MIRROR: Automatic R2RML Mapping Generation from Relational Databases

Luciano Frontino de Medeiros[1], Freddy Priyatna[2(✉)], and Oscar Corcho[2]

[1] UNINTER, Curitiba, Brasil
luciano.me@uninter.com
[2] Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain
ocorcho@fi.upm.es

Abstract. Two W3C recommendations exist for the transformation of RDB content into RDF: Direct Mapping (DM) and R2RML. The DM recommendation specifies the set of fixed transformation rules, whilst R2RML allows customising them. Here we describe the MIRROR system, which generates two sets of R2RML mappings. First, it creates a set of mappings that allow any R2RML engine to generate a set of RDF triples homomorphic to the ones that a DM engine would generate (they only differentiate in the URIs used). This allows R2RML engines to exhibit a similar behaviour to that of DM engines. Second, it produces an additional set of R2RML mappings that allow generating triples resulting from the implicit knowledge encoded in relational database schemas, such as subclass-of and M-N relationships. We demonstrate the behaviour of MIRROR using the W3C DM Test Case together with an extended version of one of its databases.

1 Introduction

The W3C RDB2RDF (Relational Database to Resource Description Framework) Working Group was created in 2009 with the mission to standardize languages for mapping relational data and relational database schemas into RDF and OWL. As a result of the work in the group, two recommendations were published in September 2012: Direct Mapping [1] and R2RML [2]. The former specifies the terms generation rules to be applied to generate automatically an RDF dataset that reflects the structure and content of the relational database. Since this may not be always adequate or optimal, especially in those cases when the relational database content needs to be transformed into RDF according to an existing ontology, R2RML allows customising the terms generation rules to be applied.

Hence R2RML provides more flexibility than its counterpart, the Direct Mapping specification. However, this comes at a cost for users interested in generating RDF from their relational databases: they need to learn how to create those R2RML mappings. Several tools have been made available to facilitate the task of mapping generation, as discussed in Section 2, but either they produce mappings in earlier RDB2RDF languages (e.g. the ODEMapster GUI, which produces R2O mappings) or are not usable enough (e.g. form-based tools that

P. Cimiano et al. (Eds.): ICWE 2015, LNCS 9114, pp. 326–343, 2015.
DOI: 10.1007/978-3-319-19890-3_21

only provide syntactic sugar to users, who still require a good knowledge of R2RML). An alternative approach to ease the burden of R2RML creation from users, making them more efficient, is to bootstrap the process with the creation of an initial R2RML mapping document that reflects the behaviour of the Direct Mapping specification, and then allow users to edit that document further, e.g. in a text editor. This has generally proven to be useful in our own work, since in many cases a large percentage of triple maps inside an R2RML mapping document are reused. This has also an additional positive side effect, which is the fact that any R2RML engine (e.g. morph-RDB) can be used to produce RDF following the Direct Mapping specification.

Furthermore, we have already pointed out that the Direct Mapping generates an RDF dataset that reflects the structure and content of the relational database. However, there are some well-known and widely-applied relational database patterns that usually encode some additional information that may be useful in this transformation process. For instance, some combinations of primary and foreign keys in relational tables are commonly used to represent parent-child, 1-N and M-N relationships between tables. This means that we may be able to push our approach further by generating as well some of that implicit information (such as subclass-of relationships, some specific object properties, etc.), in addition to the mappings generated following the Direct Mapping specification.

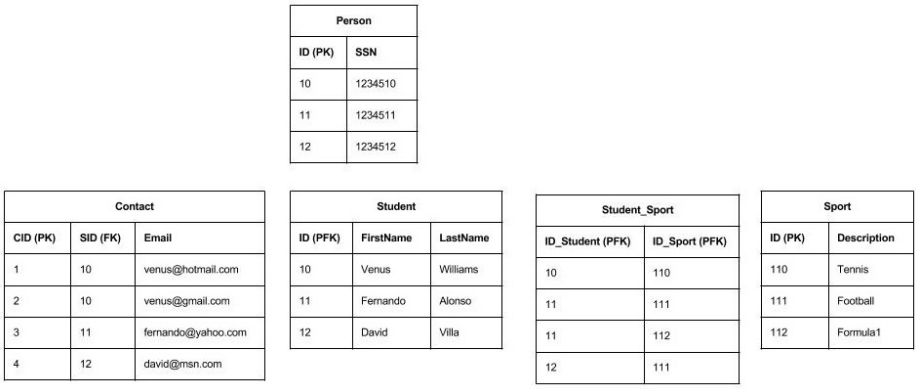
Motivational Example. Let us see an example, which will be used throughout the rest of the paper. Consider the database `D011`[1], from the W3C Direct Mapping Test Cases, with tables: `Student(ID, FirstName, LastName)` and `Sport(ID, Description)`, where `ID` is the primary key in both cases; and `Student_Sport( ID_Student, ID_Sport)`, where both columns form a composite primary key, and where `ID_Student` is a foreign key that refers to the column `ID` of the table `Student` and `ID_Sport` is a foreign key that refers to the column `ID` of the table `Sport`.

The constraints specified by the primary/foreign keys in the table `Student_Sport` represent an M-N relationship between table `Student` and table `Sport`. Thus, when transforming this database into RDF, one can expect that there will be an object property, for example `hasStudent`, with `Sport` as its domain and `Student` as its range, or viceversa with object property `hasSport`.

Now we will make the following modifications to that database, and will name the resulting database `D011B`. Figure 1 provides its graphical representation.

- Add another table `Person(ID, SSN)`, with `ID` as the primary key. We also add a foreign key constraint to the column `ID` of table `Student`, which refers to the column `ID` of the table `Person`. Then, we can see that there is a parent-child relationship, being the table `Person` the parent, and the table `Student` the child. This relationship implies that every property available in the parent is also inherited by the child, so that when we transform this database into RDF, the instances resulting from table `Student` will also have properties corresponding to the column `SSN` of the parent table `Person`.

[1] http://www.w3.org/2001/sw/rdb2rdf/test-cases/#D011-M2MRelations



Person

ID (PK)	SSN
10	1234510
11	1234511
12	1234512

Contact

CID (PK)	SID (FK)	Email
1	10	venus@hotmail.com
2	10	venus@gmail.com
3	11	fernando@yahoo.com
4	12	david@msn.com

Student

ID (PFK)	FirstName	LastName
10	Venus	Williams
11	Fernando	Alonso
12	David	Villa

Student_Sport

ID_Student (PFK)	ID_Sport (PFK)
10	110
11	111
11	112
12	111

Sport

ID (PK)	Description
110	Tennis
111	Football
112	Formula1

Fig. 1. Graphical Representation of D011B Database

– Add another table `Contact(CID, SID, Email)` with `CID` as the primary key and `SID` as the foreign key that refers to the column `ID` in the table `Student`. The relationship between `Student` and `Contact` is a 1-N relationship, so we may expect to have an object property generated for this relationship, being `Student` the domain and `Contact` the range.

Contributions. The main contribution of this paper is the design and implementation of an algorithm that takes as an input a relational database and generates as an output an R2RML mapping document that includes two groups of mappings. The first group of mappings encodes the transformations that would be done by a Direct Mapping engine, with the only exception that the generated RDF will differ in the URIs that are generated for some RDF nodes. The second group of mappings encodes an additional set of transformations that exploit the implicit information that is normally contained in relational databases (e.g. subclass-of relationships, M-N relationships) and which are not exposed by directly following the Direct Mapping approach.

The rest of the paper is structured as follows. In Section 2 we discuss pre-R2RML and R2RML-compliant mapping generation systems. In Section 3 we present the core of our approach for the automatic generation of R2RML mappings from a relational database schema. In Section 4 we present some experiments applied to the set of test cases provided by the W3C RDB2RDF working group. In Section 5 we provide some conclusions on this paper and our planned future work.

2 Background and State of The Art

We start this section by providing some background on the two W3C recommendations that we have already referred to in the introduction: Direct Mapping and R2RML. Then we move into the description of some of the approaches that have been proposed so far for the generation of RDB2RDF mappings.

2.1 Background: W3C Direct Mapping and R2RML

As discussed in the introduction, the W3C **Direct Mapping** recommendation defines simple transformation rules to generate RDF from relational data, as follows:

- A rule to generate the subject URI that corresponds to each row of a database table. These subject URIs are the result of the concatenation of a base URI with the name of the primary key column, the symbol `=` and the value of the column, e.g., `<Student/ID=10>`. Blank nodes are generated in the case of tables that do not have any primary key defined.
- A rule to generate rdf:type triples from each row of a database table, e.g., `<Student/ID=10> rdf:type <Student>`.
- A rule to generate literal triples from each row of a database table. This rule generates the subject URI as the subject, the concatenation of the table name, the symbol # and the column name as the predicate, and the column value as the object. An example of a triple generated by this rule is the following: `<Student/ID=10> <Student#FirstName> "Venus"`.
- A rule to generate reference triples from each row of a datable table that contains a foreign key. This rule generates the subject URI as the subject, the concatenation of the table name, the string #ref- and the column name as the predicate, and the concatenation of the referenced table name, the primary key of the referenced table and the column value as the object. For example, this rule would generate the following triple: `<Contact/ID=1> <Contact#ref-SID> <Student/ID=10>`.

Unlike the Direct Mapping recommendation, **R2RML** allows users to specify the transformation rules to be applied. The most important R2RML elements are:

- `rr:TriplesMap`, used to transform database rows into RDF triples.
- `rr:LogicalTable`, used to specify the table (or view) whose rows are to be transformed.
- `rr:TermMap`, used to represent the term generation rules for components of the triples (subject, predicate, and object). There are three ways to specify them: constant (`rr:constant`), column `rr:column`, or template (`rr:template`).
- `rr:SubjectMap`, used to specify the transformation rules to generate the subjects of the triples.
- `rr:PredicateObjectMap`, used to specify the transformation rules to generate the pair of predicate and object of the triples, by means of `rr:PredicateMap` and `rr:ObjectMap`, respectively.

2.2 RDB2RDF Mapping Generation Approaches

Several works in the state of the art have dealt with the automatic or manual generation of RDB2RDF mappings.

The first group of systems that we can refer to is that of early RDB2RDF systems (e.g. ODEMapster [3], D2R Server [4], Triplify [5])[2], which used their own

[2] A longer list is available at http://d2rq.org/resources#projects

mapping languages to transform relational database content to RDF. They also had associated functionalities to ease the generation of such mappings, either manually or automatically. For example, the ODEMapster GUI was a NeOn Toolkit plugin that allowed specifying in a graphical manner the most common types of mappings that may be declared in the R2O language. D2R Server provides an automatic mapping generation functionality based on table and column names, as well as constraints such as primary and foreign keys, what is very similar to the one provided for the W3C Direct Mapping. This implementation was first available for the D2R language and later for R2RML. Triplify provides mapping templates for some well-known Web applications (e.g. WordPress, Joomla!, Drupal).

There are also other approaches that are independent from the mapping language and tool used to specify and run mappings. For instance, the authors in [6,10] analyse the different types of relationships that exist between relational tables, using primary and foreign keys, so as to determine the classes and properties of the expected results in RDF. However, no mappings are generated as a result of this analysis, what means that this work cannot be reused by other RDB2RDF engines. More recent work [7] has proposed a fixed set of rules for saturating mappings, once that the mappings between the relational database and an ontology have been defined by a domain expert. In fact, the motivation example that we have described in our example in section 1 is inspired by that work.

In [8] the authors proposed a semi-automatic mapping generation process, where R2RML mappings are generated based on a set of semantic correspondences (for classes and properties) defined by domain experts. And the authors in [9] present a GUI-based R2RML mapping editor to help non-expert users to create and modify their mappings.

However, none of the aforementioned approaches and systems deals with the automatic generation, from relational databases, of R2RML mappings that encode the implicit information that can be obtained from the relational database schema.

3 Automatic Generation of R2RML Mappings

Our process for automatically generating R2RML mappings from a relational database schema is depicted in Figure 2. The system receives as an input either the connection details to an existing database or a SQL file containing the database schema (represented by SQL DDL/DML statements). Then the process consists of two main steps:

1. **Identification of Relationships between Tables**. In this step, the relationships between tables are extracted, as well as their cardinality and the columns and constraints that are present in the database schema.
2. **Generation of R2RML Mappings**. In this step, the R2RML triples maps that correspond to the patterns identified in the previous step are generated.

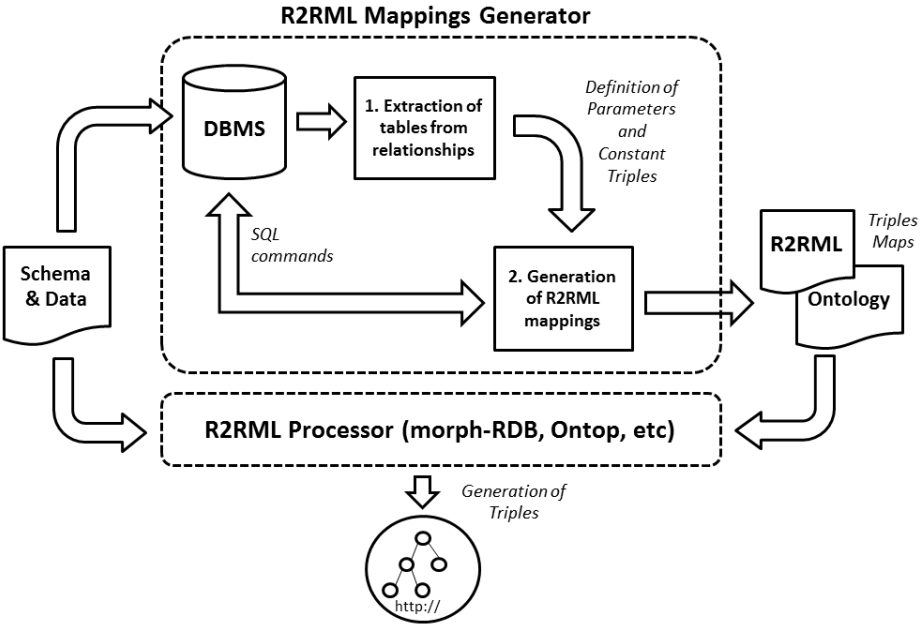


Fig. 2. A general overview of the R2RML mapping generation process in MIRROR

We consider two assumptions about the relational schema, so as to ensure completeness of the mapping process and preservation of information, as discussed in [10]:

1. The relational schema must be normalized, at least, in third normal form (3NF).
2. Primary keys must be defined as not null and unique.

Next we discuss about the typical patterns that can be found in a relational database schema and our approach to convert them into R2RML mappings.

3.1 A Catalogue of Typical Patterns in Relational Schemas

A typical database modeling process considers three types of models: **conceptual** (where the elements are described using an Entity-Relationship or an Extended Entity-Relationship diagram), **logical** (which uses the relational model and is independent of the target database management system) and **physical** (which depends on the underlying database management system, defines how data is stored and declares constraints and keys). While conceptual and logical models may in principle be the most adequate to understand the domain of a database, given their higher level of abstraction and technology independence, these models are not commonly available. Therefore, R2RML mapping generation algorithms need to be designed taking into account the information

that can be obtained from the physical model: relations (tables) and relationships between them.

We have created a catalogue (see tables 1 and 2) describing nine types of relationships between two (or more) tables that may be found in the physical implementation of a relational database. We name these tables "parent" and "child". A parent table is the one that contains a primary key that is used as a foreign key by the child table.

Catalogue Creation Process. Figure 3 outlines the steps followed to generate our catalogue. First we consider all the possible pairs (16) that describes a relationship between two entities at the conceptual model, from the perspective of each entity: `(Opt,1)`, `(Opt,N)`, `(Mand,1)` and `(Mand,N)`[3]. This list is then pruned as follows:

1. Using the reflexive property (i.e., $1 - N \equiv N - 1$), the options are pruned to 10.
2. By assumption 2 (primary keys must be defined as not null and unique), options 10, 11, 12 and 16 are not allowed, because of the parent optional feature `(Opt,X)`:
3. Finally, three special cases are added: i) **reciprocal relationships**, where one parent may be (optionally) related to only one child, and one child may be related to only one parent, respectively; ii) **self (or recursive) relationships**, where one instance of the parent may be related to another instance of the same parent; and iii) **n-ary relationships**, which involve many relationships, having many child tables connected to one parent table.

As a result, the catalogue is reduced to nine patterns, which need to be detected in the physical model of the database.

Catalogue Description. These nine patterns are graphically depicted in tables 1 and 2, which shows how they are normally specified in the conceptual (already discussed), logical and physical models. In the **logical model** we specify the number of relations (tables) involved, as well as the type of relationship between them. In the **physical model** we describe whether null values are accepted in the key column (primary or foreign) of each relation.

Now we describe each of the rows of tables 1 and 2:

1. Rows 1 and 2a get transformed into a single table in the relational model (and hence also in the physical model).
2. Row 2b may be considered as a special case of row 5a. A special case of an IS-A relationship using two tables may be also considered here.
3. Row 3 indicates a reciprocal relationship case between two tables and it matches with twice 1:N case not mandatory, as row 5a also states.

[3] Opt and Mand refer to whether the relationship is optional or mandatory, and the second item refers to the maximum cardinality, which may be 1 or uningspecified.

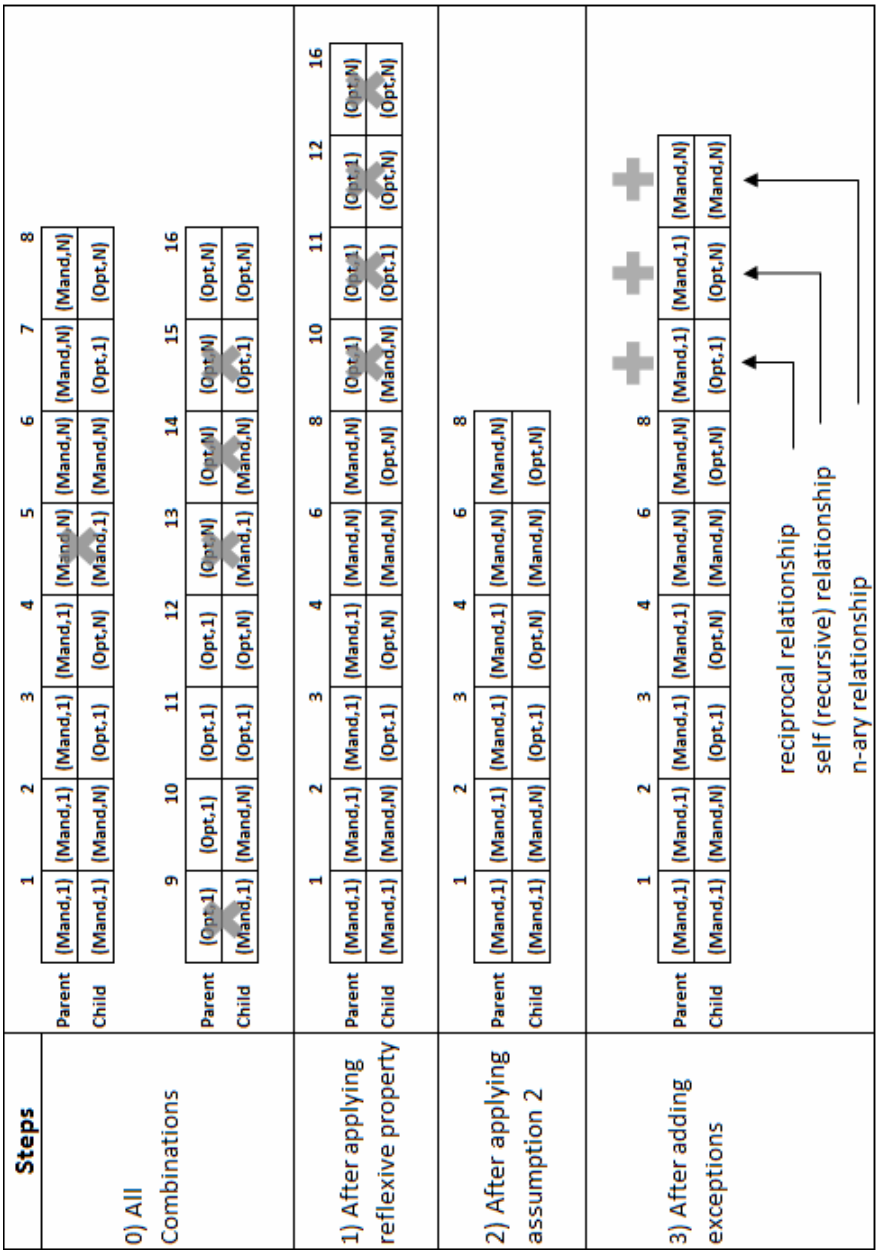
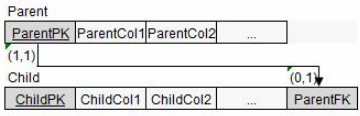
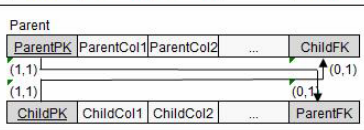
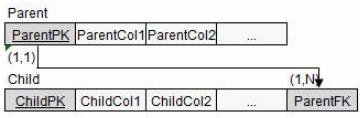
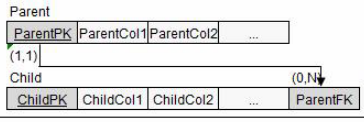
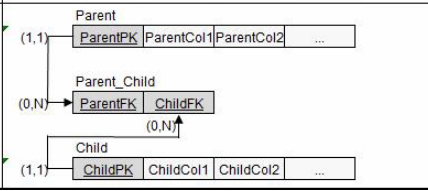


Fig. 3. Obtaining the Catalogue for Guiding the R2RML Mapping Generation

Table 1. Correspondences between conceptual, logical and physical models (rows 1 to 5)

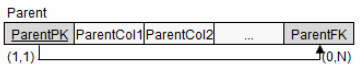
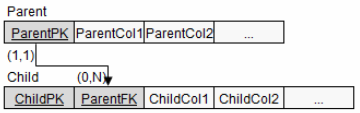
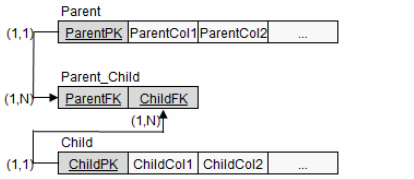
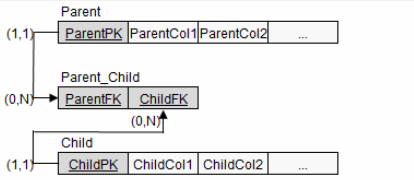
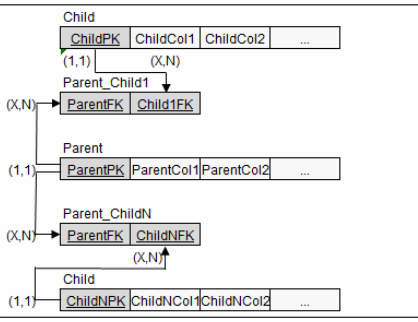
		Conceptual		Logical		Physical			Comments
						Nullable		Graphical Representation for a Generic Model	
		Parent	Child	Tables	Relationships	Parent	Child		
1		(Mand,1)	(Mand,1)	1	-	N	-		One table, no relationships
2	a	(Mand,1)	(Opt,1)	1	-	N	-		
	b			2	1-N	N	Y		Used for subclass relationships (e.g. *Person* and *Student*)
3		(Mand,1)	(Opt,1)	2	1-N	N	Y		Reciprocal relationship
4		(Mand,1)	(Mand,N)	2	1-N	N	N		(e.g. *Student* and *Contact*, mandatory case)
5	a	(Mand,1)	(Opt,N)	2	1-N	N	Y		(e.g. *Student* and *Contact*, optional case)
	b			3	M-N	N	Y		(e.g. *Student*, *Sport* and *Student_Sport*, if considering optional case)

4. Rows 7 and 8 can be considered as special cases using only two tables, settled in the row 9 as a more general form.
5. Row 6 describes a self-relationship.

The categories described next reflect the patterns of relationships that guide the SQL queries on the information schema that we will use in our algorithm, starting from, at least, two tables (patterns 1 and 2a, translated into only one table, are not considered):

1. **1:N**, **optional** entity (rows 2b, 3, 5a, 6a and 6b).
2. **1:N**, **mandatory** entity (row 4)
3. **M:N** having **2 tables**, optional or mandatory (rows 5b, 7 and 8)
4. **M:N** having **more than 2 tables**, optional or mandatory (row 9)

Table 2. Correspondences between conceptual, logical and physical models (rows 6 to 9)

		Conceptual		Logical		Physical			Comments
						Nullable		Graphical Representation for a Generic Model	
		Parent	Child	Tables	Relationships	Parent	Child		
6	a	(Mand,1)	(Opt,N)	1	1-N	N	Y		Self relationship (or recursive relationship)
	b			2	1-N	N	Y		
7		(Mand,N)	(Mand,N)	3	M-N	N	N		(e.g. tables *Student*, *Sport* and *Student_Sport*, mandatory case)
8		(Mand,N)	(Opt,N)	3	M-N	N	Y		Same as 5b
9		(Mand,N)	(Mand,N) or (Opt,N)	>=3	M-N	N	Y or N		N-ary relationships (i.e. many different combinations involving rows 7 and 8)

3.2 Algorithms for the Generation of R2RML Mappings

Two different algorithms are proposed for R2RML mapping generation: one for 1:N relationships (Algorithm 1) and another one for M:N relationships (Algorithm 2).

In Algorithm 1 (Cardinality 1-N) the outer loop goes through all primary keys from the parent table, and executes three procedures that produce R2RML mapping components:

– *triplesMap(n)*: it stores an ordered, auto-incremented triples maps, indexed by n, according to the template `<#TriplesMap{n}>`.

- *logicalTable(RS)*: it stores the mapping component `rr:logicalTable` for the parent table RS.
- *subjectMap(RS, KS)*: it stores the mapping component `rr:subjectMap`, taking in account the template `rr:template "http://IRI/RS/{KS}"`, where `IRI` is a parameter defined by the user.

The inner loop stores the mapping component `rr:predicateObjectMap`. It loops through all the columns that belong to the parent table (represented by argument $attr(RS)$). When the index n is incremented, the graph for the child table is generated, considering now:

- *triplesMap(n)*: it has the same behaviour as for the parent table.
- *logicalTable(RT)*: it stores the mapping component `rr:logicalTable` for the child table RT.
- *subjectMap(RT, KT)*: it stores the mapping component `rr:subjectMap`, taking in account the template `rr:template "http://IRI/RT/{KT}"`.

After another inner loop with respect to the `rr:predicateObjectMap` for the child table, the algorithm registers the relationship, by means of the mapping component `rr:joinCondition`, linking the primary key KS from the parent table with the foreign key KT from the child table.

Algorithm 2 (Cardinality M-N) is different, since we use one more loop on all rows obtained from evaluation $[\![\phi]\!]_I$ (categories 3 and 4).

Algorithm 1 1-N Cardinality

Require: $attr(\phi) = \{RS, KS, RT, KT\}$
1: **if** $card([\![\phi]\!]_I) = 1$ **then**
2: $n = 1$
3: **for all** KS **do**
4: triplesMap(n) ▷ Generates triples map for RS (parent table)
5: logicalTable(RS)
6: subjectMap(RS, KS)
7: **for all** $attr(RS)$ **do**
8: predicateObjectMap($attr(RS)$)
9: **end for**
10: $n \leftarrow n + 1$
11: triplesMap(n) ▷ Generates triples map for RT (child table)
12: logicalTable(RT)
13: subjectMap(RT, KT)
14: **for all** $attr(RT)$ **do**
15: predicateObjectMap($attr(RT)$)
16: **end for**
17: joinCondition(KS, KT) ▷ Generates join condition
18: $n \leftarrow n + 1$
19: **end for**
20: **end if**

Algorithm 2 M-N Cardinality

Require: $attr(\phi) = \{RS, KS, RT, KT\}$
if $card([\![\phi]\!]_I) > 1$ **then**
  for all tuples in ϕ **do**
    $n = 1$
    for all KS **do**
      triplesMap(n) ▷ Generates triples map for RS (parent table)
      logicalTable(RS)
      subjectMap(RS, KS)
      for all $attr(RS)$ **do**
        predicateObjectMap($attr(RS)$)
      end for
      $n \leftarrow n + 1$
      triplesMap(n) ▷ Generates triples map for RT (child table)
      logicalTable(RT)
      subjectMap(RT, KT)
      for all $attr(RT)$ **do**
        predicateObjectMap($attr(RT)$)
      end for
      joinCondition(KS, KT) ▷ Generates join condition
      $n \leftarrow n + 1$
    end for
  end for
end if

Subclass Identification. We use saturation to extend the set of R2RML mappings that has been initially created, exploiting subclass relationships that can be found in the database physical model. Unlike the work presented in [7], our work does not consider the use of an existing ontology to guide this saturation process. Our saturation approach considers two cases that can appear in the database physical model:

1. An IS-A relationship with cardinality 1-1 between a parent table and its child, having a common primary key table (row 2b from table 1).
2. An IS-A relationship with cardinality 1-N between a parent table and its child, becoming 1-1 after a data checking, testing whether any tuple in the parent table is related to only one tuple in the child table (it may happen with rows 4, 5a from 1, and 6a and 6b from table 2).

In these cases, the R2RML triple map for the child table is saturated with additional attributes from the parent table. An extra constant triple map is generated to feature explicitly the hierarchy, using `rdfs:subClassOf`.

Object Property Identification. Covering rows 5b from table 1; and 7, 8 and 9 from table 2, M-N relationships are represented by 3 tables. The binary table between parent and child tables, having the primary keys respectively

as foreign keys, can be understood as an object property. Our R2RML mapping generator can create constant triples maps, templated as object property `ParentHasChild` and putting also its inverse, `ChildBelongsToParent`, using `owl:ObjectProperty`, `owl:inverseOf`, `rdfs:domain` and `rdfs:range`.

Datatype Property Identification. In the wake of the object properties handling, all columns of tables in the database schema are featured as datatype properties. The mapping generator creates constants triples maps (using `owl:DatatypeProperty`) considering the table to which the column belongs as the domain (using `rdfs:domain`), and the column data type as the range (using `rdfs:range`).

4 Implementation and Experimentation

The algorithms described in this paper have been implemented in MIRROR[4] (MappIng from Relational to Rdf generatOR) and have been integrated with morph-RDB[5] [11]. We have performed some experiments in order to show that our system can obtain R2RML mappings that encode the semantics in the W3C Direct Mapping specification, and furthermore that our system can generate R2RML mappings that also lift-up the implicit semantics encoded in the database.

We use two datasets in this experimentation: the set of databases provided in the Direct Mapping Test Cases, and we extend one of the databases (D011) to encode a parent-child relationship.

4.1 Experimentation Using the Direct Mapping Test Cases

The Direct Mapping Test Cases[6] is a test suite provided by the W3C RDB2RDF Working Group, which consists of a collection of test cases covering various database schemes such as databases without tables, databases with one table, with 1-N relationships, and with M-N relationships. In each of the test cases, the triples that can be expected as a result of applying Direct Mapping rules are provided. Thus, this test suite is a suitable source for us to evaluate our system, enabling us to see if our system produces the expected Direct Mapping results. In addition to that, we can also easily see the additional triples generated by the saturated mappings resulting from the identification of pattern relationships described in Section 3.

Here we discuss the database of Test Case `D011`, which consists of three tables `Student`, `Sport`, and `Student_Sport`. The table `Student_Sport` acts as a binary table that enables the M-N relationship between table `Student` and table `Sport`. The result of applying Direct Mapping rules over the database D011 can be seen in Listing 1.1.

[4] https://github.com/oeg-upm/MIRROR
[5] https://github.com/oeg-upm/morph-rdb
[6] http://www.w3.org/2001/sw/rdb2rdf/test-cases/

Listing 1.1. The Result of Applying Direct Mapping Rules Over D011 Test Case Database

```
<Student/ID=10> rdf:type <Student> .
<Student/ID=10> <Student#FirstName> "Venus".
<Student/ID=10> <Student#ID> 10 .
<Student/ID=10> <Student#LastName> "Williams" .
<Student/ID=11> rdf:type <Student> .
<Student/ID=11> <Student#FirstName> "Fernando".
<Student/ID=11> <Student#ID> 11 .
<Student/ID=11> <Student#LastName> "Alonso" .
<Student/ID=12> rdf:type <Student> .
<Student/ID=12> <Student#FirstName> "David".
<Student/ID=12> <Student#ID> 12 .
<Student/ID=12> <Student#LastName> "Villa" .
<Student_Sport/ID_Student=10;ID_Sport=110> rdf:type <Student_Sport> .
<Student_Sport/ID_Student=10;ID_Sport=110> <Student_Sport#ID_Student> 10 .
<Student_Sport/ID_Student=10;ID_Sport=110> <Student_Sport#ref−ID_Student> <Student/ID=10> .
<Student_Sport/ID_Student=10;ID_Sport=110> <Student_Sport#ID_Sport> 110 .
<Student_Sport/ID_Student=10;ID_Sport=110> <Student_Sport#ref−ID_Sport> <Sport/ID=110> .
<Student_Sport/ID_Student=11;ID_Sport=111> rdf:type <Student_Sport> .
<Student_Sport/ID_Student=11;ID_Sport=111> <Student_Sport#ID_Student> 11 .
<Student_Sport/ID_Student=11;ID_Sport=111> <Student_Sport#ref−ID_Student> <Student/ID=11> .
<Student_Sport/ID_Student=11;ID_Sport=111> <Student_Sport#ID_Sport> 111 .
<Student_Sport/ID_Student=11;ID_Sport=111> <Student_Sport#ref−ID_Sport> <Sport/ID=111> .
<Student_Sport/ID_Student=11;ID_Sport=112> rdf:type <Student_Sport> .
<Student_Sport/ID_Student=11;ID_Sport=112> <Student_Sport#ID_Student> 11 .
<Student_Sport/ID_Student=11;ID_Sport=112> <Student_Sport#ref−ID_Student> <Student/ID=11> .
<Student_Sport/ID_Student=11;ID_Sport=112> <Student_Sport#ID_Sport> 112 .
<Student_Sport/ID_Student=11;ID_Sport=112> <Student_Sport#ref−ID_Sport> <Sport/ID=112> .
<Student_Sport/ID_Student=12;ID_Sport=111> rdf:type <Student_Sport> .
<Student_Sport/ID_Student=12;ID_Sport=111> <Student_Sport#ID_Student> 12 .
<Student_Sport/ID_Student=12;ID_Sport=111> <Student_Sport#ref−ID_Student> <Student/ID=12> .
<Student_Sport/ID_Student=12;ID_Sport=111> <Student_Sport#ID_Sport> 111 .
<Student_Sport/ID_Student=12;ID_Sport=111> <Student_Sport#ref−ID_Sport> <Sport/ID=111> .
<Sport/ID=110> rdf:type <Sport> .
<Sport/ID=110> <Sport#ID> 110 .
<Sport/ID=110> <Sport#Description> "Tennis" .
<Sport/ID=111> rdf:type <Sport> .
<Sport/ID=111> <Sport#ID> 111 .
<Sport/ID=111> <Sport#Description> "Football" .
<Sport/ID=112> rdf:type <Sport> .
<Sport/ID=112> <Sport#ID> 112 .
<Sport/ID=112> <Sport#Description> "Formula1" .
```

Listing 1.2. The Generated R2RML Mappings Correspond to Direct Mapping Triples

```
<#TriplesMap5> a rr:TriplesMap;
    rr:logicalTable [ rr:tableName "student"; ];
    rr:subjectMap [ rr:class <Student>; rr:template "Student/{ID}"; rr:termType rr:IRI; ];
    rr:predicateObjectMap [ rr:predicate <Student#ID>; rr:objectMap [ rr:column "ID"; rr:datatype xsd:integer ; ];];
    rr:predicateObjectMap [ rr:predicate <Student#FirstName>;rr:objectMap [ rr:column "FirstName"; rr:datatype xsd:string ; ];];
    rr:predicateObjectMap [ rr:predicate <Student#LastName>; rr:objectMap [ rr:column "LastName"; rr:datatype xsd:string ; ]; ].

<#TriplesMap10> a rr:TriplesMap;
    rr:logicalTable [ rr:tableName "student_sport"; ];
    rr:subjectMap [ rr:class <Student_Sport>; rr:template "Student_Sport/{ID_Student}/{ID_Sport}"; rr:termType rr:IRI;];
    rr:predicateObjectMap[rr:predicate <Student_Sport#ID_Student>;rr:objectMap[rr:column "ID_Student";rr:datatype xsd:integer ;];];
    rr:predicateObjectMap[rr:predicate <Student_Sport#ID_Sport>;rr:objectMap[rr:column "ID_Sport";rr:datatype xsd:integer ;];];
    rr:predicateObjectMap [rr:predicate <Student_Sport#ref−ID_Student>;rr:objectMap [
        rr:parentTriplesMap <#TriplesMap5>;rr:joinCondition [ rr:child "ID_Student"; rr:parent "ID"; ];];];
    rr:predicateObjectMap [ rr:predicate <Student_Sport#ref−ID_Sport>;rr:objectMap [
        rr:parentTriplesMap <#TriplesMap1>; rr:joinCondition [ rr:child "ID_Sport"; rr:parent "ID"; ];];].

<#TriplesMap1> a rr:TriplesMap;
    rr:logicalTable [ rr:tableName "Sport"; ];
    rr:subjectMap [ rr:class <Sport>;rr:template "Sport/{ID}";rr:termType rr:IRI;];
    rr:predicateObjectMap [rr:predicate <Sport#ID>;rr:objectMap [ rr:column "ID"; rr:datatype xsd:integer ; ]; ];
    rr:predicateObjectMap [rr:predicate <Sport#Description>;rr:objectMap [ rr:column "Description"; rr:datatype xsd:string ; ];].
```

When the same database schema and instance are passed to MIRROR, it generates the R2RML mappings shown in Listing 1.2.

Listing 1.3. Direct Mapping triples of morph-RDB Result Over D011 Test Case Database

```
<Student/10> rdf:type <Student> .
<Student/10> <Student#FirstName> "Venus"^^xsd:string .
<Student/10> <Student#ID> "10"^^xsd:integer .
<Student/10> <Student#LastName> "Williams"^^xsd:string .
<Student/11> rdf:type <http ://example.com/Student .
<Student/11> <Student#FirstName> "Fernando"^^xsd:string .
<Student/11> <Student#ID> "11"^^xsd:integer .
<Student/11> <Student#LastName> "Alonso"^^xsd:string .
<Student/12> rdf:type <http ://example.com/Student> .
<Student/12> <Student#FirstName> "David"^^xsd:string .
<Student/12> <Student#ID> "12"^^xsd:integer .
<Student/12> <Student#LastName> "Villa"^^xsd:string .
<Student_Sport/10/110> rdf:type <Student_Sport> .
<Student_Sport/10/110> <Student_Sport#ID_Student> "10"^^xsd:integer> .
<Student_Sport/10/110> <Student_Sport#ref-ID_Student> <Student/10> .
<Student_Sport/10/110> <Student_Sport#ID_Sport> "110"^^xsd:integer> .
<Student_Sport/10/110> <Student_Sport#ref-ID_Sport> <Sport/110> .
<Student_Sport/11/111> rdf:type <Student_Sport> .
<Student_Sport/11/111> <Student_Sport#ID_Student> "11"^^xsd:integer> .
<Student_Sport/11/111> <Student_Sport#ref-ID_Student> <Student/11> .
<Student_Sport/11/111> <Student_Sport#ID_Sport> "111"^^xsd:integer> .
<Student_Sport/11/111> <Student_Sport#ref-ID_Sport> <Sport/111> .
<Student_Sport/11/112> rdf:type <Student_Sport> .
<Student_Sport/11/112> <Student_Sport#ID_Student> "11"^^xsd:integer> .
<Student_Sport/11/112> <Student_Sport#ref-ID_Student> <Student/11> .
<Student_Sport/11/112> <Student_Sport#ID_Sport> "112"^^xsd:integer> .
<Student_Sport/11/112> <Student_Sport#ref-ID_Sport> <Sport/112> .
<Student_Sport/12/111> rdf:type <Student_Sport> .
<Student_Sport/12/111> <Student_Sport#ID_Student> "12"^^xsd:integer> .
<Student_Sport/12/111> <Student_Sport#ref-ID_Student> <Student/12> .
<Student_Sport/12/111> <Student_Sport#ID_Sport> "111"^^xsd:integer> .
<Student_Sport/12/111> <Student_Sport#ref-ID_Sport> <Sport/111> .
<Sport/110> rdf:type <Sport> .
<Sport/110> <Sport#ID> "110"^^xsd:integer> .
<Sport/110> <Sport#Description> "Tennis"^^xsd: string> .
<Sport/111> rdf:type <Sport> .
<Sport/111> <Sport#ID> "111"^^xsd:integer> .
<Sport/111> <Sport#Description> "Football"^^xsd: string> .
<Sport/112> rdf:type <Sport> .
<Sport/112> <Sport#ID> "112"^^xsd:integer> .
<Sport/112> <Sport#Description> "Formula1"^^xsd:string> .
```

Upon receiving these mappings, morph-RDB (or any other R2RML processor) generates a set of triples (see Listing 1.3) that correspond to the ones generated by Direct Mapping, hence we call them Direct Mapping triples.

Listing 1.4. R2RML Mappings that Generate the Extra Triples

```
<#TriplesMap15> a rr:TriplesMap;
  rr:logicalTable [ rr:sqlQuery
    "SELECT DISTINCT t_39025.ID_Student AS ID_Student, t_39025.ID_Sport AS ID_Sport
      FROM (sport AS t_01724 JOIN student_sport AS t_39025 ON ((t_01724.ID=t_39025.ID_Sport)))
      JOIN student AS t_83317 ON ((t_83317.ID=t_39025.ID_Student))" ];
  rr:subjectMap [ rr:termType rr:IRI; rr:template "Sport/{ID_Sport}"; ];
  rr:predicateObjectMap [ rr:predicate <SportHasStudent>;
  rr:objectMap [ rr:template "Student/{ID_Student}" ]; ].
```

In addition to the mappings above, additional mappings are also generated, as shown in Listing 1.4. These mappings produce the triples that can be seen

in Listing 1.5, which specify the relationship between Student and Sport, which are not generated by the Direct Mapping specification.

Listing 1.5. Extra triples of morph-RDB Result Over D011 Test Case Database

```
<Sport/110> <SportHasStudent> <Student/10> .
<Sport/111> <SportHasStudent> <Student/11> .
<Sport/111> <SportHasStudent> <Student/12> .
<Sport/112> <SportHasStudent> <Student/11> .
```

4.2 Experimentation Using `D011B`

For `D011B`, MIRROR generates the additional mappings shown in Listing 1.6:

- R2RML mappings that generate the triples that the instances of the class `Student` are also instances of the class `Person` (line 20-23) and that `SSN` is a property of `Student` because `Student` inherits properties of `Person` (line 24-27).
- R2RML mappings that generate relationships between `Student` and `Contact` (line 7-10).

Listing 1.6. R2RML Mappings Correspond to Subclass Generation and the Inherited `SSN` property in class `Student`

```
<#TriplesMap12> a rr:TriplesMap;
    rr: logicalTable [ rr:sqlQuery
        "SELECT DISTINCT t_37839.ID, t_11900.CID
        FROM (student AS t_37839 JOIN contact AS t_11900 ON ((t_37839.ID=t_11900.SID)))" ];
    rr:subjectMap [ rr:termType rr:IRI;
        rr:template "http :// example.com/Student/{ID}"; ];
    rr:predicateObjectMap [
        rr:predicate ex:StudentHasContact;
        rr:objectMap [ rr:template "http :// example.com/Contact/{CID}" ]; ];
    ];
].
<#TriplesMap18> a rr:TriplesMap;
    rr: logicalTable [ rr:sqlQuery
        "SELECT t_76159.ID, t_76159.FirstName, t_76159.LastName, t_40951.SSN
        FROM person AS t_40951 JOIN student AS t_76159 ON (t_40951.ID=t_76159.ID)"
    ];
    rr:subjectMap [ rr: class ex:Student; rr:termType rr:IRI;
        rr:template "http :// example.com/student/{ID}";
    ];
    rr:predicateObjectMap[
       rr:predicate rdf:type;
       rr:objectMap [ rr:constant ex:Person ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:Student#ssn;
        rr:objectMap [ rr:datatype xsd: string ; rr:column "SSN";]
    ];
     ...
].
```

5 Conclusion

We have presented a tool for the automatic generation of R2RML mappings from a relational database schema. These mappings can be used by any R2RML processor to generate a set of RDF triples that is similar to those resulting from Direct Mapping. Several types of relationships between tables in a physical model have been categorised. By means of a core query and two algorithms that extract and organize this information, mappings are generated.

MIRROR has been integrated with morph-RDB, what allows experimenting with the generated R2RML mappings. The process was tested using the DM test cases suite, with test D011 extended to D011B to cover more relationships. The additional mappings and extra triples resulting from subclasses, object properties and datatype properties have been also described.

In future works, we will cover other types of database systems, so as to make our work inline with xR2RML [12] and RML [13].

Acknowledgments. This research has been funded by Ministerio de Economía y Competitividad (Spain) under the project "4V: Volumen, Velocidad, Variedad y Validez en la Gestión Innovadora de Datos" (TIN2013-46238-C4-2-R). Luciano Frontino de Medeiros was supported by Fundación Carolina-Spain.

References

1. Arenas, M., Bertails, A., Prud, E., Sequeda, J., et al.: A direct mapping of relational data to RDF, W3C recommendation 27 september 2012 (2013)
2. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C recommendation, 27 september 2012 (2013)
3. Barrasa Rodríguez, J., Corcho, Ó., Gómez-Pérez, A.: R2O, an extensible and semantically based database-to-ontology mapping language (2004)
4. Bizer, C., Cyganiak, R.: D2R server-publishing relational databases on the semantic web. In: Poster at the 5th International Semantic Web Conference (2006)
5. Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., Aumueller, D.: Triplify: light-weight linked data publication from relational databases. In: Proceedings of the 18th international conference on World wide web, pp. 621–630. ACM (2009)
6. Sequeda, J.F., Tirmizi, S.H., Corcho, O., Miranker, D.P.: Survey of directly mapping SQL databases to the semantic web. Knowledge Engineering Review **26**, 445–486 (2011)
7. Sequeda, J.F., Arenas, M., Miranker, D.P.: OBDA: query rewriting or materialization? in practice, both!. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 535–551. Springer, Heidelberg (2014)
8. Pequeno, V.M., Vidal, V.M., Casanova, M.A., Neto, L.E.T., Galhardas, H.: Specifying complex correspondences between relational schemas and RDF models for generating customized R2RML mappings. In: Proceedings of the 18th International Database Engineering & Applications Symposium, pp. 96–104. ACM (2014)

9. Sengupta, K., Haase, P., Schmidt, M., Hitzler, P.: Editing R2RML mappings made easy. In: International Semantic Web Conference (Posters & Demos), pp. 101–104 (2013)
10. Sequeda, J.F., Arenas, M., Miranker, D.P.: On directly mapping relational databases to RDF and OWL. In: Proceedings of the 21st international conference on World Wide Web, pp. 649–658. ACM (2012)
11. Priyatna, F., Corcho, O., Sequeda, J.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In: Proceedings of the 23rd international conference on World wide web, International World Wide Web Conferences Steering Committee, pp. 479–490 (2014)
12. Michel, F., Djimenou, L., Faron-Zucker, C., Montagnat, J.: xR2RML: Non-relational databases to RDF mapping. Technical report (2015)
13. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: a generic language for integrated RDF mappings of heterogeneous data. In: Proceedings of the 7th Workshop on Linked Data on the Web (LDOW2014), Seoul, Korea (2014)