R2RML by Assertion: A Semi-Automatic Tool for Generating Customised R2RML Mappings

Luís Eufrasio T. Neto[1], Vânia Maria P. Vidal[1], Marco A. Casanova[2], José Maria Monteiro[1]

[1] Federal University of Ceará, Fortaleza, CE, Brazil
{vaniap.vidal, luis.eufrasio, jmmfilho}@gmail.com
[2] Department of Informatics – Pontifical Catholic University of Rio de Janeiro, RJ, Brazil
casanova@inf.puc-rio.br

Abstract. In this paper, we demonstrate the RBA (**R**2RML **B**y **A**ssertion) tool which automatically generates customized R2RML mappings based on a set of semantic mappings that model the relationship between the relational database schema and a target ontology in RDF. The semantic mappings are specified by a set of correspondence assertions, which are simple to understand.

Keywords: Linked Data, RDB-to-RDF user interface, R2RML mappings.

1 Introduction

The Linked Data initiative [1] promotes the publication of previously isolated databases as interlinked RDF triple sets, thereby creating a global scale data space, known as the Web of Data. However, the full potential of Linked Data depends on how easy it is to transform data stored in conventional, relational databases (RDB) into RDF triples. This process is often called RDB-to-RDF.

There are two main approaches for mapping RDB to RDF: direct mapping and customized mapping. The direct mapping approach relies on automatic methods that derive an ontology from a relational schema and transform data according to that schema. The customized mapping approach lets an expert create a mapping between the relational schema and an existing target ontology. In this approach, the RDB-to-RDF process can be divided into two steps: mapping generation and mapping implementation. The mapping generation step results in a specification of how to represent RDB schema concepts in terms of RDF classes and properties in a target vocabulary of the designer's choice. The mapping is conceptual and enables different implementation styles. For example, it can be used to materialize the RDF view or to offer virtual access through an interface that queries the underlying database.

In fact, quite a few tools that support customized mappings have been developed, such as Triplify, D2R Server [2], and OpenLink Virtuoso. Early surveys of RDB-to-RDF tools pointed out that the tools typically adopt different and proprietary mapping languages for the mapping process and do not provide a way to easily generate customized mappings between a RDB schema and a given domain ontology.

Recently, the W3C RDB2RDF Working Group proposed a standard mapping language, called R2RML [3], to express RDB-to-RDF mappings. However, R2RML is somewhat difficult to use, which calls for the development of tools to support the definition and deployment of mappings using R2RML. In this demo, we will show ***RBA***, a tool that simplifies the task of generating and deploying customized R2RML mappings. The demo video is available at http://youtu.be/Un_UIrbHmqo.

2 Generating R2RML Mappings with RBA

R2RML is a language for expressing customized mappings from relational databases to RDF datasets. An R2RML mapping refers to logical tables to retrieve data from the input database. A logical table can be: (1) a base table; (2) a view; and (3) a valid SQL query (called an "R2RML view" because it emulates an SQL view without modifying the database). The use of views is a convenient solution to deal with complex mappings, which require data transformation, computation, or filtering before generating triples from the database.

In our approach, we use a three-level architecture for mapping RDB to RDF, which is depicted in Figure 1. The exported ontology (***EO***) models the RDF view exported by the data source. The vocabulary of the exported ontology is a subset of the target ontology vocabulary. The middle layer consists of a set of relational view schemas ***VS***, where ***VS*** is a direct transformation for the exported ontology ***E.*** The view schemas in ***VS*** can be implemented as relational views or R2RML views [3].

In our framework, the use of the middle layer ***VS***, help breaking the definition of the mappings into two stages: the definition of the *SQL mappings* and the definition of the *R2RML mappings*. The R2RML mapping from the views to the exported ontology is in fact one-to-one and is automatically generated based on ***VS***. Also, it simplifies maintaining the R2RML mapping to reflect changes to the database schemas.

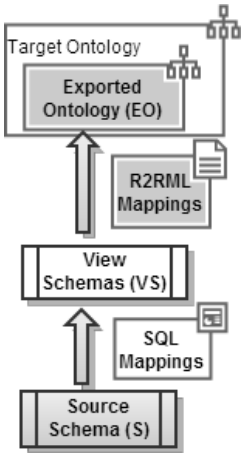

Fig. 1. 3-Level Schema Architecture

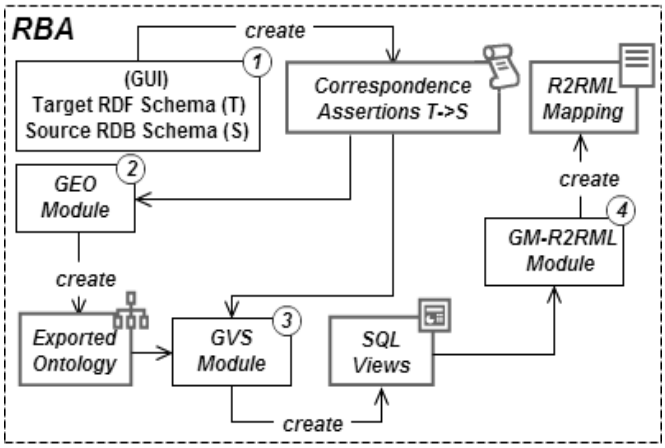

Fig. 2. ***RBA***: main components.

Figure 2 highlights the main components of *RBA*. The process for generating R2RML mappings with RBA, consists of four steps:

STEP 1: A session with RBA starts with the user loading a source and a target schema into the system. Then, the user can draw correspondence assertions (CAs) to specify the mapping between the target RDF schema and the source relational schema.

A CA can be: (i) a class correspondence assertion (CCA), which matches a class and a relation schema; (ii) an object property correspondence assertion (OCA), which matches an object property with paths (list of foreign keys) of a relation schema; or (iii) a datatype property correspondence assertion (DCA), which matches a datatype property with attributes or paths of a relation schema. CAs have a simple syntax and semantics, and yet suffice to capture most of the subtleties of mapping relational schemas into RDF schemas.

Figure 3 shows a snapshot of the GUI used for loading the schemas and defining the correspondence assertions. The target schema on the left contains 3 classes. The source relational schema on the right contains 3 tables. Given the two schemas, the user defines correspondence assertions from target to source.

As an example, consider the correspondence assertions shown in Figure 3:

CCA1: specifies that each tuple *t* in the *Authors* relation produces an RDF triple (we omit the translations from attribute values to RDF literals for simplicity):

```
http://example.com/person/t.authorID rdf:type foaf:Person .
```

CCA2 specifies that each tuple *t* in *Papers* relation produces the RDF triple:

```
<http://example.com/document/t.paperID> rdf:type foaf:Document .
```

DCA2 specifies that each tuple *t* in *Authors* produces the RDF triple:

```
<http://example.com/person/t.authorID> foaf:mbox "t.email" .
```

OCA1 specifies that for each pair <*t,t'*>, where *t* is a tuple in *Papers*, *t'* is a tuple in *Authors*, and exists *t"* in *Rel_Author_Paper* such that *t.paperID = t".paperID* and *t'. authorID = t".authorID*, it generates one triple:

```
<http://example.com/document/t.paperID>
        dc:creator <http://example.com/person/t'.authorID> .
```

OCA1 is a little more complex, but it can be graphically defined. The user just navigates through the path by clicking over the FK nodes in the source schema and the ***RBA*** tool builds the assertions (see the text field at the bottom of Figure 3).

As we will show, step 1 is in fact the the only step which is not automatic.

STEP 2: The *GEO (Generate Exported Ontology) module* automatically generates the *exported ontology (EO)*, which is induced by the correspondence assertions defined in Step 1. This task is very simple: GEO traverses the list of assertions created and includes in EO all classes and properties mapped. Figure 4 shows the exported ontology for the assertions in Fig 3.

STEP 3: The *GVS (Generate Views Schema) module* automatically generates the set of relational view schemas, which constitutes a direct transformation of the exported ontology generated in Step 2. The views schemas can be implemented as SQL views or R2RML views and ***RBA*** allows the user to decide the kind of views will be created; "Relational Views" or "R2RML Views" (see radio buttons in Figure 4). The first type creates the views in the source database and the second creates them directly in the R2RML mapping. In [4], we present an algorithm that automatically generates

the view schemas based on the correspondence assertions and exported ontology. Figure 5 shows the three SQL views generated for our case study.

STEP 4: The *GM-R2RML (Generate Mapping – R2RML) module* automatically generates a R2RML mapping from the views to the exported ontology. In [4], we present an algorithm that automatically generates the R2RML mappings from the CAs from the views to the exported ontology, which is one to one. The R2RML mapping is trivial, as shown in figure 6. After the creation of the R2RML mapping, probably the user needs to publish it to make some SPARQL queries over the RDF data. RBA emulates the D2RQ Server [2] with some customizations.

As discussed before, D2RQ has its own proprietary mapping language D2RM and until this moment there is not any production version that supports R2RML. To fill this gap, we developed a customized version of D2RQ (D2RQ-Ext) to support the main terms of R2RML and we embedded it as a component inside *RBA*. The limitation here is that D2RQ-Ext does not support "R2RML Views", but it supports "Relational Views" which allows the publication of mappings when user chooses this kind of Views on Step 3.

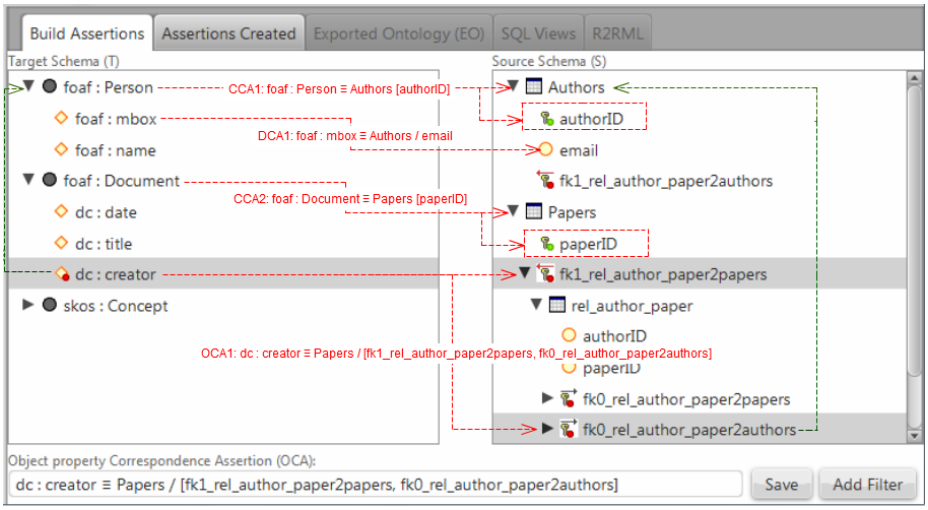

Fig. 3. RBA GUI snapshot.

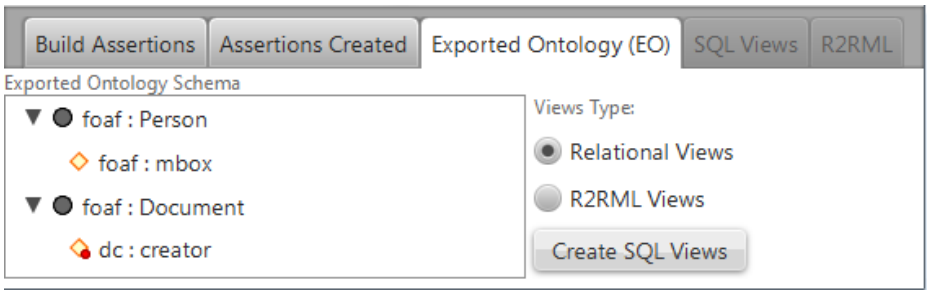

Fig. 4. Exported Ontology

```
CREATE OR REPLACE VIEW PERSON_VIEW AS
SELECT authors.AuthorID as ID, authors.Email as mbox
FROM authors;

CREATE OR REPLACE VIEW DOCUMENT_VIEW AS
SELECT papers.PaperID as ID
FROM papers

CREATE OR REPLACE VIEW DOCUMENT_CREATOR_VIEW AS
SELECT papers.PaperID as ID_DOCUMENT, authors.AuthorID as ID_PERSON
FROM papers, rel_author_paper, authors
WHERE papers.PaperID = rel_author_paper.PaperID
AND rel_author_paper.AuthorID = authors.AuthorID
```

Fig. 5. SQL Views

```
<#TriplesMapPerson>
    rr:logicalTable [ rr:tableName "PERSON_VIEW" ];
    rr:subjectMap [ rr:template "person/{ID}";
                    rr:class foaf:Person;];
    rr:predicateObjectMap [
        rr:predicate foaf:mbox;
        rr:objectMap [ rr:column "mbox" ];].
<#TriplesMapDocument>
    rr:logicalTable [ rr:tableName "DOCUMENT_VIEW" ];
    rr:subjectMap [ rr:template "document/{ID}";
                    rr:class foaf:Document;];
<#TriplesMapDocumentCreator>
    rr:logicalTable [rr:tableName "DOCUMENT_CREATOR_VIEW"];
    rr:subjectMap [rr:template "document/{ID_DOCUMENT}";];
    rr:predicateObjectMap [
        rr:predicate dc:creator;
        rr:objectMap [
            rr:parentTriplesMap <#TriplesMapPerson>;
            rr:joinCondition [rr:child "ID_PERSON";
                              rr:parent "ID";];];].
```

Fig. 6. R2RML mapping

The demo video is available at http://youtu.be/Un_UIrbHmqo. It shows how the ***RBA*** GUI helps the user graphically define CAs, and outlines the steps implemented by each module of the tool. It also shows some practical applications of the tool in a real world scenario. To the best of our knowledge, no commercial tool is available for generating customized R2RML mappings at the level of complexity of ***RBA***.

References

1. Berners-Lee, T: Linked Data, http://www.w3.org/DesignIssues/LinkedData.html (2006).
2. Bizer, C., and Cyganiak, R.: D2R Server – Publishing Relational Databases on the Semantic Web, ISWC (2006).
3. Das, S., Sundara, S., and Cyganiak, R.: R2RML: RDB to RDF Mapping Language, W3C Working Draft, http://www.w3.org/TR/r2rml/ (2012).
4. Vidal, V., Casanova, M., Neto, L.: Towards Automatic Generation of R2RML Mappings, http://goo.gl/fvZRt (2013). Submitted.