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 “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 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.

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 <t,t'>, 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 <t,t'>, 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:


The R2RML mapping in Table 5 refers to the base table Rel_Person_Paper (line 2) to link individuals of class foaf:Person with individuals of class conf:Paper via conf:hasPublication object property. Attribute paperID is used to compose the same URI of PaperTriplesMap (line 3 of Table 5 is equal to line 4 of Table 1). This mapping performs a join between Rel_Person_Paper and Papers on the paperID attribute (lines 9-11). The mapping specifies that, for each pair <t,t'>, where t is a tuple in Rel_Person_Paper, t' is a tuple in Papers, and t.paperID = t'.paperID, one triple must be generated:


The R2RML mapping in Table 6 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 orgID is used to compose the same URI of OrganizationTriplesMap (line 3 of Table 6 is equal to line 4 of Table 1). This mapping performs a join between Rel_Person_Organization and Organizations on the organizationID attribute (lines 9-11). The mapping specifies that, for each pair <t,t'>, where t is a tuple in Rel_Person_Organization, t' is a tuple in Organizations, and t.organizationID = t'.organizationID, one triple must be generated:

<http://example.com/person/t.perID> conf:hasAffiliation <http://example.com/org/t.orgID> .

The R2RML mapping in Table 7 refers to the base table Rel_Paper_Topic (line 2) to link individuals of class conf:Paper with individuals of class conf:Topic via conf:hasResearchInterests object property. Attribute tID is used to compose the same URI of TopicTriplesMap (line 3 of Table 7 is equal to line 4 of Table 1). This mapping performs a join between Rel_Paper_Topic and Topics on the topicID attribute (lines 9-11). The mapping specifies that, for each pair <t,t'>, where t is a tuple in Rel_Paper_Topic, t' is a tuple in Topics, and t.topicID = t'.topicID, one triple must be generated:

3. CORRESPONDENCE ASSERTIONS

3.1 Basic Concepts and Notation

As usual, we denote a relation scheme as $R[A_1,...,A_n]$ 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 $S=(R,\Omega)$, where $R$ is a set of relation schemes and $\Omega$ is a set of relational constraints such that:

(i) $\Omega$ has a unique primary key for each relation scheme in $R$;

(ii) $\Omega$ has a mandatory attribute statement for each attribute which is part of a key or primary key;

(iii) if $\Omega$ has a foreign key of the form $F(R:L:S:K)$, then $\Omega$ also has a constraint indicating that $K$ is the primary key of $S$.

The vocabulary of $S$ is the set of relation names, attribute names, etc. used in $S$. Given a relation scheme $R[A_1,...,A_n]$ and a tuple variable $t$ over $R$, we use $t.A_i$ to denote the projection of $t$ over $A_i$. We use selections over relation schemes, defined as usual.

Let $S=(R,\Omega)$ be a relational schema and $R$ and $T$ be relation schemes of $S$. A list $q_f=[F_1,...,F_m]$ of foreign key names of $S$ is a path from $R$ to $T$ if there is a list $R_1,...,R_n$ of relation schemes of $S$ 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 $q_f$. A path $q_f$ is an association path iff $q_f=[F_1,F_2]$, where the foreign keys are of the forms $F_1(R_2;L_2;K_2)$ and $F_2(R_2;M_2,R_3;K_2)$. 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 $V$ is a set of classes, object properties and datatype properties. An ontology is a pair $O=(V,\Sigma)$ such that $V$ is a vocabulary and $\Sigma$ is a finite set of formulae in $V$, the constraints of $O$. 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 $S=(R,\Omega)$ be a relational schema and $O=(V,\Sigma)$ 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) $\Psi: C \equiv R[A_1,...,A_n]$

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

where $\Psi$ is the name of the CCA, $C$ is a class of $V$, $R$ is a relation name of $S$, $A_1,...,A_n$ 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) $\Psi: P \equiv R/r_1/q_1$

where $\Psi$ is the name of the OCA, $P$ is an object property of $V$, $R$ is a relation name of $S$, and $q_1$ 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) $\Psi: P \equiv R/[A_1,...,A_n]$

(iii) $\Psi: P \equiv R/r_1/q_1/B$

(iv) $\Psi: P \equiv R/r_1/q_1/[B_1,...,B_m]$

where $\Psi$ is the name of the DCA, $P$ is a datatype property of $V$, $R$ is a relation name of $S$, $A$ is an attribute of $R$, $A_1,...,A_n$ are attributes of $R$, $q_1$ is a path from $R$ to a relation schema $T$, $B$ is an attribute of $T$, and $B_1,...,B_m$ are attributes of $T$.

Definition 3.4: A mapping between $V$ and $S$ is a set $A$ of correspondence assertions such that:

(i) If $A$ has an OCA of the form $P \equiv R$, then $A$ 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 $A$ has an OCA of the form $P \equiv R/q$, where $q$ is a path from $R$ to $T$, then $A$ 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 $A$ has a DCA that matches a datatype property $P$ in $V$ with a relation name $R$ of $S$, then $A$ 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 $O=(V,\Sigma)$ be an ontology and $S=(R,\Omega)$ be a relational schema, with vocabulary $U$. Let $X$ be a set of scalar variables and $T$ be a set of tuple variable, disjoint from each other and from $V$ and $U$.

A literal is a range expression of the form $R(t)$, where $R$ is a relation name in $U$ and $t$ is a tuple variable in $T$, 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 "$B[x_1,...,x_j]"$ to indicate that the tuple or
 scalar variables $x_1, \ldots, x_q$ occur in $B$ and $R(t), B[t,x_1, \ldots, x_q]$” 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:

- $C(x) \leftarrow B[x,s]$, where $C$ is a class in $V$ and $B[x,s]$ 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 $\psi_D$;
  
  
  \[
  D(s) \leftarrow R(t), B[t,s], \quad \text{where } B[t,s] \text{ stands for } \text{"HasURI}[^D]{\text{\psi_D}(s)\text{"}, if the CCA for } D \text{ follows Line 1 of Table 6, or } B[t,s] \text{ stands for } \text{"HasURI}[P]{\text{\psi_D}(t,s), \text{\psi_D}(t,s)}, \text{if the CCA for } D \text{ follows Line 2 of Table 6.}
  \]

- Translate the value of $A$ in tuple $t$, generating the literal $\nu$.

- Associate $\nu$ as the value for property $P$ of $s$.

Table 7 shows a set of correspondence assertions that specifies a mapping between $\text{CONF} \_\text{OWL}$ and $\text{ISWC} \_\text{REL}$, obtained with the help of the tool described in [8]. For example, the transformation rules induced by $\text{CCA1}$, $\text{DCA1}$ and $\text{OCA2}$ are (we omit the translations from attribute values to RDF literals for simplicity): $\text{CCA1}$ specifies that each tuple $t$ in $\text{Persons}$ generates one triple:

\[
\text{<http://example.com/person/t.perID> rdf:type foaf:Person.}
\]

$\text{DCA1}$ specifies that each tuple $t$ in $\text{Persons}$ generates one triple:

\[
\text{<http://example.com/person/t.perID> foaf:name concat(firstname, t.lastname).}
\]

$\text{OCA2}$ specifies that, for each tuple $t$ in $\text{Person}$, for each tuple $t'$ in $\text{Topics}$ such that $t'$ is referenced by $t$ through path $[\text{Fk} \_\text{Authors, Fk} \_\text{Publications, Fk} \_\text{Papers, Fk} \_\text{Topics}]$, one triple is generated:

\[
\text{<http://example.com/person/t.perID> conf:researchInterests <http://example.com/org/t.topicID>}.\]

\[
\text{Table 5. Built-in predicates}
\]

<table>
<thead>
<tr>
<th>Built-in predicate</th>
<th>Intuitive definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonNull(v)</td>
<td>nonNull(v) holds iff value $v$ is not null</td>
</tr>
<tr>
<td>RDFLiteral(u, A, R, v)</td>
<td>Given a value $u$, an attribute $A$, 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 $R$.</td>
</tr>
<tr>
<td>HasReferencedTuples[q]{t,u} where $q$ is a path from $R$ to $T$</td>
<td>Given a tuple $t$ of $R$ and tuple $u$ of $T$, HasReferencedTuples[q]{t,u} holds iff $u$ is referenced by $t$ through path $q$.</td>
</tr>
<tr>
<td>HasURI[Ψ]{t,s} where $Ψ$ is a CCA for a class $C$ of $V$, using attributes $A_1, \ldots, A_n$ of $R$</td>
<td>Given a tuple $t$ of $R$. HasURI[Ψ]{t,s} holds iff $s$ is the URI obtained by concatenating the namespace prefix for $C$ and the values of $A_1, \ldots, A_n$.</td>
</tr>
<tr>
<td>concat(v_1, v_2, v_3, v)</td>
<td>Given a list $[v_1, v_2, v_3, v]$ of string values, concat(v_1, v_2, v_3, v) holds iff $v$ is the string obtained by concatenating $v_1, v_2, v_3$, and $v$.</td>
</tr>
</tbody>
</table>

\[
\text{Table 6. Mapping Rules}
\]

<table>
<thead>
<tr>
<th>Correspondence Assertion</th>
<th>Mapping Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ψ: C = R[A_1, \ldots, A_n]$, $\sigma$</td>
<td>$C(s) \leftarrow R(t), \text{HasURI}[^P]{\text{\psi_D}(t,s)}$, $\text{nonNull}(v)$</td>
</tr>
<tr>
<td>$Ψ: O = R$ where:</td>
<td>$P(s,o) \leftarrow R(t), B[t,x], B[o,y]$</td>
</tr>
<tr>
<td>- $A$ has a CCA $Ψ_A$ that matches the domain $D$ of $O$ with $R$ and $Ψ_D$ has mapping rule $D(s) \leftarrow R(t), B[t,s]$</td>
<td></td>
</tr>
<tr>
<td>- $A$ has a CCA $Ψ_A$ that matches the range $N$ of $O$ with $R$ and $Ψ_D$ has mapping rule $N(o) \leftarrow R(t), B[o,t]$</td>
<td></td>
</tr>
<tr>
<td>$Ψ: O = R / ψ$ where:</td>
<td>$P(s,o) \leftarrow R(t), B[t,x], \text{HasReferencedTuples}[q]{t,u}, T(u), B[u,o]$</td>
</tr>
<tr>
<td>- $q$ is a path of $R$ to $T$</td>
<td></td>
</tr>
<tr>
<td>- $A$ has a CCA $Ψ_A$ that matches the domain $D$ of $O$ with $R$ and $Ψ_D$ has mapping rule $D(s) \leftarrow R(t), B[t,s]$</td>
<td></td>
</tr>
<tr>
<td>- $A$ is an attribute of $R$</td>
<td></td>
</tr>
<tr>
<td>$Ψ: P = R / ψ / A$ where:</td>
<td>$P(s,v) \leftarrow R(t), B[t,x], \text{nonNull}(v), \text{RDFLiteral}(t, A, \text{&quot;A&quot;}, \text{&quot;R&quot;}, v)$</td>
</tr>
<tr>
<td>- $q$ is a path of $R$ to $T$</td>
<td></td>
</tr>
<tr>
<td>- $A$ has a CCA $Ψ_A$ that matches the domain $D$ of $P$ with $R$ and $Ψ_D$ has mapping rule $D(s) \leftarrow R(t), B[t,s]$</td>
<td></td>
</tr>
<tr>
<td>- $A$ is an attribute of $R$</td>
<td></td>
</tr>
<tr>
<td>$Ψ: P = R / ψ / A$ where:</td>
<td>$P(s,v) \leftarrow R(t), B[t,x], \text{nonNull}(v), \text{RDFLiteral}(t, A, \text{&quot;A&quot;}, \text{&quot;T&quot;}, v)$</td>
</tr>
<tr>
<td>- $q$ is a path of $R$ to $T$</td>
<td></td>
</tr>
<tr>
<td>- $A$ has a CCA $Ψ_A$ that matches the domain $D$ of $P$ with $R$ and $Ψ_D$ has mapping rule $D(s) \leftarrow R(t), B[t,s]$</td>
<td></td>
</tr>
<tr>
<td>- $A$ is an attribute of $R$</td>
<td></td>
</tr>
<tr>
<td>$Ψ: P = R / ψ / A$ where:</td>
<td>$P(s,v) \leftarrow R(t), B[t,x], \text{nonNull}(v), \text{RDFLiteral}(t, A, \text{&quot;A&quot;}, \text{&quot;T&quot;}, v)$</td>
</tr>
<tr>
<td>- $q$ is a path of $R$ to $T$</td>
<td></td>
</tr>
<tr>
<td>- $A$ has a CCA $Ψ_A$ that matches the domain $D$ of $P$ with $R$ and $Ψ_D$ has mapping rule $D(s) \leftarrow R(t), B[t,s]$</td>
<td></td>
</tr>
<tr>
<td>- $A$ is an attribute of $R$</td>
<td></td>
</tr>
<tr>
<td>$Ψ: P = R / ψ / A$ where:</td>
<td>$P(s,v) \leftarrow R(t), B[t,x], \text{nonNull}(v), \text{RDFLiteral}(t, A, \text{&quot;A&quot;}, \text{&quot;T&quot;}, v)$</td>
</tr>
<tr>
<td>- $q$ is a path of $R$ to $T$</td>
<td></td>
</tr>
<tr>
<td>- $A$ has a CCA $Ψ_A$ that matches the domain $D$ of $P$ with $R$ and $Ψ_D$ has mapping rule $D(s) \leftarrow R(t), B[t,s]$</td>
<td></td>
</tr>
<tr>
<td>- $A$ is an attribute of $R$</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Table 7. Correspondence Assertions}
\]

| CCA1 | foaf:Person ≡ Persons[perID] |
| CCA2 | foaf:Document ≡ Papers[paperID] |
| CCA3 | conf:Organization ≡ Organizations[orgID] |
| CCA4 | conf:PostalAddress ≡ Organizations[address, location, postcode, country] |
| CCA5 | conf:Conference ≡ Conferences[confID] |
| CCA6 | skos:Concept ≡ Topics[topicID] |
| OCA1 | conf:hasAffiliation ≡ Persons / [Fk_Persons, Fk_Organizations] |
| OCA2 | conf:researchInterests ≡ Papers / [Fk_Authors, Fk_Publications, Fk_Papers, Fk_Topic] |
| OCA3 | vcard:ADR ≡ Organizations |
| OCA4 | skos:subject ≡ Papers / [Fk_Papers, Fk_Topic] |
| OCA5 | conf:conference ≡ Papers / Fk_Conferences |
| OCA6 | skos:broadTopic ≡ Topics / Fk_Parent |
| DCA1 | foaf:name ≡ Persons / [firstName, lastName] |
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 \( E \).

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_RDF domain ontology in Figure 2, 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


Table 8. Algorithm 1

Step 1: For each class C in $V_s$ where $K_1,...,K_s$ are the datatype properties of the key of C do
- Create a relational view also named C;
- Create $K_1,...,K_s$, the attributes of the primary key of view C;
- Create the subject map referring to view C using template T1 in Table 9.

Case 2.1: P has cardinality equal to 1.
- Let D be view that match to the domain of P and let $K_{D_1},...,K_{D_n}$ be the attributes of the primary key of D;
- Create object map for P using template T2 in Table 9, which is added to the triples map of view D.
- Create attribute P in view D whose type is defined according to range of property P;
- Create the predicate object map for P using template T2 in Table 9, which is added to the triples map of view D.

Case 2.2: P has cardinality greater than 1.
- Create domain object view D_P;
- Create attributes $K_{D_1},...,K_{D_n}$ in D_P whose types are defined as in D;
- Create foreign key FK_D_P(D_P:{K_{D_1},...,K_{D_n}}, D:{K_{D_1},...,K_{D_n}});
- Create attribute P in D_P whose type is defined according to range of property P;
- Create the subject map referring to view D_P, and predicate object map for P using template T3 in Table 9.

Case 3.1: P has cardinality equal to 1.
- Let D and R be the views that match to the domain and range of P, respectively, let $K_{D_1},...,K_{D_n}$ be the attributes of the primary key of D and let $K_{R_1},...,K_{R_m}$ be the attributes of the primary key of R; the views D and R were created in Step 1
- Create attributes $K_{D_1},...,K_{D_n}$ in D whose types are defined as in D;
- Create foreign key FK_D_P(D_P:{K_{D_1},...,K_{D_n}}, D:{K_{D_1},...,K_{D_n}});
- Create predicate object map for P using template T4 in Table 9, which is added to the triples map of view D.

Case 3.2: P has cardinality greater than 1.
- Create relational view D_P;
- Create attributes $K_{D_1},...,K_{D_n}$ in D_P whose types are defined as in D;
- Create foreign key FK_D_P(D_P:{K_{D_1},...,K_{D_n}}, D:{K_{D_1},...,K_{D_n}});
- Create attributes $K_{R_1},...,K_{R_m}$ in D_P whose types are defined as in R;
- Create foreign key FK_D_P(D_P:{K_{D_1},...,K_{D_n}}, R:{K_{R_1},...,K_{R_m}});
- Create the subject map referring to view D_P and predicate object map for P using template T5 in Table 9.
Table 9. Templates to translate CAs to R2RML mappings

<table>
<thead>
<tr>
<th>Template</th>
<th>Logical Table Specification</th>
<th>Subject Map Specification</th>
<th>Predicate Object Map Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td><code>&lt;#C_TriplesMap&gt;</code></td>
<td><code>[rr:logicalTable [ rr:tableName &quot;C&quot; ];</code></td>
<td><code>[rr:subjectMap [</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>rr:template &quot;namespaceOfC/{K_1}/.../{K_n}&quot;;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>rr:class C; ];</code></td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td><code>[rr:objectMap [</code></td>
<td><code>rr:column &quot;P&quot; ]; ];</code></td>
</tr>
<tr>
<td>T3</td>
<td><code>&lt;#D_P_TriplesMap&gt;</code></td>
<td><code>[rr:logicalTable [ rr:tableName &quot;D_P&quot; ];</code></td>
<td><code>[rr:subjectMap [</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>rr:template &quot;namespaceOfD/{K_1}/.../{K_n}&quot;;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>rr:class D; ];</code></td>
</tr>
<tr>
<td>T4</td>
<td></td>
<td><code>[rr:predicateObjectMap [</code></td>
<td><code>rr:predicate P;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>rr:objectMap [</code></td>
<td><code>rr:parentTriplesMap &lt;R_TriplesMap&gt;;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>rr:joinCondition [</code></td>
<td><code>rr:child &quot;K_R1&quot;;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>rr:parent &quot;K_R1&quot;; ];</code></td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td><code>&lt;#D_P_TriplesMap&gt;</code></td>
<td><code>[rr:logicalTable [ rr:tableName &quot;D_P&quot; ];</code></td>
<td><code>[rr:subjectMap [</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>rr:template &quot;namespaceOfD/{K_1}/.../{K_n}&quot;;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>rr:class D; ];</code></td>
</tr>
</tbody>
</table>

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

- `<#Person_TriplesMap>`
  - Logical Table
    - `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>`
  - Logical Table
    - `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>`
  - Logical Table
    - `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.