

# A Semi-Automatic Approach for Generating Customized R2RML Mappings

Vânia Maria Pontes Vidal  
Federal University of Ceará  
Fortaleza, CE, Brazil  
vvidal@lia.ufc.br

Marco A. Casanova  
Department of Informatics  
PUC-Rio  
Rio de Janeiro, RJ, Brazil  
casanova@inf.puc-rio.br

Luís Eufrazio T. Neto  
José Maria Monteiro  
Federal University of Ceará  
Fortaleza, CE, Brazil  
luis.eufrazio@gmail.com  
jmmfilho@gmail.com

## ABSTRACT

The Linked Data initiative brought new opportunities for building the next generation of Web applications. However, the full potential of linked data depends on how easy it is to transform data stored in relational databases into RDF triples. Recently, the W3C RDB2RDF Working Group proposed a mapping language, called R2RML, to specify mappings between relational schemas and RDF vocabularies. However, the specification of R2RML mappings is not an easy task. This paper therefore proposes a strategy to simplify the specification of R2RML mappings. The paper first introduces correspondence assertions, which provide a convenient way to manually model mappings between relational schemas and RDF vocabularies. Then, the paper describes a method to automatically generate R2RML mappings from the correspondence assertions.

## Categories and Subject Descriptors

H.2.1 [Database Management]: Logical Design – data models, normal forms, schema and subschema.

## General Terms

Design, Standardization and Languages.

## Keywords

Correspondence Assertions, RDB-to-RDF, R2RML, Linked Data.

## 1. INTRODUCTION

The Linked Data initiative [2] promotes the publication of previously isolated databases as interlinked RDF triple sets, thereby creating a global scale dataspace, known as the Web of Data. However, the full potential of linked data depends on how easy it is to transform data stored in relational databases (RDBs) into RDF triples. This process is often called RDB-to-RDF [10].

There are two main approaches for mapping relational databases into RDF. The *direct mapping approach* [12] relies on automatic methods to derive ontologies that directly reflect relational schemas and to transform the relational data into RDF triples.

The *customized mapping approach* relies on the designer to specify a mapping between the relational schema and an RDF vocabulary and results in a specification of how to represent relational schema concepts in terms of RDF classes and properties of the designer's choice. Quite a few tools have been developed to support the customized mapping approach, such as Triplify [1], D2R Server [3,4], and OpenLink Virtuoso [9]. However, early surveys of RDB-to-RDF tools [11,12] pointed out that the tools typically adopt proprietary mapping languages.

As a reaction, the W3C RDB2RDF Working Group proposed a standard mapping language, called R2RML [7], to express RDB-to-RDF mappings. However, R2RML is somewhat difficult to use, which calls for the development of methods and tools to support the deployment of mappings using R2RML.

This paper has two contributions. First, it proposes correspondence assertions [13] as a convenient way to manually specify mappings between relational schemas and RDF vocabularies. Second, it describes a method to automatically generate R2RML mappings from correspondence assertions.

Briefly, the method has two steps: (1) the manual specification of a set of correspondence assertions between the base relational schema and the vocabulary of a domain ontology of the user's choice, which results in an exported ontology; (2) the automatic generation of relational views and R2RML mappings, based on the result of the first step. The views are defined by projection-selection-equi-join queries, absorb the complexity of the mappings and are directly mapped to RDF. Thereby, the method supports most types of data restructuring used in data publishing.

This paper is organized as follows. Section 2 presents the R2RML language and a motivating example. Section 3 defines the concept of correspondence assertion. Section 4 describes the approach for the automatic generation of R2RML mappings, based on correspondence assertions. Section 5 contains the conclusions.

## 2. R2RML MAPPING LANGUAGE AND A MOTIVATING EXAMPLE

R2RML is a language designed to express 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 a base table, a view or a valid SQL query (called an "R2RML view" because it emulates an SQL view without modifying the database).

Each logical table is mapped to RDF using a *triples map*, which is a rule to map each row in a logical table to a set of RDF triples. The rule has two main parts: a subject map and multiple

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'14, March 24-28, 2014, Gyeongju, Korea.

Copyright 2014 ACM 978-1-4503-2469-4/14/03...\$15.00.

predicate-object maps. The *subject map* generates the subject of all RDF triples generated from a logical table row. The *predicate-object maps* in turn consist of *predicate maps* and *object maps* (or referencing object maps). Triples are generated by combining the subject map with a predicate map or an object map and by applying these three maps to each logical table row.

As an example, Figure 1 depicts the relational schema *ISWC\_REL*. Each table has a primary key, whose name ends with 'ID'. *Persons* and *Papers* represent the main concepts. The attribute *conference* of *Papers* is a foreign key to *Conferences*. *Rel\_Person\_Paper* represents an N:M relationship between *Persons* and *Papers*. The labels of the arcs, such as *Fk\_Publications*, are the names of the foreign keys.

Figure 2 depicts the ontology *CONF\_OWL*, which reuses terms from *FOAF* (Friend of a Friend), *SKOS* (Knowledge Organization System), *VCARD* and *DC* (Dublin Core). We use the prefix "conf" for the new terms defined in the *CONF\_OWL* ontology.

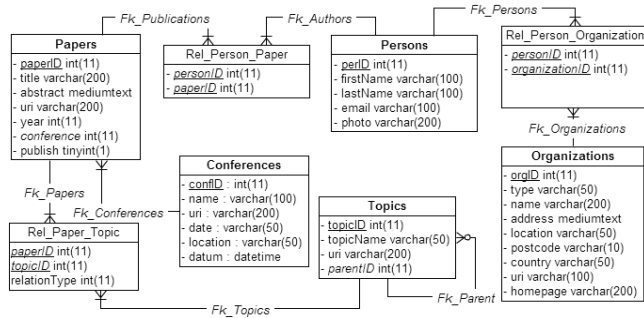


Figure 1. The ISWC\_REL Database Schema.

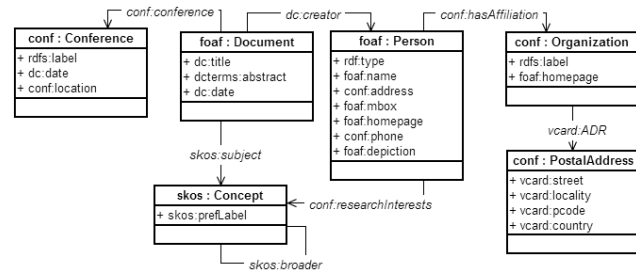


Figure 2. The CONF\_OWL Ontology.

The R2RML mapping in Table 1 refers to the base table *Persons\_Rel* (line 2). This mapping specifies that each tuple *t* in *Persons\_Rel* produces three RDF triples:

`<http://example.com/person/t.perID> rdf:type foaf:Person.`

(Generated from the subject map in lines 3-5)

`<http://example.com/person/t.perID> foaf:name concat(t.firstname, t.lastname).`

(Generated from the predicate map in lines 6-8)

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

(Generated from the predicate map in lines 9-11)

The R2RML mapping in Table 2 refers to the base table *Organizations\_Rel* (line 2). This mapping specifies that each tuple *t* in *Organizations\_Rel* produces three RDF triples:

`<http://example.com/org/t.orgID> rdf:type conf:Organization.`

(Generated from the subject map in lines 3-5)

`<http://example.com/org/t.orgID> foaf:label "t.name".`

(Generated from the predicate map in lines 6-8)

`<http://example.com/org/t.orgID> foaf:homepage t.homepage".`

(Generated from the predicate map in lines 9-11)

The R2RML mapping in Table 3 refers to the base table *Rel\_Person\_Organization* (line 2) to link individuals of class *foaf:Person* with individuals of class *conf:Organization* via *conf:hasAffiliation* object property. Attribute *perID* is used to compose the same URI of *PersonTriplesMap* (line 3 of Table 3 is equal to line 4 of Table 1). This mapping performs a join between *Rel\_Person\_Organization* and *Organizations\_Rel*, on the *organizationID* and *orgID* attributes (lines 9-11). The mapping specifies that, for each pair  $\langle t, t' \rangle$ , where *t* is a tuple in *Rel\_Person\_Organization*, *t'* is a tuple in *Organizations\_Rel*, and *t.organizationID* = *t'.orgID*, one triple must be generated:

`<http://example.com/person/t.perID> conf:hasAffiliation  
<http://example.com/org/t'.orgID> .`

The R2RML mapping in Table 4 refers to the R2RML view, *RV*, defined in lines 3-6. The object map in lines 9-12 maps tuples of *RV* to triples of the object property *conf:researchInterests*. Note that this mapping cannot be directly defined over a base table, because the *rr:joinCondition* term in an *rr:objectMap* can have only one foreign key referenced. So, in this case, we have to define a view to directly relate a person with his/her topics of interest. The mapping specifies that, for each pair  $\langle t, t' \rangle$ , where *t* is a tuple in *RV*, *t'* is a tuple in *Rel\_Paper\_Topic* and *t.idTopic* = *t'.topicID*, one triple must be generated:

`<http://example.com/person/t.idPerson> conf:researchInterests  
<http://example.com/topic/t'.topicID> .`

Table 1. R2RML Mappings referring to *Persons\_Rel*

1	<code>&lt;#PersonTriplesMap&gt;</code>
2	<code>rr:logicalTable [ rr:tableName "Persons_Rel" ];</code>
3	<code>rr:subjectMap [</code>
4	<code>  rr:template "http://example.com/person/{perID}";</code>
5	<code>  rr:class foaf:Person; ];</code>
6	<code>rr:predicateObjectMap [</code>
7	<code>  rr:predicate foaf:name;</code>
8	<code>  rr:objectMap [rr:template "{firstName} {lastName}"];];</code>
9	<code>rr:predicateObjectMap [</code>
10	<code>  rr:predicate foaf:mbox;</code>
11	<code>  rr:objectMap [ rr:column "email" ]; ].</code>

Table 2. R2RML Mappings referring to *Organizations\_Rel*

1	<code>&lt;#OrgTriplesMap&gt;</code>
2	<code>rr:logicalTable [ rr:tableName "Organizations_Rel" ];</code>
3	<code>rr:subjectMap [</code>
4	<code>  rr:template "http://example.com/org/{orgID}";</code>
5	<code>  rr:class conf:Organization; ];</code>
6	<code>rr:predicateObjectMap [</code>
7	<code>  rr:predicate rdfs:label;</code>
8	<code>  rr:objectMap [ rr:column "name" ]; ];</code>
9	<code>rr:predicateObjectMap [</code>
10	<code>  rr:predicate foaf:homepage;</code>
11	<code>  rr:objectMap [ rr:column "homepage" ]; ].</code>

**Table 3.** R2RML Mappings referring to *Rel\_Person\_Organization*

1	<#HasAffiliationTriplesMap>
2	rr:logicalTable [rr:tableName "Rel_Person_Organization"];
3	rr:subjectMap [
4	rr:template "http://example.com/person/{perID}"; ];
5	rr:predicateObjectMap [
6	rr:predicate conf:hasAffiliation;
7	rr:objectMap [
8	rr:parentTriplesMap <#OrgTriplesMap>;
9	rr:joinCondition [
10	Rr:child "organizationID";
11	Rr:parent "orgID"; ]]; ];

**Table 4.** R2RML Mappings referring to an R2RML View

1	<#ResearchInterestsTriplesMap>
2	rr:logicalTable [
3	rr:sqlQuery ""
4	SELECT pe.perID as idPerson, rpt.topicID as idTopic
5	FROM Persons_Rel as pe, Rel_Person_Paper as rpp,
6	Papers_Rel as pa, Rel_Paper_Topic as rpt
7	WHERE pe.perID = rpp.personID and rpp.paperID =
8	pa.paperID and pa.paperID = rpt.paperID; ""
9	rr:subjectMap [
10	rr:template "http://example.com/person/{idPerson}"; ];
11	rr:predicateObjectMap [
12	rr:predicate conf:researchInterests;
13	rr:objectMap [
14	rr:template "http://example.com/topic/{ idTopic }"; ]]; ];

### 3. CORRESPONDENCE ASSERTIONS

#### 3.1 Basic Concepts and Notation

As usual, we denote a *relation scheme* as  $R[A_1, \dots, A_n]$  and adopt *mandatory* (or *not null*) *attributes*, *keys*, *primary keys* and *foreign keys* as relational constraints. In particular, we use  $F(R:L, S:K)$  to denote a foreign key, named  $F$ , where  $L$  and  $K$  are lists of attributes from  $R$  and  $S$ , respectively, with the same length. We also say that  $F$  *relates*  $R$  and  $S$ .

A *relational schema* is a pair  $\mathcal{S}=(\mathbf{R}, \mathbf{Q})$ , where  $\mathbf{R}$  is a set of relation schemes and  $\mathbf{Q}$  is a set of relational constraints such that:

- (i)  $\mathbf{Q}$  has a unique primary key for each relation scheme in  $\mathbf{R}$ ;
- (ii)  $\mathbf{Q}$  has a mandatory attribute statement for each attribute which is part of a key or primary key;
- (iii) if  $\mathbf{Q}$  has a foreign key of the form  $F(R:L, S:K)$ , then  $\mathbf{Q}$  also has a constraint indicating that  $K$  is the primary key of  $S$ . The *vocabulary* of  $\mathcal{S}$  is the set of relation names, attribute names, etc. used in  $\mathcal{S}$ . Given a relation scheme  $R[A_1, \dots, A_n]$  and a *tuple variable*  $t$  over  $R$ , we use  $t.A_k$  to denote the *projection* of  $t$  over  $A_k$ . We use *selections* over relation schemes, defined as usual.

Let  $\mathcal{S}=(\mathbf{R}, \mathbf{Q})$  be a relational schema and  $R$  and  $T$  be relation schemes of  $\mathcal{S}$ . A list  $\varphi=[F_1, \dots, F_{n-1}]$  of foreign key names of  $\mathcal{S}$  is a *path from*  $R$  *to*  $T$  iff there is a list  $R_1, \dots, R_n$  of relation schemes of  $\mathcal{S}$  such that  $R_1=R$ ,  $R_n=T$  and  $F_i$  *relates*  $R_i$  and  $R_{i+1}$ . We say that tuples of  $R$  *reference* tuples of  $T$  *through*  $\varphi$ . A path  $\varphi$  is an *association path* iff  $\varphi=[F_1, F_2]$ , where the foreign keys are of the forms  $F_1(R_2:L_2, R_1:K_1)$  and  $F_2(R_2:M_2, R_3:K_3)$ . For example, consider the relational schema *ISWC\_REL* in Figure 1. The list of foreign keys names  $[Fk\_Publications, Fk\_Authors]$  is an association path from *Papers* to *Persons*, but  $[Fk\_Publications, Fk\_Persons]$  is not even a path.

We also recall a minimum set of concepts related to ontologies. A *vocabulary*  $\mathcal{V}$  is a set of *classes*, *object properties* and *datatype*

*properties*. An *ontology* is a pair  $\mathcal{O}=(\mathcal{V}, \Sigma)$  such that  $\mathcal{V}$  is a vocabulary and  $\Sigma$  is a finite set of formulae in  $\mathcal{V}$ , the *constraints* of  $\mathcal{O}$ . Among the constraints, we consider those that define the *domain* and *range* of a property, as well as *cardinality constraints*, defined in the usual way.

#### 3.2 Definition of Correspondence Assertions

This section introduces the notion of correspondence assertion, leaving examples to Section 3.3. Let  $\mathcal{S}=(\mathbf{R}, \mathbf{Q})$  be a relational schema and  $\mathcal{O}=(\mathcal{V}, \Sigma)$  be an ontology and assume that  $\Sigma$  has constraints defining the domain and range of each property.

**Definition 3.1:** A *class correspondence assertion (CCA)* is an expression of one of following forms:

- (i)  $\Psi: C \equiv R[A_1, \dots, A_n]$
- (ii)  $\Psi: C \equiv R[A_1, \dots, A_n] \sigma$

where  $\Psi$  is the *name* of the CCA,  $C$  is a class of  $\mathcal{V}$ ,  $R$  is a relation name of  $\mathcal{S}$ ,  $A_1, \dots, A_n$  are the attributes of the primary key of  $R$ , and  $\sigma$  is a selection over  $R$ . We also say that  $\Psi$  *matches*  $C$  with  $R$ .

**Definition 3.2:** An *object property correspondence assertion (OCA)* is an expression of one of following forms:

- (i)  $\Psi: P \equiv R$
- (ii)  $\Psi: P \equiv R / \varphi$

where  $\Psi$  is the *name* of the OCA,  $P$  is an object property of  $\mathcal{V}$ ,  $R$  is a relation name of  $\mathcal{S}$ , and  $\varphi$  is a path from  $R$ .

**Definition 3.3:** A *datatype property correspondence assertion (DCA)* is an expression of one of following forms:

- (i)  $\Psi: P \equiv R / A$
- (ii)  $\Psi: P \equiv R / \{A_1, \dots, A_n\}$
- (iii)  $\Psi: P \equiv R / \varphi / B$
- (iv)  $\Psi: P \equiv R / \varphi / \{B_1, \dots, B_n\}$

where  $\Psi$  is the name of the DCA,  $P$  is a datatype property of  $\mathcal{V}$ ,  $R$  is a relation name of  $\mathcal{S}$ ,  $A$  is an attribute of  $R$ ,  $A_1, \dots, A_n$  are attributes of  $R$ ,  $\varphi$  is a path from  $R$  to a relation schema  $T$ ,  $B$  is an attribute of  $T$ , and  $B_1, \dots, B_n$  are attributes of  $T$ .

**Definition 3.4:** A *mapping* between  $\mathcal{V}$  and  $\mathcal{S}$  is a set  $\mathcal{A}$  of correspondence assertions such that:

- (i) If  $\mathcal{A}$  has an OCA of the form  $P \equiv R$ , then  $\mathcal{A}$  must have a CCA that matches the domain of  $P$  with  $R$ , and a CCA that matches the range of  $P$  also with  $R$ .
- (ii) If  $\mathcal{A}$  has an OCA of the form  $P \equiv R / \varphi$ , where  $\varphi$  is a path from  $R$  to  $T$ , then  $\mathcal{A}$  must have a CCA that matches the domain of  $P$  with  $R$  and a CCA that matches the range of  $P$  with  $T$ .
- (iii) If  $\mathcal{A}$  has a DCA that matches a datatype property  $P$  in  $\mathcal{V}$  with a relation name  $R$  of  $\mathcal{S}$ , then  $\mathcal{A}$  must have a CCA that matches the domain of  $P$  with  $R$ .

#### 3.3 Transformation Rules

In this section, we introduce the notion of transformation rule and show how to interpret correspondence assertions as transformation rules. Let  $\mathcal{O}=(\mathcal{V}, \Sigma)$  be an ontology and  $\mathcal{S}=(\mathbf{R}, \mathbf{Q})$  be a relational schema, with vocabulary  $\mathcal{U}$ . Let  $\mathcal{X}$  be a set of *scalar variables* and  $\mathcal{T}$  be a set of *tuple variable*, disjoint from each other and from  $\mathcal{V}$  and  $\mathcal{U}$ .

A *literal* is a *range expression* of the form  $R(t)$ , where  $R$  is a relation name in  $\mathcal{U}$  and  $t$  is a tuple variable in  $\mathcal{T}$ , or a *built-in predicate* of one of the forms shown in Table 5. A *rule body*  $B$  is a list of literals. We use " $B[x_1, \dots, x_k]$ " to indicate that the tuple or

scalar variables  $x_1, \dots, x_k$  occur in  $B$  and  $R(t)$ ,  $B[t, x_1, \dots, x_k]$ ” to indicate that the rule body has a literal of the form  $R(t)$ .

A *transformation rule*, or simply a *rule*, is an expression of one of the forms:

- $C(x) \leftarrow B[x]$ , where  $C$  is a class in  $V$  and  $B[x]$  is a rule body
- $P(x, y) \leftarrow B[x, y]$ , where  $P$  is a property and  $B[x, y]$  is a rule body

Let  $A$  be a set of correspondence assertions that defines a mapping between  $V$  and  $S$ , that is,  $A$  satisfies the conditions stated in Definition 2.4. Assume that each class  $C$  in  $V$  is associated with a namespace prefix.

Table 6 shows the transformation rules *induced* by the correspondence assertions in  $A$ . For example, the rule on the right-hand side of Line 5 indicates that, for each tuple  $t$  of  $R$  such that  $t.A$  is not null, one should:

- Compute the URI  $s$  of the instance of domain  $D$  of  $P$  that  $t$  represents, using the class correspondence assertion  $\Psi_D$ :  $D(s) \leftarrow R(t), B_D[t, s]$ , where  $B_D[t, s]$  stands for “ $HasURI[\Psi](t, s)$ ”, if the CCA for  $D$  follows Line 1 of Table 6, or  $B_D[t, s]$  stands for “ $HasURI[\Psi](t, s), \sigma(t)$ ”, if the CCA for  $D$  follows Line 2 of Table 6
- Translate the value of  $A$  in tuple  $t$ , generating the literal  $v$
- Associate  $v$  as the value for property  $P$  of  $s$

Table 7 shows a set of correspondence assertions that specifies a mapping between *CONF\_OWL* and *ISWC\_REL*, obtained with the help of the tool described in [8]. For example, the transformation rules induced by *CCA1*, *DCA1* and *OCA2* are (we omit the translations from attribute values to RDF literals for simplicity):

*CCA1* specifies that each tuple  $t$  in *Persons* generates one triple:

`<http://example.com/person/t.perID> rdf:type foaf:Person.`

*DCA1* specifies that each tuple  $t$  in *Persons* generates one triple:

`<http://example.com/person/t.perID> foaf:name  
concat(t.firstname, t.lastname).`

*OCA2* specifies that, for each tuple  $t$  in *Person*, for each tuple  $t'$  in *Topics* such that  $t'$  is referenced by  $t$  through path  $[Fk\_Authors, Fk\_Publications, Fk\_Papers, Fk\_Topics]$ , one triple is generated:

`<http://example.com/person/t.perID> conf:researchInterests  
<http://example.com/org/t'.topicID>.`

**Table 5. Built-in predicates**

Built-in predicate	Intuitive definition
$nonNull(v)$	$nonNull(v)$ holds iff value $v$ is not null
$RDFLiteral(u, A, R, v)$	Given a value $u$ , an attribute $A$ of $R$ , a relation name $R$ , and a literal $v$ , $RDFLiteral(u, A, R, v)$ holds iff $v$ is the literal representation of $u$ , given the type of $A$ in $R$
$HasReferencedTuples[\varphi](t, u)$ where $\varphi$ is a path from $R$ to $T$	Given a tuple $t$ of $R$ and tuple $u$ of $T$ , $HasReferencedTuples[\varphi](t, u)$ holds iff $u$ is referenced by $t$ through path $\varphi$
$HasURI[\Psi](t, s)$ where $\Psi$ is a CCA for a class $C$ of $V$ , using attributes $A_1, \dots, A_n$ of $R$	Given a tuple $t$ of $R$ , $HasURI[\Psi](t, s)$ holds iff $s$ is the URI obtained by concatenating the namespace prefix for $C$ and the values of $t.A_1, \dots, t.A_n$
$concat([v_1, \dots, v_n], v)$	Given a list $[v_1, \dots, v_n]$ of string values, $concat([v_1, \dots, v_n], v)$ holds iff $v$ is the string obtained by concatenating $v_1, \dots, v_n$

**Table 6. Mapping Rules**

	Correspondence Assertion	Mapping Rule
1	$\Psi: C = R[A_1, \dots, A_n]$	$C(s) \leftarrow R(t), HasURI[\Psi](t, s)$
2	$\Psi: C = R[A_1, \dots, A_n] \sigma$	$C(s) \leftarrow R(t), HasURI[\Psi](t, s), \sigma(t)$
3	$\Psi: O = R$ where: - $A$ has a CCA $\Psi_D$ that matches the domain $D$ of $O$ with $R$ and $\Psi_D$ - $A$ has a CCA $\Psi_N$ that matches the range $N$ of $O$ with $R$ and $\Psi_N$ - $A$ has mapping rule $D(s) \leftarrow R(t), B_D[t, s]$ - $A$ has mapping rule $N(o) \leftarrow R(t), B_N[t, o]$	$P(s, o) \leftarrow R(t), B_D[t, s], B_N[t, o]$
4	$\Psi: O = R / \varphi$ where: - $\varphi$ is a path of $R$ to $T$ - $A$ has a CCA $\Psi_D$ that matches the domain $D$ of $O$ with $R$ and $\Psi_D$ - $A$ has a CCA $\Psi_N$ that matches the range $N$ of $O$ with $T$ and $\Psi_N$ - $A$ has mapping rule $D(s) \leftarrow R(t), B_D[t, s]$ - $A$ has mapping rule $N(o) \leftarrow T(u), B_N[u, o]$	$P(s, o) \leftarrow R(t), B_D[t, s], HasReferencedTuples[\varphi](t, u), T(u), B_N[u, o]$
5	$\Psi: P = R / A$ where: - $A$ has a CCA $\Psi_D$ that matches the domain $D$ of $P$ with $R$ and $\Psi_D$ - $A$ has mapping rule $D(s) \leftarrow R(t), B_D[t, s]$ - $A$ is an attribute of $R$	$P(s, v) \leftarrow R(t), B_D[t, s], nonNull(t.A), RDFLiteral(t.A, "A", "R", v)$
6	$\Psi: P = R / \varphi / A$ where: - $\varphi$ is a path of $R$ to $T$ - $A$ has a CCA $\Psi_D$ that matches the domain $D$ of $P$ with $R$ and $\Psi_D$ - $A$ has mapping rule $D(s) \leftarrow R(t), B_D[t, s]$ - $A$ is an attribute of $T$	$P(s, v) \leftarrow R(t), B_D[t, s], HasReferencedTuples[\varphi](t, u), nonNull(u.A), RDFLiteral(u.A, "A", "T", v)$
7	$\Psi: P = R / \{A_1, \dots, A_m\}$ where: - $A$ has a CCA $\Psi_D$ that matches the domain $D$ of $P$ with $R$ and $\Psi_D$ - $A$ has mapping rule $D(s) \leftarrow R(t), B_D[t, s]$ - $A_1, \dots, A_m$ are attributes of $R$	$P(s, v) \leftarrow R(t), B_D[t, s], nonNull(t.A_1), \dots, nonNull(t.A_m), RDFLiteral(t.A_1, "A_1", "R", v_1), \dots, RDFLiteral(t.A_m, "A_m", "R", v_m), concat([v_1, \dots, v_m], v)$
8	$\Psi: P = R / \varphi / \{A_1, \dots, A_m\}$ where: - $\varphi$ is a path of $R$ to $T$ - $A$ has a CCA $\Psi_D$ that matches the domain $D$ of $P$ with $R$ and $\Psi_D$ - $A$ has mapping rule $D(s) \leftarrow R(t), B_D[t, s]$ - $A_1, \dots, A_m$ are attributes of $T$	$P(s, v) \leftarrow R(t), B_D[t, s], HasReferencedTuples[\varphi](t, u), nonNull(u.A_1), \dots, nonNull(u.A_m), RDFLiteral(u.A_1, "A_1", "T", v_1), \dots, RDFLiteral(u.A_m, "A_m", "T", v_m), concat([v_1, \dots, v_m], v)$

**Table 7. Correspondence Assertions**

<i>CCA1</i>	<code>foaf:Person = Persons[perID]</code>
<i>CCA2</i>	<code>foaf:Document = Papers[paperID]</code>
<i>CCA3</i>	<code>conf:Organization = Organizations[orgID]</code>
<i>CCA4</i>	<code>conf:PostalAddress = Organizations[address, location, postcode, country]</code>
<i>CCA5</i>	<code>conf:Conference = Conferences[confID]</code>
<i>CCA6</i>	<code>skos:Concept = Topics[topicID]</code>
<i>OCA1</i>	<code>conf:hasAffiliation = Persons / [Fk_Persons, Fk_Organizations]</code>
<i>OCA2</i>	<code>conf:researchInterests = Persons / [Fk_Authors, Fk_Publications, Fk_Papers, Fk_Topics]</code>
<i>OCA3</i>	<code>vcard:ADR = Organizations</code>
<i>OCA4</i>	<code>skos:subject = Papers / [Fk_Papers, Fk_Topics]</code>
<i>OCA5</i>	<code>conf:conference = Papers / Fk_Conferences</code>
<i>OCA6</i>	<code>skos:broader = Topics / Fk_Parent</code>
<i>DCA1</i>	<code>foaf:name = Persons / {firstName, lastName}</code>

DCA2	<i>foaf:mbox</i> = <i>Persons</i> / <i>email</i>
DCA3	<i>rdfs:label</i> = <i>Organization</i> / <i>name</i>
DCA4	<i>foaf:homepage</i> = <i>Organization</i> / <i>homepage</i>
DCA5	<i>vcard:Street</i> = <i>Organizations</i> / <i>address</i>
DCA6	<i>vcard:locality</i> = <i>Organizations</i> / <i>location</i>
DCA7	<i>vcard:Pcode</i> = <i>Organizations</i> / <i>postcode</i>
DCA8	<i>vcard:country</i> = <i>Organizations</i> / <i>country</i>
DCA9	<i>dc:title</i> = <i>Papers</i> / <i>title</i>
DCA10	<i>dcterms:abstract</i> = <i>Papers</i> / <i>abstract</i>
DCA11	<i>dc:date</i> = <i>Papers</i> / <i>year</i>
DCA12	<i>skos:prefLabel</i> = <i>Topics</i> / <i>topicName</i>
DCA13	<i>rdfs:label</i> = <i>Conferences</i> / <i>name</i>
DCA14	<i>dc:date</i> = <i>Conferences</i> / <i>date</i>
DCA15	<i>conf:location</i> = <i>Conferences</i> / <i>location</i>

## 4. A STRATEGY FOR THE GENERATION OF R2RML MAPPINGS

### 4.1 Overview of the strategy

In this section, we present our strategy for the generation of customized R2RML mappings. The inputs are:

- $D = (V_D, C_D)$  is the target ontology
- $S$  is the relational schema that must be mapped to  $D$
- $A$  is a mapping, that is, a set of correspondence assertions between  $V_D$  and  $S$

The strategy has three layers, as depicted in Figure 3. The top layer features a *target ontology*, which is a domain ontology of the user's choice, and an *exported ontology E*, which models the exported RDF view. The vocabulary of the exported ontology must be a subset of the target ontology vocabulary. The middle layer consists of a set of relational view schemas  $VS$ . The middle layer helps 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  $VS$  to  $E$  can be automatically generated from the set of correspondence assertions between  $S$  and  $D$ . The bottom layer contains the source relational schema.

The strategy consists of two main steps, discussed in detail in the rest of this section: (1) the design of the exported ontology  $E$ ; and (2) the design of the set  $VS$  of relational views and the R2RML mapping  $M$  from  $VS$  to  $E$ .

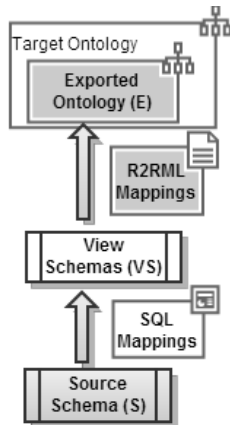


Figure 3. Three-Level Schema Strategy.

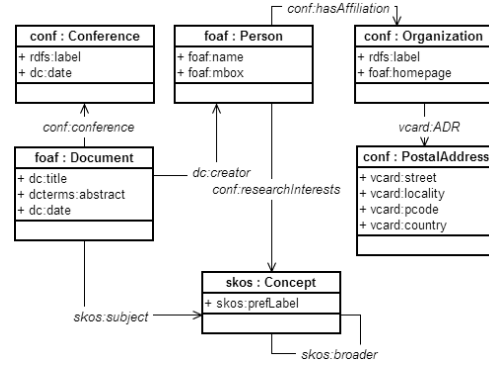


Figure 4. ISWC\_RDF Exported Ontology Schema.

### 4.2 Step 1: Design of the Exported Ontology

Recall that  $D=(V_D, C_D)$  is the target ontology,  $S$  is the relational schema that must be mapped to  $D$  and  $A$  is a set of correspondence assertions between  $V_D$  and  $S$ . We require that the exported ontology  $E=(V_E, C_E)$  be an open or a closed fragment [5] of  $D$ . This implies that  $V_E$  is a subset of  $V_D$  and that  $C_E$  can be automatically created from  $A$  and  $C_D$  [6].

We stress that the set of constraints  $C_E$  will typically include domain, range and cardinality constraints, which will be used in the second step, described in section 4.3.

Space limitations do not permit to describe Step 1 in detail. So, we illustrate Step 1 with our running example. Consider the *ISWC\_REL* schema in Figure 1, the *CONF\_OWL* domain ontology in Figure 2 and the correspondence assertions in Table 7. Figure 5 shows the exported ontology *ISWC\_RDF*. The vocabulary of *ISWC\_RDF* contains all elements of the *CONF\_OWL* ontology that match some element of *ISWC\_REL*.

### 4.3 Step 2: Design of Relational View and R2RML Mappings

Table 8 shows Algorithm 1, which automatically generates the relational view schemas  $VS$  for a given exported ontology  $E=(V_E, C_E)$  and the R2RML mappings from  $VS$  to  $E$ . Algorithm 1 has 3 main steps:

**Step 1:** For each class in  $V_E$ , Step 1 creates a relational view and the corresponding subject map, using template T1 in Table 9. The subject is obtained by concatenating the namespace for the class with the values of the primary key attributes of the relational view. So, the URI is unique.

**Step 2:** Step 2 processes datatype properties in  $V_E$ . Two cases are possible:

**Step 2.1.** If the datatype property has cardinality equal to 1, Step 2.1 generates an attribute and the corresponding predicate object map using template T2 in Table 9.

**Step 2.2.** If the datatype property has cardinality greater than 1, Step 2.2 generates a relation with a foreign key and an attribute and the corresponding subject and predicate object map using template T3 in Table 9.

**Step 3:** Step 3 processes object properties in  $V_E$ . Two cases are possible:

**Step 3.1.** If the object property has cardinality equal to 1, Step 3.1 generates a foreign key and the corresponding predicate object map using template T4 in Table 9.

**Step 3.2.** If the object property has cardinality greater than 1, Step 3.2 generates a relation with two foreign keys and the corresponding subject and predicate object maps using template T5 in Table 9.

Figure 5 depicts *ISWC Views*, the relational view schemas for the exported ontology *ISWC RDF* output by Algorithm 1. Table 10 shows some of the R2RML mappings generated using the templates in Table 9.

Finally, we observe that the view definitions for the relational view schemas can be automatically generated, based on the correspondence assertions between  $V_D$  and  $S$  (details omitted due to space limitation).

## 5. CONCLUSIONS AND FUTURE WORK

To facilitate the deployment of mappings using R2RML, we first introduced correspondence assertions to specify the mapping between a base relational schema and an RDF vocabulary. Then, we proposed an approach to automatically generate R2RML mappings, based on a set of correspondence assertions. The approach uses relational views as a middle layer, which facilitates the R2RML generation process and improves the maintainability of the mapping. The approach is backed up by a tool, described elsewhere [8].

As for the immediate future, we are extending the D2R Server to process R2RML mappings as a basis for the mapping implementation. The extension supports the mapping generation process described in Section 4.

## 6. ACKNOWLEDGEMENTS

This work was partly supported by CNPq, under grants 301497/2006-0, 475717/2011-2 and 57128/2009-9, by FAPERJ, under grants E-26/170028/2008 and E-26/103.070/2011, and by CAPES, under grant PROCAD/NF 1128/2010.

## 7. REFERENCES

- [1] Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., and Aumüller, D.: Triplify - Lightweight Linked Data Publication from Relational Databases. Proc. 18th Int'l. Conf. on World Wide Web (WWW 2009), pp. 621-630.
- [2] Berners-Lee, T: Linked Data, <http://www.w3.org/DesignIssues/LinkedData.html> (2006).
- [3] Bizer, C., and Cyganiak, R.: D2R Server – Publishing Relational Databases on the Semantic Web. Proc. 5th Int'l. Semantic Web Conf. (ISWC 2006).
- [4] Bizer, C.: D2R Map – A Database to RDF Mapping Language. Proc. 12th Int'l. Conf. on World Wide Web (WWW 2003).
- [5] Casanova, M. A., Breitman, K. K., Furtado A. L., Vidal, V. M., Macedo, J. A., Gomes, R. V., Salas, P. E.: The Role of Constraints in Linked Data. Proc. 11th Int'l. Conf. on On the move to meaningful internet systems - Volume Part II (OTM 2011), pp. 781-799.
- [6] Cullot, N., Ghawi, R., Ye' tongnon, K.: DB2OWL: A Tool for Automatic Database-to-Ontology Mapping. Proc. SEBD 2007, pp. 491-494.
- [7] Das, S., Sundara, S., and Cyganiak, R.: R2RML: RDB to RDF Mapping Language, W3C Working Draft, <http://www.w3.org/TR/r2rml/> (2012).
- [8] Neto, L., Vidal, V., Casanova, M., Monteiro J.: R2RML by Assertion: A Semi-Automatic Tool for Generating Customized R2RML Mappings. ESWC 2013.

- [9] OpenLink Virtuoso. <http://virtuoso.openlinksw.com>
- [10] Polfliet, S., Ichise, R.: Automated mapping generation for converting databases into linked data. Proc. 9th International Semantic Web Conference (ISWC2010).
- [11] Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr, T., Auer, S., Sequeda, J., Ezat, A.: A survey of current approaches for mapping of relational databases to RDF. W3C RDB2RDF Incubator Group report (2009).
- [12] Sequeda, J., Tirmizi, S., Corcho, O., Miranker, D.: Survey of directly mapping SQL databases to the Semantic Web (2011). *Knowledge Engineering Review*, 26, pp. 445-486.
- [13] Vidal, V. M., Araujo, V. S., Casanova, M. A.: Towards Automatic Generation of Rules for Incremental Maintenance of XML Views of Relational Data. Proc. Web Information Systems Engineering (WISE 2005), pp. 189-202

**Table 8. Algorithm 1**

<p><b>Step 1:</b> For each class <math>C</math> in <math>V_E</math> where <math>K_1, \dots, K_n</math> are the datatype properties of the key of <math>C</math> do</p> <p style="padding-left: 20px;">Create a relational view also named <math>C</math>;</p> <p style="padding-left: 20px;">Create <math>K_1, \dots, K_n</math>, the attributes of the primary key of view <math>C</math>;</p> <p style="padding-left: 20px;">Create the subject map referring to view <math>C</math> using template T1 in Table 9.</p> <p><b>Step 2:</b> For each datatype property <math>P</math> in <math>V_E</math> do</p> <p style="padding-left: 20px;">Let <math>D</math> be view that match to the domain of <math>P</math> and let <math>K_{D_1}, \dots, K_{D_n}</math> be the attributes of the primary key of <math>D</math>; // view <math>D</math> created in Step 1</p> <p style="padding-left: 20px;"><b>Case 2.1:</b> <math>P</math> has cardinality equal to 1.</p> <p style="padding-left: 40px;">Create attribute <math>P</math> in view <math>D</math> whose type is defined according to range of property <math>P</math>;</p> <p style="padding-left: 40px;">Create the predicate object map for <math>P</math> using template T2 in Table 9, which is added to the triples map of view <math>D</math>.</p> <p style="padding-left: 20px;"><b>Case 2.2:</b> <math>P</math> has cardinality greater than 1.</p> <p style="padding-left: 40px;">Create relational view <math>D_P</math>;</p> <p style="padding-left: 40px;">Create attributes <math>K_{D_1}, \dots, K_{D_n}</math> in <math>D_P</math> whose types are defined as in <math>D</math>;</p> <p style="padding-left: 40px;">Create foreign key <math>FK_{D_P}(D_P: \{K_{D_1}, \dots, K_{D_n}\}, D: \{K_{D_1}, \dots, K_{D_n}\})</math>;</p> <p style="padding-left: 40px;">Create attribute <math>P</math> in <math>D_P</math> whose type is defined according to range of property <math>P</math>;</p> <p style="padding-left: 40px;">Create the subject map referring to view <math>D_P</math>, and predicate object map for <math>P</math> using template T3 in Table 9.</p> <p><b>Step 3:</b> For each object property <math>P</math> in <math>V_E</math> do</p> <p style="padding-left: 20px;">Let <math>D</math> and <math>R</math> be the views that match to the domain and range of <math>P</math>, respectively, let <math>K_{D_1}, \dots, K_{D_n}</math> be the attributes of the primary key of <math>D</math> and let <math>K_{R_1}, \dots, K_{R_n}</math> be the attributes of the primary key of <math>R</math>; // views <math>D</math> and <math>R</math> were created in Step 1</p> <p style="padding-left: 20px;"><b>Case 3.1:</b> <math>P</math> has cardinality equal to 1.</p> <p style="padding-left: 40px;">Create attributes <math>K_{R_1}, \dots, K_{R_n}</math> in <math>D</math> whose types are defined as in <math>R</math>;</p> <p style="padding-left: 40px;">Create foreign key <math>FK_{D_P}(D: \{K_{R_1}, \dots, K_{R_n}\}, R: \{K_{R_1}, \dots, K_{R_n}\})</math></p> <p style="padding-left: 40px;">Create the predicate object map for <math>P</math> using template T4 in Table 9, which is added to the triples map of view <math>D</math>.</p> <p style="padding-left: 20px;"><b>Case 3.2:</b> <math>P</math> has cardinality greater than 1.</p> <p style="padding-left: 40px;">Create relational view <math>D_P</math>;</p> <p style="padding-left: 40px;">Create attributes <math>K_{D_1}, \dots, K_{D_n}</math> in <math>D_P</math> whose types are defined as in <math>D</math>;</p> <p style="padding-left: 40px;">Create foreign key <math>FK_{D_P}(D_P: \{K_{D_1}, \dots, K_{D_n}\}, D: \{K_{D_1}, \dots, K_{D_n}\})</math>;</p> <p style="padding-left: 40px;">Create attributes <math>K_{R_1}, \dots, K_{R_n}</math> in <math>D_P</math> whose types are defined as in <math>R</math>;</p> <p style="padding-left: 40px;">Create foreign key <math>FK_{D_P}(D_P: \{K_{R_1}, \dots, K_{R_n}\}, R: \{K_{R_1}, \dots, K_{R_n}\})</math>;</p> <p style="padding-left: 40px;">Create the subject map referring to view <math>D_P</math> and predicate object map for <math>P</math> using template T5 in Table 9.</p>
---

**Table 9. Templates to translate CAs to R2RML mappings**

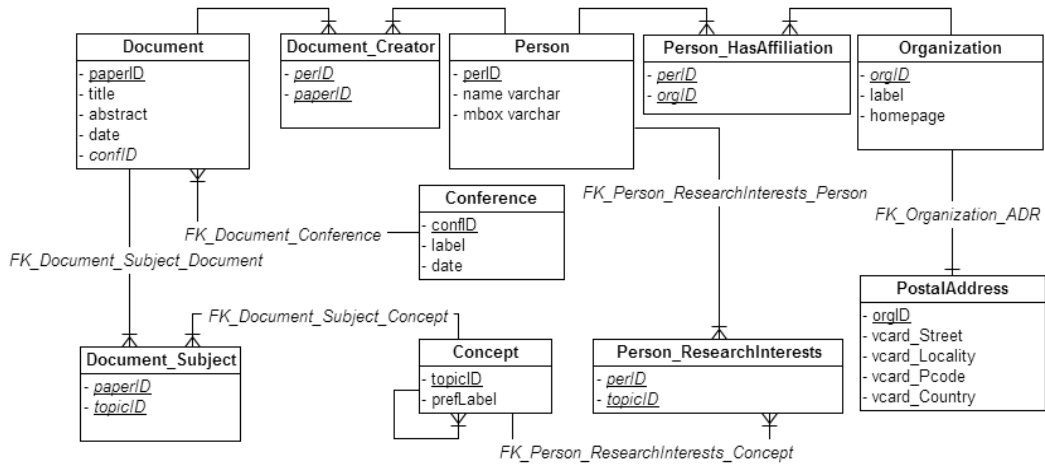
T1	<pre>&lt;#C_TriplesMap&gt; rr:logicalTable [ rr:tableName "C" ]; rr:subjectMap [   rr:template "namespaceOfC/{K1}/{K2}/.../{Kn}";   rr:class C; ];</pre>
T2	<pre>rr:predicateObjectMap [   rr:predicate P;   rr:objectMap [ rr:column "P" ]; ];</pre>
T3	<pre>&lt;#D_P_TriplesMap&gt; rr:logicalTable [ rr:tableName "D_P" ]; rr:subjectMap [   rr:template "namespaceOfD/{Kd1}/{Kd2}/.../{Kdn}";   rr:class D; ]; rr:predicateObjectMap [   rr:predicate P;   rr:objectMap [ rr:column "P" ]; ];</pre>
T4	<pre>rr:predicateObjectMap [   rr:predicate P;   rr:objectMap [     rr:parentTriplesMap &lt;R_TriplesMap&gt;;     rr:joinCondition [       rr:child "KR1";       rr:parent "KR1"; ];     ...     rr:joinCondition [       rr:child "KRn";       rr:parent "KRn"; ]; ]; ];</pre>
T5	<pre>&lt;#D_P_TriplesMap&gt; rr:logicalTable [ rr:tableName "D_P" ]; rr:subjectMap [   rr:template "namespaceOfD/{Kd1}/{Kd2}/.../{Kdn}";   rr:class D; ]; rr:predicateObjectMap [   rr:predicate P;   rr:objectMap [     rr:parentTriplesMap &lt;R_TriplesMap&gt;;     rr:joinCondition [       rr:child "KR1";       rr:parent "KR1"; ]; ];   ...   rr:joinCondition [     rr:child "KRn";     rr:parent "KRn"; ]; ]; ];</pre>

**Table 10. Two examples of R2RML mappings generated using the Templates in Table 9**

```
<#Person_TriplesMap>
rr:logicalTable [ rr:tableName "Person" ];
rr:subjectMap [
  rr:template "http://xmlns.com/foaf/0.1/person/{perID}";
  rr:class foaf:Person; ];
rr:predicateObjectMap [
  rr:predicate foaf:name;
  rr:objectMap [ rr:column "name" ]; ].

<#Concept_TriplesMap>
rr:logicalTable [ rr:tableName "Concept" ];
rr:subjectMap [
  rr:template
"http://www.w3.org/2004/02/skos/core/concept/{topicID}";
  rr:class skos:Concept; ].

<#Person_ResearchInterests_TriplesMap>
rr:logicalTable [ rr:tableName "Person_ResearchInterests" ];
rr:subjectMap [
  rr:template "http://xmlns.com/foaf/0.1/person/{perID}";
  rr:class foaf:Person; ];
rr:predicateObjectMap [
  rr:predicate conf:researchInterests;
  rr:objectMap [
    rr:parentTriplesMap <Concept_TriplesMap>;
    rr:joinCondition [
      rr:child "topicID";
      rr:parent "topicID"; ]; ]; ].
```



**Figure 5. ISWC\_View Schemas.**