Extracting Description Set Profiles from RDF Datasets using Metadata Instances and SPARQL Queries

Tsunagu Honma
Graduate School of Library, Information and Media Studies, University of Tsukuba, Japan
tsuna@slis.tsukuba.ac.jp

Kei Tanaka
NTT DATA Corporation, Japan
telekyon.official@gmail.com

Mitsuharu Nagamori
Faculty of Library, Information and Media Science, University of Tsukuba, Japan
nagamori@slis.tsukuba.ac.jp

Shigeo Sugimoto
Faculty of Library, Information and Media Science, University of Tsukuba, Japan
sugimoto@slis.tsukuba.ac.jp

Abstract

A variety of communities create and publish metadata as Linked Open Data (LOD). Users of those datasets find and use them for their own purposes and may combine the datasets to add value. Each LOD dataset uses various vocabularies, structures and constraints for describing resources. In order to improve the usability of LOD datasets, it is very important for metadata designers to enhance the interoperability of their own metadata with that of other datasets. In order to create new interoperable metadata, metadata schema designers have to understand the Application Profiles of the existing LOD datasets.

Dublin Core Description Set Profile (DSP) is a component of Dublin Core Application Profiles. A DSP describes the structures and constraints of metadata in an application (e.g., resource classes, properties cardinality and value scheme). Metadata schema registries, which collect and provide metadata schemas, have a large potential for helping metadata schema designers find, compare, and adopt existing schemas. However, most LOD datasets are not published with their DSPs. As a result, metadata schema designers have to look at each dataset and guess the DSPs.

This paper proposes a method to extract the structural constraints of metadata records automatically from metadata instances using existing metadata schema. The goal of this study is to reduce the cost of metadata schema extraction and to increase the number of metadata schemas registered in metadata schema registries. We have experimentally extracted constraints from LOD datasets using SPARQL. To evaluate, we applied our approach to 10 datasets in the DataHub. By comparing the structural constraints that were extracted using our approach with a manual approach, we found that our approach was able to extract more constraints.

Keywords: application profiles; metadata schema design; metadata schema extraction

1. Introduction

A considerable number of metadata datasets are published as Linked Open Data (LOD)[1] for sharing on the Web. LOD is widespread across many specific domains such as government, geography and e-science. Many communities create and publish LOD datasets on the Web and users are free to combine those datasets. Before designing new LOD datasets, metadata schema designers design a new application profile, which defines some constraints of metadata that are important for users of datasets. Particularly, in order to mash-up different datasets, metadata schema designers should create schema that enhance the interoperability of those metadata.

Application Profiles (Coyle and Baker, 2009) are helpful for users to understand the constraints of datasets. Dublin Core Description Set Profile (DSP) (Nilsson, 2008) is a component of an application profile, which explains the structural constraints of metadata

[1] http://linkeddata.org/

instances (Nilsson and Baker, 2008). If metadata schema designers are able to find and use DSPs, they can understand what vocabularies, structures, and constraints are used for describing datasets in that specific domain.

There are some metadata schema registries for accumulating and publishing metadata vocabularies and application profiles. Metadata schema designers can use those registries for finding existing application profiles that are similar to their own application profile. In order to cover a more specific domain, we have to increase the number of application profiles. However, most LOD datasets are not published with their profiles (Nishide, et al,. 2013). Therefore, metadata schema designers have to look into datasets and try to deduce their structural constraints. There are a lot of datasets in each specific domain, and those datasets are often too large to look into to determine structural constraints. It is therefore costly for metadata schema designers to have to make deductions about structural constraints manually.

We propose a method to extract the structural constraints of LOD datasets automatically. Creators of LOD datasets describe metadata instances based on their implicit or explicit structural constraints. Therefore, we use metadata instances, which are included in LOD datasets and existing metadata schema, for extracting structural constraints. We extract structural constraints from LOD datasets using SPARQL. We create Description Templates for each class membership, which resources are instances of. After creating Description Templates, we also extract property URIs, value types, language tags and datatypes for creating Statement Templates.

We apply our approach in practice to 10 datasets in the DataHub for evaluating our approach and clarifying issues which we need to solve for improving our method.

2. Sharing Application Profiles to Design a New Interoperable Schema

When metadata schema designers design a new application profile, they try to find existing application profiles in order to 1) reduce the cost of designing application profiles, 2) improve the interoperability of their metadata and 3) develop requirements for their metadata. Creating application profiles from scratch comes at a high cost, because metadata schema designers have to find suitable metadata vocabularies and structures for their purposes. If there are existing application profiles which have been created for similar purposes, designers can reuse those schema to reduce the cost of finding metadata vocabularies and deciding on the structure of metadata. As a result, the new application profile has improved interoperability because schema designers reuse common vocabularies and structures in the specific domain in which their metadata is used. Through reusing and customizing existing application profiles, metadata schema designers develop requirements for their metadata

In order to accomplish these goals, metadata schema designers should find and reuse existing application profiles in the same domain. Metadata schema registries are useful for metadata schema designers to find existing parts of application profiles. Metadata schema registries support the sharing of metadata schema on the web and promote reuse of metadata schemas and metadata interoperability (Nagamori et al., 2011). The Open Metadata Registry (Hillmann et al., 2006) is one such metadata schema registry. This registry can store metadata vocabularies and metadata schema in the form of element sets. MetaBridge (Nagamori et al., 2011) is also a metadata schema registry which is compatible with OWL-DSP based on DSP. If metadata creators share their application profile explicitly in those registries, metadata schema designers can use those registries as examples of metadata structures and constraints when they design new application profiles.

The number of application profiles that are registered in those registries is not enough for metadata schema designers to find and reuse those profiles. Therefore, it is important to create and register application profiles of various datasets. If metadata creators publish LOD datasets with their application profiles, schema registries can accumulate and share those application profiles. However, most LOD datasets are published without explicit application profiles. For that reason, one has to look into each LOD dataset and create its application profile manually. LOD datasets are often too large for observing as a whole, and observing those datasets and creating application profiles are difficult for metadata schema designers. It is necessary to extract application profiles from existing LOD datasets automatically.

There is related work in the area of schema extraction (Chidlovskii, 2002). Here, the researchers proposed methods for extracting XML Schema. XML Schema defines the structural constraints of metadata, which have been serialized in XML, such as the hierarchies of each XML element and its attribute. However, we would like to extract the structural constraints of resources, properties and values that are described with the RDF model, not only serialized with XML. Such constraints are independent of the serialization found in XML elements hierarchies. SchemEX (Konrath et al., 2012) is an existing approach for extracting metadata schema from LOD datasets. This approach extract schema that includes RDF type clusters and relationships between resources that are instances of type clusters. Those schema abstract structural constraints about dataset with typed resources and properties, but not define metadata value constraints, especially literal value constraints such as datatypes and language tags.

In this research, we propose a method to extract application profiles for LOD datasets automatically using metadata instances and existing schema. In the Singapore Framework, an application profile consists of five components. This research aims to extract Description Set Templates, which define the structural constraints of metadata instances. Metadata instances are described based on implicit or explicit structural constraints. We can extract those constraints from existing metadata instances.

3. Extracting Structural Constraints from Metadata Instances

Definitions of metadata vocabularies, structural constraints of metadata and description formats are all components of a metadata schema. In this research, our goal is to extract structural constraints as a DSP when a user inputs metadata instances. A DSP consists of Description Templates and Statement Templates. Description Templates define the constraints of resources, and Statement Templates define the constraints of attributes. In DSP, we are able to describe the following constraints using Description Templates and Statement Templates.

- Description Templates
  - Resource class membership constraints
  - Statement Templates which belongs to this Description Template
- Statement Templates
  - Property URI
  - Type constraint, "literal" or "non-literal"
  - Class membership of non-literal metadata values
  - Datatypes and language tags of literal metadata values

In this section, we explain our approach for extracting structural constraints with an example. Figure 1 shows an example of metadata instances. The example shows that *_:group1* is an instance of *foaf:Group ∩ foaf:Organization*. This resource has two members using *foaf:member*, *_:person1* and *_:person2* which have their own names and email addresses with *foaf:name* and *foaf:mbox*. Our goal is extracting the structural constraints of these metadata instances as seen in table 1 and table 2. Table 1 shows the constraints of resources which are instances of *foaf:Group ∩ foaf:Organization*. Table 2 shows the constraints of resources which are instances of *foaf:Person*.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

_:person1 rdf:type foaf:Person;
  foaf:name "Alice"@en;
  foaf:mbox <mailto:alice@example.com>.

_:person2 rdf:type foaf:Person;
  foaf:name "Bob"@en;
  foaf:mbox <mailto:bob@example.com>.

_:group1 rdf:type foaf:Group, foaf:Organization;
  foaf:name "University of Tsukuba"@en;
  foaf:homepage <http://www.tsukuba.ac.jp/>;
  foaf:member _:person1, _:person2 .
```

FIG. 1. An example of metadata instances for extracting structural constraints of metadata

TABLE 1: Structural constraints of an instance of foaf:Group ∩ foaf:Organization

Attribute	Property	Value Constraints
name	foaf:name	rdfs:Literal, @en
website	foaf:homepage	foaf:Document
member	foaf:member	foaf:Person

TABLE 2: Structural constraints of an instance of foaf:Person

Attribute	Property	Value Constraints
name	foaf:name	rdfs:Literal, @en
email	foaf:mbox	rdfs:Resource

Metadata instances are described based on the above constraints, and we extract them from metadata instances using the following steps. In each step, we extract resources, properties and values using SPARQL because we need to estimate the structural constraints of metadata instances. Before extracting the structural constraints, we loaded metadata instances in an RDF database.

Step 1: Get the class membership which resources are instances of

Step 2: Get the properties for each class membership

Step 3: Get a value type constraint (literal or non-literal)

Step 4: Get other value constraints

Step 4-1: Get literal value constraints (e.g., language tag and datatype)

Step 4-2: Get non-literal value constraints (e.g., resource class membership and base URI)

In the first step, we extract class memberships of resources which are described using *rdf:type* because typed resources are useful starting anchors for defining Description Templates. In our example, there are two class memberships, *foaf:Person* and (*foaf:Group ∩ foaf:Organization*). We extract those memberships using a SPARQL query, which is shown in Figure 2, and create two Description Templates.

```
SELECT DISTINCT (GROUP_CONCAT(DISTINCT(?type) ; separator = ", ") as ?types)
WHERE {
  ?s rdf:type ?type.
  ?s ?p ?o.
  FILTER(?p!=rdf:type)
}
GROUP BY ?s
ORDER BY ?type
```

FIG. 2. A SPARQL query for extracting the class membership which resources are instances of

The second step is a process for creating Statement Templates. Statement Templates are created for defining the constraints of metadata attributes. In this step, we execute queries to find properties for each class membership, which are defined by Description Templates. When we execute a SPARQL query, such as that shown in Figure 3, we get minimum Statement Templates that define only property constraints.

```
SELECT DISTINCT ?p
WHERE {
  ?s ?p ?o .
  ?s rdf:type foaf:Group .
  ?s rdf:type foaf:Organization .
  FILTER NOT EXISTS {
    ?s rdf:type ?type .
    FILTER(?type != foaf:Group)
    FILTER(?type != foaf:Organization)
  }
}
```

FIG. 3. A SPARQL query for extracting properties which instances of foaf:Group ∩ foaf:Organization have

We estimate value constraints in the third step. After we get metadata values using classes of resources and a property, we classify those values into "literal", "non-literal" and "mix". To estimate value constraints, we count the number of the three metadata values below.

A) The number of all metadata values,

B) The number of literal metadata values, and

C) The number of non-literal metadata values.

When A = B, we define value constraints as "literal". If A > B and A > C, we define value constraints as "mix". For extracting B and C, we use *isLiteral*, *isIRI* and *isBlank*, which are SPARQL functions that are shown in a SPARQL query in Figure 4.

```
SELECT (COUNT (?o) as ?count)
WHERE {
  ?s rdf:type foaf:Person .
  FILTER NOT EXISTS {
    ?s rdf:type ?type .
    FILTER(?type != foaf:Person)
  }
  {
    ?s foaf:mbox ?o .
    FILTER isBlank(?o)
  }
  UNION
  {
    ?s foaf:mbox ?o .
    FILTER isIRI(?o)
  }
}
```

FIG. 4. A SPARQL query for extracting the number of non-literal metadata values for foaf:Person and foaf:mbox

In the final step, we extract the constraints of literal and non-literal metadata values such as class memberships of non-literal resources, base URIs, language tags and datatypes of literal metadata values. This process is executed based on the result of step 3. If the metadata type value is "non-literal", we extract resource classes and the base URI of metadata values by analyzing all objects data pulled back and load to the RDF database. When we found metadata with the value "literal", we defined datatype and language tags.

4. Evaluation

We implemented a system to extract DSPs using our approach. To evaluate our system and approach, we extract DSPs from 10 datasets and verify those DSPs. We used 10 LOD datasets that are published as RDF files on the DataHub[2]. It is difficult to extract metadata schema manually, so to evaluate our method for large datasets, we chose datasets that could be accessed on the Web and were the top 10 largest in file size at the time of access. In this evaluation, we confirm only precision by comparing constraints which are extracted using our approach and a manual method, and also comparing extracted constraints and actual datasets.

First, we compared structural constraints defined by DSPs, which were extracted by our approach and a manual method. Using this comparison, we attempted to confirm if the system we implemented is running correctly based on our proposed method. A person who executes a manual method has knowledge and experience of designing metadata schema, but may not have knowledge about the specific domain of each dataset (e.g., geography, statistics, etc.). In a manual method, the process of extracting a DSP is based on 5 steps that were shown in section 3. The difference of our approach and a manual method is the data size of RDF files. For extracting a DSP from a dataset, our approach used entire RDF files belonging to that dataset, whereas the manual method used the top 200 lines from each RDF file.

Table 3 shows the number of Description Templates and Statement Templates that were extracted using our approach and a manual method. We confirmed that all of structural constraints extracted manually were included in the structural constraints extracted by our approach. The constraints that we compared are shown in section 3. We also confirmed the constraints which were extracted by our approach are not contradictory to actual datasets. There are, however, differences between numbers of templates that were extracted by our approach and a manual method. One reason is because the amount of data that was used to manually extract was smaller than our approach. Another reason is that some resources have multiple RDF types,

[2] http://datahub.io/

Table 3: The number of Description Templates and Statement Templates that were extracted by our approach and a manual method

Dataset ID in the DataHub	Description Templates		Statement Templates	
	our approach	manual method	our approach	manual method
nytimes	1	1	13	9
colinda	2	1	15	7
mondial	19	4	107	31
eurostat-rdf	9	2	75	8
linked-open-vocabularies-lov	9	4	63	15
farmers-markets-geographic-data-united-states	33	4	164	18
msc	6	1	39	4
nuts-geovocab	4	3	15	11
osm-semantic-network	3	3	44	22
parole-simple-out	168	2	669	7

and those class memberships are difference each other. For these cases, we created a Description Template for each class membership, so that most Description Templates have only a few resources. For example, we extracted 168 Description Templates from *parole-simple-out*, but 106 Description Templates have less than 10 resources. We discuss this problem in section 5.

After we confirmed our system is running correctly, we checked constraints that were extracted by our method but weren't extracted by the manual method. As a result of comparing those constraints and original datasets, those constraints were not contradictory to datasets. Finally, we looked into parts of each dataset in order to find constraints which were not extracted by our method.

In the above procedure, we confirmed that it is possible to extract most structural constraints, which described in section 3, using our approach. However, there are constraints which we could not extract using our approach. We discuss whether or not the constraints that we extracted are useful to understand existing metadata structures in the next section.

5. Discussion

We could not extract structural constraints of resources which do not have *rdf:type* using our approach. For example, *nuts-geovocab,* for describing geographical metadata, includes RDF Collections in order to describe the exterior of geospatial objects with multiple coordinates. Figure 5 shows metadata instances from *nuts-geovocab*. There are more than two coordinates for describing the exterior of the resource *"http://nuts.geovocab.org/id/AT111_geometry"*. Those coordinates are described using non-typed blank nodes which are connected with *rdf:first* and *rdf:rest*. This meant that we could not extract the Description Templates for resources that describe coordinates. When we guess the classes of each resource using existing metadata schema and definitions about metadata vocabularies which include *rdfs:domain* or *rdfs:range*, we can extract more Description Templates.

There are other issues that need to be solved in order to improve our approach. In this evaluation, we could extract a large number of Description Profiles from *farmers-markets-geographic-data-united-states* and *parole-simple-out*. We proceeded to check their Description Templates and Statement Templates. As a result, in some cases, we could merge the Description Templates into other templates. For example, *farmers-markets-geographic-data-united-states*, there are the following two class memberships, Class membership defined in Description Template A

- http://logd.tw.rpi.edu/source/data-gov/vocab/Dataset (logd:Dataset)
- http://purl.org/twc/vocab/conversion/Dataset (conversion:Dataset)
- http://purl.org/twc/vocab/conversion/MetaDataset (conversion:MetaDataset)
- http://rdfs.org/ns/void#Dataset (void:Dataset)

Class membership defined in Description Template B

- http://logd.tw.rpi.edu/source/data-gov/vocab/Dataset (logd:Dataset)
- http://purl.org/twc/vocab/conversion/Dataset (conversion:Dataset)
- http://purl.org/twc/vocab/conversion/SameAsDataset (conversion:SameDataset)
- http://rdfs.org/ns/void#Dataset (void:Dataset)

Description Template A and B have differences in the two classes conversion:MetaDataset and conversion:SameDataset. Both Description Templates have 8 Statement Templates, and those Statement Templates are similar. If there are a large number of Description Templates, metadata schema designers cannot easily understand the structural constraints of the dataset. In that case, we should define one Description Template for resources which are instance of (logd:Dataset ∩ conversion:Dataset ∩ void:Dataset).

We believe that we are unable to extract DSPs correctly if there are resources that have multiple roles in the datasets. We have created and published *Aozora Bunko LOD*[3] which is a dataset including bibliographies based on *Aozora Bunko*[4]. *Aozora Bunko* is a Japanese digital library that publishes digitized books. The bibliographies, which are published on *Aozora Bunko*, have some resources about persons, such as "creator", "translator" and "reviser". We described person as a instance of *aozora:Person*. However, instances of *aozora:Person* have different roles in that dataset as mentioned above. In that case, we can only extract one Description Template about *aozora:Person*, and in the Description Template, the metadata attributes for the persons with different roles are mixed. There are two approaches to resolve this problem. One is by

```
<geometry:Polygon
  xmlns:geometry="http://geovocab.org/geometry#"
  xmlns:wgs84pos=" http://www.w3.org/2003/01/geo/wgs84_pos#"
  rdf:about="http://nuts.geovocab.org/id/AT111_geometry">
  <geometry:exterior>
    <geometry:LinearRing>
      <geometry:posList>
        <rdf:Description>
          <rdf:first>
            <rdf:Description>
              <wgs84pos:lat>47.35300025</wgs84:lat>
              <wgs84pos:long>16.435400050000055</wgs84:long>
            </rdf:Description>
          </rdf:first>
          <rdf:rest>
            <rdf:Description>
              <rdf:first>
                <rdf:Description>
                  <wgs84:lat>47.455132750000018</wgs84:lat>
                  <wgs84:long>16.281081050000068</ns48:long>
```

FIG. 5. An example of resource which are described using non-typed resources

[3] http://mdlab.slis.tsukuba.ac.jp/lodc2012/aozoralod/
[4] http://www.aozora.gr.jp/

adding different classes for each type of person in the original datasets. Because it is required to change source data, this approach is not practical. The other is extracting a Description Template for each pair of a class membership and a property that has an instance of that class membership as a range. For example, if there are metadata instances which figure 6 shows, we should extract two Description Templates for *aozora:Person* as *dc:creator* and *aozora:Person* as *dc:translator*.

```
<book_A> dc:creator <person_X> ;
             dc:translator <person_Y> .

<person_X> rdf:type aozora:Person .
<person_Y> rdf:type aozora:Person .
```

FIG. 6. An examples of resources which are both instance of aozora:Person, and have different roles "dc:creator" and "dc:translator"

6. Conclusion

In this paper, we have proposed a method for extracting the structural constraints of LOD datasets using metadata instances and existing schema. Metadata schema about existing datasets are important for metadata schema designers to create a new interoperable schema with a low cost. However, because creating formal metadata schema is costly, there are few schema about existing LOD datasets on the web. We aim to extract metadata schema automatically, especially the structural constraints of metadata records, in order to add metadata schema to metadata schema registries.

To evaluate our approach, we compared the number of structural constraints which were extracted by our approach and manually with 10 datasets in the DataHub. That evaluation showed that our approach could extract all the structural constraints which could be extracted manually. We also compared metadata instances and structural constraints which are extracted using our approach. As a result, it has become clear that there are three issues to be solved when extracting structural constraints using our approach. One is the need to improve our method for extracting Description Templates of resources which have no *rdf:type*. The second issue is that we need to merge Description Templates when the extracted templates are similar to other templates. The last issue is that we separate templates for resources, which have same classes, but have different roles in a dataset.

References

Chidlovskii. Boris (2002). Schema extraction from XML collections. Proceedings of the 2nd ACM/IEEE-CS joint conference of Digital libraries, 2002, 291-292.

Coyle, Karen and Thomas Baker. (2009). Guidelines for Dublin Core Application Profiles. Retrieved May 15, 2014, from http://dublincore.org/documents/2009/05/18/profile-guidelines/ .

Hillmann, I. Diane I, Stuart A. Sutton, Jon Phipps and Ryan Laundry. (2006). A Metadata registry from vocabularies up: The NSDL registry project. Proceedings of the International Conference on Dublin Core and Metadata Applications, 2006.

Konrath, Mathias, Thomas Gottron, Steffen Staab and Ansgar Scherp. (2012). SchemEX – Efficient Construction of a Data Catalogue by Stream-based Indexing of Linked Data. Journal of Web Semantics, 2012, vol. 16.

Nagamori, Mitsuharu, Masahide Kanzaki, Naohisa Torigoshi and Shigeo Sugimoto. (2011). Meta-Bridge: A Development of Metadata Information Infrastructure in Japan. Proceedings of the International Conference on Dublin Core and Metadata Applications, 2011, 63-68.

Nilsson, Mikael. (2008). Description Set Profiles: A constraint language for Dublin Core Application Profiles. Retrieve May 15, 2014, from http://dublincore.org/documents/dc-dsp/ .

Nilsson, Mikael, Thomas Baker and Pete Johnston. (2008). The Singapore Framework form Dublin Core Application Profiles. Retrieve May 15, 2014, from http://dublincore.org/documents/2008/01/14/singapore-framework/ .

Nishide, Yoritsugu, Tsunagu Honma and Mitsuharu Nagamori. (2013). An Investigation of Japanese Open Data Schema and Links to Improve the Use of Datasets. Digital Library, 2014.