Visualizing RDF Data Profile with UML Diagram

Conference Paper · March 2013

DOI: 10.1007/978-1-4614-6880-6_24

2 authors:

Huiying Li
Southeast University (China)
11 PUBLICATIONS **75** CITATIONS
SEE PROFILE

Xiang Zhang
Southeast University (China)
33 PUBLICATIONS **218** CITATIONS
SEE PROFILE

Visualizing RDF Data Profile with UML Diagram

Huiying Li and Xiang Zhang

Abstract Various RDF data have been published in the Semantic Web. A key challenge for users to reuse these data is in understanding the large and unfamiliar RDF data, especially when the data schema is absent or when different schemas are mixed. In this paper, we describe a tool that can construct the actual schema, gather corresponding statistics, and can present a visual, UML diagram for RDF data sources, such as SPARQL endpoints or RDF dumps. Moreover, for datasets with complex structure, the tool can mine the class subsumptions and then provide a concise visualization that would serve as a big picture for users. The experimental results compare our approach and ExpLOD using seven datasets (including DBpedia) from the Linked Data cloud. The performance evaluations show that our approach is more efficient than ExpLOD. The concise visualization experiments on complex datasets such as DBpedia show that our approach is feasible.

1 Introduction

In recent years, the Web is being increasingly extended with more RDF data sources and links between objects. The Linking Open Data community project is promoting the emergence of linked open data (LOD) [1] and is fostering the availability of many open RDF datasets, such as DBpedia (extracted from Wikipedia), SwetoDBLP, and LinkedMDB.

The abundance of RDF data brings users with many opportunities to reuse these data. Before an RDF data can be reused, the user must understand the data and then determine whether such data can be easily reused. When handling a large and complex RDF data, users cannot easily obtain the big picture if the RDF dataset uses multiple ontologies or if the ontology cannot be obtained. Facing this challenge,

H. Li (✉) • X. Zhang
School of Computer Science and Engineering, Southeast University, Nanjing 210096, P.R.China
e-mail: huiyingli@seu.edu.cn; x.zhang@seu.edu.cn

firstly, we propose a SPARQL query-based approach for RDF data profiling. For users, data profiling is a cardinal activity when facing an unfamiliar dataset. This process helps in assessing the importance of the dataset as a whole, in finding out whether the dataset or part of the dataset can be easily reused, in improving the user's ability to query or search the dataset, and in detecting irregularities for improving data quality. Then, we visualize the RDF data profile using UML diagram, since UML is the most accepted software engineering standard.

In this paper, we consider the data structure and five kinds of descriptive statistics then visualize them using UML diagram. The contributions of our work are as follows: (1) We propose an approach to obtain the actual schema of a SPARQL endpoint or RDF dump that can be considered as a customized schema for the RDF data, to gather corresponding statistics and to present a UML-based visualization for users; (2) we present a way to deal with large and complex RDF data. An association rule mining algorithm is used to determine the class subsumptions in RDF data. Based on the class subsumptions, a concise visualization is constructed to provide users a big picture of the dataset, and (3) we compare the performance of the proposed approach and ExpLOD which is similar to our approach. The evaluations show that our approach is feasible and more efficient than ExpLOD.

The rest of this article is organized as follows: Sect. 2 introduces the related work. Section 3 introduces the SPARQL-based approach for constructing and visualizing RDF data profiling. Section 4 presents the steps to construct a concise visualization based on the class subsumptions. Section 5 details the experimental results of our approach. Section 6 concludes the study.

2 Related Work

Describing and understanding large RDF data is enabled by statistics and summary. Recently, structural summaries have been proposed in the context of RDF data in [2]. Before this work, some similar work has been carried out for XML data. These prior works have shown the usability of XML path summaries for a variety of scenarios within semi-structured XML data. But most of these works must rely on the tree nature of XML data or on acyclicity assumptions that do not hold for RDF data.

Recently, some researchers are dealing with RDF data statistics and summary, such as semantic sitemaps [3], RDFStats [4], and SCOVO vocabulary [5]. Among these works, ExpLOD [6] is the most similar work to our approach. It is a tool that supports constructing summaries of RDF usage based on the bisimulation contraction mechanism. Although ExpLOD enables coarse granularity summarization based on the hierarchical bisimulation label, it faces difficulties in handling summarizations grouped by high hierarchy labels, such as namespace, which could be too general sometimes. Moreover, ExpLOD does not care about the relation between instance blocks, which is very important in understanding the data structure.

Compared with ExpLOD, our approach is distinct in three aspects. Firstly, the proposed approach considers the class type instead of predicate usage to divide instances into different blocks. It avoids the situation whereby the block number becomes too large. Secondly, our approach not only considers the data structure but also places importance on the statistics such that users can understand the RDF data. Thirdly, for large and complex datasets, our approach mines the class subsumptions and provides a concise visualization for users.

3 Visualizing RDF Data Profile

The RDF graph can be considered as a set of RDF triples. A triple consists of a subject, a predicate, and an object. Figure 1 shows the RDF graph of a sample snippet of RDF data. The snippet gives the information about two music artists, their names, the records they have made, and the record titles. Such a small snippet uses four ontologies, namely, the FOAF ontology (with prefix label $foaf$), the Dublin Core ontology (with prefix label dc), the Music Ontology (with prefix label mo), and the RDF meta-ontology (with prefix label rdf). When dealing with a very large RDF dataset, users cannot easily understand the data if the ontology is absent. Common profiling methods and tools assume a starting point of relational data with a domain-specific schema. This assumption does not hold for Semantic Web data published on the web.

We consider the data structure and five kinds of descriptive statistics for RDF data profiling. *Data structure* contains the classes and their properties, respectively. *Class instantiation* is the number of distinct instances that are typed as a particular

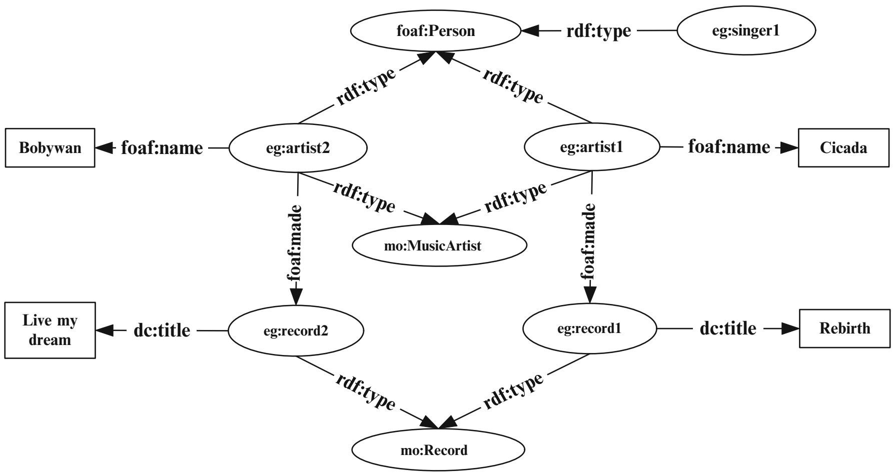

Fig. 1 RDF data snippet

class. *Property instantiation* is the number of distinct triple (s, p, o) for an object property p from class C_1 to C_2, where the type of s is C_1 and the type of o is C_2. For a datatype property p of class C, its instantiation is the number of distinct triple (s, p, o) where the type of s is C. *Multiplicity of object* denotes the multiplicity of object for an object property p from class C_1 to class C_2; four kinds of multiplicity are used: [0..1] means that every instance I_1 typed as C_1 has at most one relation p to instance I_2 typed as C_2. [0..∗] means that every instance I_1 typed as C_1 has zero or more relation p to instances I_2 typed as C_2. [1..∗] means that every instance I_1 typed as C_1 has at least one relation p to instance I_2 typed as C_2. [1] means that every instance I_1 typed as C_1 has one and only one relation p to instance I_2 typed as C_2. The datatype property can also denote the multiplicity of object. *Functional property* denotes whether a property is functional. An object property p from class C_1 to class C_2 is called a functional property if every instance I_1 typed as C_1 has a unique p value I_2 typed as C_2, i.e., there cannot be two distinct instances I_2 and I_3 typed as C_2 such that there are two triples (I_1, p, I_2) and (I_1, p, I_3). The definition of "functional" datatype property is similar. *Inverse-functional property* denotes whether a property is inverse-functional. An object property p from class C_1 to class C_2 is called an inverse-functional property if a value I_1 typed as C_2 can only be the value of p for a single instance typed as C_1, i.e., there cannot be two distinct instances I_2 and I_3 typed as C_1 such that there are two triples (I_2, p, I_1) and (I_3, p, I_1). The definition of "inverse-functional" datatype property is similar. Note that the *functional property* and *inverse-functional property* in our paper are a little different to OWL functional and inverse-functional properties. We emphasize the domain and range classes of the functional property, while the OWL functional property just requests that it has at most one value for any particular instance. The inverse-functional property is similar.

To compute the RDF data structure, we initially collect all classes used in the RDF data. In RDF data, many classes are not declared as $owl : Class$ or $rdfs : Class$ explicitly. We use the following SPARQL query to obtain all the named classes: "$SELECT\ distinct\ ?c\ WHERE\ \{?s\ rdf : type\ ?c.\}$". The classes with rdf, $rdfs$, and owl namespaces are considered as metaschema-level classes and are not collected (except for $rdfs : Seq$ because many blank nodes are declared to be $rdfs : Seq$). With these named classes, instance in RDF data can be classified into a set of named class according to their type. For each class, we explore all its instances to collect their used properties (including relations and attributes). For example, Fig. 2a queries the used properties of class $mo : Record$. Moreover, for the object property, we determine the range classes by the SPARQL query in Fig. 2b.

Some instances are without type declaration despite many typed instances. In solving this problem, an intuitive solution is to gather all the non-typed instances into one unnamed class. In providing clearer information for users, the second solution is to divide these non-typed instances into different property restriction classes according to their property restrictions. We collect the properties with non-typed

a

```
SELECT  distinct ?prop
WHERE{
            ?s rdf:type mo:Record.
            ?s ?prop ?o.
          }
```

Query property usage

b

```
SELECT  distinct ?rangeclass
WHERE{
          ?s rdf:type mo:Record.
          ?s mo:available_as?o.
          OPTIONAL{?o rdf:type ?rangeclass.}
          }
```

Query range classes

Fig. 2 Query property usage and range classes

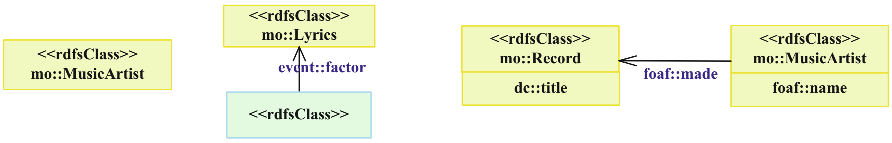

Fig. 3 RDF snippet and UML diagrams

instances as subject or object in advance. Then, we use SPARQL query to divide the non-typed instances. In the Jamendo data, the unnamed class is divided into seven restriction classes, as shown in Fig. 4. With the SPARQL queries above, the RDF data structure is constructed. Then, obtaining statistics such as class instantiation and property instantiation using SPARQL becomes easy. Moreover, the statistics of multiplicity of object and functional property are computed.

Since we have induced the actual schema and gather the corresponding statistics, visualizing them for users becomes an important issue. We adopt the notations of properties and classes proposed by [7] as well as the Ontology Definition Metamodel [8]. Figure 3 demonstrates the UML diagrams for classes and properties.

We visualize the data profile of RDF data Jamendo. Figure 4 shows the structure of, and the corresponding statistics on, Jamendo, including 18 classes, of which 7 are restriction classes. For each class, the number of instances belongs to it and the percent of all instances is provided. For every property, the instantiation and the multiplicity of object are provided, and whether the property is functional (denoted by f) or inverse-functional (denoted by inf) is also provided. Such a UML diagram provides the data structure and corresponding statistics for users. It offers convenience for users to understand the RDF data behind a SPARQL endpoint, to decide whether the data can be reused, to construct suitable SPARQL query, and to retrieve interesting detailed information.

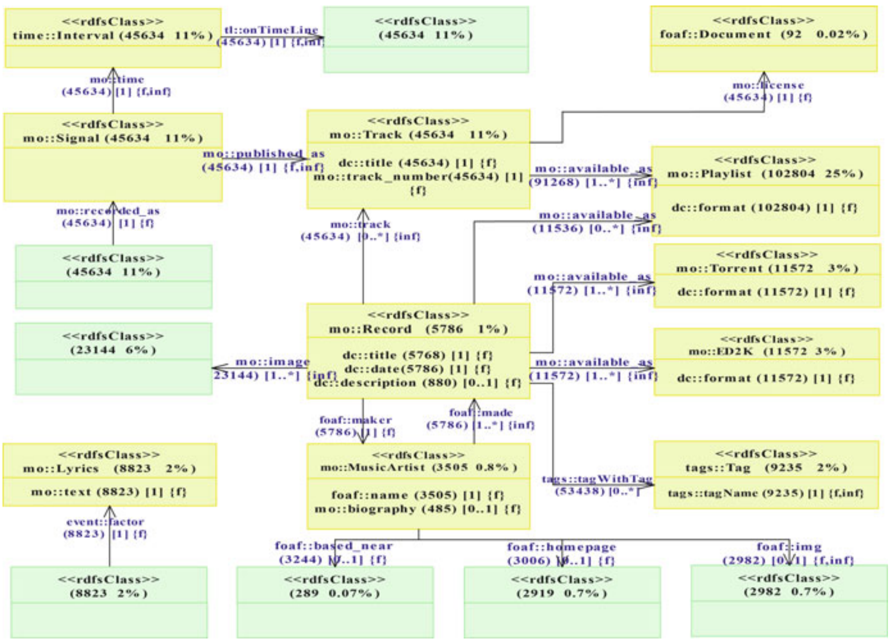

Fig. 4 Visualizing Jamendo data with UML diagram

4 Constructing Concise Visualization

When an SPARQL endpoint has too many classes, visualizing the entire set of classes is inefficient, and meanwhile, it will be difficult for users to understand the visualization. To reduce the number of restriction classes, we gather all the non-typed instances into one unnamed class. However, many classes may persist even after doing this procedure. For example, DBpedia has 326 named classes. We introduce an approach to construct a concise visualization for the dataset with a complex structure. The key problem is to select the classes and properties for constructing the concise visualization. For selecting classes, our purpose is to select a small number of classes that can cover all classes in an RDF dataset, so the class subsumptions are considered. Since the ontology is absent, we have to discover hidden class subsumptions from RDF data. We found that there are many instances typed as different classes in RDF data. So the association rule mining approach is used to extract the class subsumptions.

Association rule mining is a popular and widely researched method for discovering interesting relations between variables in large databases. Considering a transaction table, the rows represent instances, and the columns represent class types. A value of 1 in field (i, j) indicates that instance i is of type j. Association rule mining algorithm can be used to mine the class subsumptions of a large RDF dataset. After creating the transaction table locally using SPARQL query, the classic algorithm Apriori [9] is used to learn the subclass association rule. Given the class set $\mathbb{C}$ (columns in the transaction table) and the instance set $\mathbb{I}$ (rows in the transaction table), $\mathbb{X}$ is a subset of $\mathbb{C}$. The support $supp(\mathbb{X})$ is defined as the number of instance that is of $rdf : type$ every $C \in \mathbb{X}$. The Apriori finds the association rules $C_i \Longrightarrow C_j$ with high confidence value, where C_i and C_j are both classes. The confidence of an association rule is defined as follows:

$$conf(C_i \Longrightarrow C_j) = \frac{supp(\{C_i, C_j\})}{supp(\{C_i\})}$$

This formula provides evidence for the validity of the subclass relation between C_i and C_j because most instances that are of $rdf : type\ C_i$ are also of $rdf : type\ C_j$. Take the RDF data in Fig. 1 for example, the two instances declared as $mo : MusicArtist$ are also of $rdf : type\ foaf : Person$. The Apriori algorithm can mine the database and show that class $mo : MusicArtist$ is the subclass of $foaf : Person$, with a confidence value of 1. While the Apriori algorithm can also mine that class $foaf : Person$ is the subclass of $mo : MusicArtist$, with a confidence value of $\frac{2}{3}$. Given a set $\mathbb{C}$ of all named classes in an RDF dataset and a nonnegative threshold τ, the mining results of algorithm Apriori can be considered as a relation $\mathbb{R}$ on $\mathbb{C}$, where $\mathbb{R} = \{< C_i, C_j > | C_i \in \mathbb{C} \wedge C_j \in \mathbb{C} \wedge conf(C_i \Longrightarrow C_j) \geq \tau\}$; $\mathbb{R}$ is called the ***mining relation***. If $C_i \neq C_j$, $< C_i, C_j > \in \mathbb{R}$, and $< C_j, C_i > \in \mathbb{R}$, then C_i is called same as to C_j, and C_j is put into set $\mathbb{C}_{\mathbb{S}}$.

Definition 1. (***Subclass Relation***) Given the ***mining relation*** $\mathbb{R}$ on $\mathbb{C}$. The set $\mathbb{C}' = \mathbb{C} - \mathbb{C}_{\mathbb{S}}$, $\mathbb{R}'$ is a relation on $\mathbb{C}'$, where $\mathbb{R}' = \{< C_m, C_n > | < C_m, C_n > \in \mathbb{R} \wedge C_m \in \mathbb{C}' \wedge C_n \in \mathbb{C}'\}$. $rt(\mathbb{R}')$ is the reflexive and transitive closure of $\mathbb{R}'$. The relation $rt(\mathbb{R}')$ is called the ***subclass relation***, and it is denoted as $\mathbb{R}_{\mathrm{subC}}$.

Theorem 1. *The **subclass relation** is **reflexive**, **asymmetric**, and **transitive**.*

Proof. Given the ***subclass relation***, $\mathbb{R}_{\mathrm{subC}} = rt(\mathbb{R}')$, where $\mathbb{R}'$ is a relation on $\mathbb{C}'$. The ***subclass relation*** is the reflexive and transitive closure of $\mathbb{R}'$; therefore, it is **reflexive** and **transitive**. $\mathbb{R}'$ is **asymmetric**; therefore, $\mathbb{R}' = \mathbb{R}'^{-1}$. Because $(\mathbb{R}'^2)^{-1} = (\mathbb{R}' \circ \mathbb{R}')^{-1} = \mathbb{R}'^{-1} \circ \mathbb{R}'^{-1} = \mathbb{R}' \circ \mathbb{R}' = \mathbb{R}'^2$, thus $\mathbb{R}'^2$ is **asymmetric**. Hence, $\mathbb{R}'^n$ is **asymmetric**, where $|\mathbb{C}'| = n$. $rt(\mathbb{R}') = \triangle \cup \mathbb{R}' \cup \mathbb{R}'^2 \ldots \mathbb{R}'^n$, where $\triangle$ is the diagonal relation on $\mathbb{C}'$. Hence, $rt(\mathbb{R}')$ is also **asymmetric**. Therefore, the ***subclass relation*** is **reflexive**, **asymmetric**, and **transitive**. ■

Therefore, the ***subclass relation*** is a *partial order*.

Definition 2. (***Maximal Class***) Given the ***subclass relation*** $\mathbb{R}_{\mathrm{subC}}$ on set $\mathbb{C}$, $C_{\max}$ is the ***maximal class*** if $C_{\max} \in \mathbb{C}$ and there is no $C_i \in \mathbb{C}$ such that $C_{\max} \neq C_i$ and $< C_{\max}, C_i > \in \mathbb{R}_{\mathrm{subC}}$.

Definition 3. (***Minimal Class***) Given the ***subclass relation*** $\mathbb{R}_{\mathrm{subC}}$ on set $\mathbb{C}$, $C_{\min}$ is the ***minimal class*** if $C_{\min} \in \mathbb{C}$ and there is no $C_i \in \mathbb{C}$ such that $C_{\min} \neq C_i$ and $< C_i, C_{\min} > \in \mathbb{R}_{\mathrm{subC}}$.

Definition 4. (***Class Level***) Given the ***subclass relation*** $\mathbb{R}_{\text{subC}}$ on set $\mathbb{C}$, the level of ***minimal class*** is 0. The level of class C_i is $max + 1$; max is the maximum level of all C_j where $< C_j, C_i > \in \mathbb{R}_{\text{subC}}$.

In constructing a concise visualization, we select all maximal classes to represent the classes in an RDF dataset. Given an RDF dataset, $\mathbb{C}$ is the class set, and $\mathbb{C}_m$ is the maximal class set. A partition of set $\mathbb{C}$ can be computed based on the maximal classes. For every maximal class, all its subclasses that are not contained in other blocks construct a partition block. Certainly, set $\mathbb{C}$ can have many different partitions. To obtain the partition with well-distributed element number in every block, we use a heuristic method. Firstly, the maximal class with the lowest level is selected to construct a block; then, the maximal class and selected classes are removed. The process goes on until there is no class left to be selected. Therefore, a class partition is constructed based on the maximal classes; every class belongs to one and only one partition block. Every class belongs to one block because for every class C_i, there exits at least one maximal class C_m that $< C_i, C_m > \in \mathbb{R}_{\text{subC}}$. Every class belongs to only one block because we only select the class which is not selected by other blocks when constructing block.

After selecting the maximal classes to construct the concise visualization, another problem is to select the representative properties for every maximal class. We consider two kinds of representative properties: the one is the most popular property, and the other is the most distinctive property.

Definition 5. (***Most Popular Property***) Given the class C, set $\mathbb{P}$ contains all the properties of C. The ***most popular property*** of C is the property P_{pop} with the largest instantiation in $\mathbb{P}$.

Definition 6. (***Most Distinctive Property***) Given the class C, set $\mathbb{P}$ contains all the properties of C. The ***most distinctive property*** of C is the property P_{dist} with the smallest number of classes that also have the property P_{dist}.

In the concise visualization, we can select the most popular property (datatype property and object property) or the most distinctive property (datatype property and object property) for every maximal class. For the object property, if it has more than one range class, we only consider the one that has the most number of instances. Figure 5 shows the concise visualization of DBpedia data with the most distinctive property for every maximal class. Among the 326 named classes, only 35 are maximal classes. The integer after the $\ll rdfsClass \gg$ stereotype denotes the number of subclasses covered by this maximal class. The visualization is concise enough for users to grasp the big picture of the large and complex RDF dataset. If the user wants to know more about one or several classes, she can also use the approach mentioned in Sect. 3 to get the detail statistics.

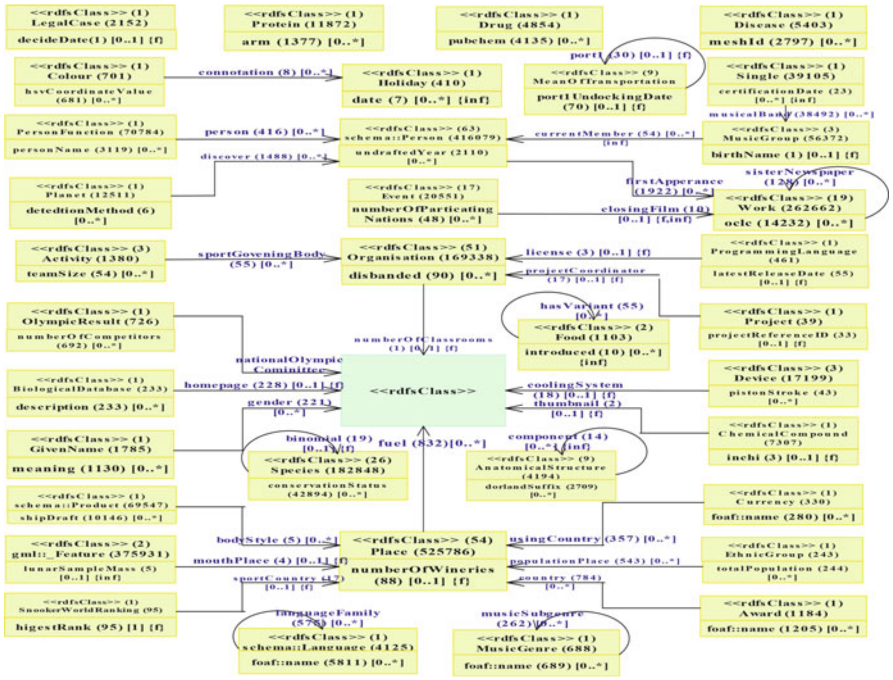

Fig. 5 Concise visualization of DBpedia data with the most distinctive property (default namespace is **dbpedia**)

5 Experimental Study

In this section, we compare the performance of our approach with that of ExpLOD. ExpLOD is a tool that supports SPARQL-based summary creation of RDF usage. We use the Jena toolkit (jena.sourceforge.net) to manage the RDF data for ExpLOD and for our approach. All the experiments were developed within the Eclipse environment and on a 64bit ThinkStation with 3.10 MHz and 16 GB of RAM (of which 14 GB was assigned to the JVM).

We chose seven datasets, which are shown in the Table 1. These datasets are selected from the LOD cloud because they vary in the amount and type of information they describe. Data Peel and Jamendo contain information on music artists and their productions. Data GeoSpecies contains information on biological orders, families, species, as well as species occurrence records and related data. Data LinkedCT contains information on linked clinical trials. Data LinkMDB contains linked data on movies. Data SwetoDblp includes information about affiliations, universities, publications, and publishers. Data DBpedia contains extracted data from Wikipedia. We concentrate on the infobox subset of DBpedia 3.7. Table 1 shows the following information about each dataset: the name, the number of triples it contains, as well as the number of properties, classes, and instances.

Table 1 Statistics of RDF datasets

Dataset	# of classes	# of properties	# of instances	# of triples
Peel	9	25	76,894	271,369
Jamendo	11	26	410,893	1,047,950
GeoSpecies	44	168	184,931	2,076,380
LinkedCT	13	90	1,169,905	9,804,652
LinkedMDB	53	222	1,326,001	6,147,995
SwetoDblp	10	145	2,394,479	13,041,580
DBpedia	326	1,378	1,839,009	26,988,054

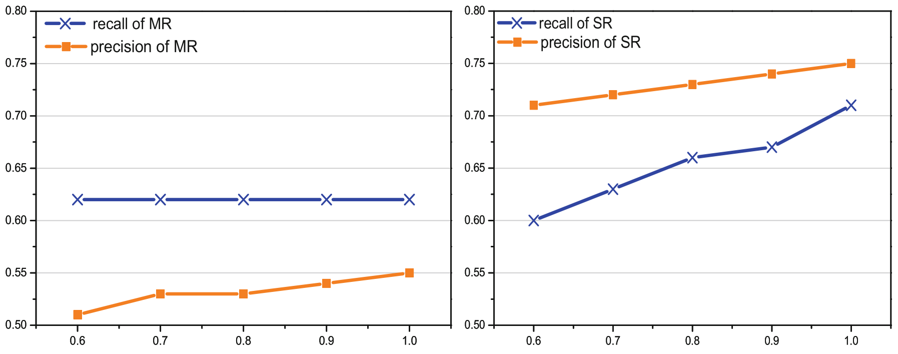

Fig. 6 Recall and precision of DBPedia data

For four of the seven datasets (Peel, Jamendo, LinkedCT, and SwetoDblp), we did not obtain any mining result because there are no instances typed as different classes in these four datasets, which is the premise to mine the class subsumptions. For the other three datasets (GeoSpecies, LinkedMDB, and DBpedia), we got the mining relations using Apriori algorithm. To evaluate the effectiveness of mining relation and subclass relation, the recall and precision scores are computed for various thresholds on the confidence values. Figure 6 illustrates the recall and precision of DBpedia data. The results of GeoSpecies and LinkedMDB data are similar and not listed. We evaluated the mining relation and subclass relation by comparing them to the DBpedia ontology (version 3.7). However, as the DBpedia ontology contains different types of axioms, we only considered the class subsumption axioms. Moreover, we considered not only the explicit class subsumption but all the inferable class subsumptions as well. DBpedia ontology contains 276 explicit subsumption axioms, 597 inferable subsumption axioms, and 873 subsumption axioms in total.

Figure 6 illustrates the precision and recall of mining relation (denoted as **MR**) and subclass relation (denoted as **SR**) with varying confidence values. We can observe that the recall of mining relation is 6.2 whatever the threshold on the confidence value is, because we do not find more correct results when the threshold is decreased. We can also observe that the precision of subclass relation is improved largely compared with that of mining relation. We can also observe that the recall

Table 2 Performance (in ms) of ExpLOD, FV, CVpop, and CVdist

	ExpLOD	**FV**		**CVpop**		**CVdist**	
	Time	**Time**	**# of queries**	**Time**	**# of queries**	**Time**	**# of queries**
Peel	62, 422	11, 359	87	–	–	–	–
Jamendo	202, 344	26, 906	85	–	–	–	–
GeoSpecies	2, 839, 031	1, 783, 593	2, 352	67, 078	538	35, 812	298
LinkedCT	860, 719	119, 953	172	–	–	–	–
LinkedMDB	645, 891	98, 968	621	46, 829	493	27, 922	388
SwetoDblp	913, 594	504, 109	566	–	–	–	–
DBpedia	–	249, 541, 131	152, 729	2, 945, 578	2, 574	1, 460, 828	1, 423

of subclass relation is lower than that of mining relation when the threshold is set to 0.6. The reason is that when the threshold on the confidence value is decreased, the Apriori algorithm finds more wrong subsumption axioms. The mining relation contains more results that show C_i as a subclass of C_j and C_j as a subclass of C_i; although one of these subsumption axioms maybe true, these results are both removed from subclass relation because of **asymmetry**. But with the increase of the threshold, the recall of subclass relation is improved, largely compared with that of mining relation. Finally, based on the subclass relation, we get 35 maximal classes out of all 326 classes when the threshold is set to 1.0. For most datasets, except DBpedia, the time needed to compute the association rules is less than 3 min when the threshold is set to 1.0. For the dataset DBpedia, when the threshold is set to 1.0, the running time is less than 20 min due to the large number of instances.

To show the efficiency of our approach and ExpLOD, we conduct a performance evaluation. In our approach, the threshold on the confidence value is set to 1.0. Performance is measured as the time taken by different approaches. For ExpLOD, the premise of computing the summary graph is to construct a labeled graph. Hence, the time taken by ExpLOD is the sum of the time for constructing a labeled graph and the time for computing the summary graph. The running time of our approach is composed of two parts: the time to obtain the data structure (full or concise) and the time to compute the corresponding statistics. We compare three means of our approach: full visualization (denoted as **FV**), concise visualization with the most popular property for every maximal class (denoted as **CVpop**), and concise visualization with the most distinctive property for every maximal class (denoted as **CVdist**). Peel, Jamendo, LinkedCT, and SwetoDblp did not yield any mining results. Thus, we list only the full visualization performance for them. For DBpedia, the running time exceeds three days when using ExpLOD; the results are not listed.

Table 2 shows the performance comparison. The running time of ExpLOD is longer than that of our approach (regardless of full visualization or concise visualization). For ExpLOD, before computing the summary graph, a bisimulation label must be assigned to each class, predicate, instance, and literal. This graph will increase the triple number largely. For the Jamendo dataset, for example, ExpLOD records 1,047,950 triples. After adding bisimulation labels, the number of triples increased to 3,940,113, which is about four times more than the original number. We can observe that the running time for constructing full visualization is much longer than that needed for constructing concise visualization. Moreover, the running time and number of SPARQL queries for constructing **CVdist** are all less than that needed for constructing **CVpop**. The reason is that it does not need to compare the instantiation for every property when constructing **CVdist**. The performance to construct concise visualization is largely improved for the datasets GeoSpecies and DBpedia. One reason is that they have relative complex structure, it reduces a large number of queries when constructing the concise visualization; the other important reason is that they have many mined class subsumptions that help to obtain a small number of maximal classes in the concise visualization.

6 Conclusion

We propose a SPARQL-based tool to construct the actual schema, gather corresponding statistics, and present a UML-based visualization for RDF data sources, such as SPARQL endpoints and RDF dumps. The proposed approach helps users understand the data structure and then construct a suitable SPARQL query. Moreover, for datasets with a complex structure, the Apriori algorithm is used to mine class subsumptions and then create a concise visualization which is convenient for understanding. Experimental results show that our approach does not need to construct a middle graph and is more efficient than ExpLOD. The concise visualization experiments on complex datasets, such as DBpedia, show that our approach is feasible.

Acknowledgements The work is supported by the National Natural Science Foundation of China under grant no. 60973024, no. 61170165, and no. 61003055.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. IJSWIS **5**(3), 1–22 (2009)
2. Maduko, A., Anyanwu, K., Sheth, A.P., Schliekelman, P.: Graph summaries for subgraph frequency estimation. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 508–523. Springer, Heidelberg (2008)
3. Cyganiak, R., Stenzhorn, H., Delbru, R., Decker, S., Tummarello, G.: Semantic sitemaps: efficient and flexible access to datasets on the semantic web. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 690–704. Springer, Heidelberg (2008)
4. Langegger, A., Woβ, W.: RDFStats-an extensible RDF statistics generator and Library. In: 20th International Workshop on Database and Expert Systems Application, pp. 79–83, 2009
5. Hausenblas, M., Halb, W., Raimond, Y., Feigenbaum, L., Ayers, D.: SCOVO: Using statistics on the web of data. In: ESWC 2009. Springer, Heidelberg (2009)

6. Khatchadourian, S., Consens, M.P.: ExpLOD: exploring interlinking and RDF usage in the linked open data cloud. In: ESWC 2010. LNCS, vol. 6089, pp. 272–287. Springer, Heidelberg (2010)
7. Brockmans, S., Volz, R., Eberhart, A., Loffler, P.: Visual modeling of OWL DL ontologies using UML. In: McIlraith, S.A., Plexousakis, D., Harmelen, F., (eds.) ISWC 2004. LNCS, vol. 3298, pp. 198–213. Springer, Heidelberg (2004)
8. Documents associated with Ontology Definition Metamodel (ODM) Version 1.0, http://www.omg.org/spec/ODM/1.0/ (2009)
9. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. of 20th International Conference on Very Large Data Bases, pp. 487–499. Morgan Kaufmann, San Francisco (1994)