Christophe Debruyne, Lucy McKenna, and Declan O'Sullivan. Extending R2RML with support for RDF collections and containers to generate MADS-RDF datasets. In Jaap Kamps, Giannis Tsakonas, Yannis Manolopoulos, Lazaros S. Iliadis, and Ioannis Karydis, editors, Research and Advanced Technology for Digital Libraries - 21st International Conference on Theory and Practice of Digital Libraries, TPDL 2017, Thessaloniki, Greece, September 18- 21, 2017, Proceedings, volume 10450 of Lecture Notes in Computer Science, pages 531–536. Springer, 2017

Extending R2RML with Support for RDF Collections and Containers to Generate MADS-RDF Datasets

Christophe Debruyne, Lucy McKenna, and Declan O'Sullivan

ADAPT Centre, Trinity College Dublin, College Green, Dublin 2, Ireland
{firstname.lastname}@adaptcentre.ie

Abstract. It is a best practice to avoid the use of RDF collections and containers when publishing Linked Data, but sometimes vocabularies such as MADS-RDF prescribe these constructs. The Library of Trinity College Dublin is building a new asset management system backed by a relational database and wants to publish their metadata according to these vocabularies. We chose to use the W3C Recommendation R2RML to relate the database to RDF datasets, but R2RML unfortunately does not provide support for collections and containers. In this paper, we propose an extension to R2RML to address this problem. We support gathering collections and containers from different fields in a row of a (logical) table as well as across rows. We furthermore prescribe how the extended R2RML engine deals with named graphs in the RDF dataset as well as empty sets. Examples and our demonstration on a part of the Library's database prove the feasibility of our approach.

Keywords. R2RML, Linked Data Publishing, MODS, MADS

1 Introduction

The Digital Resources and Imaging Services (DRIS) department of the Library of Trinity College Dublin (TCD) hosts the Digital Collections Repository of the university, providing open access to the university's growing collection of digitized cultural heritage materials. DRIS hopes to move towards publishing the bibliographic data of their digital collections as Linked Data (LD) as to increase their materials' visibility. To this end, a bespoke tool backed by a relational database has been developed that accepted URIs to other Linked Data datasets. The Library decided[1] that records should follow the Metadata Object Description Schema (MODS) as this standard was: suitable for cataloguing DRIS resources to the required level of detail, compatible with existing MAchine-Readable Cataloging (MARC) records in other catalogues and also less complex than MARC, and available as an RDF vocabulary.

Given that the information was stored in a relational database, adopting the RDB to RDF Mapping Language (R2RML) [1], a vocabulary for declaring customized mappings from relational databases to RDF datasets, is a sensible approach. During the creation of R2RML mappings, a challenge arose: complete RDF records could not be

[1] Which explains why no other models such as CIDOC-CRM were considered.

generated as, to produce such a record, the use of RDF collections was required by the ontology, but were not support by R2RML.

We thus propose a minimal extension to the R2RML language and algorithm for the generation of RDF collections *and* containers. After elaborating on our approach, we demonstrate it on a part of the tool's database. We discuss our approach with respect to related work prior to concluding this paper and formulating future directions.

2 Background

MODS and MADS. Both Metadata Object Description Schema (MODS) and the Metadata Authority Description Schema (MADS) are XML schemas to describe bibliographic metadata and share quite a few elements. An OWL ontology was developed for both schemas; MODS-RDF and MADS-RDF. The MODS-RDF ontology, however, excluded all elements it had in common with MADS. If one relied on the MODS XML schema and wants to generate "semantically" equivalent RDF, MADS-RDF has to be adopted as well. But, unlike MODS-RDF, where properties are represented individually, in MADS-RDF properties are grouped in collections.

Resource Description Framework (RDF) provides two constructs to gather RDF terms for use in statements; **RDF Containers and Collections**. The difference between RDF containers (`rdf:Bag`, `rdf:Seq`, and `rdf:Alt`) and collections (`rdf:List`) is that the latter has an explicit terminator (`rdf:nil`, or the empty list) and is therefore immutable. One can add additional elements to the former. We note that it is generally considered a bad practice to use these constructs in Linked Data publishing, but some ontologies rely on it.

We assume the reader is familiar with **R2RML**, and otherwise refer to [1]. We have chosen to adopt R2RML as it is a W3C Recommendation (i.e., a standard) and hence supported by various tools, and also because it provides us with a scalable declarative approach. However, R2RML provides no *elegant* support for creating such mappings. In some cases, one can resort to an additional triples map for creating these provided the underlying relational database has support for pivot tables (allowing one to "pivot" a table and treat a particular row as the column). We deem this approach also too complex. Another approach is to go through several pre-, or post-processing stages, but that renders RDF generation not self-contained.

3 Approach

In this section, we describe our approach to provide support for generating RDF containers and collections in R2RML. We will exemplify our approach with a simple database (see **Fig. 1**), and then cover both cases of collecting RDF terms (per row, and per column)[2] But we first formulate the following requirements: 1) **Collecting RDF terms per row from various cells** with the additional requirement that one should be able to specify what type of terms can be collected and that they can differ in a collection/container. 2) **Collect RDF terms across rows:** grouping the RDF

[2] Our prototype is available at: https://opengogs.adaptcentre.ie/debruync/r2rml/src/r2rml-col

terms that are generated from an object map for each subject. 3) **Nesting:** the ability to nest containers and collections, and both approaches. 4) **Provide support for managing empty collections and containers**. 5) **Managing named graphs**.

BOOK	
ID	TITLE
1	Frankenstein
2	The Long Earth

AUTHOR				
ID	BOOKID	TITLE	FNAME	LNAME
1	1	*NULL*	Mary	Shelley
2	2	Sir	Terry	Pratchett
3	2	*NULL*	Stephen	Baxter

Fig. 1. Two relational tables representing books and their authors.

To collect terms for each row in a logical table, we extended R2RML's vocabulary in the following ways: the introduction of a predicate `rrf:gather` to indicate which RDF terms need to be gathered into a collection or a container, and allowing the predicate `rr:termType` to refer to `rdf:Bag`, `rdf:Seq`, `rdf:Alt`, or `rdf:List`. The last is the default when a valid term type is absent. We note that these are all part of the RDF namespace, which we reused. The subject of `rrf:gather` must be a list of object maps that generate RDF terms. We thus have an object map that is comprised of object maps – which we will call a *gather map*. When none of the object maps generate a term as prescribed by the R2RML W3C Recommendation, an empty list or container is generated. One can also use `rrf:gatherAsNonEmpty` to avoid the generation of empty collections/containers.

The application of this gather-object map *g* on a row will result in the application of each object map part of *g* to create the container or collection. Using the running example described above, the R2RML snippet in **Listing 1** (top) generates the RDF shown in **Listing 1** (bottom) One can see how rows in the person table generates a bag only containing a first- and last name when a title is non-existent.

```
rr:predicateObjectMap [
 rr:predicate ex:name;
 rr:objectMap [
  rrf:gather ([ rr:column "TITLE" ] [ rr:column "FNAME" ] [ rr:column "LNAME" ]);
  rr:termType rdf:Bag;
  ];
]
```

```
person:2 ex:name [ a rdf:Bag; rdf:_1 "Sir"; rdf:_2 "Terry"; rdf:_3 "Pratchett" ].
person:1 ex:name [ a rdf:Bag; rdf:_1 "Mary"; rdf:_2 "Shelley" ].
person:3 ex:name [ a rdf:Bag; rdf:_1 "Stephen"; rdf:_2 "Baxter" ].
```

Listing 1. Collecting RDF terms for each row.

The fifth requirement will be covered here as it necessitates prescribing how different named graphs across rows that are gathered should be treated as the target graphs in the subject and predicate-object maps may differ for each row. While gathering collections or containers per row is fairly straightforward as it introduced a new object map that needs to be applied to each row of a logical table, collecting RDF terms per column is a bit more challenging in terms of coming up with an appropriate extension of the vocabulary and the algorithm, especially the latter as one needs to keep track of the rows that need to be grouped in order to generate the collection or container. We extended the algorithm as follows: the implementation keeps track of

all object maps with a `rr:collectAs` (See **Listing 2**) statement. The algorithm generates a subject for each row in the logical table (or join in case of reference-object maps). A special data structure keeps track of the RDF terms generated by the graph- and predicate-maps for each row – and thus also subject – whilst collecting the objects for the creation of the collection or container. Since each row may generate different predicates or graphs, but objects are collected across them, we have decided to store the collection or container in all possible combinations of graphs and predicates related to a particular subject. Though we think that this would be an unlikely use case, we deemed it important to think this aspect of the extension through.

```
rr:predicateObjectMap [
  rr:predicate ex:writtenby;
  rr:objectMap [
    rr:parentTriplesMap <#AuthorsTriplesMap>;
    rr:joinCondition [ rr:child "ID"; rr:parent "BOOKID"; ];
    rrf:collectAs rdf:List;
  ];
];
```

```
book:1 ex:writtenby ( person:1 ).
book:2 ex:writtenby ( person:2 person:3 ).
```

Listing 2. Collecting RDF terms across rows.

We allow nesting in the following ways: i) gather maps may be nested with gather maps; and ii) one may collect (nested) gather maps with `rrf:collectAs.` Since we have created an object map of object maps to tackle the case of gathering RDF terms for each row to cover the first case, it is fairly straightforward to nest them. For obvious reasons, however, no "cycles" are permitted in nested object maps. Due to space limitations, we will not be able to provide examples and refer to the documentation instead. What we do not allow is the use of `rrf:collectAs` in nested object maps; it does not make sense to start aggregating, for each row, terms across rows.

4 Demonstration

Here, we demonstrate our approach to generate a MADS-RDF dataset from the relational database of The Library's cataloguing system. Concepts in MODS, such as `mods:Title`, are related to a collection of `mads:Element` instances. Elements, which act as an abstract concept for something that has a label, are attributed such a label with the predicate `mads:elementValue` whose range is an `xsd:string`. The concept `mads:Element` is then specialized into a number of subclasses such as `mads:TitleElement`, which itself is an abstract concept for all elements one can find in a title. One needs to use instances of "concrete" concepts such as `mads:MainTitleElement`, and `mads:PartNameElement` in that list.

In the database, a record must have at least one TitleInfo – terminology adopted from MODS XML, which acts as a "container for all subelements related to title information. The table `TitleInfo` thus has a foreign key to a record in the table `Record`. In `TitleInfo`, all subelements are captured in the fields `nonSort`, `partName`, `partNumber`, `subtitle`, and `title`. Due to space limitations and since the structure of these mappings are the same for all subelements, we will only

describe one. We also leave out the mapping for `Record` (and also how records are then related to `TitleInfo`), and focus on the creation of title elements instead. We note a mapping was created for the whole database (including other elements). The evaluation of our approach's performance was not within the scope of this study.

Our mapping is shown in **Listing 3** (top), we use HTTP URIs for TitleInfo, but URNs for the individual elements. We chose URNs as we do not (yet) foresee a reason why users want to engage with these resources via resolvable HTTP URIs, but we also wanted to avoid the use of blank nodes. Since the actual value of the title elements are not suitable for creating URNs as they can contain illegal characters, we provided specific IDs for each element when they exist. These conditionals appear in the SQL query. The title info and title element are linked by reusing the same URN template (highlighted in yellow). **Listing 3** (bottom) contains some RDF statements that were generated of one of TCD Library's assets.

```
<#TitleInfo>
  rr:logicalTable [
    rr:sqlQuery """SELECT *, IF(nonSort IS NULL, NULL, id) AS nId, IF(subtitle IS
NULL, NULL, id) AS sId, IF(partNumber IS NULL, NULL, id) AS nuId, IF(partName IS
NULL, NULL, id) AS naId FROM TitleInfo"""; ];
  rr:subjectMap [
    rr:template "http://data.library.tcd.ie/resource/titleinfo/{id}";
    rr:class madsrdf:Title;
  ];
  # Mapping to generate rdfs:label based on "title" omitted
  rr:predicateObjectMap [
    rr:predicate madsrdf:elementList;
    rr:objectMap [
      rrf:gather (
        [ rr:template "urn:tcd:title-nonsort-{nId}" ]
        [ rr:template "urn:tcd:title-main-{id}" ]
        [ rr:template "urn:tcd:title-subtitle-{sId}" ]
        [ rr:template "urn:tcd:title-partname-{naId}" ]
        [ rr:template "urn:tcd:title-partnumber-{nuId}" ]
      );
    ];
    rr:termType rdf:List;
  ];
 .
<#TitleInfo-Title>
  rr:logicalTable [ rr:sqlQuery "SELECT id, title FROM TitleInfo"; ];
  rr:subjectMap [
    rr:template "urn:tcd:title-main-{id}"; rr:class madsrdf:MainTitleElement;
  ];
  rr:predicateObjectMap [
    rr:predicate madsrdf:elementValue; rr:objectMap [ rr:column "title"; ];
  ];
 .
```

```
<http://data.library.tcd.ie/resource/titleinfo/2> a mads:Title; mads:elementList (
<urn:tcd:title-nonsort-2> <urn:tcd:title-main-2> <urn:tcd:title-partnumber-2> ).
<urn:tcd:title-main-2> a mads:MainTitleElement; mads:elementValue
  "Transactions of the Institution of Civil Engineers of Ireland".
<urn:tcd:title-nonsort-2> a mads:NonSortElement; mads:elementValue "The".
<urn:tcd:title-partnumber-2> a mads:PartNumberElement; mads:elementValue "Vol.26".
```

Listing 3. Relating TitleInfo to MADS-RDF with our R2RML extension.

5 Related Work

We focus on related word of generating RDF datasets from *relational databases* only. To the best of our knowledge, xR2RML [3] is the only initiative that aimed to extend

R2RML with support for containers and collections. It extends both R2RML for relational databases and RML [2], itself a superset of R2RML, to handle other source data formats such as JSON, XML, and CSV. At the time of writing, the implementation of xR2RML provides no support for named graphs, nested collections and containers, and different term types in collections and containers.[3]. We consider the first a non-implemented feature rather than a real limitation. Interesting about their approach is how they handled "representation agnostic" mappings allowing one to mix representation formats. One can, for instance, treat the contents of a column as JSON. This feature allows one to generate collections or containers for tables with such columns.

6 Conclusions and Future Work

This paper provides evidence that a *minimal* extension of R2RML to support the generation of RDF collections and containers from relational databases is feasible. The Library of Trinity College Dublin, who wished to generate RDF from their relational database using MADS-RDF, provided the motivation of this study, as those vocabularies prescribe the use of RDF collections for which there is no support in R2RML.

Our approach furthermore supports a wider range of cases than the one needed for our motivating use case; nesting collections/containers, collections/containers across rows, and dealing with empty collections and containers. Though the Library did not need to gather collections across rows, this could potentially be useful to generate a collection of disjoint OWL classes when generating a taxonomy from a table, for instance. With respect to existing state of the art, our approach covers a wider range of cases, and does not intermix data representation formats. We believe that this would ease the maintenance mappings, though evidence for this needs to be gathered.

Acknowledgements. The ADAPT Centre for Digital Content Technology is funded under the SFI Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund. We also express our gratitude to the Library of Trinity College Dublin (TCD) for providing us their data and Garg Abhivan who explored the development of an earlier prototype.

References

1. Das, S., Sundara, S, Cyganiak, R.: R2RML: RDB to RDF Mapping Language. W3C Recommendation (2012)
2. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A generic language for integrated RDF mappings of heterogeneous data. In: Bizer, C., Heath, T., Auer, S., Berners-Lee, T. (eds.) Proc. of the Workshop on Linked Data on the Web (LDOW 2014), CEUR Workshop Proceedings, vol. 1184. CEUR-WS.org (2014)
3. Michel, F., Djimenou, L., Faron-Zucker, C., Montagnat, J.: Translation of relational and non-relational databases into RDF with xR2RML. In: Monfort, V., Krempels, K., Majchrzak, T.A., Turk, Z. (eds.) Proc. of the 11th International Conference on Web Information Systems and Technologies (WEBIST 2015). pp. 443–454. SciTePress (2015)

[3] See https://github.com/frmichel/morph-xr2rml, last accessed March 23, 2017