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

Tsunagu Honma†, Kei Tanaka††, Mitsuharu Nagamori†, Shigeo Sugimoto†
† University of Tsukuba, ††NTT DATA Corporation
Objective: Extraction of Metadata Schema from Metadata Instances

- Reusing existing schemas for designing new schemas
- Extracting metadata schema from metadata instances published and available on the web
- Discussing the issues of extracting Description Templates

An RDF graph → A Schema
Metadata Schema Design

• Schema = Requirements, Models, Structures, ...

• Guidelines for DCMI Application Profiles ‡
  – Create a new schema based on existing schema
  – “Are there existing community standards that need to be considered?”

• Reuse existing schemas is important for designers to create a new schema

‡ http://dublincore.org/documents/profile-guidelines/
Reuse Metadata Schemas for Designing New Schema

• Metadata schema designers select and customize existing schemas
  – to enhance metadata interoperability, and
  – to reduce the cost of designing metadata
Goal

• Accumulate existing metadata schema into metadata schema registries

• Make metadata schema accessible for schema designers to help them reuse existing schema
Metadata Schema Registry

• Accumulates metadata schemas and make them reusable on the Internet
  – Open Metadata Registry†, MetaBridge††, …

† http://metadataregistry.org/
†† https://www.metabridge.jp/
Problem: Undocumented Metadata Schema

• We search and collect metadata schemas from the web

• There are lots of datasets on the web
• However, those datasets rarely publish their schema

• Very Basic Problem: even if metadata is open its schema is rarely published or documented

• We cannot collect existing schema easily
Collect when Schemas are Undocumented

- Look at each datasets and create documents include schemas
  - It is hard to create documents because datasets are often too large to understand overviews of datasets.

- We want the easy way to make schema documented

1. publish data
2. document schema
Approach: Extract Constraints using SPARQL and Create a DSP

- We extract constraints of metadata from metadata instances using SPARQL—especially, we extract structural constraints to understand overviews of a dataset

- We create a Description Set Profile based on the extracted constraints
Dublin Core Description Set Profile

• “A DSP is a way of describing structural constraints on a description set” †

• Description Template
  – “which contain the statement templates that apply to a single kind of description as well as constraints on the described resource”

• Statement Template
  – “which contain all the constraints on the property, value strings, vocabulary encoding schemes, etc. that apply to a single kind of statement”

† http://dublincore.org/documents/dc-dsp/
DSP for RDF

• Before extracting Description/Statement Templates, we have to define
  – what is “a single kind of description”, and
  – what is “a single kind of statement” in RDF
A Single Kind of Description

• A description
  – An RDF graph which describes about an RDF subject

• A single kind of description
  – A kind of RDF graph about RDF subjects
  – Those subjects have same class or class memberships using rdf:type
  – e.g. If there are some instances of foaf:Person, a single kind of description is RDF graphs about those instances
A Single Kind of Statement

• A statement
  – An RDF triple

• A single kind of statement
  – A kind of RDF triple
  – Those triples are described using same property URI
Examples of Description/Statement Templates

```
template: schema:Book ∩ ex:Book

<table>
<thead>
<tr>
<th>attribute</th>
<th>property</th>
<th>occur</th>
<th>type</th>
<th>relation</th>
<th>lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>dc:title</td>
<td>1</td>
<td>Literal</td>
<td></td>
<td>en, ja</td>
</tr>
<tr>
<td>Author</td>
<td>dc:creator</td>
<td>0-n</td>
<td>URI</td>
<td>[foaf:Agent]</td>
<td></td>
</tr>
<tr>
<td>Subject</td>
<td>dc:subject</td>
<td>0-n</td>
<td>URI</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
template: foaf:Agent

<table>
<thead>
<tr>
<th>attribute</th>
<th>property</th>
<th>occur</th>
<th>type</th>
<th>relation</th>
<th>lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>dc:title</td>
<td>1</td>
<td>Literal</td>
<td></td>
<td>en, ja</td>
</tr>
<tr>
<td>Link</td>
<td>rdfs:seeAlso</td>
<td>0-n</td>
<td>URI</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Extraction Process

1. Get class memberships and create Description Templates
2. Get properties and create Statement Templates
3. Get a value type constraint
4. Get other value constraints
   1. Get literal value constraints
   2. Get non-literal value constraints
Step 1: Get Class Memberships

- A Description Template is a template for a single kind of description
- We regard an instance of one or more rdf:type as a described resource

A Description Template for schema:Book ∩ ex:Book
A Query for Step 1

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

- Extract co-occurred class memberships
- Create Description Templates for each class membership
Step 2: Get Properties

- A Statement Template is a template for a single kind of statement

- Resource A
  - dc:title
  - dc:creator

- Person A
  - dc:creator

- "book title"

- A Description Template
  - A Statement Template for dc:title
  - A Statement Template for dc:creator
A Query for Step 2

```
SELECT DISTINCT ?predicate
WHERE {
  ?s rdf:type schema:Book .
  ?s rdf:type ex:Book .
  FILTER NOT EXISTS {
    ?s rdf:type ?type .
    FILTER(?type != schema:Book)
    FILTER(?type != ex:Book)
  }
}
```

- Extract properties for instances of a class membership
- Create Statement Templates for each property
Step 3: Get a Value Type

- Constraints of literal values are different from constraints of non-literal values
- We extract a value type, Literal, Non-Literal or Mix

Diagram:
- dc:title
- resource A
- "book title"
- dc:creator
- person A
- Template for dc:title values are Literal
- Template for dc:creator values are Non-Literal
A Query for Step 3

```
SELECT (COUNT (?o) as ?count) WHERE {
  FILTER NOT EXISTS {
    ?s rdf:type ?type .
    FILTER(?type != schema:Book)
    FILTER(?type != ex:Book)
  }
  { ?s dc:creator ?o . FILTER isBlank(?o) } UNION
  { ?s dc:creator ?o . FILTER isIRI(?o) }
}
```

- Get numbers of literal values and non-literal values
  – this query is an example of extracting non-literal
- Literal \( \geq 1 \), non-Literal \( =0 \) \( \rightarrow \) Values is Literal
Step 4: Get other Value Constraints

• Get literal value constraints
  – RDF language tags
  – RDF datatype URI

• Get non-literal value constraints
  – Resource classes
Implementation:
Schema Extraction System

• We implement a system to extract DSPs using the proposed approach

• Input:
  – an RDF file

• Output:
  – Description Template: class memberships
  – Statement Template: property URI, class/language/datatype of values
Evaluation: Correctness of our Implementation

• Purpose:
  – confirm the system is running correctly

• Methods:
  – Extract DSPs using our method and compare with DSPs created using a manual method

• Datasets:
  – 10 datasets from the Datahub

※ manual method: extract Templates manually using the top 200 lines from each RDF file, in 10-30 minutes
Result: Comparing with our method with the manual method

- templates (our method) \(\supseteq\) templates (manual method)
- It seems that our implementation is successful

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

• We checked Description/Statement Templates which are extracted by our approach
  – from datasets in the Datahub and others

• There are three problems of extracting Description Templates
  1. Resources without rdf:type
  2. Divided similar Description Templates
  3. Multiple roles in same property
Problem 1: Resources without rdf:type

- We could not create Description Templates for resources which don’t have rdf:type
  - Because we create Description Templates for each of class memberships
Solution 1: Create Description Template for property memberships

- We create temporal class and its Description Template for property memberships
Problem 2: Extract similar Templates

- There are numbers of Description Templates
- Those templates are often similar to each other
  - same Statement Templates
  - several differences between class memberships

For
- lodg:Dataset
- conversion:Dataset
- conversion:MetaDataset
- void:Dataset

Different Template

For
- lodg:Dataset
- conversion:Dataset
- conversion:SameDataset
- void:Dataset
The Number of Resources belonging to each Template

- 66,347 Resources, 168 Description Templates
- More than 100 templates have less than 10 resources

http://datahub.io/dataset/parole-simple-ont
Solution 2

• Merge templates after we create templates

• Distinguish class constraints and attributes
  – Prevent too much divided templates before we create templates
Solution 2-1: Merge Templates

• Merge templates depending on similarity of Description Templates and their Statement Templates
  – class memberships similarity
  – property memberships and constraints similarity
Solution 2-2: Distinguish Class Constraints and Attributes

• Values of rdf:type are
  – class constraints of a kind of described resource, or
  – values of an attribute for each resource

• After we distinguish class constraints and attributes, we create Description Templates based on the former
Problem 3: Multiple Roles in same Class

• In some cases, a class have multiple roles in a dataset
  – e.g. Instances of foaf:Person indicates authors or translators in a dataset, and their properties are different each other

• We extract composite templates for authors and translators

Author A

“name”

Translator A

“language”

Template for foaf:Person

constraints of **name**

constraints of **language**
Solution 3: Create Description Templates with Classes and Properties

• Use classes, properties and inbound properties when we create Description Templates
  – e.g. create a template for instances of foaf:Person as dc:creator which have foaf:name

Diagram:
- Book A -> dc:creator -> Author A
- Author A -> foaf:name
- rdf:type -> foaf:Person

Class

Inbound Property

Property

“name”
Summary

• There are few metadata schema on the Web, so that it is difficult to reuse existing schema

• We extract structural constraints of metadata instances using SPARQL

• Create Description Set Profiles with class memberships and properties we extracted

Speaker: Tsunagu Honma
tsuna@slis.tsukuba.ac.jp