RDB2OWL: A Language and Tool for Database to Ontology Mapping

Kārlis Čerāns[1], Guntars Būmans[1]

karlis.cerans@lumii.lv, guntars.bumans@gmail.com
Institute of Mathematics and Computer Science, University of Latvia
Raina blvd. 29, Riga, LV-1459, Latvia

Abstract. We demonstrate a high-level RDB2OWL language for database-to-ontology mapping specification in a compact notation within the ontology class and property annotations, as well as a corresponding mapping implementation tool that translates the RDB2OWL mappings into executable D2RQ mappings thus allowing to create a semantic SPARQL endpoint containing conceptually organized data from a relational database.

Keywords: Relational databases, OWL ontologies, RDF schemas, Database to ontology mappings.

1 Introduction

Exposing the contents of relational databases (RDB) to semantic web standard RDF [1] and OWL [2] formats enables the integration of the RDB contents into the Linked Data [3] and Semantic web [4] information landscape. An important benefit of RDB-to-RDF/OWL mapping is also the possibility of creating a conceptual model of the relational database data on the level of RDF Schema/OWL and further on accessing the RDB contents from the created semantic/conceptual model perspective. Since the RDB-to-RDF/OWL mapping connects the technical data structure (the existing RDB schema) with the conceptual one, it would typically have not a one-to-one data structure reproduction, but would rather involve some data transformation means.

The technical task of mapping relational databases to RDF/OWL formats is nowadays well understood and widely studied, just to mention D2RQ [5], Virtuoso RDF Graphs [6], Ultrawrap [7], Spyder [8] and ontop [9] among different RDB-to-RDF/OWL mapping languages and tools, together with W3C standard R2RML [10]. The languages of most RDB-to-RDF/OWL mapping approaches offer conceptually clear mapping structure with less attention paid, however, to the concise mapping writing possibility suitable both for manual mapping information creation as well as for using the mapping information as semantic-level documentation of the RDB structure. We review here a slightly simplified variant of the RDB2OWL mapping

[1] Partially supported by Latvian State Research programme NexIT project No.1 'Technologies of ontologies, semantic web and security'

language [11,12,13] that allows creating RDB-to-RDF/OWL mapping specifications in a compact textual form with mapping descriptions placed in the annotations of the target ontology entities. The RDB2OWL language allows re-using both the target ontology and source database structure information within the mapping definition, as well as introducing user defined functions.

The central novel point of this demonstration is a tool implementing the RDB2OWL notation[2] via converting a RDB2OWL mapping into an executable D2RQ mapping that can further on be used by the D2RQ platform either to produce the RDF dump of the source RDB, or to turn it into a SPARQL endpoint. Alternative RDB2OWL implementations sharing the inner RDB2OWL mapping representation with the translation into D2RQ are underway.

A particular usage scenario for RDB2OWL mappings is a semantic re-engineering of existing relational databases (cf. e.g. [14,15]) aiming at existing data access from the data ontology conceptual model perspective; this scenario has motivated most of the mapping constructions built into RDB2OWL. The tool has been used to re-work the Latvian medicine registries example [14] into a maintainable mapping solution (the original mapping solution of [14] involved creating custom SQL scripts just for the concrete database mapping; this approach allowed to receive the first principal proof-of-concept results, however, it did not appear practical when the database schema as well as mapping definition adjustments were to be involved).

Note that a textual RDB2OWL mapping specification can be used also as a reference for a (legacy) RDB by documenting the structure of the data contained therein.

2 RDB2OWL Mapping Language

A RDB2OWL mapping defines the correspondence between a relational database and an OWL ontology or RDF schema. The mapping information consists of elementary mappings or maps that are recorded in annotation assertions for ontology entities[3] - class maps for ontology classes, data property maps and object property maps for ontology data properties and object properties respectively.

The general pattern of class map as well as object and data property map definition follow the common sense structural principle of mapping ontology classes to database tables, data properties to table columns and object properties to links between tables.

More precisely, a class map in RDB2OWL is characterized by a table expression, possibly involving a join of several tables and a filter, and a pattern for instance resource URI construction from a table expression row. A class map can be attached to a single ontology/schema class, meaning that it describes instances for this class.

An object property map contains references to its subject (source) and object (target) class maps in the property map's (extended) table expression structure; such references bring the class map table expressions as components into the property map's table expression and provide the patterns of the property triple subject resp.

[2] The tool is freely available for download from `http://rdb2owl.lumii.lv/`

[3] We use `rdb2owl:DBExpr` as the annotation property, where `rdb2owl:=<http://rdb2owl.lumii.lv/2012/1.0/rdb2owl#>`

object URI generation from the property map's table expression row. An object property map is always attached to a single object property within the ontology.

Similarly, a data property map is characterized by a corresponding (extended) table expression that contains a reference to the property map's subject (source) class map, and a value expression describing the computation of the property literal value that corresponds to the URI, as computed by the subject class map's URI pattern. A data property map is always attached to a single data property within the ontology.

The textual form of RDB2OWL maps relies on textual presentations of table expressions, instance URI generation patterns and expression values; it is optimized to take into account the available ontology and RDB schema context information.

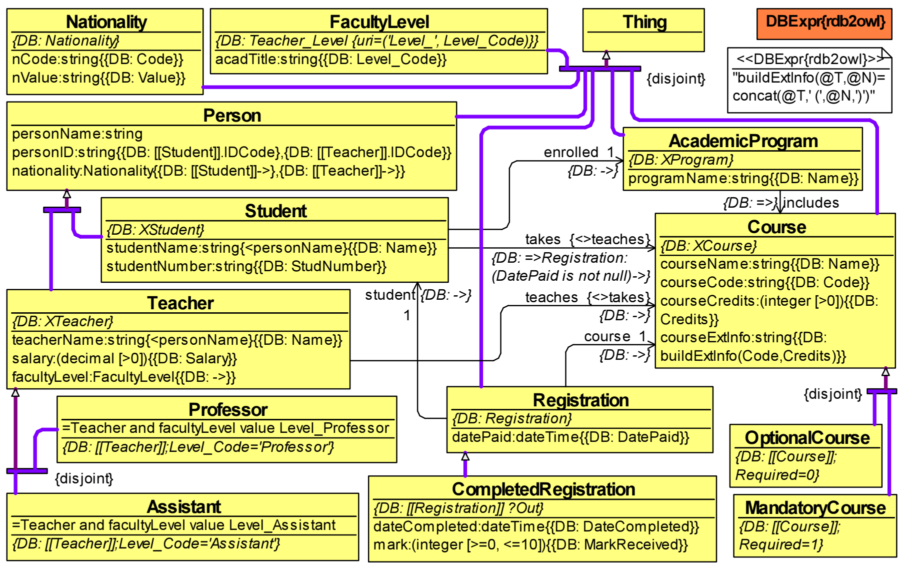
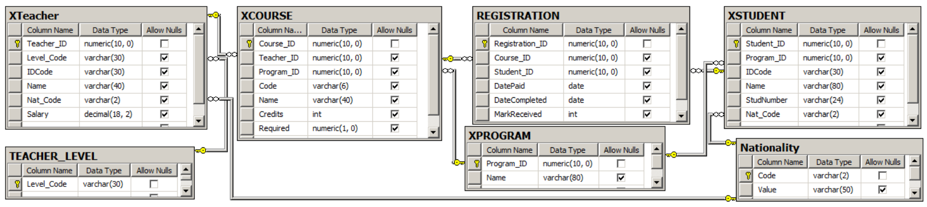

Nationality
{DB: Nationality}
nCode:string{{DB: Code}}
nValue:string{{DB: Value}}
FacultyLevel
{DB: Teacher_Level {uri=('Level_', Level_Code)}}
acadTitle:string{{DB: Level_Code}}
Thing
{disjoint}
DBExpr{rdb2owl}
<<DBExpr{rdb2owl}>>
"buildExtInfo(@T,@N)=
concat(@T,' (',@N,')')"
Person
personName:string
personID:string{{DB: [[Student]].IDCode},{DB: [[Teacher]].IDCode}}
nationality:Nationality{{DB: [[Student]]->},{DB: [[Teacher]]->}}
AcademicProgram
{DB: XProgram}
programName:string{{DB: Name}}
enrolled 1
{DB: ->}
{DB: =>}
includes
Student
{DB: XStudent}
studentName:string{<personName}{{DB: Name}}
studentNumber:string{{DB: StudNumber}}
takes {<>teaches}
{DB: =>Registration:
(DatePaid is not null)->}
Course
{DB: XCourse}
courseName:string{{DB: Name}}
courseCode:string{{DB: Code}}
courseCredits:(integer [>0]){{DB: Credits}}
courseExtInfo:string{{DB: buildExtInfo(Code,Credits)}}
Teacher
{DB: XTeacher}
teacherName:string{<personName}{{DB: Name}}
salary:(decimal [>0]){{DB: Salary}}
facultyLevel:FacultyLevel{{DB: ->}}
student
{DB: ->}
1
teaches {<>takes}
{DB: ->}
course 1
{DB: ->}
Registration
{DB: Registration}
datePaid:dateTime{{DB: DatePaid}}
{disjoint}
Professor
=Teacher and facultyLevel value Level_Professor
{DB: [[Teacher]];Level_Code='Professor'}
{disjoint}
OptionalCourse
{DB: [[Course]];
Required=0}
MandatoryCourse
{DB: [[Course]];
Required=1}
Assistant
=Teacher and facultyLevel value Level_Assistant
{DB: [[Teacher]];Level_Code='Assistant'}
CompletedRegistration
{DB: [[Registration]] ?Out}
dateCompleted:dateTime{{DB: DateCompleted}}
mark:(integer [>=0, <=10]){{DB: MarkReceived}}

Fig. 1. Mini-university ontology with full RDB2OWL mapping annotations
The '{DB: …}' notation is used for annotation assertions with rdb2owl:DBExpr property

In what follows, we outline the concrete syntax of RDB2OWL maps. Figure 1 contains an illustrative example of a mini-University OWL ontology in OWLGrEd[4] ontology editor notation [16,17] (e.g. showing OWL classes as UML classes, OWL object and data properties as roles on UML associations between property domain and range classes and attributes of property domain classes)[5]. The ontology in Figure 1 contains both its structure definition (the classes and properties) and further OWL constructs (e.g. class expressions and datatype facet restrictions); it is enriched by a custom *{DB: <annotation_value>}* notation for the database connection annotations (cf. [18]), as well. The corresponding RDB schema is outlined in Figure 2.

[4] `http://owlgred.lumii.lv/`

[5] The ontology with annotations in a standard OWL RDF/XML format is available at `http://rdb2owl.lumii.lv/demo/UnivExample.owl`



XTeacher

	Column Name	Data Type	Allow Nulls
🔑	Teacher_ID	numeric(10, 0)	☐
	Level_Code	varchar(30)	✓
	IDCode	varchar(30)	✓
	Name	varchar(40)	✓
	Nat_Code	varchar(2)	✓
	Salary	decimal(18, 2)	✓

XCOURSE

	Column Na...	Data Type	Allow Nulls
🔑	Course_ID	numeric(10, 0)	☐
	Teacher_ID	numeric(10, 0)	✓
	Program_ID	numeric(10, 0)	✓
	Code	varchar(6)	✓
	Name	varchar(40)	✓
	Credits	int	✓
	Required	numeric(1, 0)	✓

REGISTRATION

	Column Name	Data Type	Allow Nulls
🔑	Registration_ID	numeric(10, 0)	☐
	Course_ID	numeric(10, 0)	✓
	Student_ID	numeric(10, 0)	✓
	DatePaid	date	✓
	DateCompleted	date	✓
	MarkReceived	int	✓

XSTUDENT

	Column Name	Data Type	Allow Nulls
🔑	Student_ID	numeric(10, 0)	☐
	Program_ID	numeric(10, 0)	✓
	IDCode	varchar(30)	✓
	Name	varchar(80)	✓
	StudNumber	varchar(24)	✓
	Nat_Code	varchar(2)	✓

TEACHER_LEVEL

	Column Name	Data Type	Allow Nulls
🔑	Level_Code	varchar(30)	☐

XPROGRAM

	Column Name	Data Type	Allow Nulls
🔑	Program_ID	numeric(10, 0)	☐
	Name	varchar(80)	✓

Nationality

	Column Name	Data Type	Allow Nulls
🔑	Code	varchar(2)	☐
	Value	varchar(50)	✓

Fig. 2. A mini-University RDB schema

A table expression in RDB2OWL in the simplest and most common case is just a table name, such as *XTeacher*, or *Teacher_Level*; a filter expression can be added to a table expression, using a semicolon, such as '*XCourse;Required=1*'. The table expressions involving several tables can be formed either in

(a) item list notation with comma-separated, possibly alias-labeled table expressions, such as '*XTeacher T, XCourse C; T.Teacher_ID=C.Teacher_ID*', or
(b) navigation list notation, such as '*XTeacher[Teacher_ID]->[Teacher_ID] XCourse*' that can be shortened to '*XTeacher->XCourse*' by omitting navigation columns from table sole FKs to the matching PKs; the symbol => denotes PK-to-sole FK navigation, e.g. '*XProgram=>XCourse*', or
(c) notation combining both (a) and (b), where the whole navigation list can be regarded as a single item in the item list.

There is a further possibility of local filter introduction in navigation expressions, such as *XStudent=>Registration:(DatePaid is not null)->XCourse*.

The textual syntax for a class map consists of table expression and URI pattern specification, such as '*Teacher_Level {uri=('Level_', Level_Code)}*'; the URI pattern is expected to have a list of constant strings and column expressions whose values are concatenated to give the local name of the corresponding ontology class instance. The URI pattern can be omitted, if it is formed just 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, or by the name of the class for which it is the sole class map (e.g. *[[Teacher]]*).

The object property map is described as an extended table expression where two sub-expressions denote its subject (source) and object (target) class maps respectively. These sub-expressions can be either:

(a) explicitly marked by an alias <s> or <t>, respectively, in some place within the table expression's structure,
(b) defined by the mark <s> or <t> as a table expression item itself; 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, or
(c) in the case of missing explicit <s> and/or <t> marks these marks are assumed <s> for the leftmost and <t> for rightmost item within the table expression.

These conventions allow denoting e.g. the object property map simply by '->', if the property corresponds to the sole FK-to-PK mapping between the tables serving as table expressions in the sole class maps of property domain and range classes.

The data property map can be described by a column name/column expression that is to be evaluated in the context of the sole class map attached to the property domain class. Alternatively, <*table_expression*>.<*value_expression*> notation can be used for data property map description, where the table expression is required to contain either explicit or implicit reference <*s*> to the property map's subject class map.

The example in Figure 1 contains also an illustration of a function definition using an ontology level *rdb2owl:DBExpr* annotation, the defined function *buildExtInfo* is then used in a *courseExtInfo* property value expression for the *Course* class. The RDB2OWL functions can be used for repeating mapping fragments, as well as for increasing mapping readability in the case of complex mapping constructions. The function body can introduce also additional table expressions into the context of evaluating its value expression; such table expressions are joined to the table expression defining the context of evaluation of the function-calling value expression.

The decoration *?Out* in the database annotation for *CompletedRegistration* class means relating to this class only those rows of the described class map table expression that contain a value in at least one property whose domain is this class.

A more detailed RDB2OWL language design description can be found in [11] (the core concepts) and [12] (advanced constructs), as well as [13].

3 Mapping Implementation in RDB2OWL Tool

The RDB2OWL mapping tool[6] takes as the input annotated OWL ontology as well as a connection to the source database (in order to read the source database schema) and it produces a D2RQ mapping for accessing the source database via D2RQ platform[7] SPARQL endpoint or RDF dump tool. The database connection information (JDBC driver name and connection information) can be defined as an annotation in the data ontology, or it can be entered directly in the RDB2OWL tool.

The RDB2OWL mapping processing in the tool follows a number of steps activated either one-by-one or using a batch command "Create mapping from ontology":

- Load ontology: the ontology with the attached RDB2OWL annotations is loaded into the internal RDB2OWL mapping model [13]; the parsing of the annotations into the RDB2OWL model items is performed.
- Load source database schemas into the RDB2OWL model.
- Finalize abstract mapping: the advanced mapping constructs (e.g. named class maps, shorthand notation, defaults and user defined functions) are resolved into basic mapping constructions thus transforming the mapping into the semantic RDB2OWL model [13] outlined here in Figure 3.
- Generate D2RQ Mapping from the created semantic RDB2OWL model.

Although the navigation item and link construction can be translated into more basic reference item notation for table expressions corresponding to Raw RDB2OWL metamodel [11], both notations are retained in the RDB2OWL semantic model and the D2RQ mapping generation is performed directly from the notation involving the navigation items (corresponding to RDB2OWL Core MM of [11]). The model of Figure 3 is to be re-used also in upcoming alternative RDB2OWL implementations.

[6] http://rdb2owl.lumii.lv/; see this paper's example at http://rdb2owl.lumii.lv/demo/
[7] http://d2rq.org/

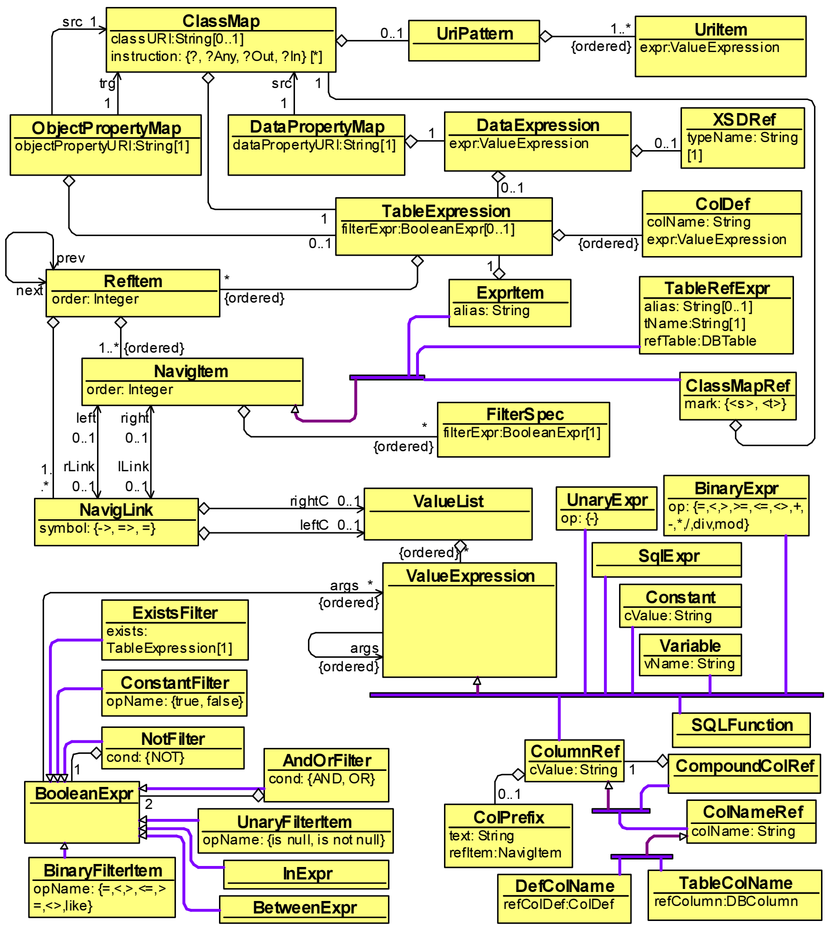


Fig. 3. Essential fragments of RDB2OWL semantic metamodel

An important parameter for the mapping generation is the "with inference" option that can be turned on or off by the user. We explain this parameter using an example. If an ontology class, say, *Student*, is said to correspond to a database table *XStudent*, one naturally expects that its superclass (e.g. *Person*) also automatically receives the instances coming from the *XStudent* table. If the RDF triples corresponding to the database are dumped and loaded into a RDF triple store (e.g. Virtuoso Server), the RDF triple store can perform the inference. If, however, the RDB data are to be accessed by D2R server on on-the-fly basis, this inference is built into the D2RQ mapping generation by the RDB2OWL tool so that together with the D2RQ class map for the *Student* class, a similar class map is generated also for the *Person* class (an alternative would be to extend the D2R server functionality with some inference capabilities). The RDB2OWL tool for D2RQ mapping generation currently supports subclass, subproperty and inverse properties inference. Similar inference capabilities are planned also for upcoming R2RML implementation.

The RDB2OWL tool is implemented in JAVA programming language, however, its current implementation uses libraries for the internal RDB2OWL model handling and transformations that are working only in MS Windows environment. The model transformations from RDB2OWL syntactic to semantic metamodel are written in LuA programming language using its lQuery model transformation library [19].

4 Discussion and Conclusions

The RDB2OWL mapping language and tool allows easy creation of database-to-ontology mappings involving basic one-to-one correspondence between ontology classes, data and object properties and database tables, columns and relations respectively, as well as involving more advanced constructs (e.g. adding filters, sub-class relations, value processing, etc.). The experience shows that for simple to medium size ontologies and corresponding databases the RDB2OWL language and tool, in combination with D2RQ Platform [5] and possibly an RDF triple store (in the case of initial database RDF dump created by the D2RQ server), allow to quickly obtain a SPARQL endpoint over conceptually structured view over RDB data up and running. The method is well applicable also if only a part on information from a larger database is to be exposed into the semantic form.

The practical experience of using RDB2OWL over larger ontologies and database schemas, as in Latvian medicine registries example of [14] with the ontology containing 173 classes, 219 object properties and 816 data properties, shows that the tool is well usable, however, some further ontology and mapping engineering means for cross-checking the validity and completeness of the created mappings would be helpful (these can be used both on the RDB2OWL annotation level, as well as on the level of the generated executable mapping code, e.g. in D2RQ).

The direct RDB2OWL mapping text size both in the considered mini-University and in the Latvian medicine registries [14] examples is about 8-9 times smaller, if compared with the generated D2RQ mapping text size, thus allowing considering RDB2OWL for manual database-to-ontology correspondence specification.

Regarding the OWL constructions allowed in the ontologies that are RDB2OWL mapping targets, we note that in fact, any OWL features also beyond the RDF Schema notation can be permitted. Figure 1 contains a number of such features, for instance equivalent class assertions and datatype facet restrictions. We can note that the equivalent class assertions for the subclasses of the *Teacher* class are “enforced” by the RDB2OWL mapping definition; while the facet restrictions in the current mapping version are left to be checked as constraints over the obtained RDF triple set.

Although the RDB2OWL tool has a well-defined task of creating a conceptual-level SPARQL endpoint for a relational database, permitting its standalone use, a particular usage context of the RDB2OWL database-to-ontology mapping tool is within a larger semantic database platform [15] involving also the OWLGrEd ontology editor [16,17], the visual custom SPARQL query generation tool ViziQuer [20] and the SPARQL endpoint browser OBIS [21].

References

1. Resource Description Framework (RDF), http://www.w3.org/RDF/
2. Motik, B; Patel-Schneider P.F; Parsia B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, 2009
3. Linked Data, http://linkeddata.org
4. Tim Berners-Lee, James Hendler and Ora Lassila, "The Semantic Web", Scientific American, May 2001, p. 29-37.
5. D2RQ Platform. Treating Non-RDF Relational Databases as Virtual RDF Graphs. http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/spec/
6. C.Blakeley: "RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping)", OpenLink Software, 2007.
7. 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)
8. Spyder tool, URL: http://www.revelytix.com/content/spyder
9. Bagosi, T., Calvanese, D., Hardi, J., Komla-Ebri, S., Lanti, D., Rezk, M., Rodriguez-Muro, M., Slusnys, M., & Xiao, G. (2014). The Ontop framework for ontology based data access. In Zhao, D., Du, J., Wang, H., Wang, P., Ji, D., & Pan, J. Z. (Eds.), CSWS 2014, Vol. 480 of Communications in Computer and Information Science, pp. 67-77. Springer.
10. R2RML: RDB to RDF Mapping Language, http://www.w3.org/TR/r2rml/
11. K.Čerāns, G.Būmans, RDB2OWL: a RDB-to-RDF/OWL Mapping Specification Language // J.Barzdins and M.Kirikova (eds.), Databases and Information Systems VI, IOS Press 2011, p.139-152.
12. G.Būmans, K.Čerāns, Advanced RDB-to-RDF/OWL mapping facilities in RDB2OWL // Proc. of BIR 2011, Riga, Latvia, October 7-8, 2011. LNBIP 90, pp. 142-157. Springer, Heidelberg, 2011 (ISBN:978-3-642-24510-7
13. G.Būmans, Relational database information availability to semantic Web technologies, Dr.sc.comp theses, University of Latvia, 2012, https://dspace.lu.lv/dspace/handle/7/2538.
14. 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.), TUT Press, pp. 89-102. (2008).
15. K. Cerans, G. Barzdins, G. Bumans, J. Ovcinnikova, S. Rikacovs, A. Romane and M. Zviedris. A Relational Database Semantic Re-Engineering Technology and Tools // Baltic Journal of Modern Computing (BJMC), Vol. 3 (2014), No. 3, pp. 183-198.
16. Barzdins, J.; Barzdins, G.; Cerans, K.; Liepins, R.; Sprogis, A.: OWLGrEd: a UML Style Graphical Notation and Editor for OWL 2. In Proc. of OWLED 2010, 2010.
17. Barzdins, J.; Cerans, K.; Liepins, R.; Sprogis, A.: UML Style Graphical Notation and Editor for OWL 2. In Proc. of BIR'2010, LNBIP, Springer 2010, vol. 64, p. 102-113.
18. K.Čerāns, J.Ovčiņņikova, R.Liepiņš, A.Sproģis, Advanced OWL 2.0 Ontology Visualization in OWLGrEd // A.Caplinskas, G.Dzemyda, A.Lupeikiene, O.Vasilecas (eds.), Databases and Information Systems VII, IOS Press, Frontiers in Artificial Intelligence and Applications, Vol 249, 2013, pp.41-54.
19. Liepiņš, R. Library for Model Querying – lQuery. In Proceedings of 2012 Workshop on OCL and Textual Modelling; 2012.
20. Zviedris M., Barzdins G., ViziQuer: A Tool to Explore and Query SPARQL Endpoints // The Semantic Web: Research and Applications, LNCS Volume 6644, 2011, pp. 441-445
21. M.Zviedris, A.Romane, G.Barzdins, K.Cerans. Ontology-Based Information System // Proceedings of JIST'2013, LNCS Volume 8388, 2014, pp. 33-47