# Towards Automatic Generation of R2RML Mappings

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

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

**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 conventional, relational databases into RDF triples. Recently, the W3C RDB2RDF Working Group proposed a standard mapping language, called R2RML, to specify how to represent existing relational data as RDF triples, expressed in a structure and in a target vocabulary of the designer's choice. In this paper, we propose an approach to automatically generate R2RML mappings based on a set on the 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, R2RML.

## 1 Introduction

The Linked Data initiative [2] brought new opportunities for building the next generation of Web applications. Indeed, it promotes the publication of previously isolated databases as interlinked RDF triple sets, thereby creating a global scale dataspace, known as the Web of Data. The success of the linked data initiative is partly due to the adoption of Web standards, such as URIs and HTTP, as well as Semantic Web standards, such as RDF and vocabularies. However, the full potential of linked data depends on how easy it is to transform data stored in conventional, relational databases (RDBs) into RDF triples. This process is often called RDB-to-RDF [12].

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 ontology from a relational schema and transform data according to that schema. A survey of systems based on direct mapping is presented in [14].

The customized mapping approach - the subject of this work - lets an expert create a mapping between the relational schema and an existing target ontology that is used to make the transformation into RDF. 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. It provides the ability to view exist-

ing relational data as RDF triples, expressed in a structure and 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 mapping have been developed, such as Triplify [1], D2R Server [3, 4], and OpenLink Virtuoso [9, 10]. Early surveys of RDB-to-RDF tools [12, 14] 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 [8], 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.

This paper has two major contributions. First, we propose the use of correspondence assertions [15] for specifying the semantic mapping between a target vocabulary and a base RDB schema. Then, we propose an approach to automatically generate R2RML mappings based on a set of correspondence assertions.

The remainder of this paper is organized as follows. Section 2 presents our mapping formalism. Section 3 introduces the case study used throughout the paper. Section 4 presents an overview of the R2RML Mapping Language. Section 5 exposes our approach for the automatic generation of customized R2RML mappings. Section 6 contains the conclusions and directions for future work.

## 2 Correspondence Assertions

### 2.1 Basic Concepts and Notation

In this section, we first recall a minimum set of concepts from the relational model and from ontologies, and introduce some basic notation.

A *relational alphabet* $\boldsymbol{U}$ consists of a set of *relation names*, *attribute names*, *key names*, *primary key names*, *foreign key names* and *tuple variables*.

A *relation scheme* is an expression of the form $R[A_1,....,A_n]$, where $R$ is a relation name in $\boldsymbol{U}$ and $A_1,....,A_n$ is a list of distinct attribute names in $\boldsymbol{U}$, called the *list of attributes* of $R$.

Let $R[A_1,....,A_n]$ *and* $S[B_1,....,B_m]$ be two relation schemes. The notions of *key*, *primary key*, and *mandatory* (or *not null*) attribute are defined as usual. A *foreign key* for $R$ *with respect to* $S$ is a statement of the form $FK(R:L,S:K)$, where $L$ is a list of attributes of $R$ and $K$ is a list of attributes of $S$ with the same length as $L$. A *relational constraint* is a key, a primary key, a foreign key or a mandatory attribute constraint.

A *relational schema* is a pair $\boldsymbol{S}=(\boldsymbol{R},\Omega)$, where $\boldsymbol{R}$ is a set of relation schemes and $\Omega$ is a set of relational constraints. The *vocabulary* of $\boldsymbol{S}$ is the set of relation names, attribute names, key names, primary key names, and foreign key names that occur in relation schemes in $\boldsymbol{R}$ or in constraints in $\Omega$. We also say that a relation name of the vocabulary of $\boldsymbol{S}$ is a relation name of $\boldsymbol{S}$, and likewise for the rest of the terms.

Let $R_1,...,R_n$ be relation schemes of a relational schema $\boldsymbol{S}$. Suppose that there exists a list of foreign keys $FK_i(R_i{:}F_i, R_{i+1}{:}F_{i+1})$ or $FK_i(R_{i+1}{:}F_{i+1}, R_i{:}F_i)$, for $1 \leq i \leq n\text{-}1$. Then, $\varphi = [FK_1,... ,FK_{n-1}]$ is a *path* from $R_1$ to $R_n$. We also say that tuples of $R_1$ *reference tuples of* $R_n$ *through* $\varphi$.

Given a relation scheme $R[A_1,....,A_n]$ and a tuple variable $t$ in $\boldsymbol{U}$, we use $t.A_k$ to denote the *projection* of $t$ over an attribute $A_k$ of $R$.

Finally, we use *selections* over relation schemes, defined as usual.

We also recall a minimum set of concepts related to ontologies. A *vocabulary* $\boldsymbol{V}$ is a set of *classes*, *object properties* and *datatype properties.* An *ontology* is a pair $\boldsymbol{O}=(\boldsymbol{V},\Sigma)$ such that $\boldsymbol{V}$ is a vocabulary and $\Sigma$ is a finite set of formulae in $\boldsymbol{V}$, the *constraints* of $\boldsymbol{O}$. 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.

### 2.2 Definition of the Correspondence Assertions

We propose the use of correspondence assertions (CA) for specifying the mapping between a target ontology and a relational schema.

A correspondence assertion 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 attributes or paths 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.

In what follows, let $\boldsymbol{O}=(\boldsymbol{V},\Sigma)$ be an ontology and $\boldsymbol{S}=(\boldsymbol{R},\Omega)$ be a relational schema.

**Definition 2.1:** A *class correspondence assertion (CCA)* is an expression of one of following forms:

(i) $\Psi: C \equiv R[A_1,...,A_n]$

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

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

**Definition 2.2:** An *object property correspondence assertion (OCA)* is an expression of one of following forms:

(i) $\Psi: O \equiv R / \varphi$

(ii) $\Psi: O \equiv R / NULL$

where $\Psi$ is the *name* of the OCA, $O$ is an object property of $\boldsymbol{V}$, $R$ is a relation name of $\boldsymbol{S}$ and $\varphi$ is a path of $R$. We also say that $\Psi$ *matches* $O$ with $R$.

**Definition 2.3:** A *datatype property correspondence assertion (DCA)* is an expression of one of following forms:

(i) $\Psi: P \equiv R / A$

(ii) $\Psi: P \equiv R / \{A_1,...,A_n\}$

(iii) $\Psi: P \equiv R / \varphi / B$

(iv) $\Psi: P \equiv R / \varphi / \{B_1,...,B_n\}$

where $\Psi$ is the name of the DCA, $P$ is a datatype property of $\boldsymbol{V}$, $R$ is a relation name of $\boldsymbol{S}$, $A$ is an attribute of $R$, $A_1,...,A_n$ are attributes of $R$, $\varphi$ is a path from $R$ to $R'$, $B$ is an attribute of $R'$, and $B_1,...,B_n$ are attributes of $R'$. We also say that $\Psi$ *matches* $P$ with $R$.

**Definition 2.4:** A *mapping* between $\boldsymbol{V}$ and $\boldsymbol{S}$ is a set $\boldsymbol{M}$ of correspondence assertions such that:

(i) If $\boldsymbol{M}$ has a DCA of the form $P \equiv R/\varepsilon$, then $\boldsymbol{M}$ has a CCA that matches the domain of $P$ with $R$;

(ii) If $\boldsymbol{M}$ has an OCA of the form $P \equiv R/\varphi$, where $\varphi$ is a path from $R$ to $R'$, then $\boldsymbol{M}$ has a CCA that matches the domain of $P$ with $R$, and a CCA that matches the range of $P$ with $R'$.

### 2.3 Transformation Rules generated by Correspondence Assertions

In this section, we first introduce the notion of transformation rule and then show how to interpret correspondence assertions as transformation rules.

In what follows, let $\boldsymbol{O}=(\boldsymbol{V},\Sigma)$ be an ontology, and $\boldsymbol{S}=(\boldsymbol{R},\Omega)$ be a relational schema over a relational alphabet $\boldsymbol{U}$. Let $\boldsymbol{X}$ be a set of *simples variables*, disjoint from $\boldsymbol{V}$ and $\boldsymbol{U}$.

A *literal* is a *range expression* of the form $R(t)$, where $R$ is a relation name in $\boldsymbol{U}$ and $t$ is a tuple variable in $\boldsymbol{U}$, or a *built-in predicate* of one of the forms shown in Table 1.

A *rule body* $B$ is a list of literals. When necessary, we use $B[x_1,...,x_k]$ to indicate that the tuple or simple variables $x_1,...,x_k$ occur in $B$.

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 $\boldsymbol{V}$ 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

In what follows, assume that each class $C$ in $\boldsymbol{V}$ is associated with a namespace prefix. Table 2 shows the transformation rules *induced* by the correspondence assertions. The transformations are rather straightforward to understand. For example, consider line 5 of the Table 2, repeated below:

| | |
|---|---|
| $\Psi: P \equiv R / A$ | $P(s,o) \leftarrow R(t), nonNull(t.A)$ |
| where $A$ is an attribute of $R$ and $\Psi_D$ is the CCA that matches the domain of $P$ with $R$ | $HasURI[\Psi_D](t,s),$ $RDFLiteral(t.A,"A","R",o)$ |

Intuitively, the rule on the right-hand side indicates that, for each tuple $t$ of $R$ such that $t.A$ is not null:

- one should compute $s$, the URI of the instance of domain $D$ of $P$ that $t$ represents, using the class correspondence assertion $\Psi_D$
- translate the value of $A$ in tuple $t$, generating the literal $o$
- associate $o$ as the value for property $P$ of $s$

Line 4 represents a special case, where the domain and range of property $P$ come from the same relation scheme $R$. It covers the case where a relation scheme corresponds to two classes and an object property that relates the two classes.

**Table 1.** 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[φ](t₁, tₙ)* where $\varphi$ is a path from $R_1$ to $R_n$ | Given a tuple $t_1$ of $R_1$ and tuple $t_n$ of $R_n$, *HasReferencedTuples[φ](t₁, tₙ)* holds iff $t_n$ is referenced by $t_1$ through path $\varphi$ |
| *HasURI[Ψ](t,u)* where $\Psi$ is a CCA for a class $C$ of $\boldsymbol{V}$, using attributes $A_1,...,A_n$ of $R$ (see Def. 2.1) | Given a tuple $t$ of $R$, *HasURI[Ψ](t,u)* holds iff $u$ is the URI obtained by concatenating the namespace prefix for $C$ and the values of $t.A_1,...,t.A_n$ |
| *concat([v₁,.., vₙ], o)* | Given a list $[v_1,.., v_n]$ of string values, *concat([v₁,.., vₙ], o)* holds iff $o$ is the string obtained by concatenating $v_1,..,v_n$ |

**Table 2.** Mapping Rules

| | | |
|---|---|---|
| 1 | $\Psi$: $C \equiv R[A_1,...,A_n]$ | $C(s) \leftarrow R(t), HasURI[\Psi](t,s)$ |
| 2 | $\Psi$: $C \equiv R[A_1,...,A_n]\sigma$ | $C(s) \leftarrow R(t), HasURI[\Psi](t,s),\sigma(t)$ |
| 3 | $\Psi$: $P \equiv R_1/\varphi$<br>where:<br>- $\varphi$ is a path of $R_1$ to $R_n$<br>- $\Psi_D$ is the CCA that matches the domain of $P$ with $R_1$<br>- $\Psi_R$ is the CCA that matches the range of $P$ with $R_n$ | $P(s,o) \leftarrow R_1(t_1),$<br>$HasReferencedTuples[\varphi](t_1,t_n),$<br>$HasURI[\Psi_D](t_1,s),$<br>$HasURI[\Psi_R](t_n,o)$ |
| 4 | $\Psi$: $P \equiv R / NULL$<br>where:<br>- $\Psi_D$ is the CCA that matches the domain of $P$ with $R$<br>- $\Psi_R$ is the CCA that matches the range of $P$ with $R$ | $P(s,o) \leftarrow R(t),$<br>$HasURI[\Psi_D](t,s),$<br>$HasURI[\Psi_R](t,o)$ |
| 5 | $\Psi$: $P \equiv R / A$<br>where:<br>- $A$ is an attribute of $R$<br>- $\Psi_D$ is the CCA that matches the domain of $P$ with $R$ | $P(s,o) \leftarrow R(t),$<br>$nonNull(t.A)$<br>$HasURI[\Psi_D](t,s),$<br>$RDFLiteral(t.A,"A","R",o)$ |
| 6 | $\Psi$: $P \equiv R_1/\varphi/A$<br>where:<br>- $\varphi$ is a path of $R_1$ to $R_n$<br>- $A$ is an attribute of $R_n$<br>- $\Psi_D$ is the CCA that matches the domain of $P$ with $R_1$ | $P(s,o) \leftarrow R_1(t_1),$<br>$HasURI[\Psi_D](t_1,s),$<br>$HasReferencedTuples[\varphi](t_1,t_n),$<br>$nonNull(t_n.A),$<br>$RDFLiteral(t_n.A,"A","R_1",o)$ |

| | | |
|---|---|---|
| 7 | $\Psi$: $\boldsymbol{P} \equiv R / \{A_1,...,A_m\}$<br>where:<br>- $\{A_1,...,A_m\}$ are attributes of $R$<br>- $\Psi_D$ is the CCA that matches the domain of $P$ with $R$ | $P(s,o) \leftarrow R(t)$,<br>$HasURI[\Psi_D](t, s)$,<br>$nonNull(t.A_1),\ldots,nonNull(t.A_m)$,<br>$RDFLiteral(t.A_1,"A_1","R",o_1)$,<br>…,<br>$RDFLiteral(t.A_m,"A_m","R",o_m)$,<br>$concat([o_1,..,o_m],o)$ |
| 8 | $\Psi$: $P \equiv R_1 / \varphi / \{A_1,...,A_m\}$<br>where:<br>- $\varphi$ is a path of $R_1$ to $R_n$<br>- $\{A_1,...,A_m\}$ are attributes of $R$<br>- $\Psi_D$ is the CCA that matches the domain of $P$ with $R$ | $P(s,o) \leftarrow R_1(t_1)$,<br>$HasURI[\Psi_D](t_1,s)$,<br>$HasReferencedTuples[\varphi](t_1,t_n)$,<br>$nonNull(t_n.A_1),\ldots,nonNull(t_n.A_m)$<br>$RDFLiteral(t_n.A_1,"A_1","R",o_1)$,<br>…,<br>$RDFLiteral(t_n.A_m,"A_m","R",o_m)$,<br>$concat([o_1,..,o_m],o)$ |

## 3 Running Example

In this Section, we present the relational database schema *ISWC_REL* and the ontology *CONF_OWL*, which are used as a case study throughout the paper. We also present a set of correspondence assertions, which specifies the mapping between *CONF_OWL* and *ISWC_REL*.

**Relational Schema.** Figure 1 depicts the database schema *ISWC_REL*, which represents authors and their publications, organizations which authors are affiliated to, topics referenced by publications and conferences which publications were submitted to. Each table has a distinct primary key, whose name ends with '*ID*', and which are of type integer. Tables *Persons_Rel* and *Papers_Rel* represent the main concepts. The attribute *conference* of table *Papers_Rel* is a foreign key to table *Conferences_Rel*. Table *Rel_Person_Paper* represents an N:M relationship between *Persons_Rel* and *Papers_Rel*.

**Ontology.** Figure 2 depicts the ontology *CONF_OWL*, which reuses terms from four well-known vocabularies: *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.

**Correspondence Assertions.** To begin to understand the relationship between the *ISWC_REL* schema and *CONF_OWL*, we may invoke a schema matcher to generate a set of matchings between *CONF_OWL* and *ISWC_REL*. Alternatively, we could ask a data designer familiar with the schemas to draw arcs between elements that contain related data. Table 3 shows a set of correspondence assertions which specifies the semantic mapping between *CONF_OWL* and *ISWC_REL*.

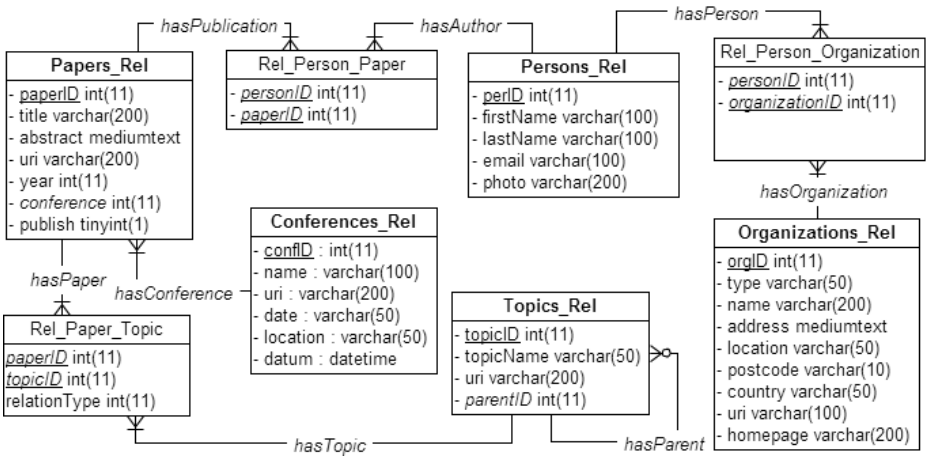


**Fig. 1.** ISWC_REL Database Schema

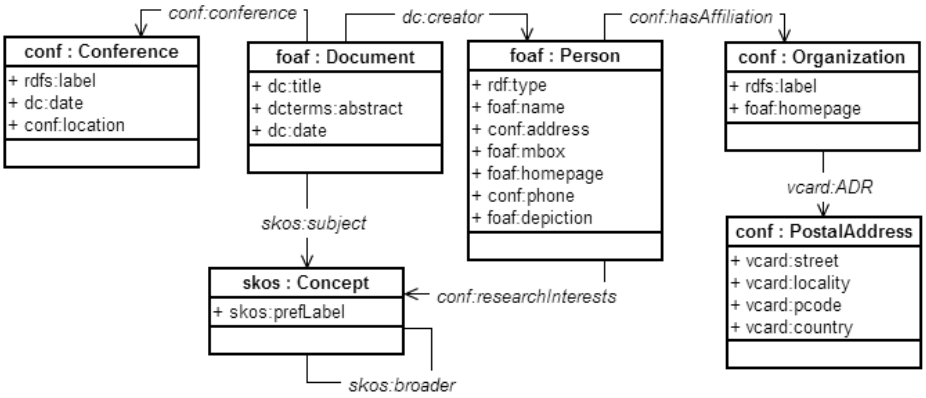


**Fig. 2.** CONF_OWL Target Ontology

**Table 3.** Correspondence Assertions

| | |
|---|---|
| *CCA1* | *foaf:Person ≡ Persons_Rel[perID]* |
| *CCA2* | *foaf:Document ≡ Papers_Rel[paperID]* |
| *CCA3* | *conf:Organization ≡ Organizations_Rel[orgID]* |
| *CCA4* | *conf:PostalAddress ≡ Organizations_Rel[orgID]* |
| *CCA5* | *conf:Conference ≡ Conferences_Rel[confID]* |
| *CCA6* | *skos:Concept ≡ Topics_Rel[topicID]* |
| *OCA1* | *conf:hasAffiliation ≡ Persons_Rel / [hasPerson, hasOrganization]* |
| *OCA2* | *conf:researchInterests ≡ Persons_Rel / [hasAuthor, hasPublication, hasPaper, hasTopic ]* |
| *OCA3* | *vcard:ADR ≡ Organizations_Rel / NULL* |
| *OCA4* | *skos:subject ≡ Papers_Rel / [hasPaper, hasTopic]* |

| | |
|---|---|
| *OCA5* | *conf:conference ≡ Papers_Rel / hasConference* |
| *OCA6* | *skos:broader ≡ Topics_Rel / hasParent* |
| *DCA1* | *foaf:name ≡ Persons_Rel / { firstName, lastName }* |
| *DCA2* | *foaf:mbox ≡ Persons_Rel / email* |
| *DCA3* | *rdfs:label ≡ Organization_Rel / name* |
| *DCA4* | *foaf:homepage ≡ Organization_Rel / homepage* |
| *DCA5* | *vcard:Street ≡ Organizations_Rel / address* |
| *DCA6* | *vcard:locality ≡ Organizations_Rel / location* |
| *DCA7* | *vcard:Pcode ≡ Organizations_Rel / postcode* |
| *DCA8* | *vcard:country ≡ Organizations_Rel /country* |
| *DCA9* | *dc:title ≡ Papers_Rel / title* |
| *DCA10* | *dcterms:abstract ≡ Papers_Rel / abstract* |
| *DCA11* | *dc:date ≡ Papers_Rel / year* |
| *DCA12* | *skos:prefLabel ≡ Topics_Rel / topicName* |
| *DCA13* | *rdfs:label ≡ Conferences_Rel / name* |
| *DCA14* | *dc:date ≡ Conferences_Rel / date* |
| *DCA15* | *conf:location ≡ Conferences_Rel / location* |

## 4 Translating Correspondence Assertions to R2RML Mappings

This section briefly overviews the R2RML mapping language, and illustrates how to translate correspondence assertions to R2RML mappings.

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).

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 predicate-object maps. The *subject map* generates the subject of all RDF triples that will be generated from a logical table row. The subjects are often URIs generated from the primary key values. 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 and object map, and applying these three maps to each logical table row.

In order to translate correspondence assertions to R2RML mappings, a subject map should be defined for each CCA, a predicate map for each DCA, and an object map for each OCA. In following, we show some examples of R2RML mappings generated for the CAs of our case study.

The R2RML mapping in Table 4 specifies R2RML mappings for table *Persons_Rel*. The mappings are generated from the CAs as follows:

- *CCA1* translates to the subject map for the target class *foaf:Person* (lines 3-5)

- *DCA1* translates to the predicate map for the datatype property *foaf:name* (6-8)
- *DCA2* translates to the predicate map for the datatype property *foaf:mbox* (9-11)

This mapping specifies that each tuple *t* in *Persons_Rel* produces three RDF triples (we omit the translations from attribute values to RDF literals for simplicity):

*<http://example.com/person/t.perID> rdf:type foaf:Person.*
*<http://example.com/person/t.perID> foaf:name concat(t.firstname, t.lastname).*
*<http://example.com/person/t.perID> foaf:mbox " t.email".*

Table 5 specifies the subject map for target class *conf:organization* generated from *CCA3*. This mapping specifies that each tuple *t* in *Organizations_Rel* produces one RDF triple:

*<http://example.com/org/t.orgID> rdf:type conf:organization.*

Table 6 specifies the object map for object property *conf:hasAffiliation* generated from *OCA1*. This mapping performs a join between *Rel_Person_Organization* and *Organizations_Rel*, on the *organizationID* and *orgID* attributes. The mapping specifies that for each pair *<t,t'>*, where *t* is a tuple in *Rel_Person_Organization*, *t'* is a tuple in *Organizations_Rel*, and *t.organizationID = t'.orgID*, it generates one triple:

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

Table 7 specifies the object map for the object property *conf:researchInterests* generated from *OCA2*. The object map in lines 8-14 maps tuples of the R2RML view, defined in lines 3-6, to triples of the object property *conf:researchInterests*. Note that the path of *OCA2* is composed of four foreign keys: *hasAuthor*, *hasPublication*, *hasPaper*, and *hasTopic*. The problem in this example is that the *rr:joinCondition* term in an *rr:objectMap* may 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.

**Table 4.** R2RML Mappings generated from *CCA1*, *DCA1* and *DCA2*

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

**Table 5.** Subject Map generated from *CCA3*

```
1  <#OrgTriplesMap>
2    rr:logicalTable [ rr:tableName "Organizations_Rel" ];
3    rr:subjectMap [
4      rr:template "http://example.com/org/{orgID}";
5      rr:class conf:Organization; ].
```

**Table 6.** R2RML Mappings generated from *CCA1* and *OCA1*

```
<#HasAffiliationTriplesMap>
  rr:logicalTable [rr:tableName "Rel_Person_Organization" ];
  rr:subjectMap [ rr:template"http://example.com/person/{perID}"; ];
  rr:predicateObjectMap [
    rr:predicate conf:hasAffiliation;
    rr:objectMap [
        rr:parentTriplesMap <#OrgTriplesMap>;
        rr:joinCondition [
          rr:child “organizationID”;
          rr:parent “orgID”;]; ]; ].
```

**Table 7.** Object map generated from *OCA2*

```
<#ResearchInterestsTriplesMap>
  rr:logicalTable [
    rr:sqlQuery """
    SELECT pe.perID as id, rpt.topicID as idTopic
    FROM Persons_Rel as pe, Rel_Person_Paper as rpp,
          Papers_Rel as pa, Rel_Paper_Topic as rpt
    WHERE
      pe.perID = rpp.personID and rpp.paperID = pa.paperID and
      pa.paperID = rpt.paperID];
  rr:subjectMap [rr:template "http://example.com/person/{id}";
  rr:predicateObjectMap [
    rr:predicate iswc:researchInterests;
    rr:objectMap [rr:template http://example.com/topic/{ idTopic }];];
```

# 5 Our Strategy for R2RML Mapping Generation

## 5.1 Overview

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

- ***D*** = $(V_D,C_D)$: the target ontology
- ***S***: the data source schema that must be mapped to ***D***
- ***A***: a set of correspondence assertions between ***S*** and ***D***

As shown in Figure 3, the mapping generation process consists of two main steps. The first step involves the design of the *exported ontology* ***E*** = $(V_E,C_E)$, which models the RDF view exported by the data source. The vocabulary $V_E$ of the exported ontology is a subset of $V_D$, which can be automatically generated based on the matching assertions between the ***D*** and ***S*** [5].

The second step involves the design of: (i) a set of relational view schemas ***VS***, where ***VS*** is a direct transformation for the exported ontology ***E***; and (ii) a R2RML

mapping ***M*** which maps ***VS*** to ***E***. The views schemas in ***VS*** can be implemented as SQL views or R2RML views. In Table 9, we present an algorithm that automatically generates ***VS*** and ***M*** from the process inputs.

The benefits of using relational views as a middle layer are twofold: (i) given that ***VS*** is a direct transformation of ***E***, the mapping ***M*** can be automatically generated based on ***VS***; (ii) it becomes easier to change the R2RML mapping ***M*** to accommodate changes to the database view schemas.

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

The design of the exported ontology depends on what we require from the data exported by the data source. As proposed in [5], we require the exported ontology be an open or a closed fragment of the domain ontology.

Again, let $\boldsymbol{D} = (V_D, C_D)$ denote the domain ontology and $\boldsymbol{E} = (V_E, C_E)$ denote the exported ontology. Assume that $V_E$ is a subset of $V_D$. Then, $\boldsymbol{E} = (V_E, C_E)$ can be automatically generated, based on the correspondence assertions between the ***D*** and ***S*** [7]. Briefly, a term $T$ (class or property) of $V_D$ is in $V_E$ if and only if there is a matching assertion for $T$ in ***A***. The constraints $C_E$ are generated using the procedure **openFragment** introduced in [5].

Consider, for example, the ISWC_REL database schema in Figure 1, the CONF_OWL Domain Ontology in Figure 2 and the correspondence assertions in Table 3. Figure 4 shows the exported ontology ISWC_RDF. The vocabulary of ISWC_RDF contains all the elements of the CONF_OWL ontology that matches an element of ISWC_REL.

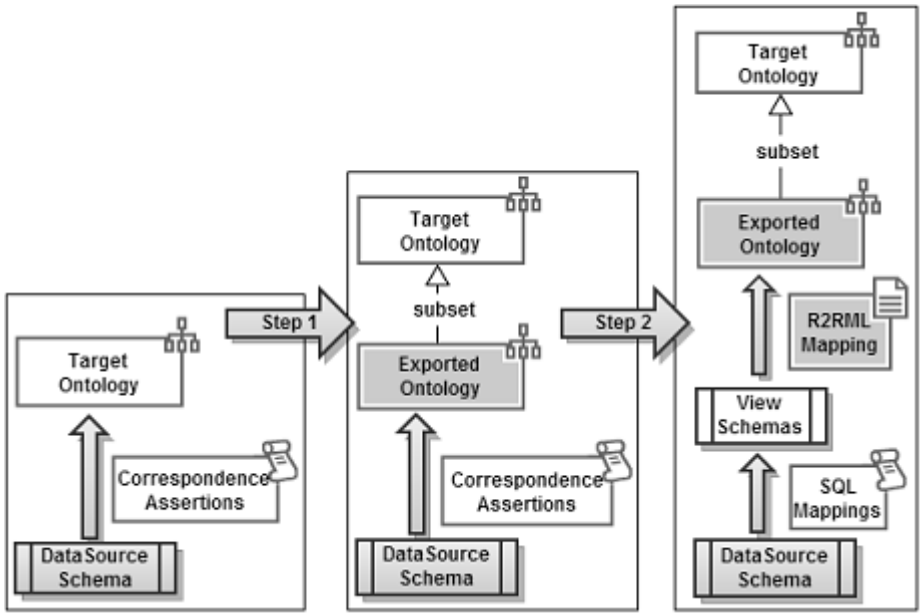


**Fig. 3.** R2RML Mapping Generation Process

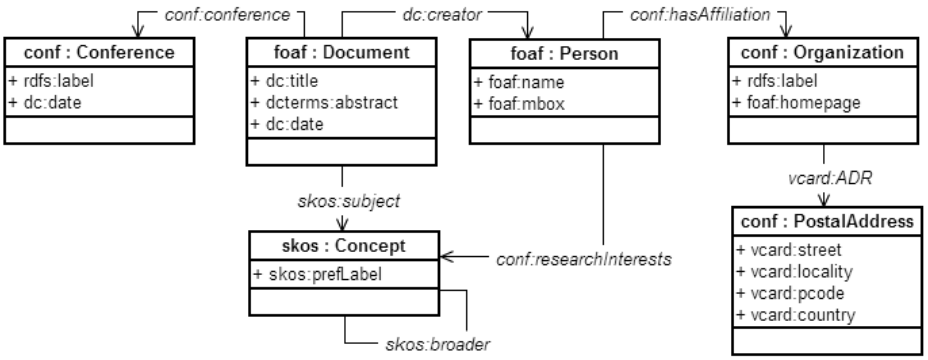


**Fig. 4.** ISWC_RDF Exported Ontology Schema

### 5.3 Step 2: Design of relational view schemas and R2RML mappings

Table 8 shows Algorithm 1, which automatically generates the relational view schemas and R2RML mappings for a given exported ontology $\boldsymbol{E} = (V_E, C_E)$. The algorithm has 3 main steps:

- Step 1: For each class in $V_E$, create a relational view and the CCA that matches the class with the view.
- Step 2: Deals with datatype properties in $V_E$. Two cases are possible:
  - Case 2.1. If the datatype property has maxcardinality equal to 1, generate an attribute and a DCA that matches the datatype property with the attribute.
  - Case 2.2. If the datatype property has maxcardinality greater than1, generate a link relation with a foreign key and an attribute, and generate a DCA that matches the datatype property with a path composed by the foreign key and the attribute.
- Step 3: Deals with object properties in $V_E$. Two cases are possible:
  - Case 2.1. If the object property has maxcardinality equal to 1, generate a foreign key and an OCA that matches the datatype property with the foreign key.
  - Case 2.2. If the object property has maxcardinality greater than1, generate a link relation with two foreign keys and the OCA that matches the datatype property with a path composed by the two foreign keys.

Figure 5 depicts *ISWC_Views*, the relational view schemas for the exported ontology *ISWC_RDF*, and Table 9 contains some of the CAs between the *ISWC_RDF* vocabulary and *ISWC_Views*. In Figure 5, the relations *Conference*, *Person*, *Document*, *Concept*, *Organization* and *PostalAddress* are generated for the classes in *ISWC_RDF*; and the link relations *Document_Creator*, *Document_Subject*, *Person_Research_Interests*, *Person_Has_Afilliation* are generated for the object properties *creator*, *subject*, *researchInterests*, *hasAfilliation* respectively.

Table 10 contains some of the R2RML mappings automatically generated from the CAs in Table 9, as discussed in Section 4.

**Table 8.** Algorithm 1

Step 1: **For each** class *prefixC:C* in $V_E$ **do**
Create a relational view *C*;
Create attribute *ID* as primary key of *C*;
Create CCA *prefixC:C ≡ C[ID];*
Step 2: **For each** datatype property *prefixP:P* in $V_E$ **do**
Let *prefixD:D* be the domain of *prefixP:P*;
Case 2.1: *prefixP:P* has maxCardinality equal to 1.
Create attribute *P* in view *D* whose type is defined according to range of *prefixP:P*;
Create DCA *prefixP:P ≡ D/P*;
Case 2.2: *prefixP:P* has maxCardinality greater than 1.
Create intermediate relational view *D_P*;
Create attribute *ID_D* in *D_P* defined as a foreign key to *D* (parent table);
Create attribute *P* in *D_P* whose type is defined according to range of *prefixP:P*;
Create DCA *prefixP:P ≡ D_P/[ID_D]/P*;
Step 3: **For each** object property *prefixP:P* in $V_E$ **do**
Let *prefixD:D* and *prefixR:R* be the domain and range of *prefixP:P*, respectively;
Case 3.1: *prefixP:P* has maxCardinality equal to 1.
Create attribute *ID_R* in table *D* which is a foreign key to *R* (parent table);
Create OCA *prefixP:P ≡ D/[ID_R];*
Case 3.2: *prefixP:P* has maxCardinality greater than 1.
Create intermediate table *D_P*;
Create attribute *ID_D* in *D_P* which is a foreign key to *D*;
Create attribute *ID_R* in *D_P* which is a foreign key to *R*;
Create OCA *prefixP:P ≡ D_P/[ID_D, ID_R];*

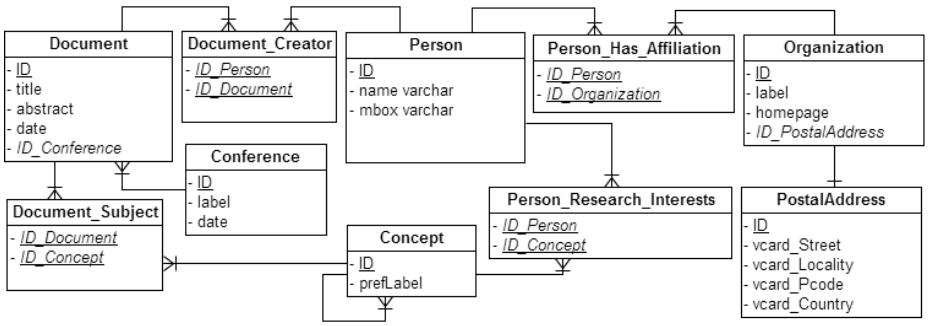


**Fig. 5.** ISWC_View Schemas

**Table 9.** Correspondence Assertions between *ISWC_RDF* and *ISWC_Views*

| | |
|---|---|
| *CCA1* | *foaf:Person ≡ Person[ID]* |
| *CCA2* | *foaf:Document ≡ Document[ID]* |
| *CCA3* | *conf:Organization ≡ Organization[ID]* |

| | |
|---|---|
| *CCA4* | *conf:PostalAddress ≡ PostalAddress[ID]* |
| *CCA5* | *conf:Conference ≡ Conference[ID]* |
| *CCA6* | *skos:Concept ≡ Concept[ID]* |
| *OCA1* | *conf:conference ≡ Document / [ID_Conference]* |
| *OCA2* | *conf:researchInterests ≡ Person_Research_Interests /[ID_Person, ID_Concept]* |
| *OCA3* | *vcard:ADR ≡ Organization /[ID_PostalAddress]* |
| *OCA4* | *skos:subject ≡ Document_Subject /[ID_Document, ID_Concept]* |
| *DCA1* | *foaf:name ≡ Person / name* |
| *DCA2* | *foaf:mbox ≡ Person / mbox* |
| *DCA3* | *skos:prefLabel ≡ Concept / prefLabel* |

**Table 10.** R2RML Mapping Generated from ACs in Table 9

```
<#PersonTriplesMap>  // R2RML mappings generated from CCA1 and DCA1
   rr:logicalTable [ rr:tableName "Person" ];
   rr:subjectMap [
      rr:template "http://example.com/person/{ID}";
      rr:class foaf:Person;];
  rr:predicateObjectMap [
      rr:predicate foaf:name;
      rr:objectMap [ rr:column "name" ];];
   ...
<#DocumentTriplesMap> // R2RML mappings generated from CCA2 and OCA1
   rr:logicalTable [ rr:tableName "Document" ];
   rr:subjectMap [
      rr:template "http://example.com/Document/{ID}";
      rr:class foaf:Document;];
   rr:predicateObjectMap [
      rr:predicate conf:conference;
      rr:objectMap [rr:template http://example.com/Conference /{ID_Conference}];];
   ...
<#Person_Research_InterestsTriplesMap> // R2RML mappings generated from OCA2
   rr:logicalTable [rr:tableName “Person_Research_Interests”];
   rr:subjectMap [
      rr:template  "http://example.com/person/{ID_Person}";
   rr:predicateObjectMap [
      rr:predicate conf:researchInterests;
      rr:objectMap [rr:template "http://example.com/Concept/{ID_Concept}"];];
```

## 6 Conclusion and Future Work

Motivated by the need to develop tools that facilitate the deployment of mappings using R2RML, we first introduced correspondence assertions to specify the mapping between a target vocabulary and a base RDB schema. We then proposed an approach to automatically generate R2RML mappings, based on a set of correspondence asser-

tions. The approach uses relational views as a middle layer, which facilitates the R2RML generation process, and improves the maintainability of the mapping.

We introduced correspondence assertions in earlier papers [15] to investigate XML views. Therefore, it would be natural to adopt the same approach to address RDB-to-RDF mappings. In fact, the latter problem proved to be much simpler than the former, since XML views are fairly complex.

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 5.

## References


1. Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., and Aumueller, D.: Triplify - Light-Weight Linked Data Publication from Relational Databases. In Proceedings of the 18th International World Wide Web Conference (2009).
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, ISWC (2006).
4. Bizer, C.: D2R Map – A Database to RDF Mapping Language, World Wide Web Conference (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. OTM Conferences (2) 2011: 781-799.
6. Cerbah, F.: Learning highly structured semantic repositories from relational databases. The Semantic Web: Research and Applications pp. 777–781 (2008).
7. Cullot,N.,Ghawi,R.,Ye´tongnon,K.:DB2OWL:AToolforAutomaticDatabase-to-Ontology Mapping, pp. 491–494 (2007).
8. Das, S., Sundara, S., and Cyganiak, R.: R2RML: RDB to RDF Mapping Language, W3C Working Draft, http://www.w3.org/TR/r2rml/ (2012).
9. Erling,O.,Mikhailov,I.:Rdfsupportinthevirtuosodbms.NetworkedKnowledge-Networked Media pp. 7–24 (2009).
10. OpenLink Virtuoso. http://virtuoso.openlinksw.com
11. Polfliet, S., Ichise, R.: Automated mapping generation for converting databases into linked data. Proc. of ISWC2010.
12. Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr, T., Auer, S., Sequeda, J., Ez- zat, A.: A survey of current approaches for mapping of relational databases to rdf. W3C RDB2RDF Incubator Group report (2009).
13. Sequeda, J.F., Depena, R., Miranker, D.P.: Ultrawrap: Using sql views for rdb2rdf. Proc. of ISWC2009.
14. Sequeda, J., Tirmizi, S., Corcho, O., Miranker, D.: DMSurvey Survey of directly mapping SQL databases to the Semantic Web (2011).
15. Vidal, V. M., Araujo, V. S., Casanova, M. A.: Towards Automatic Generation of Rules for Incremental Maintenance of XML Views of Relational Data. WISE 2005: 189-202.