A Semi-Automatic Approach for Generating Customized R2RML Mappings

Vania Maria P. Vidal1, Marco A. Casanova2, Luís Eufrasio T. Neto1

1 Federal University of Ceará, Fortaleza, CE, Brazil

{vaniap.vidal, [luis.eufrasio}@gmail.com](mailto:luis.eufrasio%7d@gmail.com)

2 Department of Informatics – Pontifical Catholic University of Rio de Janeiro, RJ, Brazil

[casanova@inf.puc-rio.br](mailto: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 customized mappings between relational schemas and target RDF vocabularies. However, the generation of customized R2RML mappings is not an easy task, which calls for the development of tools to support the definition and deployment of customized mappings. This paper therefore proposes a strategy to simplify the specification of R2RML mappings. It first introduces correspondence assertions to manually model the relationship between a relational database schema and a target RDF vocabulary. Then, the paper describes a method to automatically generate R2RML mappings from the correspondence assertions.

**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. As the name implies, the direct mapping approach relies on automatic methods to derive ontologies that directly reflect relational schemas and to transform the relational data into RDF triples. A survey of systems based on the direct mapping approach is presented in [14].

The customized mapping approach ­ the subject of this work ­ relies on the designer to specify a mapping between the relational schema and an existing target ontology that is used to transform the relational data into RDF triples. This approach divides the RDB-to-RDF process into two steps: mapping generation and mapping implementation. The mapping generation step results in a specification of how to represent relational schema concepts in terms of RDF classes and properties. It provides the ability to view existing relational data as RDF triples 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 triples or to offer virtual access through an interface that queries the underlying database.

In fact, quite a few tools that support the customized mapping approach 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] to specify a mapping between a target vocabulary and a base relational schema. Then, we introduce a method, backed up by a tool, to automatically generate R2RML mappings based on a set of correspondence assertions.

The remainder of this paper is organized as follows. Section 2 defines presents our mapping formalism the correspondence assertions and additional concepts. 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, based on the correspondence assertions. Section 6 contains the conclusions and directions for future work.

1. Correspondence Assertions
   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* ***U***consists of a set of *relation names, attribute names*, *key names, primary key names*, *foreign key names* and *tuple variables*.

A *relation scheme* in ***U*** is an expression of the form *R[A1,....,An],* where *R* is a relation name in ***U*** and *A1,....,An* is a list of distinct attribute names in ***U***, called the *list of attributes* of *R*.

Let *R[A1,....,An]* and *S[B1,....,Bm]* be two relation schemes in ***U***. The notions of *key*, *primary key*, and *mandatory* (or *not null*) attribute in ***U*** are defined as usual; we just observe that keys and primary keys have a name taken from ***U***. A *foreign key* in ***U*** for *R* *with respect to* *S* is a statement of the form *F(R:L,S:K)*, where *F* is the *name* of the foreign key, taken from ***U***, *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* in ***U*** is a key, a primary key, a foreign key or a mandatory attribute constraint in ***U***.

A *relational schema* in ***U*** is a pair ***S****=(****R****,****Ω****)*, where ***R*** is a set of relation schemes in ***U*** and ***Ω*** is a set of relational constraints in ***U*** such that: (i) ***Ω*** has a unique primary key for each relation scheme in ***R***; (ii) ***Ω*** has a mandatory attribute statement for each attribute which is part of a key or primary key; (iii) if ***Ω*** has a foreign key of the form *F(R:L,S:K)*, then ***Ω*** also has a constraint indicating that *L* is the primary key of *R*.

Let *R1,...,Rn* be relation schemes of a relational schema ***S***. Suppose that there exists a list of foreign keys *Fi(Ri:Ki,Ri+1:Ki+1)* or *Fi(Ri+1:Ki+1,Ri:Ki)*, for *1 ≤ i ≤ n-1*. Then,   
*ϕ* = *[F1,... ,Fn-1]* is a *path* from *R1* to *Rn*. We say that tuples of *Rn* *are referenced by tuples of R1**through φ*. We also say that *ϕ* is an *association path* iff *φ=[F1,F2]*, where the foreign keys are of the form *F1(R2:K2,R1:K1)* and *F2(R2:K2,R3:K3)*.

Given a relation scheme *R[A1,....,An]* and a tuple variable *t* in ***U***, we use *t.Ak* to denote the *projection* of *t* over an attribute *Ak* 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* ***V*** is a set of *classes*, *object* *properties* and *datatype properties.* An *ontology* is a pair ***O****=(****V****,Σ)* such that ***V*** is a vocabulary and *Σ* 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.

* 1. Definition of the Correspondence Assertions

As mentioned before, we propose a strategy to simplify the specification of R2RML mappings based on correspondence assertions (CAs).

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 ***O****=(****V****,Σ)*  be an ontology and ***S****=(****R****,Ω)* be a relational schema. In what follows, we assume that *Σ* defines the domain and range of each property.

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

*Ψ: C* ≡ *R[A1,...,An]*

*Ψ: C* ≡ *R[A1,...,An]σ*

where *Ψ* is the *name* of the CCA, *C* is class of ***V***, *R* is a relation name of ***S***, *A1,...,An* are the attributes of the primary key of *R*, and *σ* is a selection over *R*. We also say that *Ψ* *matches* *C* with *R*.

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

*Ψ: O* ≡ *R / ϕ*

*Ψ: O* ≡ *R*

where *Ψ* is the *name* of the OCA, *O* is an object property of ***V*** and *ϕ* is a path from *R*. We also say that *Ψ* *matches* *O* with *R*.

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

*Ψ: P ≡ R / A*

*Ψ: P ≡ R / {A1,...,An}*

*Ψ: P ≡ R / ϕ / B*

*Ψ: P* ≡ *R / ϕ / {B1,...,Bn}*

where *Ψ* is the name ofth*e* DCA, *P* is a datatyp*e* property of ***V***, *R* is a relation name of ***S***, *A* is an attribute of *R*, *A1,...,An* are attributes of *R*, *ϕ* is a path from *R* to *R’*, *B* is an attribute of *R’*, and *B1,...,Bn* are attributes of *R’*. We also say that *Ψ* *matches* *P* with *R*.

**Definition 2.4:** A *Simple Correspondence Assertion* is a correspondence assertion of the following type:

*A CCA* of the form *Ψ: C* ≡ *R[A1,...,An].*

*A DCA* of the form *Ψ: P ≡ R / A.*

*A DCA* of the form *Ψ: P ≡ R / ϕ / B*, *where ϕ has only one foreign key.*

*An OCA* of the form *Ψ: P ≡ R /ϕ*, *where ϕ has only one foreign key* or *ϕ* is an association path.

**Definition 2.5:** A *mapping* between ***V*** and ***S*** is a set ***M*** of correspondence assertions such that:

1. For each class *C* in ***V***, ***M*** has at most one CCA that matches *C* with some relation scheme in ***S***.
2. For each datatype property *P* in ***V***, ***M*** has at most one DCA that matches *P* with some relation scheme *R* in ***S***. Furthermore, ***M*** must have a CCA that matches the domain of*P* with *R*.
3. For each object property *O* in ***V***, ***M*** has at most one OCA *Ψ* that matches *O* with some relation scheme *R* in ***S***. Furthermore, if *Ψ* is of the form *O ≡ R/ϕ,* where *ϕ* is a path from *R* to *R’*, then ***M*** must have a CCA that matches the domain of*O* with *R*, and a CCA that matches the range of *O* with *R’*; if *Ψ* is of the form *O ≡ R,* then ***M*** must have a CCA that matches the domain of*O* with *R*, and a CCA that matches the range of *O* also with *R*.
   1. 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 ***O****=(****V****,Σ)*  be an ontology, and ***S****=(****R****,Ω)* be a relational schema over a relational alphabet ***U***. Let ***X*** be a set of *scalar variables*, disjoint from ***V*** and ***U*** (we recall that ***U*** has a set of tuple variables).

**Table .** 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](t,u)*  where *ϕ* is a path from *R* to *T* | Given a tuple *t* of *R* and tuple *u* of *T*,  *HasReferencedTuples[ϕ](t,u)* holds iff  *u* is referenced by *t* through path *ϕ* |
| *HasURI[Ψ](t,s)*  where *Ψ* is a CCA for a class *C* of ***V***, using attributes *A1,...,An* of *R*  (see Def. 2.1) | 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 *t.A1,...,t.An* |
| *concat([v1,.., vn],v)* | Given a list *[v1,.., vn]* of string values,  *concat([v1,.., vn],v)* holds iff  *v* is the string obtained by concatenating *v1,..,vn* |

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 ***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[x1,…,xk]”* to indicate that the tuple or scalar variables *x1,…,xk* occur in *B*. We also use *“R(t), B[t,x1,…,xk]”* 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) ← B[x]*, where *C* is a class in ***V*** and *B[x]* is a rule body
* *P(x,y)←B[x,y]*, where *P* is a property and *B[x,y]* is a rule body

Let ***M*** be a set of correspondence assertionsthat defines a mapping between ***V*** and ***S*.** Assume that each class *C* in ***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:

|  |  |
| --- | --- |
| *Ψ: P* ≡ *R / A*  where:   * *A* is an attribute of *R* * *ΨD*is the CCA that matches the domain *D* of *P* with *R* and has mapping rule  *D(s)* ← *R(t), B[t,s]* | *P(s,v)*  *R(t), B[t,s],*  *nonNull(t.A),*  *RDFLiteral(t.A,“A”,“R”,v)* |

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 *ΨD*
* translate the value of *A* in tuple *t*, generating the literal *v*
* associate *v* as the value for property *P* of *s*

Line 4 represents a special case, where the domain and range of an object property *O* come from the same relation scheme *R*.

**Table .** Mapping Rules

|  |  |  |
| --- | --- | --- |
|  | **Correspondence Assertion** | **Mapping Rule** |
| 1 | *Ψ: C* ≡ *R[A1,...,An]* | *C(s)* ← *R(t), HasURI[Ψ](t,s)* |
| 2 | *Ψ: C* ≡ *R[A1,...,An]σ* | *C(s)* ← *R(t), HasURI[Ψ](t,s),σ(t)* |
| 3 | *Ψ: O* ≡ *R / ϕ*  where:  - *ϕ* is a path of *R* to *T*  - *ΨD*is the CCA in ***M*** that matches the domain *D* of *O*  with *R* and has mapping rule *D(s) ← R(t), BD[t,s]*  - *ΨN* is the CCA in ***M*** that matches the range *N* of *O*  with *T* and has mapping rule *N(o) ← T(u), BN[u,o]* | *P(s,o)* ←  *R(t), BD[t,s],*  *HasReferencedTuples[ϕ](t,u),*  *T(u), BN[u,o]* |
| 4 | *Ψ: O* ≡ *R*  where:  - *ΨD*is the CCA in ***M*** that matches the domain *D* of *O*  with *R*and has mapping rule *D(s) ← R(t), BD[t,s]*  - *ΨN* is the CCA in ***M*** that matches the range *N* of *O*  with *R*and has mapping rule *N(o) ← R(t), BN[t,o]* | *P(s,o)* ← *R(t), BD[t,s], BN[t,o]* |
| 5 | *Ψ: P* ≡ *R / A*  where:  - *A* is an attribute of *R*  - *ΨD*is the CCA in ***M*** that matches the domain *D* of *P*  with *R* and has mapping rule *D(s) ← R(t), BD[t,s]* | *P(s,v)* ← *R(t), BD[t,s],*  *nonNull(t.A)*  *RDFLiteral(t.A,“A”,“R”,v)* |
| 6 | *Ψ: P* ≡ *R / ϕ / A*  where:  - *ϕ* is a path of *R* to *T*  - *A* is an attribute of *T*  - *ΨD*is the CCA in ***M*** that matches the domain *D* of *P*  with *R* and has mapping rule *D(s) ← R(t), BD[t,s]* | *P(s,v)* ← *R(t), BD[t,s],*  *HasReferencedTuples[ϕ](t,u),*  *nonNull(u.A),*  *RDFLiteral(u.A,“A”,“T”,v)* |
| 7 | *Ψ: P* ≡ *R / {A1,...,Am}*  where:  - *A1,...,Am* are attributes of *R*  - *ΨD*is the CCA in ***M*** that matches the domain *D* of *P*  with *R* and has mapping rule *D(s) ← R(t), BD[t,s]* | *P(s,v)* ← *R(t), BD[t,s],*  *nonNull(t.A1),…,nonNull(t.Am),*  *RDFLiteral(t.A1,“A1”,“R”,v1),*  *…,*  *RDFLiteral(t.Am,“Am”,“R”,vm),*  *concat([v1,..,vm],v)* |
| 8 | *Ψ: P* ≡ *R / ϕ / {A1,...,Am}*  where:  - *ϕ* is a path of *R* to *T*  - *A1,...,Am* are attributes of *T*  - *ΨD*is the CCA in ***M*** that matches the domain *D* of *P*  with *R* and has mapping rule *D(s) ← R(t), BD[t,s]* | *P(s,v)* ← *D(t), BD[t,s],*  *HasReferencedTuples[ϕ](t,u),*  *nonNull(u.A1),…,nonNull(u.Am)*  *RDFLiteral(u.A1,“A1”,“T”,v1),*  *…,*  *RDFLiteral(u.Am,“Am”,“T”, vm),*  *concat([v1,..,vm],v)* |

1. Running Example

In this Section, we present the relational database schema *ISWC\_REL* and the ontology *CONF\_OWL*, which we use 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 is of type integer. Tables *Persons* and *Papers* represent the main concepts. The attribute *conference* of table *Papers* is a foreign key to table *Conferences*. Table *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.

### 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 relational schema and the ontology to draw arcs between elements that contain related data. Table 3 shows a set of correspondence assertions that specifies the semantic mapping between *CONF\_OWL* and *ISWC\_REL*.

In following, we discuss the transformation rules induced by some of the CAs in Table 3.

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

*<*[*http://example.com/person/t.perID*](http://example.com/person/t.perID)*> rdf:type*  *foaf:Person*.

* *DCA1* specifies that each tuple *t* in *Persons* produces one RDF triple:

*<http://example.com/person/t.perID>*

*foaf:name* *concat(t.firstname, t.lastname)*.

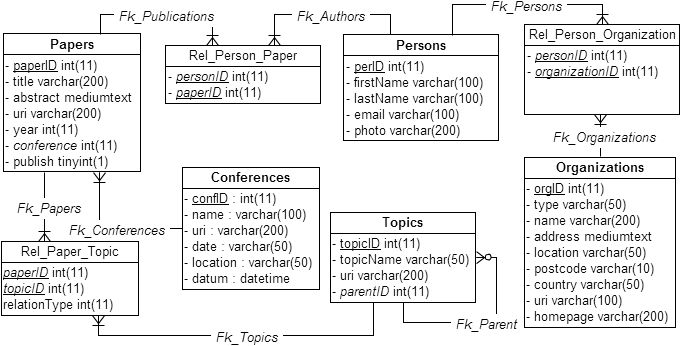
* *DCA2* specifies that each tuple *t* in *Persons* produces one RDF triple:

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

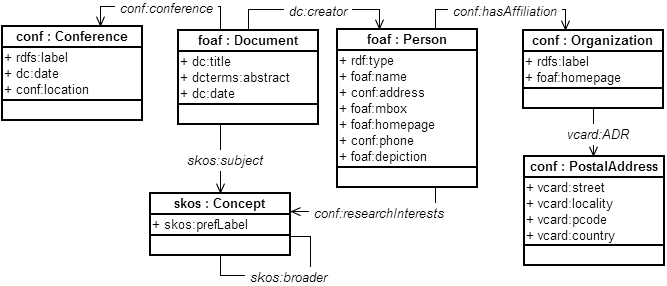
* *OCA2* specifies that, for each tuple *t* in *Person*, for each tuple *t’* in *Topics* such that *t’* is referenced by *t* through path *[Fk\_Authors, Fk\_Publications, Fk\_Papers, Fk\_Topics]*, one triple of the form is generated:

*<http://example.com/person/t.perID>*

*conf:ResearchInterests http://example.com/org/t’.topicID.*



**Fig. .** ISWC\_REL Database Schema



**Fig. .** CONF\_OWL Target Ontology

**Table .** Correspondence Assertions

|  |  |
| --- | --- |
| *CCA1* | *foaf:Person*  *Persons[perID]* |
| *CCA2* | *foaf:Document*  *Papers[paperID]* |
| *CCA3* | *conf:Organization*  *Organizations[orgID]* |
| *CCA4* | *conf:PostalAddress*  *Organizations[orgID]* |
| *CCA5* | *conf:Conference*  *Conferences[confID]* |
| *CCA6* | *skos:Concept*  *Topics[topicID]* |
| *OCA1* | *conf:hasAffiliation*  *Persons / [Fk\_Persons, Fk\_Organizations]* |
| *OCA2* | *conf:researchInterests*  *Persons / [Fk\_Authors, Fk\_Publications, Fk\_Papers, Fk\_Topics ]* |
| *OCA3* | *vcard:ADR*  *Organizations* |
| *OCA4* | *skos:subject*  *Papers / [Fk\_Papers, Fk\_Topics]* |
| *OCA5* | *conf:conference*  *Papers / Fk\_Conferences* |
| *OCA6* | *skos:broader*  *Topics / Fk\_Parent* |
| *DCA1* | *foaf:name*  *Persons / { firstName, lastName }* |
| *DCA2* | *foaf:mbox*  *Persons / email* |
| *DCA3* | [*rdfs:label*](conf:Organization/rdfs:label) *Organization / name* |
| *DCA4* | *foaf:homepage*  *Organization / homepage* |
| *DCA5* | [*vcard:Street*](conf:PostalAddress/vcard:Street) *Organizations / address* |
| *DCA6* | *vcard:locality*  *Organizations / location* |
| *DCA7* | *vcard:Pcode*  *Organizations / postcode* |
| *DCA8* | *vcard:country*  *Organizations /country* |
| *DCA9* | *dc:title*  *Papers / title* |
| *DCA10* | *dcterms:abstract*  *Papers / abstract* |
| *DCA11* | *dc:date*  *Papers / year* |
| *DCA12* | *skos:prefLabel*  *Topics / topicName* |
| *DCA13* | *rdfs:label*  *Conferences / name* |
| *DCA14* | *dc:date*  *Conferences / date* |
| *DCA15* | *conf:location*  *Conferences / location* |

1. R2RML Mapping Language

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.

The use of views in R2RML mappings is a convenient solution to deal with complex mappings, which require data transformation, computation, or filtering before generating triples from the database. For example, the object map for the object property *conf:researchInterests* shown in Table 7 refers to an R2RML view. 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*.

**Table 4.** Triples map generated from *OCA2*

|  |  |
| --- | --- |
| 1 | <#ResearchInterestsTriplesMap> |
| 2 | rr:logicalTable [ |
| 3 | rr:sqlQuery """ |
| 4 | SELECT pe.perID as id, rpt.topicID as idTopic |
| 5 | FROM Persons as pe, Rel\_Person\_Paper as rpp,  Papers as pa, Rel\_Paper\_Topic as rpt |
| 6 | WHERE  pe.perID = rpp.personID and rpp.paperID = pa.paperID and  pa.paperID = rpt.paperID]; |
| 7 | rr:subjectMap [rr:template "http://example.com/person/{id}"; |
| 8 | rr:predicateObjectMap [ |
| 9 | rr:predicate conf:researchInterests; |
| 10 | rr:objectMap [rr:template [http://example.com/topic/{ idTopic }](http://example.com/topic/%7b%20idTopic%20%7d) ]; ]; |

1. Our Strategy for R2RML Mapping Generation
   1. Overview

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

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

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

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

The mapping generation process consists of three main steps. The first step involves the design of the *exported ontology* ***EO***. The second step involves the design of a set of a relational view schemas ***V*** and a set of direct correspondence assertions between VS and EO. The third step generates the R2RML mapping ***M*** which maps ***VS*** to ***EO***.

|  |  |
| --- | --- |
| **Fig. .** 3-Level Schema Architecture | **Fig. .** ISWC\_RDF Exported Ontology Schema |

* 1. 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 target ontology.

Again, let ***D*** = *(VD,CD)* denote the target ontology and ***E*** = *(VE,CE)* denote the exported ontology. Assume that *VE* is a subset of *VD*. Then, ***E*** = *(VE,CE)* can be automatically generated, based on the correspondence assertions between the ***D*** and ***S***[7]. Briefly, a term *T* (class or property) of *VD* is in *VE* if and only if there is a matching assertion for *T* in ***A***. The constraints *CE* 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.

* 1. Step 2: Design of relational view schemas

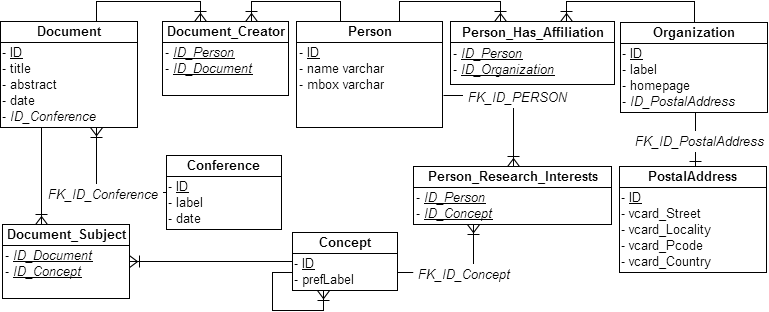
Table 8 shows Algorithm 1, which automatically generates the relational view schemas for a given exported ontology ***EO*** = *(VE,CE )* a set of direct correspondence assertions between VS and EO. The algorithm has 3 main steps:

* Step 1: For each class in *VE*, create a relational view and the CCA that matches the class with the view.
* Step 2: Deals with datatype properties in *VE*. Two cases are possible:
* Case 2.1. If the datatype property has maxcardinality equal to 1, generate an attribute and a simple 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 direct DCA that matches the datatype property with a path composed by the foreign key and the attribute.
* Step 3: Deals with object properties in *VE*. Two cases are possible:
* Case 3.1. If the object property has maxcardinality equal to 1, generate a foreign key and a direct OCA that matches the object property with the foreign key.
* Case 3.2. If the object property has maxcardinality greater than1, generate a link relation with two foreign keys and a direct OCA that matches the object 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 8.** Algorithm 1

|  |
| --- |
| Step 1: **For each** class *prefixC:C* in *VE* **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 *VE* **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* whose type is defined according to attribute ID of D;  Create foreign key *FK\_ID\_D(D\_P:{ID\_D},D:{ID})*  Create attribute *P* in *D\_P* whose type is defined according to range of *prefixP:P*;  Create DCA *prefixP:P* ≡*D/[ FK\_ID\_D]/P*; |
| Step 3: **For each** object property *prefixP:P* in *VE* **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 view *D* whose type is defined according to attribute *ID* of *R*.  Create foreign key *FK\_ID\_R(D:{ID\_R},R:{ID})*  Create OCA *prefixP:P* ≡*D/[ FK\_ID\_R];*  Case 3.2: *prefixP:P* has maxCardinality greater than 1.  Create intermediate table *D\_P*;  Create attribute *ID\_D* in *D\_P* whose type is defined according to attribute ID of D;  Create attribute *ID\_R* in *D\_P* whose type is defined according to attribute ID of R;  Create foreign key *FK\_ID\_D(D\_P:{ID\_D},D:{ID})*  Create foreign key *FK\_ID\_R(D\_P:{ID\_R}, R:{ID})*  Create OCA *prefixP:P* ≡*D/[FK\_ID\_D, FK\_ID\_R];* |



**Fig. .** ISWC\_View Schemas

**Table 9.** Simple 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 / [FK\_ID\_Conference]* |
| *OCA2* | *conf:researchInterests*  *Person /[FK\_ID\_Person, FK\_ID\_Concept]* |
| *OCA3* | *vcard:ADR*  *Organization /[FK\_ID\_PostalAddress]* |
| *OCA4* | *skos:subject*  *Document /[FK\_ID\_Document, FK\_ID\_Concept]* |
| *DCA1* | *foaf:name*  *Person / name* |
| *DCA2* | *foaf:mbox*  *Person / mbox* |
| *DCA3* | *skos:prefLabel*  *Concept / prefLabel* |

* 1. Step 3: Generation of R2RML mappings

This section shows how to translate direct correspondence assertions to R2RML mappings. In order to translate direct 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.

Table 9 shows the templates to translate simple correspondence assertions to R2RML mappings. The parameters are in bold and they are preceded by ‘$’ character. Line 1 shows the CCA direct template. ***$C*** is the class name that is the same name of the view generated after the algorithm execution. Thus, ***$C*** is used to define mapping name, table name, uri template and class name. ***$attList*** is the list of view *C* key attributes (*A1,...,An*) and the each attribute into this list is used to build the URI of the class. Line 2 depicts the DCA simple template. ***$p*** is the prefix of datatype property ***$P***. As this is a simple assertion, ***$P*** also represents the name of the column defined in view *C*. Line 3 shows the DCA simple template with a foreign key (fk). The existence of the fk tells us that the maxCardinality of the datatype property is greater than one. So the algorithm has created a view for this property and this view is used as the logical table of the subject map. This is the solution for this special case of triples generation where the cardinality of the object is greater than one. Line 4 exposes the OCA simple template with just one foreign key (*fk*) into the path **$*ϕ***. ***$p*** is the prefix of object property ***$P***. The length of **$*ϕ*** equals to one indicates that the maxCardinality of ***$P*** is also one, so the algorithm has not created a new view for this object property. The object map refers the triples map defined to ***$P*** range class and for each pair of *fk* matching attributes a join condition is created. By the end, Line 5 depicts the OCA simple template with more than one fk into the path **$*ϕ***. So the object property has maxCardinality greater than one and the algorithm has created a view for this property. Similarly to the explanation of Line 3, we have to create a new triples map. The detail here is the creation of join conditions. Our solution considers just the attributes of the last FK to build the join conditions. This happens because the view created by the algorithm already includes the joins through the whole path and projects just the key attributes of the last relation referenced by the last FK.

**Table 9.** Templates to translate simple correspondence assertions to R2RML mappings

|  |  |
| --- | --- |
| 1 | *Ψ: prefix:C* ≡ *C[A1,...,An]* |
|  | <#**$C**TriplesMap>  rr:logicalTable [ rr:tableName "**$C**" ];  rr:subjectMap [  rr:template "**$C**/ #foreach ($att in **$attList**) {$att}/ #end";  rr:class **$p**:**$C**;  ]; |
| 2 | *Ψ: p:P ≡ C / P, where P is a datatype property* |
|  | rr:predicateObjectMap [  rr:predicate **$p**:**$P**;  rr:objectMap [ rr:column "**$P**" ];  ]; |
| 3 | *Ψ: p:P ≡ C / ϕ / B, where ϕ has only one foreign key (fk)* |
|  | <#**$C\_$P**TriplesMap>  rr:logicalTable [ rr:tableName "**$C\_$P**" ];  rr:subjectMap [  rr:template "**$C**/{ID\_**$C**}";  rr:class **$p**:**$C**;  ];  rr:predicateObjectMap [  rr:predicate **$p**:**$P**;  rr:objectMap [ rr:column "**$B**" ];  ]; |
| 4 | *Ψ: p:O ≡ C /ϕ, where ϕ has only one foreign key (fk) and P is an object property* |
|  | rr:predicateObjectMap [  rr:predicate **$p**:**$O**;  rr:objectMap [  rr:parentTriplesMap <**$O.rangeClass()**TriplesMap>;  #foreach ($pair in **$*ϕ****.get*Fk())  rr:joinCondition [  rr:child “$pair.childAtt”;  rr:parent “$pair.parentAtt”;  ];  #end  ];  ]; |
| 5 | *Ψ: p:O* ≡ *R / ϕ , where ϕ is an associated path with two foreign keys* |
|  | <#**$C\_$P**TriplesMap>  rr:logicalTable [ rr:tableName "**$C\_$P**" ];  rr:subjectMap [  rr:template "**$C**/{ID\_**$C**}";  rr:class **$p**:**$C**;  ];  rr:predicateObjectMap [  rr:predicate **$p**:**$O**;  rr:objectMap [  rr:parentTriplesMap <**$O.rangeClass()**TriplesMap>;  #foreach ($pair in **$*ϕ****.getLast*Fk())  rr:joinCondition [  rr:child “$pair.childAtt”;  rr:parent “$pair.parentAtt”;  ];  #end  ];  ]; |

Table 10 contains some of the R2RML mappings automatically generated from the direct CAs in Table 9, using the templates in Table 9.

**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 "[*person*](http://example.com/person/t.perID)/{ID}";  rr:class foaf:Person; ]; |
| rr:predicateObjectMap [  rr:predicate foaf:name;  rr:objectMap [ rr:column "name" ]; ] . |
| **...** |
| <#ConceptTriplesMap> # **R2RML mapping generated from *CCA6***  rr:logicalTable [ rr:tableName "Concept" ];  rr:subjectMap [  rr:template "*concept*/{ID}";  rr:class skos:Concept; ] . |
| **...** |
| <#Person\_researchInterestsTriplesMap> # **R2RML mappings generated from *OCA2***  rr:logicalTable [rr:tableName “Person\_researchInterests”];  rr:subjectMap [  rr:template "[*person*](http://example.com/person/t.perID)/{ID\_Person}";  rr:class foaf:Person; ];  rr:predicateObjectMap [  rr:predicate conf:researchInterests;  rr:objectMap [  rr:parentTriplesMap <ConceptTriplesMap>;  rr:joinCondition [  rr:child “ID\_Concept”;  rr:parent “ID”;  ]; ]; ] . |

1. 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 assertions. 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](http://www.inf.puc-rio.br/%7Ecasanova/Publications/Papers/2011-Papers/2011-ODBASE.pdf). [OTM Conferences (2) 2011](http://www.informatik.uni-trier.de/%7Eley/db/conf/otm/otm2011-2.html#CasanovaBFVMGS11): 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.