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. This paper therefore proposes a strategy to simplify the specification of R2RML mappings. It first introduces correspondence assertions to manually model, in a convenient way, the relationship between a relational database schema and an RDF vocabulary. Then, the paper describes a method to automatically generate R2RML mappings from the correspondence assertions.

**Keywords:** Correspondence Assertions, Linked Data, RDB-to-RDF, R2RML.

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 conventional, 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* 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 [12].

The *customized mapping approach* divides the RDB-to-RDF process into two steps. The *mapping generation step* relies on the designer to specify a mapping between the relational schema and a target ontology 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 that support the customized mapping approach have been developed, such as Triplify [1], D2R Server [3, 4], and OpenLink Virtuoso [8]. 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 definition and deployment of mappings using R2RML.

This paper has two contributions. First, we propose correspondence assertions [13] as a convenient way to specify customized mappings between target RDF vocabularies and base relational schemas. Then, we introduce a method, backed up by a tool, to automatically generate R2RML mappings from a set of correspondence assertions.

The remainder of this paper is organized as follows. Section 2 introduces the correspondence assertions. Section 3 presents the example used throughout the paper. Section 4 describes our approach for the automatic generation of R2RML mappings, based on the correspondence assertions. Section 6 contains the conclusions.

1. Correspondence Assertions
   1. Basic Concepts and Notation

As usual, we denote a *relation scheme* as *R[A1,....,An]*, 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* *S* *and* *T*.

A *relational schema* is a pair ***S****=(****R****,****Ω****)*, where ***R*** is a set of relation schemes and ***Ω*** is a set of relational constraints 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 *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[A1,....,An]* and a *tuple variable t* over *R*, we use *t.Ak* to denote the *projection* of *t* over *Ak*. We use *selections* over relation schemes, defined as usual.

Let ***S****=(****R****,****Ω****)* be a relational schema and *R* and *T* be relation schemes of ***S***. A list *ϕ*=*[F1,...,Fn-1]* of foreign key names of ***S*** is a *path* *from* *R to T* iff there is a list *R1,...,Rn* of relation schemes of ***S*** such that *R1=R*, *Rn=T* and *Fi* *relates* *Ri**and* *Ri+1*. We say that tuples of *S* *reference* tuples of *T**through φ*. A path *φ* is an *association path* iff *φ*=*[F1,F2]*, where the foreign keys are of the forms *F1(R2:L2,R1:K1)* and *F2(R2:M2,R3:K3)*. 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****,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

This section introduces the notion of correspondence assertion, leaving examples to Section 3. Let ***S****=(****R****,****Ω****)* be a relational schema and ***O****=(****V****,Σ)* be an ontology and and assume that *Σ* has constraints defining 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 a 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: P* ≡ *R*

*Ψ: P* ≡ *R / ϕ*

where *Ψ* is the *name* of the OCA, *P* is an object property of ***V*** and *ϕ* is a path from *R*. We also say that *Ψ* *matches* *P* 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 ofthe DCA, *P* is a datatype 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 *T*, *B* is an attribute of *T*, and *B1,...,Bn* are attributes of *T*. We also say that *Ψ* *matches* *P* with *R*.

**Definition 2.4:** A correspondence assertion is *simple* iff it is of one of the forms:

*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: O ≡ 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. If ***M*** has an OCA of the form *P ≡ R,* then ***M*** must have a CCA that matches the domain of*P* with *R*, and a CCA that matches the range of *P* also with *R*.
2. If ***M*** has an OCA of the form *P ≡ R/ϕ*, where *ϕ* is a path from *R* to *T*, then ***M*** must have a CCA that matches the domain of*P* with *R* and a CCA that matches the range of *P* with *T.*
3. If ***M*** has a DCA that matches a datatype property *P* in ***V*** with a relation name *R* of ***S***, then ***M*** must have a CCA that matches the domain of*P* with *R*.
   1. Transformation Rules generated by Correspondence Assertions

In this section, we introduce the notion of transformation rule and show how to interpret correspondence assertions as transformation rules. Let ***O****=(****V****,Σ)* be an ontology and ***S****=(****R****,****Ω****)* 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 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***, that is, ***M*** satisfies that the conditions stated in Definition 2.5. Assume that each class *C* in ***V*** is associated with a namespace prefix*.*

Table 2 shows the transformation rules *induced* by the correspondence assertions in ***M***. 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 *ΨD*: *D(s)←R(t),BD[t,s]*, where *BD[t,s]* stands for *“HasURI[Ψ](t,s)”*, if the CCA for *D* follows Line 1 of Table 2, or *BD[t,s]* stands for *“HasURI[Ψ](t,s),σ(t)”*, if the CCA for *D* follows Line 1;
* Translate the value of *A* in tuple *t*, generating the literal *v*;
* Associate *v* as the value for property *P* of *s*.

**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* | 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* |

**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*  where:  - ***M*** has a CCA *ΨD*that matches the domain *D* of *O*  with *R* and *ΨD*has mapping rule *D(s)←R(t),BD[t,s]*  - ***M*** has a CCA *ΨN* that matches the range *N* of *O*  with *R* and *ΨN* has mapping rule *N(o)←R(t),BN[t,o]* | *P(s,o)* ← *R(t), BD[t,s], BN[t,o]* |
| 4 | *Ψ: O* ≡ *R / ϕ*  where:  - *ϕ* is a path of *R* to *T*  - ***M*** has a CCA *ΨD*that matches the domain *D* of *O*  with *R* and *ΨD*has mapping rule *D(s)←R(t),BD[t,s]*  - ***M*** has a CCA *ΨN* that matches the range *N* of *O*  with *T* and *ΨN* 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]* |
| 5 | *Ψ: P* ≡ *R / A*  where:  - ***M*** has a CCA *ΨD*that matches the domain *D* of *P*  with *R* and *ΨD*has mapping rule *D(s)←R(t),BD[t,s]*  - *A* is an attribute of *R* | *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*  - ***M*** has a CCA *ΨD*that matches the domain *D* of *P*  with *R* and *ΨD*has mapping rule *D(s)←R(t),BD[t,s]*  - *A* is an attribute of *T* | *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:  - ***M*** has a CCA *ΨD*that matches the domain *D* of *P*  with *R* and *ΨD*has mapping rule *D(s)←R(t),BD[t,s]*  - *A1,...,Am* are attributes of *R* | *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*  - ***M*** has a CCA *ΨD*that matches the domain *D* of *P*  with *R* and *ΨD*has mapping rule *D(s)←R(t),BD[t,s]*  - *A1,...,Am* are attributes of *T* | *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 example we will use in the rest of the paper.

Figure 1 depicts the relational schema *ISWC\_REL*. Each table has a distinct 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 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.

Table 3 shows a set of correspondence assertions that specifies a mapping between *CONF\_OWL* and *ISWC\_REL*, obtained with the help of the tool described in [8]. For example, the transformation rules induced by *CCA1*, *DCA1* and *OCA2* are (we omit the translations from attribute values to RDF literals for simplicity):

### *CCA1* specifies that each tuple *t* in *Persons* produces one RDF triple:

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

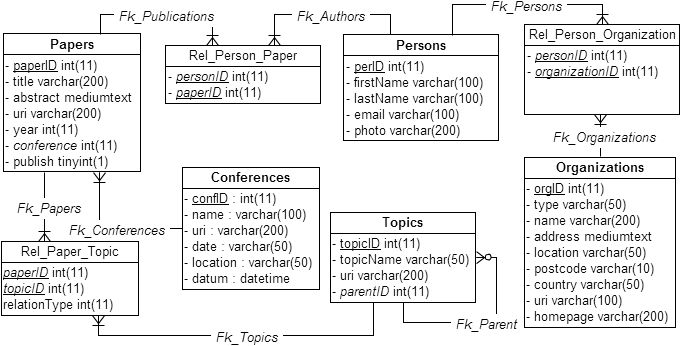
* *DCA1* specifies that each tuple *t* in *Persons* produces one RDF triple:

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

*concat(t.firstname, t.lastname)*.

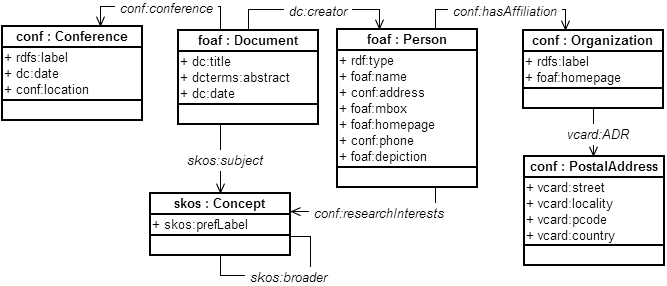
* *OCA2* specifies that, for each tuple *t* in *Person*, for each tuple *t’* in *Topics* such that *t’* is referenced by *t* through path *[Fk\_Authors, Fk\_Publications, Fk\_Papers, Fk\_Topics]*, one triple of the form is generated:

*<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. A Strategy for R2RML Mapping Generation
   1. R2RML Mapping Language

An *R2RML mapping* refers to logical tables to retrieve data from the input database. A *logical table* is a base table, a view or a valid SQL query (called an *R2RML view*). Each logical table is mapped to RDF using a *triples map*, which has a *subject map* and multiple *predicate-object maps*. The subject map generates the subject of the RDF triples. The predicate-object maps consist of *predicate maps* and *object maps* (or *referencing object maps*). Triples are created by combining the subject map with a predicate map and an object map, and by applying such maps to the logical table rows.

The use of views in R2RML mappings facilitates the definition of complex mappings. For example, the object map for the object property *conf:researchInterests* shown in Table 4 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. Overview of the strategy

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***, such that the mapping between VS and EO is defined by a set of simple correspondence assertions. 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 VS to the EO can be automatically generated based on the set of simple correspondence assertions between ***VS*** and EO. Also, it simplifies maintaining the R2RML mapping to reflect changes to the database schemas.

The mapping generation process consists of two 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 ***VS*** and 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. As *VE* is a subset of *VD*. Then, ***E*** = *(VE,CE)* can be automatically generated, based on the correspondence assertions between the ***D*** and ***S***[6]. 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 match 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 VS for a given exported ontology ***EO*** = *(VE,CE )* and the R2RML mappings from VS to 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 cardinality equal to 1, generate an attribute, a simple DCA that matches the datatype property with the attribute, and the predicate object map for the DCA using template T2 in Table 6;
* Case 2.2. If the datatype property has cardinality greater than 1, generate a relation with a foreign key and an attribute, generate a simple DCA that matches the datatype property with a path composed by the foreign key and the attribute, and the subject map and predicate object map for the DCA using template T3 in Table 6.
* Step 3: Deals with object properties in *VE*. Two cases are possible:
* Case 3.1. If the object property has cardinality equal to 1, generate a foreign key, a simple OCA that matches the object property with the foreign key, and the predicate object map for the OCA using template T4 in Table 6;
* Case 3.2. If the object property has cardinality greater than 1, generate a link relation with two foreign keys, a simple OCA that matches the object property with a path composed by the two foreign keys, and the subject map and predicate object map for the OCA using template T5 in Table 6.

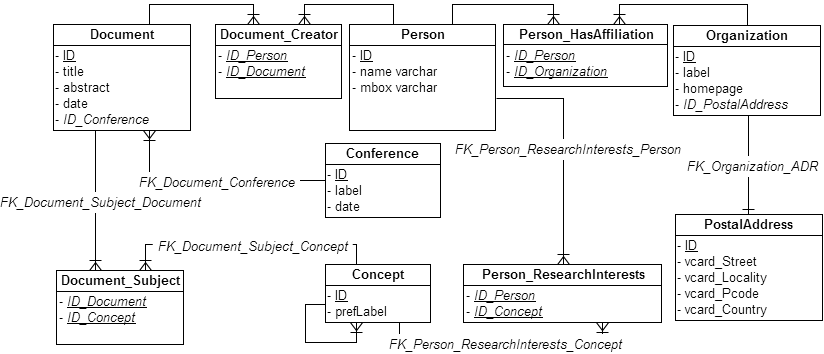
Figure 5 depicts *ISWC\_Views*, the relational view schemas for the exported ontology *ISWC\_RDF*, and Table 9 contains some of the simple CAs between the *ISWC\_RDF* vocabulary and *ISWC\_Views*. In Figure 5, the relations *Conference*, *Person*, *Document,* *Concept*, *Organization* and *PostalAddress* are generated for the classes in *ISWC\_RDF*; and the link relations *Document\_Creator*, *Document\_Subject*, *Person\_ResearchInterests*, *Person\_HasAfilliation* are generated for the object properties *creator*, *subject*, *researchInterests*, *hasAfilliation* respectively.

**Table 5.** Algorithm 1

|  |
| --- |
| Step 1: For each class *C* in *VE* where *K1,...,Kn* are the data type properties of the key of *C* do  Create a relational view *C*;  Create attributes *K1, … , Kn* which are the attributes of the primary key of view *C*;  Create CCA *Ψ: C* ≡*C[K1, … , Kn];*  Create the subject map for *Ψ* using template T1 in Table 9. |
| Step 2: For each datatype property *P* in *VE* do  Let *D* be view that match to the domain of *P and let* *KD1,...,KDn* be the attributes of the primary key of *D*; **// *view D was created in Step 1***  Case 2.1: *P* has cardinality equal to 1.  Create attribute *P* in view *D* whose type is defined according to range of property *P*;  Create DCA as *Ψ: P* ≡*D* / *P*;  Create the predicate object map for *Ψ* using template T2 in Table 6;  Attach the predicate object map created above to the subject map of view D.  Case 2.2: *P* has cardinality greater than 1.  Create relational view *D\_P;*  Create attributes *KD1,...,KDn* in *D\_P* as in D;  Create foreign key *FK\_D\_P(D\_P:{KD1,...,KDn}, D:{KD1,...,KDn* *});*  Create attribute *P* in *D\_P* whose type is defined according to range of property *P*;  Create DCA as *Ψ: P* ≡*D / [FK\_D\_P] / P*;  Create the subject map and predicate object map for *Ψ* using template T3 in Table 6. |
| Step 3: For each object property *P* in *VE* do  Let *D* and *R* be the views that match to the domain and range of *P*, respectively, let *KD1,...,KDn* be the attributes of the primary key of *D* and let *KR1,...,KRn* be the attributes of the primary key of *R*;  **// views D and R were created in Step 1**  Case 3.1: *P* has cardinality equal to 1.  Create attributes *KR1,…, KRn* in D as in *R*;  Create foreign key *FK\_D\_R(D:{KR1,…, KRn}, R:{KR1,…, KRn})*  Create OCA as *Ψ: P* ≡*D / [FK\_D\_R];*  Create the predicate object map for *Ψ* using template T4 in Table 6;  Attach the predicate object map created above to the subject map of view D.  Case 3.2: *P* has cardinality greater than 1.  Create relational view *D\_P*;  Create attributes *KD1,...,KDn* in *D\_P* as in D;  Create foreign key *FK\_D\_P\_D(D\_P:{KD1,...,KDn}, D:{KD1,...,KDn* *});*  Create attributes *KR1,…, KRn* in *D\_P* as in *R*;  Create foreign key *FK\_D\_P\_R(D\_P:{KR1,…, KRn}, R:{KR1,…, KRn})*;  Create OCA *Ψ: P* ≡*D / [FK\_D\_P\_D, FK\_D\_P\_R];*  Create the subject map and predicate object map for *Ψ* using template T5 in Table 6. |

**Table 6.** Templates to translate simple correspondence assertions to R2RML mappings

|  |  |
| --- | --- |
| T1 | <#**C**TriplesMap>  rr:logicalTable [ rr:tableName "**C**" ];  rr:subjectMap [  rr:template "http://**uriOfC**/{**K1**}/{**K2**}/... /{**Kn**}/";  rr:class **C**; ]; |
| T2 | rr:predicateObjectMap [  rr:predicate **P**;  rr:objectMap [ rr:column "**P**" ]; ]; |
| T3 | <#**D**\_**P**TriplesMap>  rr:logicalTable [ rr:tableName "**D\_P**" ];  rr:subjectMap [  rr:template "http://**uriOfD**/{**KD1**}/{**KD2**}/... /{**KDn**}/";  rr:class **D**; ];  rr:predicateObjectMap [  rr:predicate **P**;  rr:objectMap [ rr:column "**P**" ]; ]; |
| T4 | rr:predicateObjectMap [  rr:predicate **P**;  rr:objectMap [  rr:parentTriplesMap <**R**TriplesMap>;  rr:joinCondition [  rr:child “**KR1**”;  rr:parent “**KR1**”; ];  …  rr:joinCondition [  rr:child “**KRn**”;  rr:parent “**KRn**”; ]; ]; ]; |
| T5 | <#**D\_P**TriplesMap>  rr:logicalTable [ rr:tableName "**D\_P** " ];  rr:subjectMap [  rr:template "http://**uriOfD**/{**KD1**}/{**K D2**}/... /{**KDn**}/";  rr:class **D**; ];  rr:predicateObjectMap [  rr:predicate **P**;  rr:objectMap [  rr:parentTriplesMap <**R**TriplesMap>;  rr:joinCondition [  rr:child “**KR1**”;  rr:parent “**KR1**”; ];  …  rr:joinCondition [  rr:child “**KRn**”;  rr:parent “**KRn**”; ]; ]; ]; |



**Fig. .** ISWC\_View Schemas

**Table 7.** 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\_Document\_Conference]* |
| *OCA2* | *conf:researchInterests*  *Person / [FK\_Person\_ResearchInterests\_Person,*  *FK\_Person\_ResearchInterests\_Concept]* |
| *OCA3* | *vcard:ADR*  *Organization / [FK\_Organization\_ADR]* |
| *OCA4* | *skos:subject*  *Document / [FK\_Document\_Subject\_Document, FK\_Document\_Subject\_Concept]* |
| *DCA1* | *foaf:name*  *Person / name* |
| *DCA2* | *foaf:mbox*  *Person / mbox* |
| *DCA3* | *skos:prefLabel*  *Concept / prefLabel* |

**Table 8.** Some R2RML mappings generated from CAs in Table 7 using Templates in Table 6

|  |
| --- |
| <#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 Proc. 18th World Wide Web Conf. (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, Proc. ISWC (2006).
4. Bizer, C.: D2R Map – A Database to RDF Mapping Language. Proc. 12th World Wide Web Conf. (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 Conf. (2) 2011: 781-799.
6. Cullot,N.,Ghawi,R.,Ye ́tongnon,K.:DB2OWL:ATool for Automatic Database-to-Ontology Mapping, pp. 491–494 (2007).
7. Das, S., Sundara, S., and Cyganiak, R.: R2RML: RDB to RDF Mapping Language, W3C Working Draft, <http://www.w3.org/TR/r2rml/> (2012).
8. Neto, L., Vidal, V., Casanova, M., Monteiro J.: R2RML by Assertion: A Semi-Automatic Tool for Generating Customised R2RML Mappings. ESWC (2013).
9. OpenLink Virtuoso. <http://virtuoso.openlinksw.com>
10. Polfliet, S., Ichise, R.: Automated mapping generation for converting databases into linked data. Proc. ISWC 2010.
11. 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).
12. Sequeda, J., Tirmizi, S., Corcho, O., Miranker, D.: DMSurvey Survey of directly mapping SQL databases to the Semantic Web (2011).
13. 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.