RDB2OWL: a RDB-to-RDF/OWL Mapping Specification Language

Kārlis ČERĀNS[a] and Guntars BŪMANS[b,1]
[a] *Institute of Mathematics and Computer Science, University of Latvia*
Raiņa bulvāris 29, Riga LV-1459, Latvia
karlis.cerans@mii.lu.lv
[b] *University of Latvia, Raiņa bulvāris 19, Riga LV-1586, Latvia*
guntars.bumans@gmail.com

Abstract. We present a RDB2OWL mapping specification language that is aimed at presenting RDB-to-RDF/OWL mappings possibly involving advanced correspondences between the database and ontology in a human comprehensible way. The RDB2OWL mappings can be regarded as documentation of the database-to-ontology relation. The RDB2OWL language reuses the OWL ontology structure as a backbone for mapping specification by placing the database link information into the annotations for ontology classes and properties. It features reuse of database table key information, user defined and table functions, as well as multiclass conceptualization that is essential for keeping the mapping compact in case of creating a conceptual partitioning of large database tables. We report also on initial implementation experience for a subset of RDB2OWL constructs.

Keywords. Relational databases, RDF, OWL ontologies, mappings

Introduction

With the increasing use of semantic technologies both on world wide web scale, and locally within enterprises, supported by the development of open definitions and standards such as RDF [1], SPARQL 1.1 [2], OWL 2.0 [3] and many others, and the growing number and capacity of the tools for semantic contents management, the need of smooth and efficient information integration between the "old data world" organized primarily along the relational database (RDB) paradigm and the "new information world" inspired by the semantic standards and their support technologies, arises to be of utmost importance. There has been both a long-going and recent intensive research and technology development on bridging these two worlds by RDB-to-RDF/OWL mapping definition languages and techniques, going back to [4] and featuring a number of successful and promising approaches including R_2O [5], D2RQ [6], Virtuoso RDF Views [7], DartGrid [8] as well as recent work on UltraWrap [9] and Triplify [10]. There is W3C RDB2RDF Working Group [11] related to standardization of RDB to RDF mappings, as well as a related published survey of mapping RDBs to RDF [12]. Most of these approaches are concentrating on efficient machine processing of the mappings, often preferably querying RDBs on-the-fly from an SPARQL-enabled

[1] Partially supported by ESF project 2009/0138/1DP/1.1.2.1.2/09/IPIA/VIAA/004

endpoint. Much less attention, however, has been given to creating high-level mapping definitions that are oriented towards readability for a human being and that have a capacity to handle complex database-to-ontology/RDF schema relations.

The concise and human readable RDB-to-RDF/OWL mappings in a situation of an involved schema correspondence is essential e.g. for relational database semantic reengineering task, where a possibly legacy database is to be mapped to RDF/OWL, and the mapping information itself together with the ontology model is required as a documentation of the existing RDB data structure. As an existing approach in this area Semantic SQL [13] can be mentioned, still its relations to the open SPARQL standard, as well as its abilities to handle complex dependencies within a mapping are unclear.

Defining human readable mappings has been long an issue within MOF-centered [14] model transformation community. A model transformation language such as MOF QVT [15], MOLA [16] or AGG [17] (there are many other languages available) may be used for structural presentation of a mapping information; however, these languages are not generally designed to benefit from the mapping specifics that arise in RDB-to-RDF/OWL setting, partially due to simple target model structure (RDF triples). We note an interesting practical experience report of this kind in [18].

The possibility to define RDB-to-RDF/OWL mappings efficiently has emerged as an issue of primary importance also in practical semantic re-engineering of medical domain data in Latvia [19,20,21]. This approach proposes creating ontology (ontologies) for data that are available in a specific domain (e.g., government data, or medical data), using visual graphical notation offered by OWLGrEd [22,23] or UML/OWL profile [24], followed by RDB data integration into the format of the defined conceptual ontology, then followed by providing tools that are able to access the semantic data by means of a visual SPARQL query endpoint [20,25].

In this paper we propose a high level declarative RDB-to-RDF/OWL mapping specification language RDB2OWL that is based on re-using the target ontology structure as a backbone for mapping structure by storing the database link information in the annotations to ontology classes and properties, as well as to the ontology itself. Since the ontology and mapping structures may match imperfectly, some secondary mapping structuring means are made available, as well. The RDB2OWL language features also:

- reuse of RDB table column and key information, whenever that is available,
- concrete human readable syntax for mapping expressions that is very simple and intuitive in the simple cases, and can also handle more advanced cases,
- built-in and user defined functions (including column-valued functions),
- advanced mapping definition primitives, e.g. multiclass conceptualization that avoids the need of specifying long filtering conditions arising due to fixing a missing conceptual structure on large database tables,
- possibility to resort to auxiliary structures defined on SQL level (e.g. user defined permanent and temporary tables, as well as SQL views), still maintaining the principle that the source RDB is to be kept read only.

A previous work by the authors [26,27] on the basis of mapping abstract syntax structures has shown the viability of the central concepts of RDB2OWL by providing an implementation for a subset of RDB2OWL that allowed to efficiently generate the RDF data corresponding to databases from Latvian medical domain [20,21].

In the following sections we introduce three layers of the RDB2OWL mapping language – the Raw language (structure specification, not actually meant for end-users), the Core language (basic mechanisms) and Extended language. Finally, we make some implementation notes and conclude the paper.

1. Raw Mapping Language

A RDB2OWL mapping is a relation between a source relational database schema ***S*** and target OWL ontology ***O***. The mapping specifies the correspondence between the concrete source database data (table row cell values) and RDF triples "conforming" to the target ontology. We present the abstract syntax structure of a raw RDB2OWL mapping in a form of MOF-style [14] metamodel in Figure 1, with additional expression and filter metamodel in Figure 2 (the dashed items in Figure 2 are not included within the Raw version of RDB2OWL metamodel).

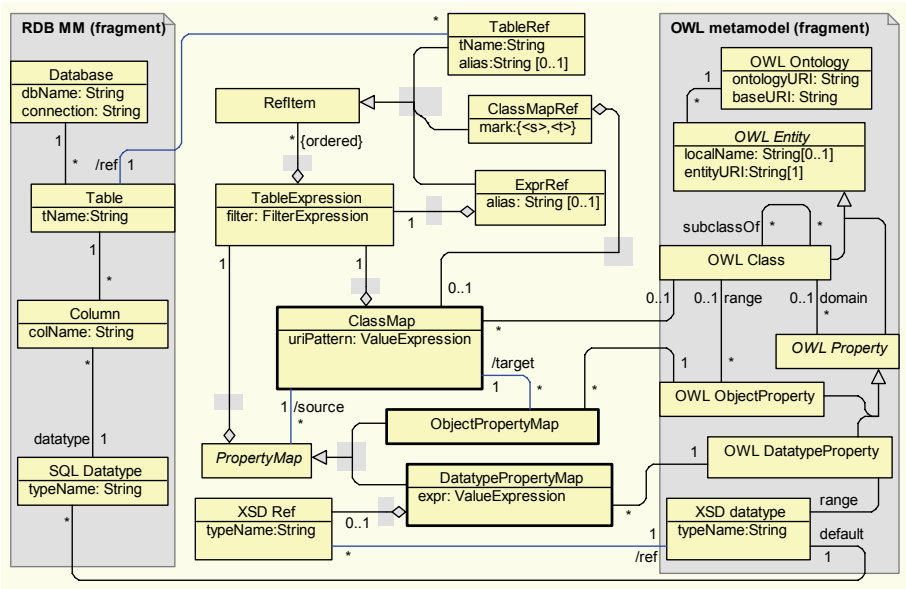

Figure 1. Raw RDB2OWL mapping metamodel

The metamodel refers to RDB schema and OWL ontology structure descriptions, presented here as the RDB MM fragment and the OWL metamodel fragment. The RDB2OWL mapping classes themselves are shown in the middle part of Figure 1. We note that in the raw RDB2OWL only the table and column structure of the source RDB is reflected, disregarding any primary and foreign key information (all table linking information has to be contained in the mapping itself). This approach allows applying RDB2OWL to legacy databases without presuming any normalization features of those.

The OWL metamodel fragment includes domain resp. range information for an OWL object or datatype property, if the property can be identified to have a single domain resp. range that is a named class or a data range (a subset of a known datatype).

An RDB2OWL mapping consists of "elementary mappings", or maps, that are instances of *ClassMap*, *ObjectPropertyMap* and *DatatypePropertyMap* classes in the mapping metamodel. The class maps (*ClassMap* instances) are responsible for *Table*-to-*OWL Class* mappings (with options to add filtering expressions and linked tables).

The datatype property maps (*DatatypePropertyMap* instances) provide *Column*-to-*OWL DatatypeProperty* mappings. Each datatype property map is based on a source class map and can access the class map's table information; it can introduce further linked tables and filters into the table context for column expression evaluation.

The object property maps (*ObjectPropertyMap* instances) establish OWL object property links that correspond to related tables in the database. The tables to be related generally come from source and target class maps of the object property map; they are joined using explicit join condition specification in the object property map's table expression's *filter* attribute, with option to include further linked tables and filters.

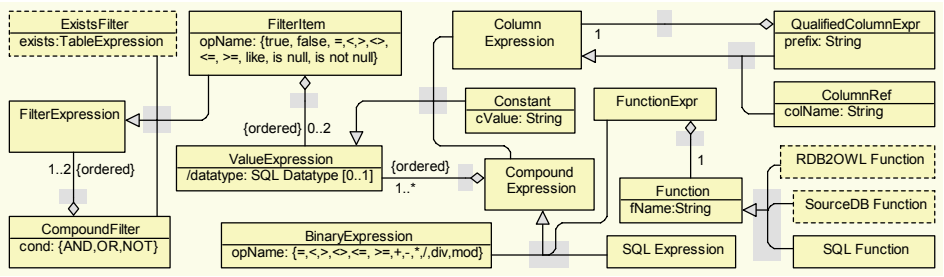

Figure 2. Expression and filter metamodel

1.1. Syntax of Expressions and Entity Maps

Syntactically the RDB2OWL mapping definition is achieved by storing textual class map and property map descriptions in the annotations to the respective OWL classes and properties (we assume a fixed annotation property *DBExpr* is used for this purpose). An OWL class may have several annotations each describing a class map; an OWL datatype or object property may have several annotations describing datatype or object property maps respectively. We start mapping syntax explanation by table expressions.

A table expression description consists of a comma-separated list of reference items, followed by optional filter expression that is separated from the reference item list by a semicolon. Each reference item can be (i) a table name possibly followed by an alias, (ii) a class map reference (one of strings '<*s*>' or '<*t*>', optionally preceded by a class map description), or (iii) a table expression enclosed in parentheses possibly followed by an alias string. The filter expression is built in accordance to the abstract syntax of expression and filter metamodel of Figure 2.

The concrete syntax of expressions is based on SQL expression syntax, however not including SQL-style subqueries. Since the RDB2OWL table expressions form a hierarchical structure where every hierarchy level can be identified by an alias, we let the *fully qualified names* (*fqn*, for short) for columns be expressions of the form $a_n.(a_{n-1}.(a_{n-2}. \ldots .(a_1.(a_0.c))\ldots))$, where *c* is a source database table column name, a_0 is a table name and $a_1 \ldots a_n$ are prefixes; each prefix is an alias or a class map reference mark. We let a column be identified within a table context not only by its *fqn*, but also by shorter forms (some of prefixes omitted) if that allows unique column identification.

Presenting the syntax we presume the use of parentheses to allow unique identification of abstract syntax structure from the expression text. Some table expressions are:

- *Person*
- *Person P, Address A; P.AdrID_FK=A.AdrID*
- *(Person P, Address A; P.AdrID_FK=A.AdrID) PA, (Company, WorksFor; CID=CID_FK); PA.(P.PersonID)= WorksFor.PersonID_FK*
- *<s>, <t>; <s>.AdrID_FK = <t>.AdrID*

A class map description is obtained by adding to a table expression description an uri pattern description in the form *{uri=(<item$_1$>,...,<item$_k$>)}*, where each *item$_i$* is a

value expression (typically, a textual constant, or a reference to a database table column); such pattern describes a conversion to uri form and concatenation of all values $item_i$. Some class map examples are:

- *Person {uri=('XPerson',PersonID)}*
- *Person P, Address A; P.AdrID_FK=A.AdrID {uri=('XPerson',PersonID)}.*

An object property map is described by a table expression, containing exactly one source class map reference (a class map reference with *mark*=<*s*>) and exactly one target class map reference (*mark*=<*t*>) within the expression's declaration structure. Some table expressions that describe object property maps are:

- *<s>, <t>; <s>.AdrID_FK = <t>.AdrID*
- *(Person {uri=('XPerson',PersonID)}) <s>, (Address {uri=('XAddress',AddressID)}) <t>; <s>.AdrID_FK = <t>.AdrID*

(<*s*> and <*t*> are class map reference marks preceded by class map definitions).

A class map reference mark <*s*> or <*t*> can be included into object property *p* map expression structure either with a preceding class map description, or without it. The inclusion of a class map description within an object property map expression means defining in-place a new class map that the object property map is going to refer to as its source or target. The most common usage of the construct, however, is without the explicit class map description; in this case the mark <*s*> (resp. <*t*>) refers to the single class map that is ascribed to the domain (resp. range) class of *p*.

A datatype property map is described by a table expression which is required to contain a single <*s*>-marked reference to the source class map, followed by a value expression that is attached to the table expression using a dot notation and further on by an optional datatype specification preceded by the string '^^'. Some datatype property description examples are *<s>.Name*, *<s>.Name^^xsd:String* and *(XPerson {uri=('XPerson',PersonID)} <s>).Name*.

In the case, if the table expression part for a datatype property map is just <*s*>, we allow omitting it together with the following dot symbol when using a short form of mapping specification. Similarly, the declaration part (together with the following semicolon) may be omitted for an object property map, if it is just <*s*>,<*t*>. These conventions are used in the following example in Section 1.2.

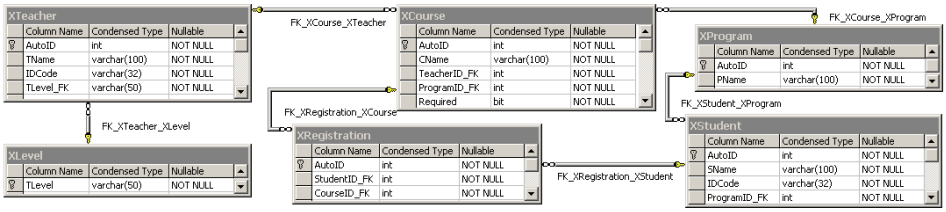

Figure 3. A mini-university database schema

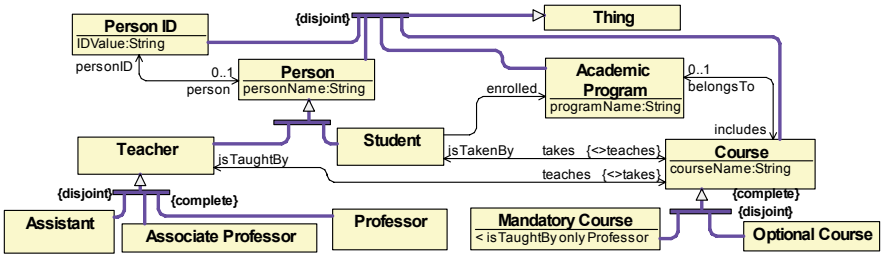

Figure 4. OWL ontology corresponding to the mini-University

1.2. A Mini-University Example

As a larger example, consider a mini-university database schema in Figure 3 and a corresponding OWL ontology in Figure 4. The OWL ontology is created using OWLGrEd notation and editor [22, 23]. The example is adopted from [28] and [29].

Observe the table splitting (*XCourse*, *XTeacher*) and table merging (*Person* from *XStudent* and *XTeacher*) in the ontology using the subclass relations. OWL class *PersonID* is based on non-primary key columns in each of *XStudent* and *XTeacher* tables. The n:n relation *takes* reflects a student-to-course association that in the RDB is implemented using *XRegistration* table.

Figure 5 shows a full mapping definition in Raw RDB2OWL notation for the mini-University example. We use here a custom extension of OWLGrEd editor [23] and we depict *DBExpr* annotations in the form '*{DB: <annotation_text>}*' to show graphically the ontology together with the annotations. The class and object property annotations in the example are shown in italics, while the datatype property annotations use plain text.

The raw RDB2OWL mapping format reveals the structure of the information that needs to be specified in order to define the mapping. The syntactic presentation of the mapping, however, is less than satisfactory, especially for not 1:1 correspondence cases between the RDB and ontology structure (e.g. a *personID* property, where the explicit class map definitions have to be included within each of property maps description). These issues, as well as more compact and better structure revealing forms for class and property map definitions are handled in RDB2OWL Core notation in Section 2.

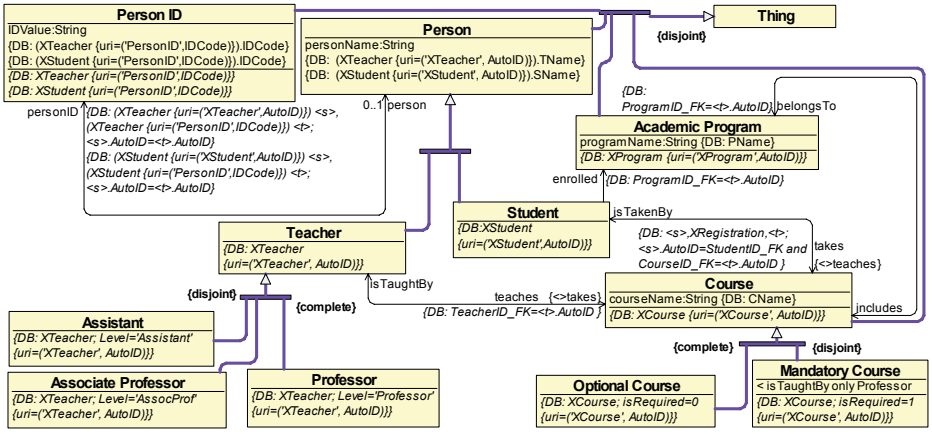

Figure 5. Annotated mini-University ontology using RDB2OWL Raw model

1.3. Mapping Semantics

For the mapping semantics purpose we assume that the source (resp. target) class map descriptions for property maps are textually included behind the corresponding <*s*> (resp. <*t*>) marks and treat the marks <*s*> and <*t*> themselves as ordinary aliases.

The *context C(t)* for the table expression *t* is built inductively following *t* structure:

(a) if *t* is a reference to a table with name *tName*, then *C(t)* consists of expressions *tName.cName* for *cName* ranging over all *t* column names;

(b) if *t* is of the form *t' a* for table or a table expression *t'* and an alias *a*, then $C(t) = \{a.x \mid x \in C(t')\}$;

(c) if t consists of items t_1, …, t_n with no aliases specified then $C(t) = C(t_1) \cup C(t_2) \cup ... \cup C(t_n)$; if for some t_i there is an alias specified, the rule (b) is to be applied on this item before (c).

We require that the fully qualified names in the table expression context be distinct; if that is not the case, the table expression is not well formed.

Given a table expression t for a property map m, we denote by *src(t)* (resp. *trg(t)*) the column prefix within *C(t)* that ends in <*s*> (resp. <*t*>); the requirements on t structure ensure that *src(t)* (resp. *trg(t)*) is uniquely defined. Note that, for instance, for a table expression *t=(A <s>) E1, B <t>* we have *src(t)=E1.<s>* and *trg(t)=<t>*.

For a value expression x and a string a we define the a-lifted form of x by replacing every column reference t within the x structure by *a.t*.

The semantics *R(t)* of a table expression t on the source database $\boldsymbol{S}$ is defined as a set of rows with columns corresponding to the table expression context *C(t)* the following way. If there is no filter expression specified as part of t, then

(a) if t is a table, then *R(t)* consists of all its rows
(b) if t is of the form *t' a* for a table expression *t'* and an alias a, then *R(t)* is obtained by renaming *R(t')* columns via adding the prefix '*a.*'
(c) if t consists of items t_1, …, t_n with no aliases involved then *R(t)* is formed by taking all row combinations from the row sets $R(t_1), ..., R(t_n)$.

If, however, there is a filter specified as part of t, only the rows that satisfy the filter are retained in *R(t)*, as obtained above.

For every row $r \in R(t)$ there is defined notion of value expression evaluation: the column expressions are looked up within the row; the constants and standard operators have their usual meaning. Let *concat* be a function concatenating all its arguments.

Given the source database schema $\boldsymbol{S}$, the corresponding RDF triples are defined for each class map and property map separately.

For a class map or property map m let t be the table expression contained in m and let e be the OWL entity (OWL class or OWL property) that m is ascribed to (we consider only class maps ascribed to OWL classes here). Let r be the *baseURI* specified for the target OWL ontology. In order to form the RDF triples that correspond to $\boldsymbol{S}$, we form in each case the row set *R(t)* by evaluating t on $\boldsymbol{S}$. For each row in *R(t)* we then proceed, as follows:

- if m is a class map, evaluate the expression contained in m's *uriPattern* attribute obtaining a string value v; then form the RDF triple <*concat(r,v),* 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type', *e.entityURI*>;
- if m is an object property map, evaluate the *src(t)*-lifted form of m's source class map *uriPattern* to obtain u and the *trg(t)*-lifted form of m's target class map *uriPattern* to obtain v, form the triple <*concat(r,u), e.entityURI, concat(r,v)*>;
- if m is a datatype property map, let d be the value of m's *expr* attribute evaluation; let s be obtained by evaluating the *src(t)*-lifted form of m's source class map *uriPattern* value. Further on we find *dt*: XSD datatype corresponding to d the following way:
    - if there is an XSD datatype specified within m, take this datatype
    - if the XSD datatype can be found as a default XSD datatype for d's SQL datatype, take this datatype
    - if the XSD datatype has been specified as *e.range*, take this datatype
    - if none of the above applies take *dt* to have *typeName='xsd:String'*.

  The resulting triple is <*concat(r,s),e.entityURI,concat(d,'^^',dt.typeName)*>.

2. The Core Language

We start presenting RDB2OWL core language by augmenting RDB2OWL raw metamodel (Figures 1 and 2) with a possibility to label a reference item (*RefItem* class element) a *top*-constrained element of a table expression. The semantics of such an element *el*, if designated for a table expression *t,* is reflected in the row set *R(t)* construction for *t* in that only the specified rows (e.g., only a top row) for columns corresponding to *el* is included into *R(t)* for a combination of values in other *t* columns. Syntactically we write *(XStudent <s>, XProgram <t> {top 1 PName asc}; PName > '')* to denote all students and the first program with non-empty name for each of them.

The RDB2OWL core language constructs are summarized in Figure 6. The core language constructs are semantically explained via their translation into the augmented RDB2OWL raw language. The principal novelties here are:

- a more refined table expression structure (each table expression reference list item now may be expressed as list-like navigation item and link structure);
- the primary key and foreign key information within RDB MM; this allows:
  (i) introducing default entity URI patterns on the basis of RDB schema structure: the default uri pattern for a class map, whose table expression is a single table *X* having a sole primary key column *P,* is defined as *uri('X',P)*; and
  (ii) avoiding the necessity to specify column names in linked table conditions, if these correspond to a unanimously clear table key information;
- the naming of class maps (*defName* attribute), together with a possibility to refer within a table expression item (a *NamedRef* element) either to such a defined class map, or to the sole un-named class map defined for a certain OWL class *c*; a named reference syntactically is represented as '*[[R]]*', where '*R*' is either the name of an owl class, or the defined name of a class map.

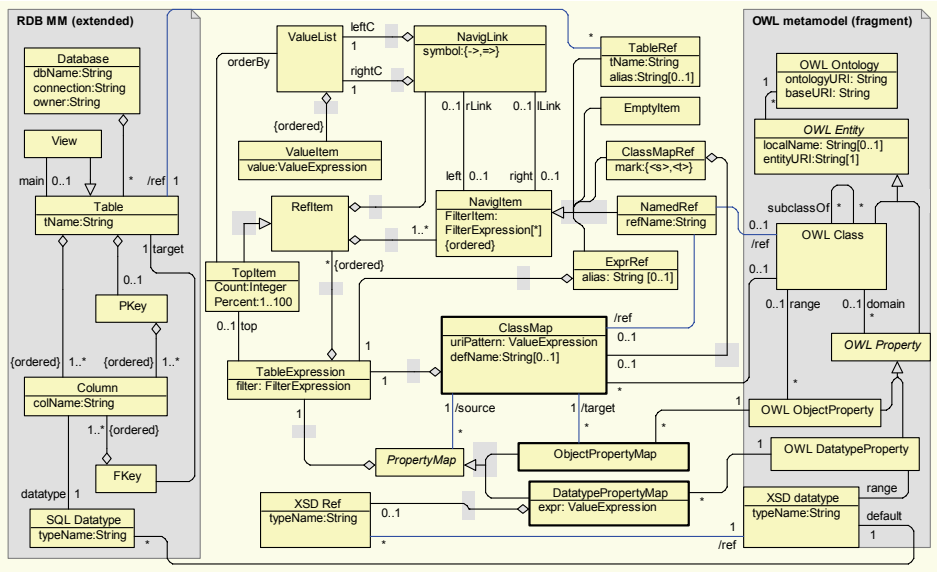

Figure 6. RDB2OWL Core metamodel

The principal building blocks of the RDB2OWL Core metamodel, as in the model Raw version, are class maps and property maps that are based on table expressions,

consisting of reference items. A reference item can contain a singleton list of navigation items thus subsuming the reference item structure of Raw metamodel. The navigation item and navigation link structure of a reference item allows encoding filters in simple table expressions as equality conditions between columns in two adjacent navigation items. For instance, *XStudent[ProgramID_FK]->[AutoID]XProgram* is a reference item presentation in a navigation item and link form, where *XStudent* and *XProgram* are navigation items and *[ProgramID_FK]->[AutoID]* is a navigation link with *ProgramID_FK* and *AutoID* being its left and right value lists (and the sole value items within these lists) respectively. The presented navigation item and link structure can be translated into an equivalent table expression *XStudent E1, XProgram E2; E1.ProgramID_FK=E2.AutoID*, where *E1* and *E2* are unique system generated aliases introduced to avoid potential table name conflicts. The navigation structures can form also longer chains as, for example, the following (*):

<s>[AutoID]->[StudentID_FK]XRegistration[CourseID_FK]->[AutoID]<t>.

We allow an empty navigation item only in linked navigation structures (the ones containing link symbol -> or => adjacent to the empty item); in this case the empty item is shorthand for a class map reference *<s>* or *<t>*, and it is replaced by the reference during the expression's short form unwinding, as explained below in steps 1. .. 4.

For a navigation link *A[F1]->[F2]B* between two single table items *A* and *B* (possibly included in expressions with filters and aliases, or being class map references or named references to single table class maps) the table column list *[F2]* may be omitted, if it corresponds to the primary key of *B*. Furthermore, *[F1]* may be omitted, as well, if there is a single foreign-to-primary key reference in the source RDB from *A* to *B* and it is based on the equality of the foreign key columns *F1* to primary key columns *F2*. For the case when *A[F1]->[F2]B* is a primary-to-foreign key relation, we allow omitting both *[F1]* and *[F2]*, in this case presenting the expression as *A=>B*. The above navigation structure (*), given the mini-University database schema of Figure 3, for the property *takes* can then be presented as *<s>=>XRegistration-><t>*.

The table expressions corresponding to the RDB2OWL Core metamodel are transformed into the (augmented) RDB2OWL Raw format via the following steps:

A. Unwinding of the short form of table expressions (insertion of *<s>* and *<t>* class map references, where appropriate) for datatype and object properties, following the rules explained below in steps 1. .. 4.
B. Insertion of explicit uri pattern definitions for class maps, in place of default uri expressions (including both class maps ascribed to classes and defined inline).
C. Replacing named references by their referred class maps defined inline.
D. Insertion of explicit column information definition in navigation links.
E. Converting the navigation expression structure into reference item structure.

For a table expression reference list structure consisting of expression and navigation items we define its *leftmost* (resp. *rightmost*) item by recursively choosing expression's leftmost (resp. rightmost) reference or navigation item, until an item that is an empty item, a table reference, a class map reference (with or without an explicit class map specification), or a named reference, is found.

The rules for unwinding the short form of a table expression *e* corresponding to an object property map are, as follows:

1. if *e*'s reference item list is empty define this list to be *<s>,<t>*;
2. if the leftmost (resp. rightmost) item is an empty item, replace this item by *<s>* (resp. *<t>*); e.g. any of '*<s>->*', '*-><t>*' and '*->*' is replaced by '*<s>-><t>*' and '*(XPerson <s>)->*' is replaced by '*(XPerson <s>)-><t>*';

3. if <*s*> (resp. <*t*>) is not present in *e* reference list structure and is explicitly referenced from the filter expression, add <*s*> (resp. <*t*>) as a new leftmost (resp. rightmost) reference item;
   for instance '*XRegistration;<s>.AutoID=StudentID_FK*' is replaced by '*<s>, XRegistration;<s>.AutoID= StudentID_FK*';
4. if <*s*> (resp. <*t*>) is not present in *e* reference list structure, add <*s*> (resp. <*t*>) as an alias for the *e*'s leftmost (resp. rightmost) item; for instance '*XPerson, XProgram*' is replaced by '*XPerson <s>, XProgram <t>*'.

Similar rules (in the part regarding <*s*>) apply also for unwinding of the short form of table expressions involved in datatype property map definitions.

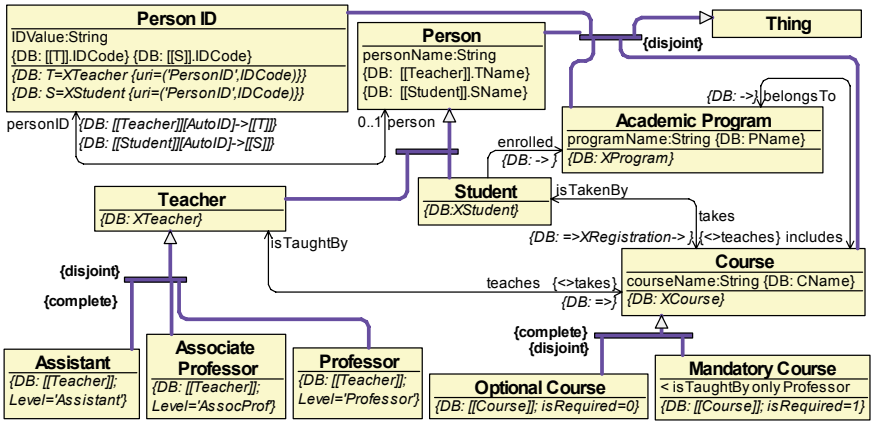

Figure 7. Annotated mini-University ontology using RDB2OWL Core model

Figure 7 shows annotated mini-university ontology of Figure 5, this time being re-worked in accordance to the core language constructs. Note the simple '->' annotations for object properties *enrolled* and *belongsTo*, as well as '=>' for *teaches*. We note also the use of named references expressed as *[[Teacher]]* (referring to the class map defined for the *Teacher* class) and *[[S]]* and *[[T]]* referring to the class maps defined as named class maps for *PersonID* class.

We demonstrate the steps A .. E, as defined above, on the object property map expression *[[Teacher]][AutoID]->[[T]]* of object property *personID* in Figure 7.

A. *([[Teacher]] <s>)[AutoID]->([[T]] <t>)* (unwinding rule 4. used).
B. Applies to converting *XTeacher* into *XTeacher {uri=('XTeacher',AutoID)}* within *Teacher* class (*[[T]]* already denotes an expression with *uri* specified).
C. *((XTeacher {uri=('XTeacher',AutoID)}) <s>) [AutoID] ->*
   ((XTeacher {uri=('PersonID',IDCode)}) <t>);
D. *((XTeacher {uri=('XTeacher',AutoID)}) <s>) [AutoID] ->*
   [AutoID] ((XTeacher {uri=('PersonID',IDCode)}) <t>)
E. *((XTeacher {uri=('XTeacher',AutoID)}) <s>) E1,*
   ((XTeacher {uri=('PersonID',IDCode)}) <t>) E2;
   E1.(<s>.AutoID)=E2.(<t>.AutoID) (the filter can be written also as *E1.AutoID=E2.AutoID* or *<s>.AutoID=<t>.AutoID*).

Figure 7 shows that mapping definition for a simple (still not completely straightforward) mapping case of mini-University can be done in RDB2OWL Core language in a very compact, yet intuitive way.

3. The Extended Language

We present here the mapping language and framework extensions that are essential for mapping applicability to practical use cases, maintaining the compact and intuitive mappings. We relate the explanation of these concepts here to their practical use in a case study of using RDB2OWL approach to migration into RDF format of 6 Latvian medical registries [20,21] data, involving 106 source database tables, 1353 table columns (in total more than 3 million rows, altogether 3 GB of data), as reported in [26,27]. The corresponding OWL ontology in the case study had 172 OWL classes, 814 OWL datatype properties and 218 OWL object properties [26].

3.1. Multiclass Conceptualization

The motivation for the feature comes from the cases when some large database table during the mapping is conceptually split into a set of OWL classes, each class responsible for a certain subset of the table columns. For instance, there may be different groups *A*, *B* and *C* of measurements taken during a clinical anamnesis, all recorded into a single table *T* as fields *A1..A10*, *B1..B10* and *C1..C10*. The conceptual model may indicate classes *A*, *B* and *C* corresponding to *T*, with the condition that an ontology individual corresponding to a *T* record has a type *A* (resp. *B*, resp. *C*) if and only if at least one of fields *A1..A10* (resp. *B1..B10*, resp. *C1..C10*) has been filled. This can be solved in RDB2OWL Core notation by adding a filter *(A1 is not null) OR ... OR (A10 is not null)* to the class map for *A* and similar filters for *B* and *C*. However, as the field groups may be very large (up to 35 fields in the medicine registries example), writing these conditions becomes a tedious and error-prone task.

To offer an user-friendlier notation for this situation we introduce into RDB2OWL specific decorations that can be attached to the class maps and/or OWL classes. The decoration '*?Out*' that can be shortened to '*?*' invokes the semantics of specifying a correspondence to the source database data only for those <*x*,'rdf:type',*o*> instance triples, where a corresponding triple <*x*,*p*,*y*> exists for some property *p* whose domain is *o* (i.e. some property that is going out of *o*).

In the considered example, the class maps for each of *A*, *B* and *C* will involve no explicitly specified filtering condition, but they will be decorated by '*?*'.

There are also '*?In*' and '*?Any*' decorations available for class maps and OWL classes, requiring existence of incoming or any property for an OWL class instance. Furthermore, we provide '*?Dom*' decorations for property maps and '*?Ran*' decorations for object property maps that are used to validate that the subject or object of a triple corresponding to the property (or a property map) has its 'rdf:type'- triple to the corresponding property domain resp. range ontology class.

For the Latvian medicine registries migration the multiclass conceptualization feature has been extensively used: the decoration '*?Out*' has been attached to 54 out of 172 OWL classes having 542 outgoing datatype properties (out of the total 814 OWL datatype properties in the case study).

3.2. Auxiliary Database Objects

As can be observed from the presented examples, the mapping language provides a format of mapping expression that is more convenient than a mapping encoding within

SQL; however, for the sake of universality, the possibility to create additional tables and the full power of SQL needs to be made available within mapping definition.

The practical need in auxiliary database objects has been observed for *Numbers* or *Tally* table (cf. [30]) as well as the classifier tables that have not been explicitly created during the relational database design time. We propose creating a new auxiliary database schema for the mapping definition, where the auxiliary tables and views, as well as some temporary data structures, if they were necessary, could be stored. The mapping annotations within the ontology then contain the information pointing to:

1) the auxiliary database,
2) the information about the code that is to be executed every time the RDF triple generation from the source database is run (this may be e.g. some complicated data processing, or filling temporal tables).

3.3. Built-in and User-defined Functions

There is a number of built-in RDB2OWL functions that allow a higher level expression forming in the mapping specification. We mention here the single-argument conversion functions: *#varchar(_)*, *#xvarchar(_)* (converts to varchar and eliminates leading and trailing spaces), *#uri(_)*, as well as multi-argument *#concat(...)*, *#xconcat(...)* and *#uriConcat(...)* that combine conversion and concatenation. Furthermore, *#exists(...)* takes any number of arguments and returns *1* iff at least one of them is not null and *#iif(a,b,c)* chooses the value of *b* or *c* depending on *a* value being *1* or *0*. The list of built-in functions is to be finalized within upcoming RDB2OWL reference manual.

An important RDB2OWL feature is a possibility to define user-defined functions. Technically these functions are placed in *DBExpr*-annotations on the ontology level. A function may take a fixed number of arguments, whose names start with '@' symbol. A simple function body consists of a value expression (including optional data type specification), as, for instance *BoolT(@X) =(#iif(@X,'true','false')^^xsd:Boolean)* that converts a bit value 0 resp. 1 into '*true*^^*xsd:Boolean*' resp. '*false* ^^*xsd:Boolean*'.

A principal feature of user-defined functions is a possibility to include also a table expression within the function body (the function body then corresponds to datatype property syntax). If a function *f(@X)=((F;filter).val)* is called as *f(V)* in the context of a table *A*, we evaluate the expression *(A,F;filter[V/@X]).val[V/@X]*, where *[V/@X]* denotes the substitution of the value *V* for the variable *@X*. As a result, a single or several resulting values are obtained; all of the resulting values are considered for making RDF triples corresponding to the source database information.

A practically important example of this kind has been the function *split4(@X) = ((Numbers;len(@X)>=N*4).substring(@X,N*4-3,4))*, where *Numbers* is a table with single integer column filled by numbers from *1* to *8000* [30]. The application *split4(FieldX)* of such a function ensures splitting of a character string into a set of its blocks of length 4 (e.g., '*199219982004*' → {'*1992*', '*1998*', '*2004*'}). Note that the *Numbers* table allows implementing a wide range of string splitting tasks [30].

A simpler application of table expressions in functions (not creating "column-valued" functions) would correspond to "translation table" functionality of D2RQ [6].

The full version of RDB2OWL admits also referring to database tables themselves as "translation functions", as well as built-in and user-defined aggregate functions not detailed here. We note that the aggregate function functionality (although, using lower level constructs) can be achieved also using SQL means, as described in Section 3.2.

4. Implementation and Experience

The RDF triple generation from RDB schema has been implemented for a subset of RDB2OWL language constructs, on the basis of manually created abstract syntax representation of mapping information. The implemented language constructs involve raw RDB2OWL model (with a restriction of a single table for a class map), with some support for automated URI pattern construction, as well as multiclass conceptualization and reliance on external database schema. The initial modeling and implementation effort using RDB2OWL approach has been reported in [26,27], where we report also on successful experience of semantic re-engineering of Latvian Medical registries data [20,21]. The mapping of the registries has been implemented by generating SQL statements that generate the corresponding RDF triples from the source RDB; the running time for approximately 42.8 million RDF triple generation on Intel Mobile Core 2 Duo T6570 processor running Windows Vista, 3 GB of RAM has been less than 19 minutes [27] – a result that is completely satisfactory for the foreseen application.

There is work in progress on creating a context-aware parser for the concrete syntax of RDB2OWL mapping language, as well as for supporting the full set of language features presented here.

Our primary interest is in running the RDF triple generation in a batch mode (this is sufficient for a wide number of applications, e.g. in government data area). However, the RDB2OWL mapping information can enable alternative implementations, possibly with generating the queries over the source SQL database online within the context of SPARQL queries over the target RDF data store (this should be at least theoretically possible for a large subset of RDB2OWL constructs; the temporal tables and user-defined aggregate functions are going to be among the first exclusions).

5. Conclusions

We have presented RDB2OWL approach to RDB-to-RDF/OWL mapping specification that re-uses the ontology structure as the backbone of the mapping specification by putting the mapping definitions in the OWL ontology entity annotations.

As the mapping examples show, the approach can be used for a convenient mapping definition. Combining the power of the RDB2OWL mapping definition approach with visual ontology modeling means such as OWLGrEd [22,23] notation and editor can be a viable mechanism for the RDB semantic re-engineering task. Since the ontology annotation mechanism is a part of ontology definition means, the RDB2OWL-annotated ontologies can be used also outside the concrete ontology editor.

The RDB2OWL approach has been successfully used for a “real-size” task of semantic re-engineering of databases in Latvian medical domain. There is work in progress towards the implementation of the full set of RDB2OWL constructs, including RDB2OWL parsing on a concrete syntax level and integrating into OWLGrEd editor.

It seems to be a plausible and interesting task to adapt the mapping constructions considered here also for RDF/OWL-to-RDF/OWL mapping definition that may be useful in transformation from the “technical data ontology” to the conceptual one, after the initial data – be these in RDB or some other format – have been exposed to the RDF format using a straightforward and technical structure preserving embedding.

References

[1] Resource Description Framework (RDF), http://www.w3.org/RDF/
[2] SPARQL 1.1 Query Language, http://www.w3.org/TR/2010/WD-sparql11-query-20100601/
[3] OWL 2 Web Ontology Language, Structural Specification and Functional-Style Syntax http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/
[4] T.Berners-Lee, Relational Databases on the Semantic Web, http://www.w3.org/DesignIssues/RDB-RDF.html, 1998.
[5] J.Barrasa, O.Corcho, G.Shen, A.Gomez-Perez: R2O: An extensible and semantically based database-to-ontology mapping language. In: *SWDB'04, 2nd Workshop on Semantic Web and Databases* (2004).
[6] D2RQ Platform, http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/spec/
[7] C.Blakeley: RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping), OpenLink Software, 2007.
[8] W.Hu, Y.Qu: Discovering Simple Mappings Between Relational Database Schemas and Ontologies, In *Proceedings of 6th International Semantic Web Conference (ISWC 2007), 2nd Asian Semantic Web Conference (ASWC 2007)*, LNCS 4825, pages 225-238, Busan, Korea, 11-15 November 2007.
[9] Sequeda, J.F., Cunningham, C., Depena, R., Miranker, D.P. Ultrawrap: Using SQL Views for RDB2RDF. *In Poster Proceedings of the 8th International Semantic Web Conference* (ISWC2009), Chantilly, VA, USA. (2009)
[10] 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* (2009).
[11] W3C RDB2RDF Working Group, http://www.w3.org/2001/sw/rdb2rdf/
[12] A Survey of Current Approaches for Mapping of Relational Databases to RDF, http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf
[13] Semantic SQL: http://semanticsql.com/
[14] OMG's MetaObject Facility, http://www.omg.org/mof/
[15] MOF QVT, http://www.omg.org/spec/QVT/1.0/
[16] MOLA project, http://mola.mii.lu.lv/
[17] The <AGG> Homepage, http://user.cs.tu-berlin.de/~gragra/agg/
[18] S.Rikacovs, J.Barzdins, Export of Relational Databases to RDF Databases: a Case Study, in P. Forbrig and H. Günther (eds.), *Perspectives in Business Informatics Research*, Springer LNBIP **64** (2010), 203-211.
[19] J.Barzdins, G.Barzdins, R.Balodis, K.Cerans, et.al., Towards Semantic Latvia. In *Communications of 7th International Baltic Conference on Databases and Information Systems* (2006), 203-218.
[20] G.Barzdins, E.Liepins, M.Veilande, M.Zviedris: Semantic Latvia Approach in the Medical Domain. *Proc. 8th International Baltic Conference on Databases and Information Systems*. H.M.Haav, A.Kalja (eds.) Tallinn University of Technology Press (2008), 89-102.
[21] G.Barzdins, S.Rikacovs, M.Veilande, and M.Zviedris, Ontological Re-engineering of Medical Databases, *Proc. of the Latvian Academy of Sciences*, Section B, Vol. **63** (2009), No. 4/5 (663/664), 20–30.
[22] J.Barzdins, G.Barzdins, K.Cerans, R.Liepins, A.Sprogis: UML Style Graphical Notation and Editor for OWL 2, in P. Forbrig and H. Günther (eds.), *Perspectives in Business Informatics Research*, Springer LNBIP 64 (2010), 102-113.
[23] OWLGrEd, http://owlgred.lumii.lv/
[24] Ontology Definition Metamodel. OMG Adopted Specification. Document Number: ptc/2007-09-09, November 2007. http://www.omg.org/docs/ptc/07-09-09.pdf
[25] G.Barzdins, S.Rikacovs, M.Zviedris: Graphical Query Language as SPARQL Frontend. In Grundspenkis, J., Kirikova, M., Manolopoulos, Y., Morzy, T., Novickis, L., Vossen, G. (Eds.), *Proc. of 13th East-European Conference (ADBIS 2009)*, Riga Technical University, Riga (2009), 93–107.
[26] G.Būmans, K.Čerāns, RDB2OWL: Mapping Relational Databases into OWL Ontologies - a Practical Approach. *Databases and information systems: Proc. of the Ninth International Baltic Con-ference, Baltic DB&IS 2010*, Riga, Latvia, July 5-7, 2010. Riga, University of Latvia (2010), 393-408.
[27] G.Būmans, K.Čerāns, RDB2OWL: a Practical Approach for Transforming RDB Data into RDF/OWL, in *Proceedings of the 6th International Conference on Semantic Systems*, Graz, Austria, September 2010, ACM International Conference Proceeding Series, ISBN 9781450300148 (2010) Article No.25.
[28] G.Barzdins, J.Barzdins, K.Cerans: From Databases to Ontologies, *Semantic Web Engineering in the Knowledge Society*; J.Cardoso, M.Lytras (Eds.), IGI Global, (2008), 242-266.
[29] G.Bumans, Mapping between Relational Databases and OWL Ontologies: an Example, *Scientific Papers, University of Latvia*, Vol. **756**, Computer Science and Information Technologies (2010) 99-117.
[30] J.Moden. The "Numbers" or "Tally" Table: What it is and how it replaces a loop. http://www.sqlservercentral.com/articles/T-SQL/62867/