 Graph based Model

Several approaches allow the graph-based schema construction, such as: *LD-VOWL* (Weise, Lohmann, & Haag, 2016), *LODSight* (Dudáš, Svátek, & Mynarz, 2015) and *LODeX* (Benedetti, Bergamaschi, & Po, 2014).

LD-VOWL is 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 RDF data which is then gradually added to an interactive VOWL graph visualization (Weise, Lohmann, & Haag, 2016).

LODSight is a dataset summary visualization tool. It uses also SPARQL to find all type-property and datatype-property paths in the dataset. The RDF schema will be represented as a graph and visualized in a format allowing the user to see the generalized structure of the dataset. The visualization is

[1] https://www.w3.org/TR/rdf11-primer/

[2] https://www.w3.org/TR/sparql11-query/

interactive, and the user can also filter the displayed paths to show only those with lower or higher frequency (Dudáš, Svátek, & Mynarz, 2015).

LODeX summaries the schema as a graph where the vertices are the classes. An undirected non-tagged edge relates two classes if there exists a property linking at least two of their respective instances. LODeX provides also the average number of a property or an attribute in the instances of a given class. It offers a visual query editor where the user can add/remove filters.

The schema summary relies on RDF schema, meaning that it assumes all instances are linked to their classes in the dataset. This assumption is not true for most existing RDF datasets, where the type declaration is often missing. Thus, the data source sometimes does not follow an explicit schema and, in this case, the tool does not really support such RDF datasets. In addition, we could not see the properties that link two classes directly from the graph, because we have to select one class at a time to display its properties.

LODSight, LD-VOWL, and LODeX do not handle untyped resources. Subsequently, the schema of the dataset is not complete. Then, these tools would only show a schema composed of typed resources only, which provides a better understanding of the data structure covering typed resources only. However, they do not cover all RDF data in the dataset. Finally, with the exception of LOdDeX, most of these tools do not offer the possibility to manipulate SPARQL queries.

2.1.2 UML Class Diagram based Schema

There is surprisingly few work on extracting and visualizing schema information from linked data as a UML diagram, principally (Li & Zhang, 2013) and (Jin-Sung & Mi-Kyung, 2005).

(Li & Zhang, 2013) proposes a SPARQL-based tool that, given an RDF dataset, it builds a data inferred schema represented as a UML class diagram, extended with a collection of statistics. The number of instances per class and the number of instances of each property allow users to better understand the RDF dataset.

This type of exploration makes it possible to present the schema in a visual way that we judge to be the best for understanding the data structure. However, it does not provide the means to write effective queries and to explore the data locally in the RDF graph. The authors mentioned the problem of processing untyped instances but did not propose a well-explained solution.

Figure 1: Graph pattern builder (Auer, et al., 2007).

2.2 Query Construction

We can classify SPARQL query construction tools into three main categories according to their approaches: semantic browsers based, forms based and visual composition based query.

2.2.1 Semantic Browsers based Query

Semantic browsers provide a GUI that implicitly supports query composition in a text editor while the user is navigating the RDF dataset.

As a tree-based semantic browser, Tabulator (Tim Berners-Lee & Sheets, 2016) displays an increasing level of refinement as the user navigates the tree structure. It features assisted SPARQL query composition and editing, supporting simple and complex queries as well.

Query composition based on semantic browser is designed to be easy and fast to learn for new users and for developers who would like to expand their own ideas about a dataset. However, before exploring the RDF dataset and formulating queries, the user needs to understand the structure of the dataset to determine the different links.

The user needs to explore an entire tree, which can be a tedious task and expensive in time and effort. On the other hand, if the user needs a specific property/resource, he has to navigate through the entire data tree every time in order to select what he needs; a non-simple task since RDF data can be large and eventually the path may be too long. Having a schema of the RDF dataset would have made things easier for users.

2.2.2 Form-based Query

This is a popular approach where queries are created from the elements of a form such as text fields, drop-down lists, etc. Examples of this approach are *SPARQL Viz* (Jethro, 2006), *Konduit VQB* (Ambrus, Möller, & Handschuh, 2010) and *Graph Pattern Builder* (Auer, et al., 2007). Graph Pattern Builder (Figure 1) was developed specifically to query Wikipedia data. Users query the knowledge base through a basic graph that consists of a set of triple patterns where each one is composed three form fields. These fields represent the subject, predicate and object of a triple pattern. Each field can receive either a variable, an identifier or a filter.

Form based query is an effective approach for data exploration and for SPARQL query composition. However, the user must first understand the structure of the RDF dataset: the classes and their optional/mandatory properties. Such information, that represents the schema of the dataset, is generally not easily grasped with these methods of data exploration. Therefore, the user will be forced to try to understand and collect the schema pieces manually, a task not easy and not obvious for the user as the schema is not fixed, as recourses may be untyped and have optional properties.

2.2.3 Visual Query Construction

A visual query construction tool, such as NITELIGHT (Smart, Russell, & Braines, 2008), *RDF-GL* (Hogenboom, Milea, Frasincar, & Uzay, 2010) and *LUPOSDATE* (Groppe, Groppe, & Schleifer, 2011), defines a visual language in which a query is represented with a graph that the tool translate into a SPARQL query.

Visual query construction can cover most of the expressiveness of SPARQL while maintaining an intuitive simplicity. However, the same problem persists, since the user has no idea about the schema nor links between different classes, he will be forced at first to understand the structure of RDF dataset by making SPARQL queries. A tedious and repetitive task.

In this section, we have pointed out several limitations of the presented approaches. The main limitation common to all these approaches, regardless of the category to which they belong, is the fact that no one combines the RDF schema construct and the RDF data query composition, although they are complementary and dependent on one another. Their complementarity is a consequence of the flexibility of RDF that does not impose a fixed schema, which makes it difficult to query RDF datasets.

3 RDF DATASET SCHEMA CONSTRUCTION

Our goal is to identify classes and their properties in an RDF graph in order to construct a model which summarises the RDF dataset. We call this summary a schema[3] and its main purpose is query construction not dataset design.

The identification of the classes in an RDF graph can be easy if its vocabulary was properly declared using RDFS. Each resource would have a class to which it belongs using `rdf:type` property.

However, the specification of the class of a resource is not mandatory in the RDF model. Consequently, real world RDF dataset may show untyped resources, the ones that are without any `rdf:type` property. Untyped resources make it difficult to identify their classes to be represented in the schema summarizing the RDF dataset. That's why we'll be talking about a group of a resource rather than the class of a resource. Of course, declared classes are groups of ressources. Other groups will be defined by clustering untyped resources.

We can formulate the problem of identifying resource clustering as follows: Given an RDF graph, group identification is the discovery of resource groups, not necessarily disjoint, each representing a set of resources having at least a property in common. The set of properties of the resources of a group define the group's properties. A resource may have more than one type and subsequently may belong to more than one group, making groups not necessarily disjoint.

To illustrate our approach, we consider throughout this section a small RDF graph (Figure 2), adapted from Jamendo RDF dataset (Raimond, 2016), which describes artists and their recordings.

3.1 Groups of Typed Resources

For typed resources, their groups are defined by their classes. The groups and their properties are determined using the following SPARQL query:

[3] Calling schema this model is inspired by relational DB and should not be confused with RDFS (RDF Schema) by W3C.

```
SELECT distinct ?class ?prop
WHERE {
  ?ressource  rdf:type  ?class .
  OPTIONAL { ?ressource ?prop ?objet } .
  Filter (?prop != rdf:type).
}
```

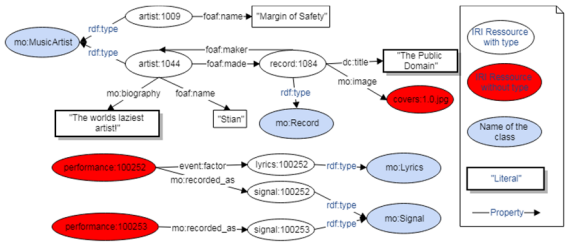


Figure 2: An RDF graph adapted from Jamendo dataset.

Applied on the RDF graph in Figure 2, the former SPARQL query returns the results shown in Table 1. There are four classes (`mo:MusicArtist`, `mo:Record`, `mo:Lyrics` and `mo:Signal`) each one will represent a group. For example, `mo:MusicArtist` is a group having two resources (`artist:1009` and `artist:1044`), and characterized with three optional properties (`mo:biography`, `foaf:made` and `foaf:name`.)

Table 1: Typed resource groups and their properties.

class	prop
mo:MusicArtist	mo:biography
mo:MusicArtist	foaf:made
mo:MusicArtist	foaf:name
mo:Record	dc:title
mo:Record	mo:image
mo:Record	foaf:maker
mo:Lyrics	
mo:Signal	

3.2 Groups of Untyped Resources

Assigning untyped resources to groups is not a straightforward task. The resources of a group should share a similarity from schema point of view. To solve this problem, we studied many alternatives. A simple solution would be to group all untyped resources in a single group. The group of untyped resource would be composed of heterogeneous unrelated resources. This basic representation does not help the user to understand the structure of the RDF graph.

On the other extremum, another basic solution, would be to create a group for each property in untyped resources. In this case, a resource with many properties will be assigned to many groups. Consequently, it would lead to an explosion of the number of groups which is against the principle of constructing a schema, summarizing the RDF dataset.

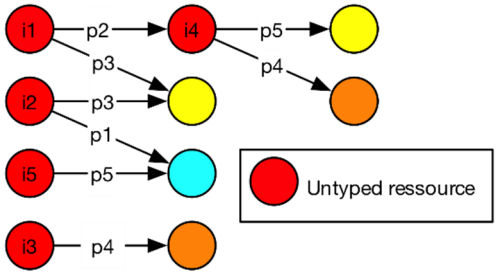


Figure 3: Example of an RDF graph with untyped resources.

By avoiding the drawbacks of the two previous trials, the chosen solution is to group untyped resources based on their common properties. Two untyped instances belong to the same group if they have at least one property in common. In order to capture business domain groups in the RDF dataset, properties such as `rdfs:label`, `rdfs:comment`, etc are excluded.

Using a binary relation R, we formalize the extraction of the groups of untyped resources in an RDF graph. Two instances i and j are R-related if and only if $prop(i) \cap prop(j) \neq \emptyset$, where prop(i) represents the set of properties of resource i. A property p belongs to prop(i) if and only if there exists in the dataset an RDF triple having i as its subject and p as its property (i.e. predicate).

Figure 3 shows an RDF graph having five untyped resources. For this dataset, we draw in Table 2 a matrix presenting the binary relation R.

We define the relation R* to be the transitive closure of R. The transitive closure R* is the smallest transitive binary relation that contains R. The binary relation R* is an equivalence relation by construction and its equivalence classes are exactly the groups of untyped resources in the RDF dataset and can be obtained by computed the transitive closure of R.

In the case of R in Table 2, there are two equivalence classes {i1, i2} and {i3, i4, i5} in R*. These two classes represent the groups of untyped instances in the RDF graph. Since groups are equivalence classes, they are necessarily disjoint. This disjointness was not targeted, but it is a good feature resulted from our formalization of the clustering problem. We characterize each group by the properties of their respective untyped resources.

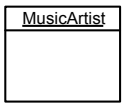
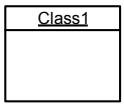
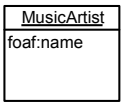
Table 2: A matrix representation of the relation R based on the RDF graph in Figure 3.

Ressources	i1	i2	i3	i4	i5
i1	1	1	0	0	0
i2	1	1	0	0	0
i3	0	0	1	1	0
i4	0	0	1	1	1
i5	0	0	0	1	1

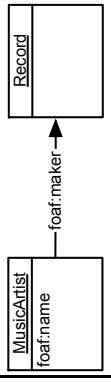
The class diagram graphical representation enables visual exploration of the RDF dataset structure and facilitates query construction.

Table 3 defines and illustrates our detailed translation rules for constructing the UML class diagram from the RDF dataset and the groups of typed and untyped resources.

Table 3: Translation rules for constructing the UML class diagram from RDF data.

Scenarios	UML Mapping
UML Classes: Groups of typed instances	A UML class whose name is the type specified in the RDF source.
Example 	`artist:1044 rdf:type mo:MusicArtist .` MusicArtist is a UML class
UML Classes: Groups of untyped instances	A UML class whose name will be Class 1 in which 1 is representing the group index. These classes will be represented with another color to distinguish them from the UML classes representing typed resource groups.
Example 	`performance:100252 mo:recorded_as signal:100252. performance:100252` belongs to group1 presented by Class1.
UML Attributes: The object is a literal	The property targeting this object will be an attribute in the UML class representing the subject of an RDF triple.
Example 	`artist:1044 rdf:type mo:MusicArtist . artist:1044 foaf:name "Stian" .` As `"Stian"` is a literal, so `foaf:name` property will be an attribute in class MusicArtist.
UML Association: The object is a resource	The property targeting this object will be an association relating two classes.

Example




```
artist:1044 rdf:type
                mo:MusicArtist .
artist:1044 foaf:maker
                    record:1084 .
record:1084 rdf:type
                    mo:Record .
```

As `record:1084` is a resource then the class `MusicArtist` and `Record` are related with an association labeled with property `foaf:maker`.

3.2.1 Extending the Class Diagram with Statistics

We add statistics to the schema such as the number of resources in each group and the number of resources that are the subjects of a property. These statistics are computed through the following SPARQL queries:

Query1 :

```
SELECT  ?C, count (distinct ?i) as ?count
WHERE { ?r rdf:type ?C . }
```

Query2 :

```
SELECT  ?C,?p, count(distinct ?r) as ?count
WHERE { ?r ?p ?o .
        ?r rdf:type ?C.
        ?o rdf:type ?C1. }
GROUP BY  ?C,?p
```

It is obvious that Query1 and Query2 computes statistics for typed resource groups. However, a simple trick allowed us using the same queries for untyped resource groups as well. The trick consists of inserting in the RDF dataset fictive triples to make each untyped resource typed with its group.

Statistics allow the user to have an idea about the structure of the RDF dataset. Since resources within the same group may not have the same properties, statistics may inform whether a property is optional by comparing the number of resources in a group with the number of participating resources in the property. If they are the same, all of the resources of the group participate in the property otherwise, the property is optional. SPARQL queries can be refined in case of optional properties using the OPTIONAL graph pattern, for example.

3.2.2 Example of Application

We have applied our approach for RDF dataset schema construction on Jamendo, an RDF dataset, as an example of application. After clustering typed and untyped resources into groups, we constructed the

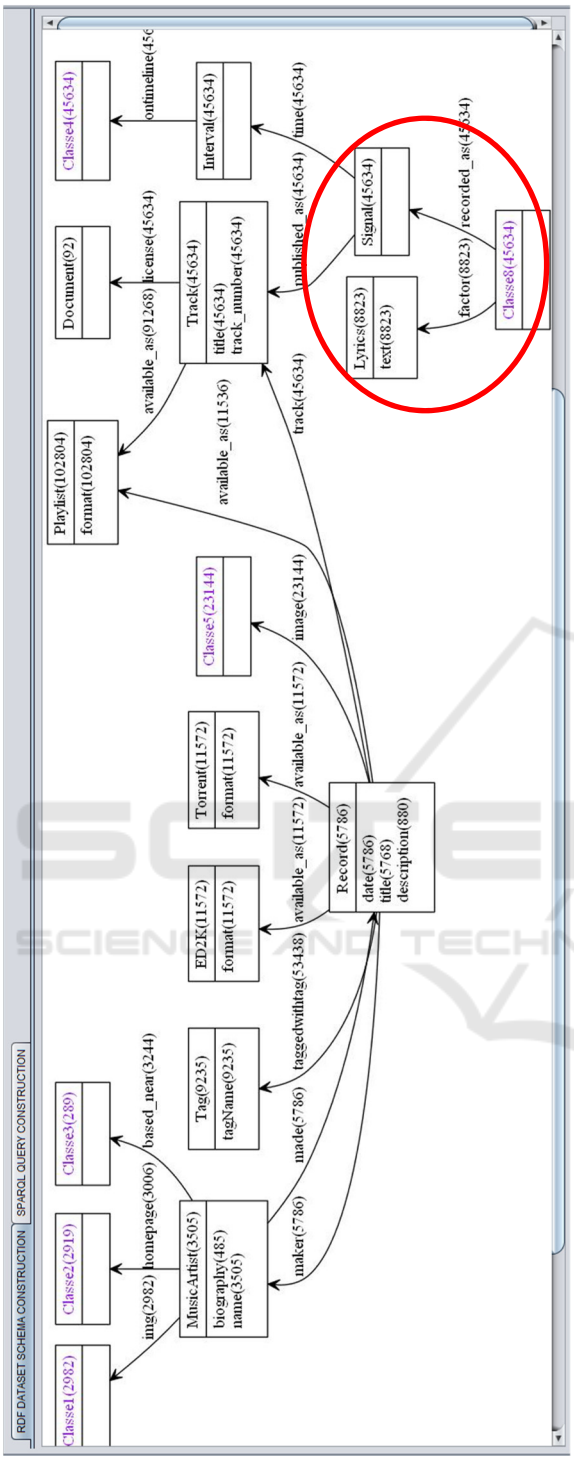


Figure 4: Jamendo RDF dataset schema.

dataset schema showed in Figure 4 by applying the transformation rules explained in Table 3.

The schema contains 17 classes of which 11 classes represent typed resource groups and 6 classes represent untyped resource groups, obtained using the equivalence relation R*. For each class, the number of instances is provided as well as the number of instances for each property of each class. This diagram is convenient for users since it allows to explore and understand the structure of the RDF dataset in a fast, intuitive, efficient and visual way which facilitates the formulation of appropriate SPARQL queries and the retrieval of interesting detailed information.

4 GUI BASED SPARQL QUERY CONSTRUCTION

Our approach to construct SPARQL queries is guided by the RDF dataset schema and uses two friendly GUI based features: keyword search and form-based query construction. Keyword search is an easy way to explore the neighbourhood of a resource in the RDF graph in order to assist the user in constructing SPARQL queries by filling triple patterns slots.

The approach consists of three tasks: indexing the RDF dataset, searching RDF data by keywords and assisting in the construction of SPARQL queries.

4.1 Indexing the RDF Dataset and Keyword Search

The RDF dataset is composed of triples that the SPARQL engine try to match with triple patterns in a SPARQL query. A triple pattern in the WHERE clause has three slots, one for the subject, the property and the object. In Figure 5, landmark B pinpoints triple pattern slots in the GUI of our prototype, EXPLO-RDF. Each slot in a triple pattern can receive as input either a variable (when it starts with '?'), a URI or a literal. A literal in the triple pattern triggers a keyword search and retrieves relevant matching values in the position of its slot from the RDF dataset. For example, if there is a literal in the subject slot of the triple pattern, only information about subjects in the RDF dataset are looked into to retrieve a matching candidate list. The user can select from the retrieved list the value she wants, either a literal or a URI to build a triple pattern to add to the query under construction.

Consequently, we built three indices: subject index, property index and object index. Information from `rdfs:label` and `rdfs:comment` triples are also indexed with their resources and can be used to retrieve relevant resources in keyword search.

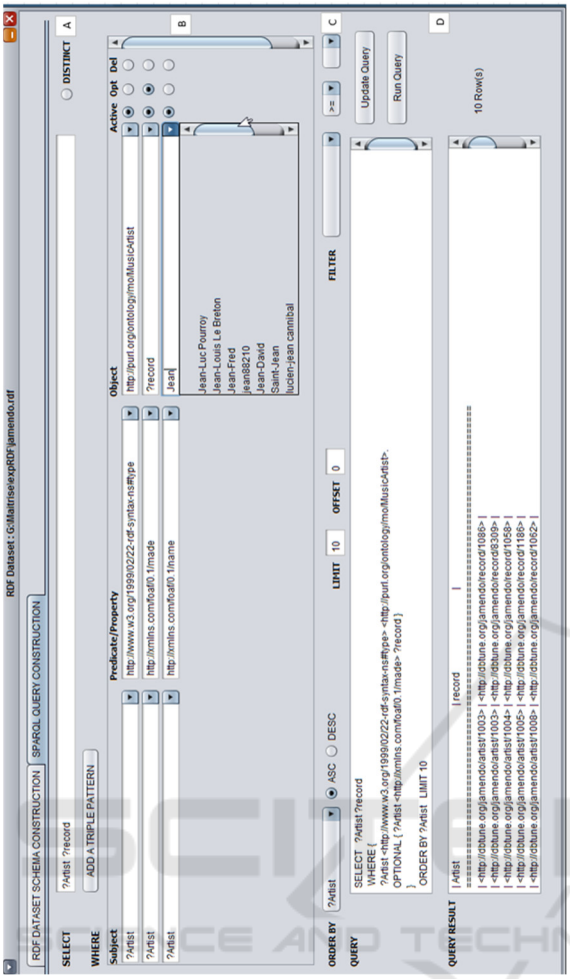


Figure 5: GUI of EXPLO-RDF, our prototype.

4.2 SPARQL Query

The GUI of EXPLO-RDF allows SPARQL query assisted composition. The WHERE clause (Figure 5:B) is constructed as the triplet patterns are filled. If the user indicates that a triple pattern is optional then this information will be added for this triple in the query. The select clause is constructed based on the list of user-defined variables. The "distinct" option can be activated to remove duplicates from the results. LIMIT, OFFSET and ORDER BY clauses (Figure 5:C) can also be added to the query. Triple patterns can be activated/deactivated (with radio buttons “Active” in Figure 5:B) to allow the exploration a neighbourhood in the RDF graph.

```
475531  <rdf:Description rdf:about="http://dbtune.org/jamendo/performance/100252">
475532    <event:factor rdf:resource="http://dbtune.org/jamendo/lyrics/100252"/>
475533    <mo:recorded_as rdf:resource="http://dbtune.org/jamendo/signal/100252"/>
475534  </rdf:Description>
475535
475536  <rdf:Description rdf:about="http://dbtune.org/jamendo/performance/100253">
475537    <mo:recorded_as rdf:resource="http://dbtune.org/jamendo/signal/100253"/>
475538  </rdf:Description>
```

Figure 6: Excerpt from Jamendo RDF dataset.

Once a SPARQL query is composed through the GUI, it is translated into a text format and showed in the editor text box. The user can eventually create and update SPARQL queries manually in the editor text box. EXPLO-RDF submits queries to the SPARQL engine and displays the results.

5 VALIDATION AND DISCUSSION

The EXPLO-RDF (Figure 5) supports two tasks. The first one is the automatic construction of the schema summarizing the RDF dataset under use, in the form of an extended UML class diagram (Figure 4). The second task consists of assisting the user in composing SPARQL queries based on triple pattern form GUI. The schema is very important for query construction as it provides the user with a blue print of the RDF dataset which inspires a first version of the query. When executed, the first version query may return no results. This is very common because of the flexibility of RDF. The user then studies the triple patterns of the first version using EXPLO-RDF forms and incrementally fine-tunes his query to make it right for extracting targeted information from the RDF dataset. Fine-tuning triple patterns uses keyword search and neighbourhood exploration within the RDF graph.

In summary the schema provides a global view of the RDF dataset while the triple pattern form GUI in EXPLO-RDF compensates with local views for the construction of effective SPARQL queries.

5.1 Validation

To validate our schema construction, we converted a relational database into an RDF dataset and constructed its schema. Then we compared the RDF dataset schema we have constructed with EXPLO-RDF with the relational database schema. It was a perfect match.

In our approach, resource groups (Figure 4) can be compared to the classes in the UML class diagram by (Li & Zhang, 2013) modelling a schema for Jamendo RDF dataset. Their schema shows 18 classes of which 7 are anonymous classes grouping untyped resources. EXPLO-RDF, our tool identified 17 classes (Figure 4) of which 16 classes match perfectly the schema of (Li & Zhang, 2013). The only difference (red-circled classes in Figure 4) was the group of untyped class8 in which EXPLO-RDF

gathered the resources participating in properties `mo:recorded_as` and `event:factors` because there exists a resource having both properties as shown by the excerpt from Jamendo RDF dataset in Figure 6.
In (Li & Zhang, 2013) schema there was two anonymous classes one for `mo:recorded_as` and the other for `event:factor`. The fact that EXPLO-RDF produces less untyped resource groups (i.e. anonymous classes) in its schemas than (Li & Zhang, 2013), make our schemas more concise and consequently easier to grasp and comprehend for users.

5.2 Complementarity of Form-based Query Construction and RDF Dataset Schema

The final goal of a user is to formulate queries. On the one hand, with a tool that just provides an RDF dataset schema, the user will be forced to manually create queries, a tedious time-consuming task that requires expertise. In addition, these queries could return empty result because of optional properties. Our prototype EXPLO-RDF with its form-based query construction offers the possibility to explore data in detail that helps to check the RDF data structure and validate the class diagram as it is a valuable summary of the RDF graph.

On the other hand, in the absence of the schema, the user will have to get an idea about the schema manually to understand the RDF data structure. A simple method is to explore the RDF dataset through simple queries using pattern triples in the form-based query construction GUI. In this case, the user will look for typed resources and their classes and explore their neighbourhood to determine their properties. The user is in fact unconsciously trying to build a schema for the RDF dataset. Providing the user with a well-constructed schema simplifies her task and saves him time and effort.

6 CONCLUSION

In this article, we've presented an approach that allows assisted SPARQL query composition. Our main contribution is to combine two approaches, namely the construction of a schema that summarises the structure of the RDF dataset, and a form-based query construction tool, supporting keyword search and neighbourhood exploration. Our experiments showed the relevance and the complementarity of the two tasks.

We project the extension of work into three axes: implementation environment, usability and schema design. Although EXPLO-RDF can be used to build queries, the user has to install GraphViz and some Java libraries for the SPARQL engine. It would be more convenient if EXPLO-RDF could be used as a web application. Currently, EXPLO-RDF support only RDF dumps, an extension to support SPARQL endpoints will make other RDF data sources easily usable.

On the usability axis, feedback from users is needed in order to improve EXPLO-RDF GUI and its features to meet their expectations. For example, it would be possible to rank keyword search completion list according to retrieval information metrics such as TF-IDF.

EXPLO-RDF builds a schema for the RDF dataset under query. Such schema is constructed for the purpose of querying only and it is a useful summary of the RDF dataset. Reengineering the RDF dataset in order to create real RDF schema can start from EXPLO-RDF schema. The question would be: how to break down an untyped resource group to obtain real world classes?

REFERENCES

Ambrus, O., Möller, K., & Handschuh, S. (2010). Konduit VQB: A visual query builder for SPARQL on the social semantic desktop. *Proceedings of the Workshop on Visual Interfaces to the Social and Semantic Web(VISSW 2010)*, (pp. 1-5).

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Zachary, I. (2007). DBpedia: A Nucleus for a Web of Open Data. *ISCW*, pp. 722–735.

Beek, W., & Folmer, E. (2017). An Integrated Approach for Linked Data Browsing. *International Society for Photogrammetry and Remote sensing*, pp. 35-38.

Benedetti, F., Bergamaschi, S., & Po, L. (2014). A Visual Summary for Linked Open Data sources. *In Proceedings of the ISWC 2014 Posters & Demonstrations, 15*(24), pp. 173-176.

Dudáš, M., Svátek, V., & Mynarz, J. (2015, 5 9). Dataset Summary Visualization with LODSight. *The Semantic Web: ESWC 2015 Satellite Events, 15*(20), pp. 36-40.

Groppe, J., Groppe, S., & Schleifer, A. (2011). Visual Query System for Analyzing Social Semantic Web. *International World Wide Web Conference Committee*, pp. 217-222.

Hitzler, P., Krotzsch, M., & Rudolph, S. (2011). *Foundations of semantic web technologies.* Chapman and Hall/CRC.

Hogenboom, F., Milea, V., Frasincar, F., & Uzay, K. (2010). RDF-GL: A SPARQL-based graphical. *Emergent Web Intelligence: Advanced Information Retrieval*, pp. 87–116.

Jethro, B. (2006). *Graphical query composition and natural language processing in an RDF visualization.* Erasmus University Rotterdam.

Jin-Sung, K., & Mi-Kyung, L. (2005). Object Modeling of RDF Schema for Converting UML Class Diagram. *In Proceedings of the International Conference on Computational Science and Its Applications*, pp. 31-41.

Li, H., & Zhang, X. (2013, Mai 2). Visualizing RDF Data Profile with UML Diagram. *Semantic Web and Web Science*, pp. 273-285.

Raimond, Y. (2016, July 30). *DBTune.org Jamendo RDF Server*. Retrieved from Datahub: https://old.datahub.io/dataset/jamendo-dbtune

Smart, P., Russell, A., & Braines, D. (2008). A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer. *EKAW 2008*, pp. 275 – 291.

Tim Berners-Lee, Y. C., & Sheets, D. (2016). Tabulator: Exploring and Analyzing linked data on the Semantic Web. *In 3rd Int. Semantic Web User*.

Weise, M., Lohmann, S., & Haag, F. (2016). LD-VOWL: Extracting and Visualizing Schema Information for Linked Data. *International Conference on Knowledge Engineering and Knowledge Management*, pp. 120-127.