Baltic J. Modern Computing, Vol. 7 (2019), No. 4, 475-486
https://doi.org/10.22364/bjmc.2019.7.4.02

RDB2OWL: A Language for Database to OWL Mapping and its Implementation

Guntars BŪMANS[1], Kārlis ČERĀNS[2]

[1]Faculty of Science and Engineering, Liepaja University,
Kr. Valdemāra iela 4, Liepaja LV-3401, Latvia,
[2]Institute of Mathematics and Computer Science, University of Latvia,
Raina blvd. 29, Riga, LV-1459, Latvia

Abstract: Most data in industry still resides in relational databases (RDB) but Semantic web uses standard RDF and OWL formats. We consider a previously developed mapping language RDB2OWL allowing compact and end-user readable mapping specification. We describe the RDB2OWL implementation by translation into R2RML standard language that avoids the view rowset multiplication due to several 1-to-n relations from a single class in the ontology.

Keywords: Database to ontology mapping, ontologies, RDF, mapping patterns

1. Introduction

Most data in industry still resides in relational databases (RDB) but Semantic web uses standard RDF (WEB, a) and OWL (Motik et al., 2012) formats. Therefore, exposing the contents of RDB to RDF and OWL formats enables the integration of the RDB contents into the Linked Data (Speicher, 2014) and Semantic web (Berners-Lee et al., 2001) information landscape. An important benefit of RDB to-RDF/OWL mapping is also the possibility of creating a conceptual model of the RDB data on the RDF Schema/OWL level and further on accessing the RDB contents from the created semantic/conceptual model perspective.

The task of mapping relational databases to RDF/OWL formats is well understood, widely studied and technically implemented, for example in D2RQ (WEB, b), Virtuoso RDF Graphs (Blakeley, 2007), Ultrawrap (Sequeda et al., 2009), Spyder (WEB, c) and W3C standard R2RML (Das et al., 2012) among different RDB-to-RDF/OWL mapping languages and tools. Most of the RDB-to-RDF/OWL mapping approaches offer languages for conceptually clear mapping structure with less attention paid, however, to the concise mapping writing.

In earlier works (Čerāns and Būmans, 2011), (Būmans and Čerāns, 2011), (Būmans and Čerāns, 2016) on RDB2OWL language the authors described a possibility of reusing both the target ontology and source database schemas in the mapping specifications. RDB2OWL mapping information is written in a compact textual form into the annotations of the target ontology entities. The RDB2OWL mapping processing is able to use implicit information about e.g. subclass relations, property domain and range classes, database tables, columns, foreign and primary keys.

RDB2OWL language has a possibility to define and call user defined functions and specify meta-level features such as multi-class conceptualization to avoid lengthy SQL filter expressions if many OWL data properties have a common domain class but maps to linked DB table. So, compact and end-user-readable mapping specifications are obtained.

The currently available RDB2OWL mapping implementation is by translating the mappings into D2RQ to be executed by D2R server (WEB, b)), or into the W3C standard R2RML language (Das et al., 2012), supported by several tools, including ontop (Calvanese et al., 2015) and R2RML Parser (WEB, d). The tools supporting the RDB2OWL mapping translation target notations D2RQ or R2RML allow to create a SPARQL endpoint to query the data from the relational database directly, or produce the RDF dump of the source RDB.

The RDB2OWL mapping translation process also has some inferencing facilities such as subclass, subproperty and inverse property inference.

To enable the practical applicability of the generated R2RML mappings, their efficiency has to be considered, what is done originally for the first time in this paper.

In what follows, Section 2 introduces RDB2OWL language syntax. Section 3 describes the implementation, including the mapping optimization concerns. Section 4 concludes the paper.

2. RDB2OWL mapping language review

RDB2OWL mappings describe correspondence between elements of OWL ontology/RDF schema and relational DB schema. The RDB2OWL mapping elements describe the connections to the database for ontology/RDF schema classes (class maps), object properties (object property maps) and data properties (data property maps), ascribed by annotation assertions to the respective ontology/schema entities.

```
classMap:  (VarName '=')? tableExpr uriPattern?;
tableExpr:  tableRefList (';' tableFilterExpr )?;
uriPattern:  '{uri=(' valueExpr (',' valueExpr)* ')}';
tableRefList:  tableRefItem (',' tableRefItem)*;
tableRefItem: (tableNavigItem tableRefItemLink?
            | tableRefItemLink);
tableNavigItem:  tableExprPlain (classMapRef)?;
tableRefItemLink : linkExpr (tableNavigItem tableRefItemLink?)?;
linkExpr: ('[' valueList ']')? ('->'|'=>') ('[' valueList ']')?;
tableExprPlain:  simpleTableExpr | '(' tableExpr ')';
simpleTableExpr:  (classMapRef | namedRef | tableUseExpr) alias?;
classMapRef:  '<s>' | '<t>';
namedRef:  '[[' VarName ']]';
tableUseExpr:  (dbAlias '::')? VarName;
objectMap:  tableExpr;
dataMap: ('[' tableExpr '].')? valueExpr ('^^' xsdRef)?;
```

Figure 1. RDB2OWL EBNF grammar fragment

A RDB2OWL class map has a table expression, possibly involving a join of several tables and a filter, and a pattern for instance resource URI (uriPattern) construction from a DB table expression. A class map attached to an ontology class describes instances for this class. The EBNF grammar code in Figure 1 shows some essential parts of RDB2OWL mapping language grammar for class map, object map and data map expressions.

Figure 2 illustrates the parse tree for tableExpr expression "*Course c, Teacher t;c.Teacher_ID=t.Teacher_ID*" The expression contains a table list followed by a filter expression after semicolon.

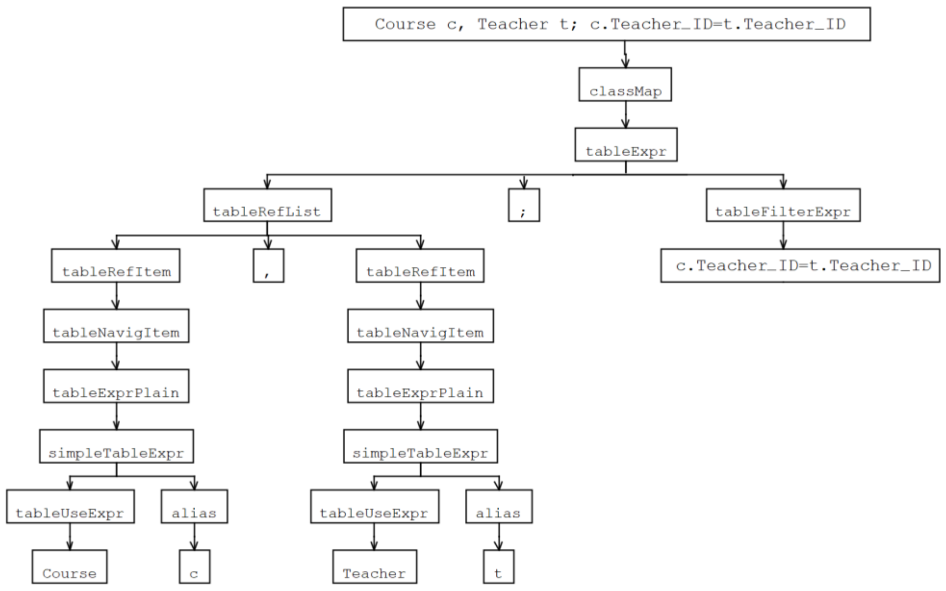

Figure 2. The parse tree for tableExpr "Course c, Teacher t; c.Teacher_ID=t.Teacher_ID"

The same table expression can be written with table navigation list containing just one navigation link "*Teacher[Teacher_ID]->[Teacher_ID] Course*" the parse tree of which is shown in Figure 3.

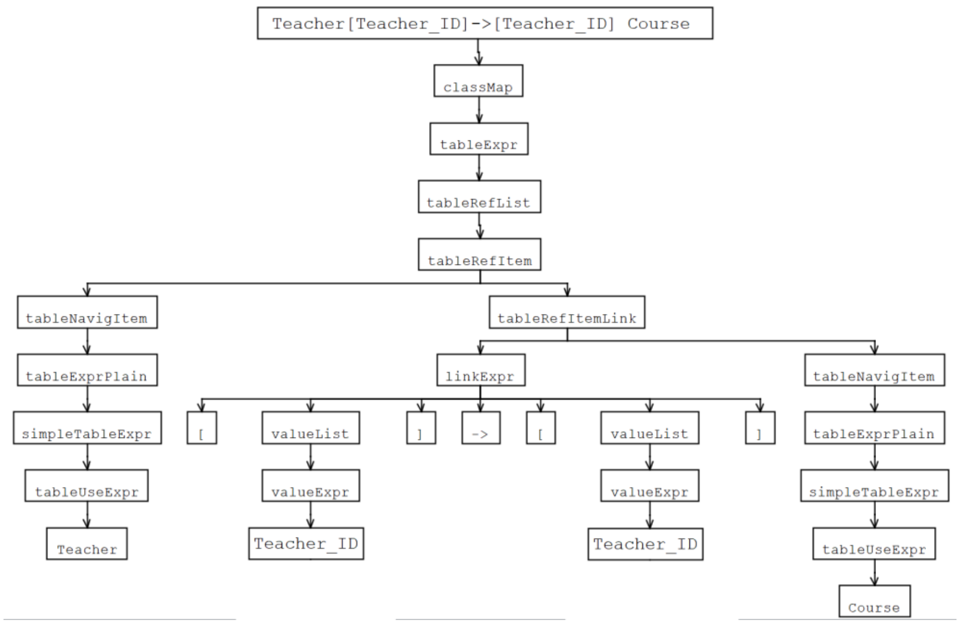

Figure 3. The parse tree for tableExpr "Teacher[Teacher_ID]->[Teacher_ID] Course"

An object property map (*objectMap*) refers to its source and target class maps- that are attached to the resp. domain and range classes of the property, so reusing URI generation patterns. An object property map has a table expression to specify how to link tables attached to the domain and range classes. Similarly, a data property map (*dataMap*) reuses URI generation pattern from domain class map. A data property map can have additional table expression that can add additional linked tables to the tables mapped to the domain class of the property. A data property map also has a value expression describing the computation of the property literal value.

We illustrate the basic RDB2OWL mapping constructs on a simple mini-University ontology and mapping example created in OWLGrEd[1] ontology editor, shown in Figure 4 where mappings are written as annotations in form *DB("<mapping expression>")*. The corresponding RDB schema is shown in Figure 5. The ontology, besides the intuitive and "common sense" properties for the classes, contains also multi-valued properties *hasMark* and *dateCompleted* for the *Student* class, linking each student directly to all marks and completion dates for all courses taken by the student.

On a more detailed level, a class map syntax consists of table expression and optional URI pattern specification in the form of list of comma separated expressions to be evaluated and concatenated, e.g. "*Student s {uri=('Student', s.IDCode)}*". Here "*Student"* is <*tableUseExpr*> as subexpression of <*simpleTableExpr*> in <*tableExprPlain*> and *"s"* is <*alias*> followed by <*uriPattrn*> expression. If URI pattern is omitted, the

[1] The ontology editor can be downloaded from `http://owlgred.lumii.lv/`

default one is formed from table expression's leftmost table name followed by its primary key column(s) value. A class map's table expression can refer also to a defined class map either by its explicit name assigned to variable, or by the name of the class for which it is the sole class map by using <*namedRef*> expression (e.g. *[[Teacher]]*), as in the mappings for the *personID* property in Figure 4.

A RDB2OWL table expression in simple cases is just a table name (*tableUseExpr*), e.g. "*Teacher".* A filter expression (<*tableFilterExpr*>) can be added to a table expression, after a semicolon, e.g. "*Course;Required=1*". The table expressions (<*tableExpr*>) can introduce additional tables in the following ways:

a. item list notation with comma-separated, optionally alias-labelled table expressions, for example,
"*Teacher T, Course C; T.Teacher_ID=C.Teacher_ID"*

b. navigation list notation (<*tableRefItem*> with <*linkExpr*>), such as *"Teacher[Teacher_ID]->[Teacher_ID] Course"* which can be shortened to "*Teacher->Course"* by omitting navigation columns that are target table's PK and their only matching source table's FK to the target table, that is, FK-to-PK link. The mark '=>' is used for reverse order, i.e. PK-to-FK link

c. notation putting (a) or (b) in brackets, regarding whole navigation list as a single item in the item list thus forming nested table expressions "*(Teacher->Course) tc, Registration r; tc.Course_Id=r. Course_Id"*. In grammar to this case corresponds <*tableExprPlain*> expression that contains another <*tableExpr*> in brackets.

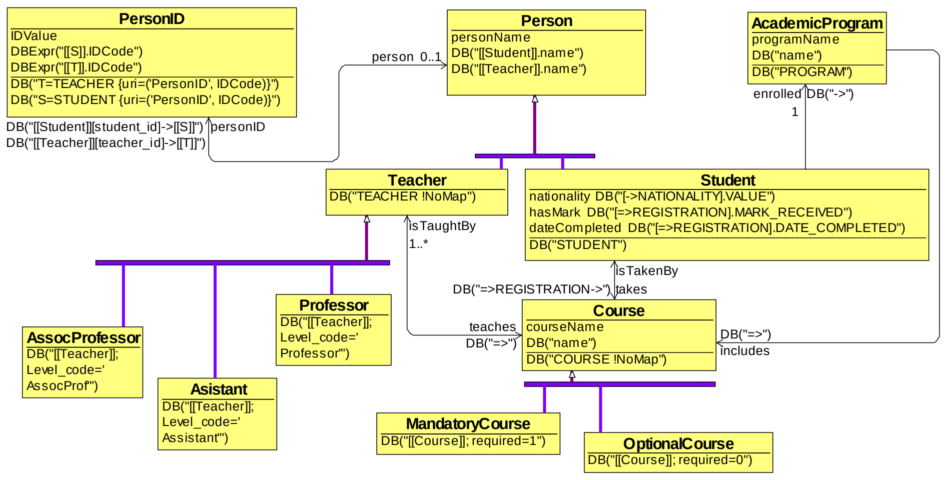
PersonID
IDValue
DBExpr("[[S]].IDCode")
DBExpr("[[T]].IDCode")
DB("T=TEACHER {uri=('PersonID', IDCode)}")
DB("S=STUDENT {uri=('PersonID', IDCode)}")
DB("[[Student]][student_id]->[[S]]")
DB("[[Teacher]][teacher_id]->[[T]]")
personID
person 0..1
Person
personName
DB("[[Student]].name")
DB("[[Teacher]].name")
AcademicProgram
programName
DB("name")
DB("PROGRAM")
enrolled
DB("->")
1
Teacher
DB("TEACHER !NoMap")
Student
nationality DB("[->NATIONALITY].VALUE")
hasMark DB("[=>REGISTRATION].MARK_RECEIVED")
dateCompleted DB("[=>REGISTRATION].DATE_COMPLETED")
DB("STUDENT")
isTaughtBy
1..*
isTakenBy
DB("=>REGISTRATION->")
takes
Professor
DB("[[Teacher]]; Level_code=' Professor'")
AssocProfessor
DB("[[Teacher]]; Level_code=' AssocProf'")
Asistant
DB("[[Teacher]]; Level_code=' Assistant'")
teaches
DB("=>")
Course
courseName
DB("name")
DB("COURSE !NoMap")
DB("=>")
includes
MandatoryCourse
DB("[[Course]]; required=1")
OptionalCourse
DB("[[Course]]; required=0")

Figure 4. Mini-university ontology with RDB2OWL mapping annotations with DB(…) notion

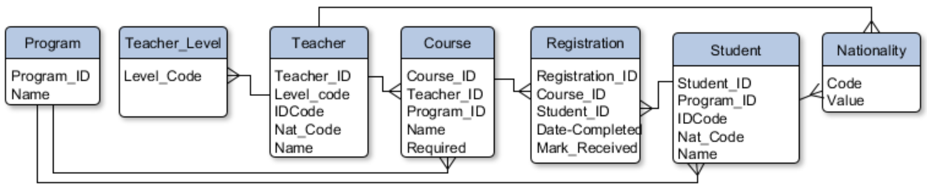

Figure 5. Mini-university RDB schema

The navigation links can chain many tables and introduce local filters attached to the individual navigation link element: "*Student:(name='Bob')=>Registration -> Course*"

An *object property map* is a table expression that has two subexpressions to denote its subject and object class maps respectively. Each of these subexpressions can be:

a. explicitly marked by an alias <s> or <t>
b. followed by the mark <s> or <t>; in this case the sole class maps defined explicitly for the property domain or range class are considered the subject and object class maps for the property map
c. if explicit <s> and/or <t> marks are not specified these marks are assumed <s> for the leftmost and <t> for rightmost item within the table expression.

These conventions on object property map and table expression syntax allow writing object property map in a concise way "->", if the property corresponds to the sole FK-to-PK mapping between the tables mapped. The concise form of "=>" is for sole PK-to-FK link. For example, long form of expression for *takes* property is "*Student[Student_ID]->[Student_ID]Registration[Course_ID]->[Course_ID]Course*" but the short form is just "*=>Registration->*"

A data property map is described as column name or column expression that is to be evaluated in the table context mapped to the sole class map of the property's domain class.

There are also more advanced mapping specification options available in RDB2OWL language, including user defined and RDB2OWL functions, multiclass conceptualization, auxiliary database objects please see (Būmans and Čerāns, 2011) for their detailed description.

3. RDB2OWL language implementation

The RDB2OWL mapping tool[2] reads an annotated OWL ontology, makes a connection to the source database, reads the database schema information. The database connection information can be defined as an annotation within the data ontology, or it can be supplied as a parameter within the RDB2OWL tool configuration.

The RDB2OWL mapping processing and translation to D2RQ/R2RML is done in the steps shown in Figure 6.

[2] http://rdb2owl.lumii.lv/

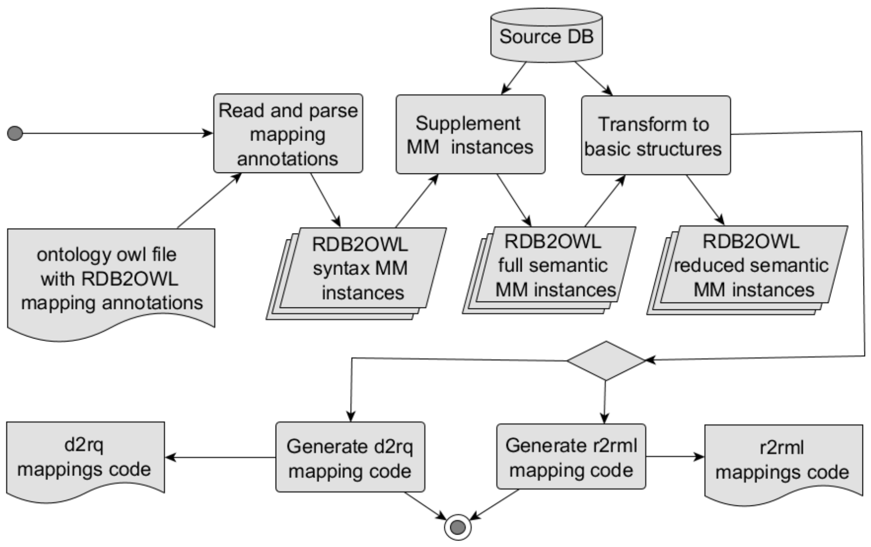
Source DB
Read and parse mapping annotations
Supplement MM instances
Transform to basic structures
ontology owl file with RDB2OWL mapping annotations
RDB2OWL syntax MM instances
RDB2OWL full semantic MM instances
RDB2OWL reduced semantic MM instances
d2rq mappings code
Generate d2rq mapping code
Generate r2rml mapping code
r2rml mappings code

Figure 6. RDB2OWL tool implementation activity diagram

The steps details are:

- **Read and parse mapping annotations**: The ontology with the attached RDB2OWL annotations is loaded, each annotation is parsed and stored in the internal RDB2OWL mapping model (Čerāns and Būmans, 2011). In this step only mere syntax information is stored.
- **Supplement MM instances:** Load source database schema information into the model.
- **Transform to basic structures:** Finalize abstract mapping: the advanced mapping constructs (e.g. named class maps, shorthands (e.g. omitted navigation columns), defaults (e.g., URI patterns) and user defined functions) are resolved into basic mapping constructions bringing the mapping into the "reduced semantic" RDB2OWL metamodel, outlined here in Figure 7; this model is used further on as the basis for D2RQ/R2RML mapping code generation,
- **Generate D2RQ/R2RML mapping code**: generation based on the created reduced semantic RDB2OWL model.

The RDB2OWL mapping translation processes all *ClassMap*, *ObjectPropertyMap* and *DataPropertyMap* objects that are stored in the internal RDB2OWL mapping model. The manipulation of model data and translation into D2RQ/R2RML is done by using lQuery tool (Liepiņš, 2011) and LuA scripting language. More on translation process to D2RQ and R2RML see (Būmans and Čerāns, 2016).

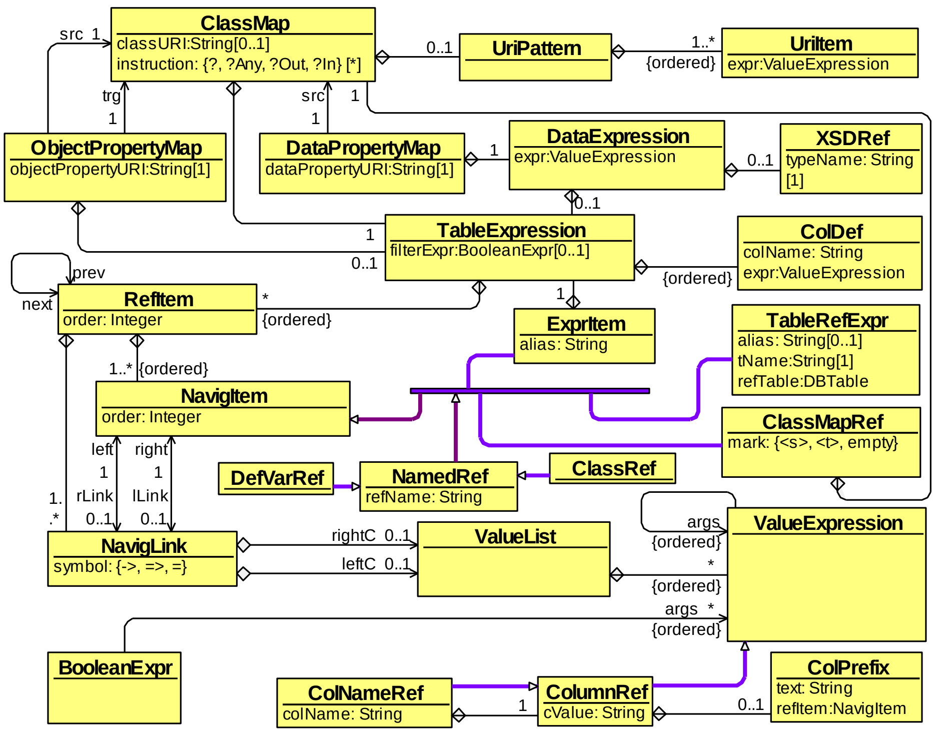

Figure 7. Essential fragments of RDB2OWL metamodel (cf. Būmans and Čerāns, 2016)

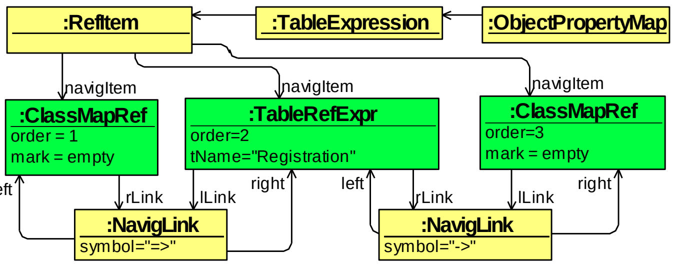

Figure 8. "=>Registration: ->" expression in RDB2OWL syntax metamodel

As an example, we demonstrate the translation of the "=>*Registration* ->" mapping expression for the *takes* property in the Mini-university example. After parsing this expression is stored in internal RDB2OWL syntax model, see **Error! Reference source not found.** 8.

During finalizing the abstract mappings, the syntax model is augmented with information not specified explicitly: missing link columns through *ValueList* by using

RDB schema information (PK-s, FK-s), finding which are source (<s>) and target (<t>) navigation items and linking them to the corresponding *ClassMap*-s. The result is *takes* property in the semantic metamodel, part of which is shown in **Error! Reference source not found.**9. The finalizing corresponds to expanding the short form of the mapping expressions and adding ontology and RDB schema information: from "*=>Registration:->*" to "*Student[Student_ID] -> [Student_ID]Registration[Course_ID]->[Course_ID]Course*".

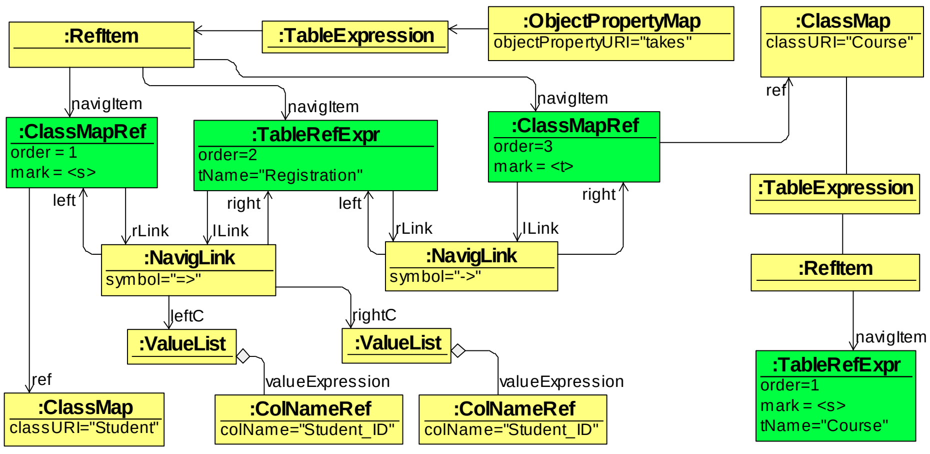

Figure 9. "=>Registration: ->" expression in the RDB2OWL semantic metamodel (part)

The RDB2OWL semantic metamodel is a more refined than the syntactic one since it uses classes that are not used on the syntax level. For example, consider "*[[Student]][student_id]->[[S]]*" annotation for *personID* property in **Error! Reference source not found.**4. After the syntactic parsing both references *[[Student]]* and *[[S]]* are stored as *NamedRef* instances in the metamodel. After the semantic processing (finding that *Student* is a class and that *S* is variable defined in the annotation in *PersonID* class) the first one is changed to become a *ClassRef* and the other one – the *DefVarRef* class instance.

The RDB2OWL translation into D2RQ is well described e.g. in (Būmans and Čerāns, 2016), we focus here on description of translation into the R2RML standard.

The basic RDB2OWL translation (from the semantic model) into R2RML involves creating a triple map for every RDB2OWL class map; such a triple map shall involve predicate object maps for all properties available in the context of the triple map subject class. The following example demonstrates a fragment of the generated R2RML code for the *Student* class from the mini-University ontology:

```
<#View1> rr:sqlQuery """
SELECT t.*,DPM2.MARK_RECEIVED AS "DPM2_MARK_RECEIVED"
   ,DPM3.DATE_COMPLETED AS "DPM3_DATE_COMPLETED"
   ,DPM4.VALUE AS "DPM4_VALUE"
   ,OPM1.COURSE_ID AS "OPM1_COURSE_ID"
FROM STUDENT t
LEFT OUTER JOIN REGISTRATION DPM2 ON
  t.STUDENT_ID=DPM2.STUDENT_ID
LEFT OUTER JOIN REGISTRATION DPM3 ON
  t.STUDENT_ID=DPM3.STUDENT_ID
LEFT OUTER JOIN NATIONALITY DPM4 ON t.NAT_CODE=DPM4.CODE
LEFT OUTER JOIN REGISTRATION OPM1
  ON t.STUDENT_ID=OPM1.STUDENT_ID;""".
<#TriplesMap1> a rr:TriplesMap; rr:logicalTable <#View1>;
  rr:subjectMap [
    rr:template "STUDENT/{STUDENT_ID}"; rr:class
ont:Student;];
  rr:predicateObjectMap [ rr:predicate ont:hasMark;
    rr:objectMap [ rr:column "DPM2_MARK_RECEIVED" ]
  ];
  rr:predicateObjectMap [ rr:predicate ont:dateCompleted;
    rr:objectMap [ rr:column "DPM3_DATE_COMPLETED" ]
  ];
   rr:predicateObjectMap [ rr:predicate ont:takes;
    rr:objectMap [ rr:parentTriplesMap <#TriplesMap2>;
    rr:joinCondition
       [rr:child "OPM1_COURSE_ID" ; rr:parent "COURSE_ID"]
];
```

It can be observed that in the case of multiple-valued properties (e.g. the object property *takes* and the data properties *hasMark* and *dateCompleted* for the *Student* class), the respective join expression in the view for the class map create a Cartesian product to provide rows for all linked value combinations (so, if a student would have 10 courses taken, 10 marks and 10 completion dates, this would result in 1000 rows in the view for the R2RML class map). The joins in the view arise from the properties included in the class map for the *Student* class: each of the properties *hasMark*, *dateCompleted* and *takes* introduces a join of the STUDENT table with the REGISTRATION table (cf. Fig. 2 and Fig. 3).

To cope with this situation, the RDB2OWL translation into R2RML has a new optimization option to generate separate triple maps for each multiple-valued data and object property. With the optimization option no extra table links are generated if more than one property introducing linked tables has a common domain class. In this case for each of these properties a separate *rr:TriplesMap* is generated with an only one table join: joining a table for the domain class with the table introduced by the property.

```
<#View1> rr:sqlQuery """
SELECT t.*, OPM1.COURSE_ID AS "OPM1_COURSE_ID"
FROM STUDENT t
LEFT OUTER JOIN REGISTRATION OPM1
  ON t.STUDENT_ID=OPM1.STUDENT_ID
; """.
<#TriplesMap1> a rr:TriplesMap;
```

```
  rr:logicalTable <#View1>;
  rr:subjectMap [
    rr:template "STUDENT/{STUDENT_ID}";
       rr:class ont:Student;
  ];
  rr:predicateObjectMap [ rr:predicate ont:takes;
    rr:objectMap
      [ rr:parentTriplesMap <#TriplesMap2>;
    rr:joinCondition [rr:child "OPM1_COURSE_ID" ;
    rr:parent "COURSE_ID"]]];.

<#View2> rr:sqlQuery """
SELECT t.*, DPM2.MARK_RECEIVED AS "DPM2_MARK_RECEIVED"
FROM STUDENT t
LEFT OUTER JOIN REGISTRATION DPM2 ON
t.STUDENT_ID=DPM2.STUDENT_ID
; """.
<#TriplesMap1_hasMark> a rr:TriplesMap;
  rr:logicalTable <#View2>;
  rr:subjectMap [
    rr:template "STUDENT/{STUDENT_ID}";
       rr:class ont:Student;
  ];
  rr:predicateObjectMap [ rr:predicate ont:hasMark;
    rr:objectMap [ rr:column "DPM2_MARK_RECEIVED" ]
  ];.
```

The RDF triples created by the optimized R2RML mapping for each concrete database would coincide with the triples created by the R2RML mapping before optimisation. On the other hand, parallel multiple joins in the triple map views leading to unnecessary row set explosion, are avoided.

4. Conclusions

The RDB2OWL mapping language and tool allows creation of wide range of database-to-ontology mappings and translation these mappings into executable D2RQ and R2RML mappings. So, the tools supporting either the D2RQ format, or the W3C standard R2RML format, can be used to obtain either the RDF dump of the source relational database, or an SPARQL endpoint on-the-fly serving the data from the source relational database.

The initial experience using RDB2OWL tool over larger ontologies and RDB schemas, for example, in Latvian medicine registries example (Barzdins et al., 2008), has demonstrated the tool usability. RDB2OWL mapping specifications are much smaller and easier to write than the corresponding D2RQ and R2RML code.

The newly implemented RDB2OWL to R2RML translation optimization addresses the row set blow-up issue due to parallel multiple-valued (1-to-n) joins that can be obtained in R2RML views during the naïve translation.

It can be expected that further practical use cases of the proposed mapping methodology can suggest also further translation process improvements. In this respect we consider achieving a fully practical RDB2OWL implementation on the R2RML basis and applying it to concrete use cases as a future work.

Acknowledgements

This article publishes a revised and extended material of the authors' presentation at the conference 'INNOVATIONS AND CREATIVITY' held in Liepāja on April 6-7, 2018.

References

Barzdins, G., Liepins, E., Veilande M., Zviedris M. (2008). Semantic Latvia Approach in the Medical Domain. In: Proc. 8th International Baltic Conference on Databases and Information Systems. H.M.Haav, A.Kalja (eds.), TUT Press, pp. 89-102. (2008).

Berners-Lee, T., Hendler, J., Lassila, O. (2001). "The Semantic Web", Scientific American, May 2001, p. 29-37.

Blakeley, C. (2007). "RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping)", OpenLink Software, 2007

Būmans, G., Čerāns, K. (2011). Advanced RDB-to-RDF/OWL Mapping Facilities in RDB2OWL In: Proc. of BIR 2011, Riga, Latvia, October 7-8, 2011. LNBIP 90, pp. 142-157. Springer, Heidelberg, 2011, ISBN:978-3-642-24510-7

Būmans, G., Čerāns, K. (2016). Database to Ontology Mappings in RDB2OWL: Notation and Implementation. In: G. Arnicans, V.Arnicane, J.Borzovs, L.Niedrite (eds.), Databases and Information Systems, IOS Press 2016, vol. 291, p. 31-42., ISBN: 978-1-61499-713-9

Calvanese, D., Cogrel, B., Komla-Ebri, S,, Lanti, D., Rezk, M., Xiao, G. (2015) How to Stay Ontop of Your Data. In: Databases, Ontologies and More. ESWC (Satellite Events) 2015: 20-25.

Čerāns, K.. Būmans, G. (2011) RDB2OWL: a RDB-to-RDF/OWL Mapping Specification Language. In: J.Barzdins and M.Kirikova (Eds.), Databases and Information Systems VI, IOS Press 2011, p.139-152.

Das, S., Sundara, S., Cyganiak, R. (2012). R2RML: RDB to RDF Mapping Language. http://www.w3.org/TR/r2rml/

Liepiņš, R. (2011). lQuery: A Model Query and Transformation Library, Scientific Papers, University of Latvia, 2011. Vol. 770, p. 27-45.

Motik, B; Patel-Schneider P.F; Parsia B. (2012) OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. https://www.w3.org/TR/owl2-syntax/

Sequeda, J.F., Cunningham, C., Depena, R., Miranker, D.P. Ultrawrap (2009). Using SQL Views for RDB2RDF. In: Poster Proceedings of the 8th International Semantic Web Conference (ISWC2009), Chantilly, VA, USA.

Speicher, S., Arwe, J., Malhotra, A. (2014.) Linked Data Platform 1.0. https://www.w3.org/TR/2015/REC-ldp-20150226/

WEB (a) Resource Description Framework (RDF). http://www.w3.org/RDF/

WEB (b) D2RQ. Accessing Relational Databases as Virtual RDF Graphs. http://d2rq.org/

WEB (c) Revelytix Spyder Tool (2012). http://www.revelytix.com/content/spyder

WEB (d) R2RML Parser. https://github.com/nkons/r2rml-parser

Received October 20, 2018, revised August 1, 2019, accepted October 22, 2019