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.

"Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IDEAS ’14, July 07 – 09 2014, Porto, Portugal
Copyright ©2014 ACM 978-1-4503-2627-8/14/07 $15.00"
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 SquirrelRDF1). 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

1url: 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 Virtuoso3 and Ontop 4. 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 creation 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. 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.
3. SPECIFYING CAs

3.1 Basic concepts and notation

As usual, we denote a relation schema as $R[(A_1, \ldots, A_n)]$ 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 $S = (R_S, C_S)$, where $R_S$ is a set of relation schemas and $C_S$ is a set of integrity constraints such that:

1. $C_S$ has a unique primary key for each relation schema in $R_S$;
2. $C_S$ has a mandatory attribute constraint for each attribute, which is part of a key or primary key;
3. If $C_S$ has a foreign key of the form $F(R : L, S : K)$, then $C_S$ 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 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 $S = (R_S, C_S)$ be a relational schema and $R$ and $T$ be relation schemas in $R_S$. A list $\rho = [F_1, \ldots, F_n-1]$ of foreign key names of $S$ is a path from $R$ to $T$ if there is a list $R_1, \ldots, R_n$ of relation schemas in $R_S$ 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 $\rho$. A path $\rho$ is an association path if $\rho = [F_1, F_2]$, where the foreign keys are of the form $F_1(R_1 : K_1, R_2 : K_2)$ 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 $O = (V_O, C_O)$ such that $V_O$ is a vocabulary and $C_O$ is a finite set of formulae in $V_O$, the constraints of $O$. Among the constraints, we consider those that define the domain and range of a property, as well as cardinality constraints, defined in the usual way.

3.2 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 $S = (R_s, C_s)$ be a relational schema and $O = (V_O, C_O)$ be an ontology. Assume that $C_O$ has constraints defining the domain and range of each property in $V_O$.

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

1. $\psi : C \equiv R[A_1, \ldots, A_n]$
2. $\psi : C \equiv R[A_1, \ldots, A_n][\sigma]$

where $\psi$ is the name of the CCA, $C$ is a class in $V_O$, $R$ is a relation name in the vocabulary of $S$, $A_1, \ldots, A_n$ are the attributes of the primary key of $R$ and $\sigma$ is a selection over $R$. We also say that $\psi$ matches $C$ with $R$.

**Definition 2.** An Object Property Correspondence Assertion (OCA) is an expression of one of the following forms:

1. $\psi : P \equiv R$
2. $\psi : P \equiv R/\rho$

where $\psi$ is the name of the OCA, $P$ is an object property in $V_O$, $R$ is a relation name in the vocabulary of $S$ and $\rho$ is a path from $R$ to some relation schema in $R_s$. We also say that $\psi$ relates $P$ 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. $\psi: P \equiv R/A$
2. $\psi: P \equiv R/\{A_1, \ldots, A_n\}$
3. $\psi: P \equiv R/\{B_1, \ldots, B_n\}$

Examples of CAs are shown in Table 1 and are explained as follows. $\psi_1$ and $\psi_2$ are examples of CAs. $\psi_1$, for instance, matches the class `bibo:Book` with the relation `book`, the path `publisher` to the value “publisher”, the head `H` specifies that the value of `publisher` should be in uppercase (through the build-in predicate “Upper”). $\psi_2$ matches the object property `dc:title` with the attribute `author`.

Similarly, we define algebraic OCAs (AOCAs) and algebraic DCAs (ADCAs). Examples of ACCAs are shown in Table 2. $\psi_1$ defines a researcher as a professor or a graduate student, while $\psi_2$ defines a researcher as a professor or a graduate student, which is also a graduate student. $\psi_3$ and $\psi_4$ are ACCA bodies.

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 object 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.

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

1. A CCA as in Definition 1, whose body is called an atomic ACCA body; or, recursively,
2. The ACCA head of $\psi$ is a class in $V_o$ and the ACCA body of $\psi$ is of one of the forms:
   - $B_1 \cup B_2$
   - $B_1 \cap B_2$
   - $B_1 - B_2$

where $B_1$ and $B_2$ are ACCA bodies.

Given a CA of the form $\psi: H \equiv B$, we say that $H$ is the head and $B$ 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 $\psi: H \equiv B$, where $H$ is the ACCA head and $B$ is the ACCA body of $\psi$, such that $\psi$ is either

1. A CCA as in Definition 1, whose body is called an atomic ACCA body; or, recursively,
2. The ACCA head of $\psi$ is a class in $V_o$ and the ACCA body of $\psi$ is of one of the forms:
   - $B_1 \cup B_2$
   - $B_1 \cap B_2$
   - $B_1 - B_2$

where $B_1$ and $B_2$ are ACCA bodies.

Similarly, we define algebraic OCAs (AOCAs) and algebraic DCAs (ADCAs). Examples of ACCAs are shown in Table 2. $\psi_1$ defines a researcher as a professor or a graduate student, which is also a graduate student, and $\psi_2$ 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 $S$. 

![Three-level schema architecture](image)
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].

### Table 1: Example of Correspondence Assertions (CAs)

<table>
<thead>
<tr>
<th>CCA</th>
<th>$\psi_1$: bibo:Book $\equiv$ book[bookid]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCA</td>
<td>$\psi_2$: org:Organization $\equiv$ organization[orgID]</td>
</tr>
<tr>
<td>DCA</td>
<td>$\psi_3$: foaf:name $\equiv$ student/name</td>
</tr>
<tr>
<td>DCA</td>
<td>$\psi_4$: foaf:name $\equiv$ professor/name</td>
</tr>
<tr>
<td>DCA</td>
<td>$\psi_5$: dc:format $\equiv$ book/format</td>
</tr>
<tr>
<td>DCA</td>
<td>$\psi_6$: participation:role $\equiv$ student/“student”</td>
</tr>
<tr>
<td>DCA</td>
<td>$\psi_7$: participation:role $\equiv$ professor/“professor”</td>
</tr>
<tr>
<td>DCA</td>
<td>$\psi_8$: dc:title $\equiv$ book/Upper/[FK_book]/title</td>
</tr>
<tr>
<td>DCA</td>
<td>$\psi_9$: rdfs:label $\equiv$ organization/name</td>
</tr>
<tr>
<td>DCA</td>
<td>$\psi_{10}$: org:purpose $\equiv$ organization/type</td>
</tr>
<tr>
<td>OCA</td>
<td>$\psi_{11}$: dc:creator $\equiv$ book/[FK_book, FK_publication, FK_author]</td>
</tr>
<tr>
<td>OCA</td>
<td>$\psi_{12}$: org:memberOf $\equiv$ author/[FK_author_org, FK_organization]</td>
</tr>
</tbody>
</table>

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

| $\psi_{13}$: lsc:Researcher $\equiv$ professor[aid] $\cup$ student[aid] / level = graduate |
| $\psi_{14}$: ex:InTraining $\equiv$ professor[aid] $\cap$ student[aid] / level = graduate |
| $\psi_{15}$: ex:EligibleFaculty $\equiv$ professor[aid] $-$ student[aid] / level = graduate |

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

```
< # researcher_TriplesMap>
rr:logicalTable [ rr:tableName:“Researcher” ];
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” ]; ]
```
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 V_E, with \( \psi_C \) being an ACCA of C, and consists of the two main steps:

- **Step 1:** ACCA body of \( \psi_C \) 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: \( B_1 \ op \ B_2 \), where \( op \in \{ \cup, \cap, - \} \). Step 2 recursively runs Algorithm 1 to create at least three relational views: \( V_{B_1} \) obtained from \( B_1 \), \( V_{B_2} \) obtained from \( B_2 \) and V obtained from \( B_1 \ op \ B_2 \). The views are generated using templates in Table 3.

In Table 3, we have: N is a class name; T, J, LA, \( \theta \) 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; \( \sigma \) 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 2 Generate_SQL_DCA() processes the datatype properties in 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 3 Generate_SQL_OCA() processes the object properties in V_E and generates the corresponding SQL statements. It has two main steps:

- **Step 1:** If \( \psi_C \) is an atomic ACCA body then
  - Generate_SQL_DCA(\( \psi_C \), C)
  - Generate_SQL_OCA(\( \psi_C \), C)
- **Step 2:** If \( \psi_C \) is an atomic ACCA body then
  - Generate_SQL_DCA(\( \psi_C \), C)
  - Generate_SQL_OCA(\( \psi_C \), C)

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.

<table>
<thead>
<tr>
<th>TV1</th>
<th>create or replace view N as select LA[1], LA[2],..., LA[n] from T[1]</th>
<th>ψ: C ⊇ R[A_1,...,A_n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>TV2</td>
<td>create or replace view N as select LA[1], LA[2],..., LA[n] from T[1]</td>
<td>ψ: C ⊇ R[A_1,...,A_n]</td>
</tr>
<tr>
<td></td>
<td>where σ</td>
<td></td>
</tr>
<tr>
<td>TV3</td>
<td>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]</td>
<td>ψ: C ⊇ R[A_1,...,A_n] and ∃ψ_P</td>
</tr>
<tr>
<td>TV4</td>
<td>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</td>
<td>ψ: C ⊇ R[A_1,...,A_n]</td>
</tr>
<tr>
<td>TV5</td>
<td>create or replace view N as select * from T[1] outer join RJ[1] on θ</td>
<td>ψ: C ⊇ B_1 ∪ B_2</td>
</tr>
<tr>
<td>TV6</td>
<td>create or replace view N as select * from T[1] inner join RJ[1] on θ</td>
<td>ψ: C ⊇ B_1 ∩ B_2</td>
</tr>
<tr>
<td>TV7</td>
<td>create or replace view N as select * from T[1] left join RJ[1] on θ</td>
<td>ψ: C ⊇ B_1 - B_2</td>
</tr>
</tbody>
</table>

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.

```sql
CREATE OR REPLACE VIEW researcher_view AS
SELECT * FROM professor_view
OUTER JOIN student_view
ON professor_view.ID1 = student_view.ID1
```

Listing 2 presents the SQL query to create the view lsql:researcher_view, which was generated based on the CAs ψ_13, ψ_4, ψ_6, and ψ_17. 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_17). 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 lsl:researcher_view.

Listing 3: SQL query to create professor_view.

```sql
CREATE OR REPLACE VIEW professor_view AS
SELECT professor.aid AS ID1,
   professor.name AS foaf:name
   "professor" AS participation:role
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.

Algorithm 3 Generate_SQL_OCA() processes the object properties in VE 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 Lap 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, Lap are lists to keep, respectively, the relational 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
  if O has cardinality equal to 1 then
    for all Foreign key FK in O do
      add an expression to LA based on FK
    end if
  else
    add an expression to LA based on type of O
  end if
end for
else
  Cardinality of O is greater than 1
  Let [K_1, ..., K_n] be data properties included in IRI of C
  add an expression to LAO based on [K_1, ..., K_n]
  update Jo with a join condition obtained from FK
  update To with relation names obtained from FK
  if FK is the last key in O then
    add an expression to LAO based on FK
  end if
end if

end for

Listing 5: SQL query to create book_view.

```sql
CREATE OR REPLACE VIEW book_view AS
SELECT UPPER(book.title) AS dc_title,
       book.format AS dc:format,
       book.bookid AS ID1
FROM book, publication
WHERE book.bookid = publication.pubid
```

The SQL query shown in Listing 6 was generated based on the OCA ψ11. The view book_view_creator keeps two foreign keys: ID1_bibobook_view that refers to book and ID1_wrote that refers to author.

Listing 6: SQL query to create book_view_creator.

```sql
CREATE OR REPLACE VIEW book_view_creator AS
SELECT bookid AS ID1_book,
       author.aid AS ID1_wrote
FROM book, publication, wrote, author
WHERE book.bookid = publication.pubid AND
      publication.pubid = wrote.publication AND
      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 through national funds through FCT - Fundação para a Ciência e a Tecnologia, under the project PEst-OE/EEI/LA0021/2013, DataStorm Research Line of Excellence funding (EXCL/EEI-ESS/0257/2012) and the grant SFHR/BPD/76024/2011; by CNPq, under grants 303332/2013-1 and by FAPERJ, under grant E-26/103.070/2011.

8. REFERENCES


