Specifying Complex Correspondences between Relational Schemas and *RDF* Models for Generating Customized *R2RML* Mappings

Valéria M. Pequeno
INESC-ID
Taguspark, Portugal
vmp@inesc-id.pt

Vania M. P. Vidal
Federal University of Ceará
Fortaleza, CE, Brazil
vvidal@lia.ufc.br

Marco A. Casanova
Pontifical Catholic University
of Rio de Janeiro
RJ, Brazil
casanova@inf.puc-rio.br

Luís Eufrasio T. Neto
Federal University of Ceará
Fortaleza, CE, Brazil
luis.eufrasio@gmail.com

Helena Galhardas*
Instituto Superior Técnico,
Universidade de Lisboa
Taguspark, Portugal
helena.galhardas@ist.utl.pt

ABSTRACT

The W3C RDB2RDF Working Group proposed a standard language to map relational data into RDF triples, called R2RML. However, creating R2RML mappings may sometimes be a difficult task because it involves the creation of views (within the mappings or not) and referring to them in the R2RML mapping. To overcome such difficulty, this paper first proposes *algebraic correspondence assertions*, which simplify the definition of relational-to-RDF mappings and yet are expressive enough to cover a wide range of mappings. Algebraic correspondence assertions include data-metadata mappings (where data elements in one schema serve as metadata components in the other), mappings containing custom value functions (e.g., data format transformation functions) and union, intersection and difference between tables. Then, the paper shows how to automatically compile algebraic correspondence assertions into R2RML mappings.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Semantic Web*

General Terms

Design, Standardization and Languages

Keywords

Correspondence Assertions, Relational-to-RDF mapping, Relational Model, RDF Model, Ontologies.

*Helena Galhardas is also researcher at INESC-ID.

IDEAS '14, July 07 - 09 2014, Porto, Portugal

http://dx.doi.org/10.1145/2628194.2628233"

1. INTRODUCTION

The growing interest in the evolution of the current Web of documents into a Web of Data has attracted attention from the database research community, since a vast quantity of data is stored in Relational Databases (RDBs) and, therefore, publishing this data to the Web of Data assumes great importance. Since that Resource Description Framework (RDF) has become the *de facto* standard for publishing structured data over the Web, it seems obvious to publish relational data in RDF format. A flexible and general way to implement this task is to create RDF views over the underlying relational data [13]. These RDF views may be either virtual or materialized. In the former, the RDF triples of the view are obtained dynamically, whereas in the latter normally a batch process is used to create the RDF repository from the RDB. The materialized view approach is used to improve query performance and data availability [4, 5, 13]. The virtual approach has the advantage that the query is executed against the most current version of the data, which is convenient when the base data is often updated.

In order to create RDF views, it is necessary to transform data stored in conventional RDB into RDF triples. This process is usually named RDB-to-RDF [10] and may follow two approaches: 1) the *direct mapping approach* and 2) the *customized mapping approach*. The former relies on automatic methods to obtain ontologies that directly reflect relational schemas (c.f. [11] and SquirrelRDF[1]). In the customized mapping approach, an expert creates a mapping between the relational schema and an existing target ontology. Here the RDB-to-RDF process can be divided into two steps: the *mapping generating step* and the *mapping implementation step*. The mapping generation step relies on the designer to specify a mapping between the RDB and the target ontology [13], resulting in a specification of how to represent RDB concepts as RDF concepts. The mapping implementation step generates the actual code to transform RDB data into RDF triples. There are various types of mapping implementations, which can be generated in this step in accordance

[1]url:http://jena.sourceforge.net/SquirrelRDF/

to the designer's choose. Virtual RDF views, RDF dumps (or materialized RDF views) and a Linked Data interface are some examples of mapping implementations.

There are many tools and applications to automate the RDB-to-RDF process, among which we may cite: SquirrelRDF, Triplify [1], D2R server [2], OpenLink Virtuoso[2] and Ontop[3]. SquirrelRDF is an implementation of the direct mapping approach, meaning that classes and properties of the target RDF vocabulary are generated from names in the RDB schema. Its mapping language is very simple and easy to understand. Triplify is a special-purpose mapping tool, which was developed for a specific use: mapping HTTP-URI requests onto RDB queries and translating the resulting relations into RDF statements. The mapping is encoded inside SQL queries using, for example, *aliases*. D2R server, OpenLink Virtuoso and Ontop are general-purpose mapping tools that heavily rely on SQL to implement the mapping. Each one adopts a different and proprietary mapping language, even though the features that they support are very similar (see surveys [6, 7, 10]). Furthermore, understanding the mapping languages may require considerable time.

In order to overcome the inconvenience of the existing RDB-to-RDF mapping tools, the W3C RDB2RDF Working Group proposed a standard language to map relational data into RDF triples, called R2RML. Some tools, such as Virtuoso and Ontop, support R2RML by translating the R2RML syntax to their own mapping language. However, in some situations (e.g., when we need to join three or more tables), creating R2RML mappings is difficult because it involves the creating of views (within the mapping or not) and referring to them in the R2RML mapping. In [12], Vidal et al. proposes a strategy to simplify the specification of R2RML mappings. The strategy has two steps: (1) the specification of a set of correspondence assertions between the base relational schema and the vocabulary of a domain ontology chosen by the user, which results in an exported ontology; and (2) the automatic generation of relational views and R2RML mappings, based on the result of the first step.

This paper has two contributions. First, it extends the work in [9, 12] by considering more complex mappings. More specifically, we investigate the usage of Correspondence Assertions (CAs) to map data-metadata (where data elements in one schema serve as metadata components in the other), mappings containing custom value functions (e.g., use transformation functions to change the data formats) and union, intersection and difference between tables. Then, it proposes algorithms to automatically generate the SQL view definition based on correspondence assertions. The automatic generation of the SQL view definitions was not addressed in our previous work [9, 12].

The remainder of the paper is as follows. Section 2 presents the running example. Section 3 defines the concept of Correspondence Assertion (CA): the necessary background and the proposed extension. Section 4 describes the approach for the automatic generation of R2RML mappings, based on CAs. Section 5 shows the algorithms for generating the SQL statements for the relational view based on CAs. Section 6 summarizes the conclusions and future work.

[2]http://virtuoso.openlinksw.com.
[3]http://ontop.inf.unibz.it/

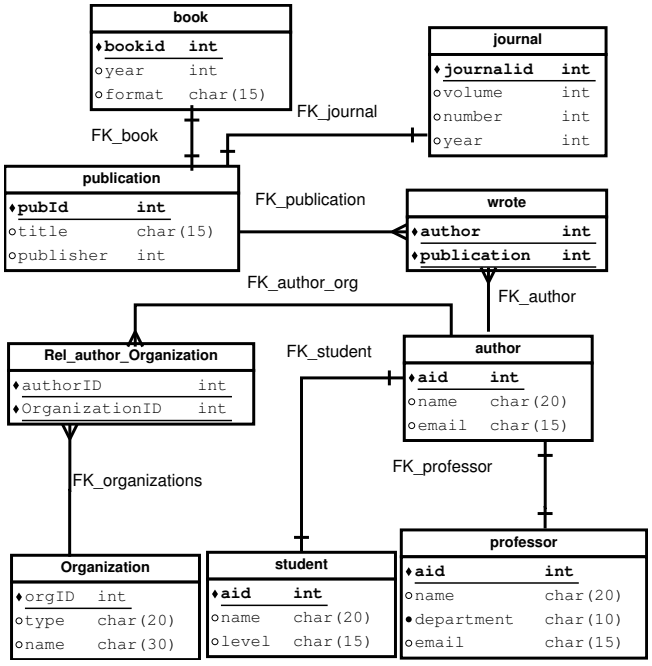

Figure 1: Source relational schema *LIBRARY_REL*.

2. RUNNING EXAMPLE

Throughout the paper, we use as a running example the relational schema presented in Figures 1 and the ontology shown in Figure 2. Figure 1 depicts the source schema *LIBRARY_REL* containing a set of publications (books and journals), each one identified by a unique number (*pubid*). Publications of type books are stored in the relation **book**; publications of type journal are stored in the relation **journal**. One or more authors write the publications. This information is modelled by the relation **wrote**. The authors are stored in the relation **author** and can be students modelled by the relation **student** or professors modelled by the relation **professor**. Each relation has a primary key, which is underlined. The labels of the arcs, such as FK_student, are the names of the foreign keys.

Figure 2 depicts the ontology *ACADEMIC_OWL*, which reuses terms from well-known vocabularies: FOAF (Friend of a Friend), DC (Dublin Core), BIBO (Bibliographic Ontology Specification), ORG (Organization), LSC (Linked Science Core Vocabulary) and PARTICIPATION. We use the prefix "ex" for new terms defined in the *ACADEMIC_OWL* ontology. *ex:InTraining* models teachers in training (teachers that are also graduate students). *ex:ElegibleFaculty* keeps faculty members eligible for administrative positions, for example, a professor that is not a graduate student.

Our aim is to create R2RML mappings from the relational schema *LIBRARY_REL* to the ontology *ACADEMIC_OWL*.

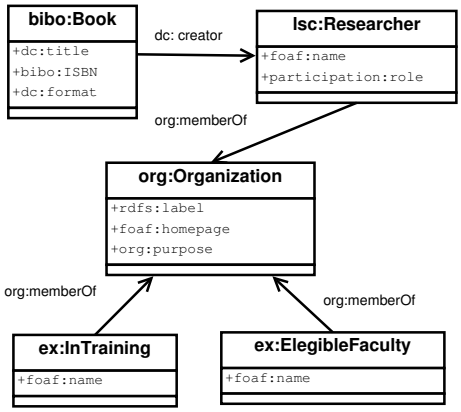

Figure 2: *ACADEMIC_OWL* target ontology schema modelled in UML.

3. SPECIFYING CAs

3.1 Basic concepts and notation

As usual, we denote a *relation schema* as $R[\mathcal{A}_1, \ldots, \mathcal{A}_n]$ and consider *mandatory* (or *not null*) *attributes, keys, primary keys* and *foreign keys* as integrity 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 $\mathbf{S} = (\mathbf{R_S}, \mathbf{C_S})$, where $\mathbf{R_S}$ is a set of relation schemas and $\mathbf{C_S}$ is a set of integrity constraints such that:

1. $\mathbf{C_S}$ has a unique primary key for each relation schema in $\mathbf{R_S}$;
2. $\mathbf{C_S}$ has a mandatory attribute constraint for each attribute, which is part of a key or primary key;
3. If $\mathbf{C_S}$ has a foreign key of the form $F(R : L, S : K)$, then $\mathbf{C_S}$ also has a constraint indicating that K is the primary key of S.

The *vocabulary* of $\mathbf{S}$ is the set of relation names, attribute names, etc. used in $\mathbf{S}$. Given a relation schema $R[A_1, \ldots, A_n]$ and a *tuple variable* t over R, we use $t.A_k$ to denote the *projection* of t over A_k. We use *selections* over relation schemas, defined as usual.

Let $\mathbf{S} = (\mathbf{R_S}, \mathbf{C_S})$ be a relational schema and R and T be relation schemas in $\mathbf{R_S}$. A list $\varrho = [F_1, \ldots, F_{n-1}]$ of foreign key names of $\mathbf{S}$ is a *path from* R *to* T iff there is a list $R_1, \ldots, R_n$ of relation schemas in $\mathbf{R_S}$ such that $R_1 = R$, $R_n = T$ and F_i relates R_i and R_{i+1}. We say that tuples of R *refer tuples of* T *through* ϱ. A path ϱ is an *association path* iff $\varrho = [F_1, F_2]$, where the foreign keys are of the form $F_1(R_1 : K_1, R_2 : L_2)$ and $F_2(R_2 : M_2, R_3 : K_3)$. For example, consider the relational schema *LIBRARY_REL* in Figure 1. The list of foreign keys names [*Fk_publication*, *Fk_author*] is an association path from **publication** to **author**, but [*Fk_student*, *Fk_book*] is not even a path.

We also recall a minimum set of concepts related to ontologies. A *vocabulary* is a set of names of *classes, object properties* or *datatype properties*. An *ontology* is a pair $\mathbf{O} = (\mathbf{V_O}, \mathbf{C_O})$ such that $\mathbf{V_O}$ is a vocabulary and $\mathbf{C_O}$ is a finite set of formulae in $\mathbf{V_O}$, the *constraints* of $\mathbf{O}$. 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 Basic Correspondence Assertion (CA)

In this section, we define the basic set of CAs, first introduced in [12] and present an extension, which is the first contribution of this paper. Let $\mathbf{S} = (\mathbf{R_S}, \mathbf{C_S})$ be a relational schema and $\mathbf{O} = (\mathbf{V_O}, \mathbf{C_O})$ be an ontology. Assume that $\mathbf{C_O}$ has constraints defining the domain and range of each property in $\mathbf{V_O}$.

DEFINITION 1. *A Class Correspondence Assertion (CCA) is an expression of one of the following forms:*

1. ψ: C $\equiv$ R$[\mathcal{A}_1, \ldots, \mathcal{A}_n]$
2. ψ: C $\equiv$ R$[\mathcal{A}_1, \ldots, \mathcal{A}_n]\sigma$

where ψ *is the name of the CCA,* C *is a class in* $\boldsymbol{V_O}$, R *is a relation name in the vocabulary of* $\boldsymbol{S}$, $\mathcal{A}_1, \ldots, \mathcal{A}_n$ *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. *An Object Property Correspondence Assertion (OCA) is an expression of one of the following forms:*

1. ψ: $\mathcal{P}$ $\equiv$ R
2. ψ: $\mathcal{P}$ $\equiv$ R$/\varrho$

where ψ *is the name of the OCA,* $\mathcal{P}$ *is an object property in* $\boldsymbol{V_O}$, R *is a relation name in the vocabulary of* $\boldsymbol{S}$ *and* ϱ *is a path from* R *to some relation schema in* $\boldsymbol{R_S}$. *We also say that* ψ *relates* $\mathcal{P}$ *with* R. □

The Datatype Property Correspondence Assertions (DCAs) introduced in [12] left out some types of mappings that can be useful in practice. For example, consider the relational schema *LIBRARY_REL* and the domain ontology *ACADEMIC_OWL* shown, respectively, in Figure 1 and 2.

1. The datatype property *participation:role* keeps the job of a person. The relational schema *LIBRARY_REL* does not have this information, but we know that any tuple in **student** has as job "student" and any tuple in **professor** has as job "professor".
2. *dc:title* corresponds to **book**.title, since they represent the same concept in the real world. However, we want to guarantee that the text is in uppercase. In [12], the authors deal only with concatenation of attribute values.

We extend DCAs in order to deal with both situations, as follows:

DEFINITION 3. *A Datatype Property Correspondence Assertion (DCA) is an expression of one of the following forms:*

1. ψ: $\mathcal{P} \equiv \mathrm{R}/\mathcal{A}$
2. ψ: $\mathcal{P} \equiv \mathrm{R}/\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$
3. ψ: $\mathcal{P} \equiv \mathrm{R}/\varrho/\mathcal{B}$
4. ψ: $\mathcal{P} \equiv \mathrm{R}/\varrho/\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$
5. ψ: $\mathcal{P} \equiv \mathrm{R}/\phi/\mathcal{A}$
6. ψ: $\mathcal{P} \equiv \mathrm{R}/\phi/\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$
7. ψ: $\mathcal{P} \equiv \mathrm{R}/\varrho/\phi/\mathcal{B}$
8. ψ: $\mathcal{P} \equiv \mathrm{R}/\varrho/\phi/\{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$
9. ψ: $\mathcal{P} \equiv \mathrm{R}/\boldsymbol{v}$

where ψ is the name of the DCA, $\mathcal{P}$ is a datatype property in $\boldsymbol{V_O}$, R is a relation name in the vocabulary of $\boldsymbol{S}$, $\mathcal{A}$ is an attribute of R, $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are attributes of R, ϱ is a path from R to some relation schema T in $\boldsymbol{R_S}$, $\mathcal{B}$ is an attribute of T and $\mathcal{B}_1, \ldots, \mathcal{B}_n$ are attributes of T, ϕ is a built-in predicate and $\boldsymbol{v}$ is a value (integer, string, etc.). We also say that ψ relates $\mathcal{P}$ with R. □

Examples of CAs are shown in Table 1 and are explained as follows. ψ_1 and ψ_2 are examples of CCAs. ψ_1, for instance, matches the class *bibo:Book* with the relation **book**. ψ_3 to ψ_{10} are examples of DCAs. ψ_3, for example, matches the datatype property *foaf:name* with the attribute *name* of relation **student**. ψ_6 matches the datatype property *participation:role* to the value "student", when the instance comes from the relation **student**, while ψ_7 matches the datatype property *participation:role* to the value "professor", when the instance comes from the relation **professor**. ψ_8 matches the datatype property *dc:title* with the attribute **publisher**.title and specifies that the value of *dc:title* should is in uppercase (through the build-in predicate "Upper"). ψ_{11} and ψ_{12} are examples of OCAs. ψ_{11}, for instance, matches the object property *dc:creator* to the relational path ϱ= [FK_book, FK_publication, FK_author], the path ϱ relates a book (in **book**) to a author (in **author**).

It is important to note that the algorithms in [12] do not cover the cases when there is more than one CA to the same datatype property or to the same objet property. We drop this restriction, which configures as a contribution of this paper.

3.3 Algebraic CAs

In this Section we define a set of CAs, named *Algebraic Correspondence Assertions*, which facilitates the definition of mappings that spam multiple database schemas, for example, and do not unduly increase the problem of compiling R2RML mappings.

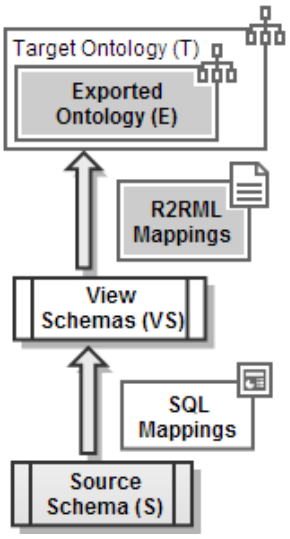

Figure 3: Three-level schema architecture.

Given a CA of the form $\psi : \mathcal{H} \equiv \mathcal{B}$, we say that $\mathcal{H}$ is the head and $\mathcal{B}$ is the body of the CA. We then recursively define Algebraic Correspondence Assertions as follows:

DEFINITION 4. *An Algebraic Class Correspondence Assertion (ACCA) is an expression of the form $\psi : \mathcal{H} \equiv \mathcal{B}$, where $\mathcal{H}$ is the ACCA head and $\mathcal{B}$ is the ACCA body of ψ, such that ψ is either*

1. *A CCA as in Definition 1, whose body is called an atomic ACCA body; or, recursively,*
2. *The ACCA head of ψ is a class in $\boldsymbol{V_O}$ and the ACCA body of ψ is of one of the forms:*
   - $\mathcal{B}_1 \cup \mathcal{B}_2$
   - $\mathcal{B}_1 \cap \mathcal{B}_2$
   - $\mathcal{B}_1 - \mathcal{B}_2$,

   where $\mathcal{B}_1$ and $\mathcal{B}_2$ are ACCA bodies. □

Similarly, we define algebraic OCAs (AOCAs) and algebraic DCAs (ADCAs). Examples of ACCAs are shown in Table 2. ψ_{13} defines a researcher as a professor or a graduate student, ψ_{14} defines a person in training as a professor, which is also a graduate student, and ψ_{15} defines a faculty member eligible to administrative positions, say, as a professor, which is not a graduate student.

4. OVERVIEW OF THE STRATEGY FOR GENERATION OF R2RML MAPPINGS

In this section we extend the strategy proposed in [12] to simplify the specification of R2RML mappings. The strategy has three layers, as depicted in Figure 3. The top layer features a *target ontology* **T**, 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 bottom layer has the source relational schema **S**.

Table 1: Example of Correspondence Assertions (CAs)

CCA	ψ_1: *bibo:Book* ≡ **book**[bookid]
CCA	ψ_2: *org:Organization* ≡ **organization**[orgID]
DCA	ψ_3: *foaf:name* ≡ **student**/*name*
DCA	ψ_4: *foaf:name* ≡ **professor**/*name*
DCA	ψ_5: *dc:format* ≡ **book**/*format*
DCA	ψ_6: *participation:role*≡ **student**/"student"
DCA	ψ_7: *participation:role*≡ **professor**/"professor"
DCA	ψ_8: *dc:title* ≡ **book**/Upper/[FK_book]/*title*
DCA	ψ_9: *rdfs:label* ≡ **organization**/*name*
DCA	ψ_{10}: *org:purpose* ≡ **organization**/*type*
OCA	ψ_{11}: *dc:creator* ≡ **book**/[FK_book, FK_publication, FK_author]
OCA	ψ_{12}: *org:memberOf* ≡ **author**/[FK_author_org, FK_organization]

Table 2: Example of Algebraic Class Correspondence Assertions (ACCAs)

ψ_{13}: *lsc:Researcher* ≡ **professor**[aid] ∪ **student**[aid] / level = graduate
ψ_{14}: *ex:inTraining* ≡ **professor**[aid] ∩ **student**[aid] / level = graduate
ψ_{15}: *ex:EligibleFaculty* ≡ **professor**[aid] − **student**[aid] / level = graduate

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 *SQL mappings* and the *R2RML mapping* can be automatically generated from the set of correspondence assertions between **S** and **T**.

The process for generating the R2RML mappings consists of four main steps:

- **Step 1:** The specification of a set of correspondence assertions between **S** and **T**.
- **Step 2:** The design of the exported ontology **E**.
- **Step 3:** The design of the set of relational view schemas **VS** and the R2RML mappings.
- **Step 4:** The generation of the SQL mappings for the views in **VS**.

Consider, for example, the *LIBRARY_REL* schema and the *ACADEMIC_OWL* domain ontology shown, respectively, in Figures 1 and 2, and the CAs in Table 1. Figure 4 shows the exported ontology *LIBRARY_RDF* generated by the algorithm presented in [3]. The vocabulary of *LIBRARY_RDF* contains all elements of the *ACADEMIC_OWL* ontology that match some elements of *LIBRARY_REL*.

Figure 5 depicts *LIBRARY_VIEW*, the relational view schema generated for the exported ontology *LIBRARY_RDF* output by Algorithm 1 in [12]. Listing 1 shows the R2RML mapping generated to class *lsc:Research* using the templates proposed in [12].

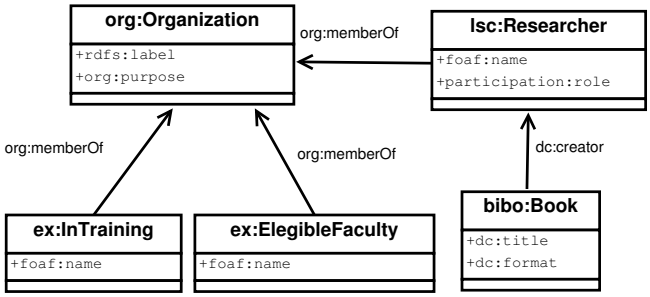

Figure 4: Exported Ontology *LIBRARY_RDF*.

Listing 1: Example of a R2RML mapping generated to class *lsc:Research*

```
<# researcher_TriplesMap>
rr:logicalTable[rr:tableName:"Researcher"];
rr:subjectMap[
    rr:template "http//linkedscience.org/
                /lsc/ns/researcher/{aid}";
    rr:class lsc:Researcher;];
rr:predicateObjectMap[
rr:predicate foaf:name;
    rr:ObjectMap [rr:column "name";];]
rr:predicateObjectMap[
rr:predicate participation:role;
    rr:ObjectMap [rr:column "role";];]
```

Finally, we observe that steps 2 and 3 are not affected by the algebraic correspondence assertions proposed in this paper, since all the complexity of the mappings are embedded in the SQL mappings defined in step 4. In the following section we present an algorithm to automatically generate

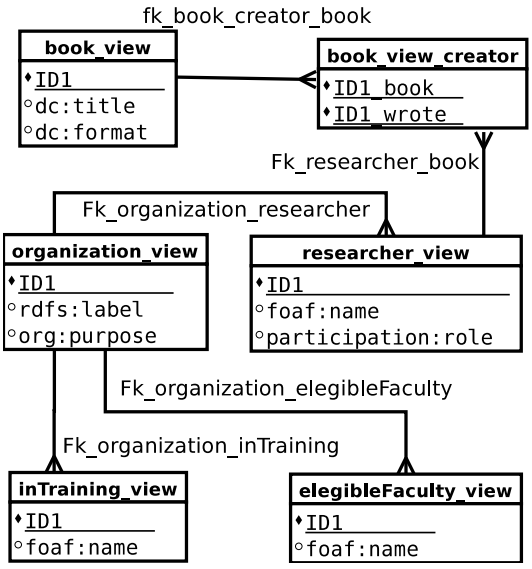

Figure 5: *LIBRARY_VIEW* relational view schema.

SQL mappings, based on the correspondence assertions between **S** and **T**. It is worth to mention that step 4 was not addressed in [12].

5. GENERATION OF SQL MAPPINGS

In this section we present Algorithm 1 Generate_RView() that automatically generates the SQL mappings for the set of views **VS**. Algorithm 1 is executed for each class C in $\mathbf{V_E}$, with ψ_C being an ACCA of C, and consists of the two main steps:

- **Step1:** ACCA body of ψ_C is atomic. Step 1 creates the SQL query for generating a relational view using templates in Table 3.
  - **Step 1.1:** processes the datatype properties. Step 1.1 generates the SQL statements obtained from datatype properties whose domain is C. This is performed by Algorithm 2 *Generate_SQL_DCA()*.
  - **Step 1.2:** processes the object properties. Step 1.2 generates the SQL statements obtained from object properties whose domain is C. This is performed by Algorithm 3 *Generate_SQL_OCA()*.
- **Step 2:** ACCA body is not atomic and has the form: $\mathcal{B}_1$ *op* $\mathcal{B}_2$, where $op \in \{\cup, \cap, -\}$. Step 2 recursively run Algorithm 1 to create at least three relational views: $\mathbf{V}_{\mathcal{B}1}$ obtained from $\mathcal{B}_1$, $\mathbf{V}_{\mathcal{B}2}$ obtained from $\mathcal{B}_2$ and **V** obtained from $\mathcal{B}_1$ *op* $\mathcal{B}_2$. The views are generated using templates in Table 3.

In Table 3, we have: **N** is a class name; **T**, **J**, **LA**, θ and **RJ** are lists to keep, respectively, the relation schemas that will be included in the FROM clause, the *join conditions* that will be included in the WHERE clause, the attributes that will be included in the SELECT clause, the *join attributes* that will be included in the ON clause, and the relation schema that will be included in (inner, outer or left) JOIN clause; σ is the selection condition specified by a CA. The value of these variables are obtained after Algorithm 1 is executed. We do not show the algorithm details here to simplify the reading. For more details see [8].

Algorithm 1 Generate_Rview.

Input: a class C of the exported ontology **OE**= $(\mathbf{V_E}, \mathbf{C_E})$, $[K_1, \ldots, K_n]$ data proprieties included in IRI of C, ψ_C: $\mathcal{H} \equiv \mathcal{B}$ be a ACCA of C with $\mathcal{H}$ is the ACCA head and $\mathcal{B}$ is the ACCA body, and a set **A** of CAs
Output: The SQL of a relational view belonging to view schema $\mathbf{V_S}$
Let $\mathcal{P}$ be a property of C and $\psi_\mathcal{P}$ be a CA of $\mathcal{P}$
if $\mathcal{B}$ is an atomic ACCA body **then**
  Generate_SQL_DCA(C, ψ_C, **A**)
  Genertate_SQL_OCA(C, ψ_C, **A**)
  Create **V** using a template TV in Table 3
  /* TV is choice based on a) type of ψ_C and b) if, for some $\mathcal{P}$ */
  /*of C, $\psi_\mathcal{P}$ has a path or not */
else
  Let $\mathcal{B}$ be of the form $\mathcal{B}_1$ *op* $\mathcal{B}_2$, where $op \in \{\cup, \cap, -\}$
  $\mathbf{V}_{\mathcal{B}1}$ = Generate_RView(C, $[K_1, \ldots, K_n]$,$\psi_C : C \equiv \mathcal{B}_1$,**A**)
  $\mathbf{V}_{\mathcal{B}2}$ = Generate_RView(C, $[K_1, \ldots, K_n]$,$\psi_C : C \equiv \mathcal{B}_2$,**A**)
  Create **V** using a template TV in Table 3
  /* TV is choice based on the type of ψ_C */
end if
Returns **V**

Algorithm 2 *Generate_SQL_DCA()* processes the datatype properties in $\mathbf{V_E}$ and generates the corresponding SQL statements. It has two main steps:

- **Step 1:** For each datatype property whose cardinality is equal to 1, step 1 generates an attribute to the relational view being created.
- **Step 2:** For each datatype property whose cardinality is greater than 1, step 2 creates a relational view containing a foreign key and an attribute.

Algorithm 2 Generate_SQL_DCA

Input: a class C of the exported ontology **OE**, the CCA ψ_C of C that matches C with a relation R, and a set of CAs between **OE** and the relational schema **S**
Output: SQL statements to the relational view C_view belonging to view schema **V**
for all data property $\mathcal{P}$ in **OE**, where $Dom(\mathcal{P})$ = C **do**
  Let $\psi_\mathcal{P}$ be a DCA of $\mathcal{P}$, such $\psi_\mathcal{P}$ relates $\mathcal{P}$ with R
  if $\mathcal{P}$ has cardinality equal to 1 **then**
    add an expression to **LA** in accordance to type of $\psi_\mathcal{P}$
    if $\psi_\mathcal{P}$ has a path ϱ **then**
      for all Foreigh key FK in ϱ **do**
        update **Jp** with join conditions obtained from FK
        update **Tp** with relation names obtained from FK
      end for
    end if
  else $\mathcal{P}$ has cardinality greater than 1 and $\psi_\mathcal{P}$ has a path ϱ
    Let $[K_1, \ldots, K_n]$ be data proprieties included in IRI of C
    add an expression to **LAp** based on $[K_1, \ldots, K_n]$
    add an expression to **LAp** in accordance to type of $\psi_\mathcal{P}$
    for all Foreigh key FK in ϱ **do**
      update **Jp** with join conditions obtained from FK
      update **Tp** with relation names obtained from FK
    end for
    Create $\mathbf{V}'$ using a template TV in Table 3
    /* TV is choice based on the type of ψ_C */
  end if
end for

In Algorithm 2, we use the variables **Np**, **Tp**, **Jp**, and **LAp** to keep the values to the creation of the new view. We assume that **Np** is a variable to keep the name to the new view

Table 3: Templates to generate SQL Statements induced by CAs.

TV1	*create or replace view* **N** *as* *select* **LA**[1], **LA**[2], ..., **LA**[n] *from* **T**[1]	ψ: C $\equiv$ R[$\mathcal{A}_1, \ldots, \mathcal{A}_n$]
TV2	*create or replace view* **N** *as* *select* **LA**[1], **LA**[2], ..., **LA**[n] *from* **T**[1] *where* σ	ψ: C $\equiv$ R[$\mathcal{A}_1, \ldots, \mathcal{A}_n$]$\sigma$
TV3	*create or replace view* **N** *as* *select* **LA**[1], **LA**[2], ..., **LA**[n] *from* **T**[1], **T**[2], ..., **T**[m] *where* **J**[1] *and* **J**[2] *and* ... *and* **J**[t]	ψ: C $\equiv$ R[$\mathcal{A}_1, \ldots, \mathcal{A}_n$] and $\exists\psi_{\mathcal{P}} \mid \psi_{\mathcal{P}}$ has a path
TV4	*create or replace view* **N** *as* *select* **LA**[1], **LA**[2], ..., **LA**[n] *from* **T**[1], **T**[2], ..., **T**[m] *where* **J**[1] *and* **J**[2] *and* ... *and* **J**[t] *and* σ	ψ: C $\equiv$ R[$\mathcal{A}_1, \ldots, \mathcal{A}_n$]$\sigma$ and $\exists\psi_{\mathcal{P}} \mid \psi_{\mathcal{P}}$ has a path
TV5	*create or replace view* **N** *as* *select* * *from* **T**[1] *outer join* **RJ**[1] *on* θ	ψ : C $\equiv \mathcal{B}_1 \cup \mathcal{B}_2$
TV6	*create or replace view* **N** *as* *select* * *from* **T**[1] *inner join* **RJ**[1] *on* θ	ψ : C $\equiv \mathcal{B}_1 \cap \mathcal{B}_2$
TV7	*create or replace view* **N** *as* *select* * *from* **T**[1] *left join* **RJ**[1] *on* θ	ψ : C $\equiv \mathcal{B}_1 - \mathcal{B}_2$

and **Tp**, **Jp**, **LAp** are lists to keep, respectively, the relation schemas that will be included in the FROM clause, the *join conditions* that will be included in the WHERE clause and the attributes that will be included in the SELECT clause.

Listing 2: SQL query to create researcher_view.

```
1 CREATE OR REPACE VIEW researcher_view AS
2     SELECT * FROM professor_view
3     OUTER JOIN student_view
4     ON professor_view.ID1 = student_view.ID1
```

Listing 2 presents the SQL query to create the view *lsc:researcher_view*, which was generated based on the CAs ψ_3, ψ_4, ψ_6, ψ_7 and ψ_{13}. Since ψ_{13} is an ACCA of union, the SQL query generated is the union of two views: *professor_view* and *student_view*. *professor_view*, shown in Listing 3, is a view generated from the relation **professor** and contains all tuples of this relation (obtained from the left-hand of the ACCA body ψ_{13} and DCAs ψ_4 and ψ_7). *student_view*, shown in Listing 4, is a view generated from the relation **student** and contains the tuples of this relation whose level = "graduate" (obtained from the right-hand of the ACCA body ψ_{13} and DCAs ψ_3 and ψ_6). We use the SQL operator OUTER JOIN to create the view *lsc:researcher_view*.

Listing 3: SQL query to create *professor_view*.

```
1 CREATE OR REPLACE VIEW professor_view AS
2     SELECT professor.aid AS ID1,
3            professor.name AS foaf:name
4            "professor" AS participation:role
5     FROM professor
```

In Listing 3, the select clause (line 02-04) was derived based on, respectively, CCA ψ_{13} and DCAs ψ_4 and ψ_7. The FROM clause (line 05) was derived based on the CCA ψ_{13}. The query in Listing 4 was created in similar way using the CAs: ψ_{13}, ψ_3 and ψ_6.

Listing 4: SQL query to create *professor_view*.

```
1 CREATE OR REPLACE VIEW student_view AS
2     SELECT student.aid AS ID1,
3            student.name AS foaf:name
4            "student" AS participation:role
5     FROM student
6     WHERE student.level = "graduate"
```

Algorithm 3 *Generate_SQL_OCA()* processes the object properties in $\mathbf{V_E}$ and generates the corresponding SQL statements. It has two main steps:

- **Step 1:** For each object property whose cardinality is equal to 1, step 1 generates a foreign key to the relational view being created.
- **Step 2:** For each object property whose cardinality is greater than 1, step 2 creates a relational view containing two foreign keys.

In Algorithm 3, we use the variables **No**, **To**, **Jo**, and **LAo** to keep the values to the creation of the new view. We assume that **No** is a variable to keep the name to the new view and **To**, **Jo**, **LAo** are lists to keep, respectively, the relation schemas that will be included in the FROM clause, the *join conditions* that will be included in the WHERE clause, the attributes that will be included in the SELECT clause.

Algorithm 3 Generate_SQL_OCA

Input: a class C of the exported ontology **OE**, and a set of CAs between **OE** and the relational schema **S**
Output: SQL statements to the relational view C_view belonging to view schema **V**

```
for all object property O in OE, where Dom(O) = C do
    Let ψ_O be a OCA of O
    if O has cardinality equal to 1 then
        if ψ_O has a path ϱ and length(ϱ) > 1 then
            for all Foreigh key FK in ϱ do
                update J with join conditions obtained from FK
                update T with relation names obtained from FK
                if FK is the last key in ϱ then
                    add an expression to LA based on FK
                end if
            end for
        else
            add an expression to LA based on type of ψ_O
        end if
    else Cardinality of O is greater than 1
        Let [K_1, ..., K_n] be data proprieties included in IRI of C
        add an expression to LAo based on [K_1, ..., K_n]
        for all Foreigh key FK in ϱ do
            update Jo with join conditions obtained from FK
            update To with relation names obtained from FK
            if FK is the last key in ϱ then
                add an expression to LAo based on FK
            end if
        end for
        Create V' using a template TV in Table 3
        /*TV is choice based on the type of ψ_C */
    end if
end for
```

Listing 5 shows the SQL query to create the view *book_view*. It was generated based on the CAs ψ_1, ψ_5, ψ_8 and ψ_{11}. The SELECT clause (line 02-04) was derived based on, respectively, DCAs ψ_8 (dc:title), ψ_5 (dc:format) and CCA ψ_1 (ID1). The FROM clause (line 05) and WHERE clause (line 06) both were derived based on CCA ψ_1. Since object property *dc:creator* has cardinality greater than 1, a new view should be generated, named *book_view_creator*. This new view is shown in Listing 6.

Listing 5: SQL query to create *book_view*.

```
1 CREATE OR REPLACE VIEW book_view AS
2   SELECT UPPER(publication.title) AS dc_title,
3          book.format AS dc:format,
4          book.bookid  AS ID1
5   FROM book, publication
6   WHERE book.bookid = publication.pubid
```

The SQL query shown in Listing 6 was generated based on the OCA ψ_{11}. The view *book_view_creator* keeps two foreign keys: ID1_bibo:book_view that refers to **book** and ID1_wrote that refers to **author**.

Listing 6: SQL query to create *book_view_creator*.

```
1 CREATE OR REPLACE VIEW book_view_creator AS
2   SELECT bookId AS ID1_book,
3          author.aid AS ID1_wrote
4   FROM book, publication, wrote, author
5   WHERE book.bookId = publication.publd AND
6     publication.publd = wrote.publication AND
7     wrote.author = author.aid
```

6. CONCLUSIONS

We introduced in this paper an extended set of correspondence assertions, which includes data-metadata mappings, mappings containing custom value functions and union, intersection and difference between tables. This extension covers a wider range of mappings and yet maintains the usability aspects of correspondence assertions. Furthermore, we described how to automatically compile algebraic correspondence assertions into R2RML mappings with the help of SQL views, used as an intermediated mapping step. This constitutes the major contributions of the paper, which significantly enhances the results reported in [12].

Another important contribution of this paper is the automatic generation of SQL mappings from the algebraic correspondence assertions. This simplifies the SQL view creation for the users by automating tedious tasks and guarantees the correctness of the SQL view query.

We are currently working on the development of heuristics to discover new correspondence assertions from those already defined, based on a collection of inference rules. This mechanism is intended to help the designer define new correspondence assertions, providing insights and suggestions, in an interactive way. We are also working on the development of an authoring tool, which supports extended correspondence assertions and incorporates the heuristics.

7. ACKNOWLEDGMENTS

This work was partially supported by national funds through FCT - Fundação para a Ciência e a Tecnologia, under the project PEst-OE/EEI/LA0021/2013, DataStorm Research Line of Excellency funding (EXCL/EEI-ESS/0257/2012) and the grant SFRH/BPD/76024/2011; by CNPq, under grant 303332/2013-1 and by FAPERJ, under grant E-26/103.070/2011.

8. REFERENCES

[1] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: Light-weight linked data publication from relational databases. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 621–630, New York, NY, USA, 2009. ACM.

[2] C. Bizer and R. Cyganiak. D2r server – publishing relational databases on the semantic web. Poster at the 5th International Semantic Web Conference (ISWC2006), 2006.

[3] M. A. Casanova, K. K. Beitman, A. L. Furtado, V. M. P. Vidal, J. A. F. Macedo, R. V. A. Gomes, and P. E. R. Salas. The role of constraints in linked data. In *Proceedings of the 2011th Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part II*, OTM'11, pages 781–799, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. Materialized View-Based Processing of RDF Queries. In *Bases de Données Avancées*, Toulouse, France, Oct. 2010.

[5] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. RDFviews: a storage tuning wizard for

RDF applications. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM'10, pages 1947–1948, New York, NY, USA, 2010. ACM.

[6] M. Hert, G. Reif, and H. C. Gall. A comparison of RDB-to-RDF mapping languages. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics'11, pages 25–32, New York, NY, USA, 2011. ACM.

[7] F. Michel, J. Montagnat, and C. Faron-Zucker. A survey of RDB to RDF translation approaches and tools. Technical Report Document ID. CrEDIBLE-13-2-v1 – I3S/RR 2013-04-FR, Univ. Nice Sophia Antipolis, France, 2013.

[8] L. E. T. Neto. Uma abordagem para publicação de visões RDF de dados relacionais. Master's thesis, Universidade Federal do Ceará, 2014.

[9] L. E. T. Neto, V. M. P. Vidal, M. A. Casanova, and J. M. Monteiro. R2RML by assertion: A semi-automatic tool for generating customised R2RML mappings. In P. Cimiano, M. Fernández, V. Lopez, S. Schlobach, and J. Völker, editors, *The Semantic Web: ESWC 2013 Satellite Events*, volume 7955 of *Lecture Notes in Computer Science*, pages 248–252. Springer Berlin Heidelberg, 2013.

[10] S. Sahoo, W. Halb, S. Hellmann, K. Idehen, J. T. Thibodeau, S. Auer, J. Sequeda, and A. Ez-zat. A survey of current approaches for mapping of relational databases to RDF. Technical report, W3C RDB2RDF Incubator Group report, 2009.

[11] J. F. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to RDF and OWL. In *Proceedings of the 21st International Conference on World Wide Web*, WWW'12, pages 649–658, New York, NY, USA, 2012. ACM.

[12] V. Vidal, M. Casanova, J. Monteiro, and L. Neto. A semi-automatic approach for generating customized R2RML mappings. In *Proceedings of 29th Symposium On Applied Computing*, Gyeongju, Korea, March 2014.

[13] V. M. P. Vidal, M. A. Casanova, and D. S. Cardoso. Incremental maintenance of RDF views of relational data. In *Proc. The 12th International Conference on Ontologies, DataBases, and Applications of Semantics*, Graz, Austria, 2013.