

A Relational Database Semantic Re-Engineering Technology and Tools

Kārlis ČERĀNS^{1,2}, Guntis BĀRZDIŅŠ, Guntars BŪMANS²,
Jūlija OVČIŅNIKOVA², Sergejs RIKAČOVŠ, Aiga ROMĀNE²,
Mārtiņš ZVIEDRIS

Institute of Mathematics and Computer Science, University of Latvia
Raina blvd. 29, Riga, LV-1459, Latvia

karlis.cerans@lumii.lv, guntis.barzdins@lumii.lv,
guntars.bumans@gmail.com, juliija.ovcinnikova@lumii.lv,
sergejs.rikacovs@lumii.lv, aiga.romane@inbox.lv,
martins.zviedris@gmail.com

Abstract. We describe a relational database semantic re-engineering technology and the tools that are available for its implementation. The semantic re-engineering technological process starts with creating a conceptual data ontology together with its annotations for database schema and user interface mappings. The database schema mappings are implemented in RDB2OWL and D2RQ server thus creating a SPARQL-endpoint for “semantic” access to the relational database contents. The SPARQL endpoint can be explored by the user interface automatically generated by OBIS system, or ViziQuer tool may be used for custom SPARQL query generation in a graphical way. We report on successful application of the approach on the Latvian medical data with the ontology containing 172 OWL classes, 138 object properties, 814 data properties, and about 40 million data level RDF triples.

Keywords. relational database re-engineering, OWL ontologies, database to ontology mappings, user interface generation

1. Introduction

The relational databases (RDB) are widely used as a main data storage platform in numerous state institutions and agencies, companies and research institutes. The RDB data are successfully accessed and modified via the client applications that are created by programmers and that foresee specific patterns of their usage. There are means of direct accessing of RDB also outside the defined patterns (e.g. for various statistical or other ad-hoc queries), using the standard SQL query language, however, this is a way too low level for a domain expert without specific IT knowledge. The semantic technologies, based on RDF (WEB, f), RDFS (WEB, e), SPARQL (WEB, i; WEB, j) and OWL (Motik et al., 2009) offer a much higher-level view on the contained data, thus increasing the hope of more direct participation of domain experts in the data exploration

¹Partially supported by ESF project 2013/0005/1DP/1.1.1.2.0/13/APIA/VIAA/049

²Partially supported by ERAF project 2014/0020/2DP/2.1.1.1.0/14/APIA/VIAA/072

and data management (see e.g. (WEB, h) for comparison of RDB/SQL, XML and RDF/SPARQL technologies). There is also an important possibility of using semantically structured data (i.e. the data in RDF/OWL format) on the web scale within the Semantic Web (Berners-Lee et al., 2001) and Linked Data (WEB, c) developments.

We describe here a methodology and an integrated tool chain covering all steps of integrating relational database data into the semantic technology information landscape.

The “embedding” of a relational database within a semantic data framework involves creation of a data ontology (usually, as an OWL ontology or RDFS schema) that contains a data structure description from the semantic perspective. While there exist approaches of automated creation of data ontology that corresponds to the given RDB schema (see e.g. (De Laborda and Conrad, 2006) and the W3C standard (Arenas et al., 2012)), our focus is on explicit creation of data ontology in parallel with the RDB schema. Such data ontology with its conceptual-oriented design will be able to (i) reflect the hierarchical structure of the model using sub-class hierarchy, (ii) use conceptually named class and relation entities, and (iii) introduce semantic concepts and relations that are indirectly mapped to the elements of the concrete database schema. The relational databases (often legacy databases), as they are owned by different data holding subjects, most likely will need at least some semantic re-structuring before they can be fully exploited on the semantic level by domain experts that are not IT professionals. This approach of parallel data ontology creation has been advocated also in the vision-setting “Semantic Latvia” paper (Barzdins et al., 2006) and has been further elaborated in e.g. (Barzdins et al., 2008a; Barzdins et al., 2008b).

A number of RDB-to-RDF/OWL mapping languages and tools exist that support defining and execution of indirect mappings between relational database and ontology formats. One can see e.g. D2RQ (WEB, b), Virtuoso RDF Graphs (Blakeley, 2007), Ultrawrap (Sequeda et al., 2009) and Spyder (WEB, k) among others. The W3C has established a R2RML standard (WEB, g) for the database-to-relational mappings that is supported by a large number of these tools. These notations and tools allow, given an RDB, to create a (real or virtual) RDF-format database that is able to communicate with its user in a form of SPARQL endpoint whose structure is not directly replicating the original RDB structure. While these approaches and tools are oriented towards well-structured computer-readable mapping specifications, they pay less attention to the human readability and the human write-ability of the mapping definitions (e.g. a mapping definition can be lengthy, with repeating standard text fragments). Our approach here is to use RDB2OWL (Cerans and Bumans, 2011; Bumans and Cerans, 2011) mapping language that offers a more compact form of database-to-ontology entity mapping specification that is well integrated within the data ontology structure (the RDB2OWL mapping specifications are placed within the data ontology annotations). The graphical OWL ontology editor OWLGrEd (Barzdins et al., 2010a; Barzdins et al., 2010b) whose usage we recommend within the database semantic re-engineering process contains means for smooth integration of RDB2OWL annotations.

We offer an implementation of the RDB2OWL mappings via their translation into a more technical D2RQ mapping format that is further on handled by a D2RQ server either in the batch processing or on-the-fly query processing mode.

The RDB semantic re-engineering tool chain further on involves a browser for the SPARQL endpoint reflecting the RDB structure; we offer here using OBIS tool (Zviedris et al., 2013) that has been meant initially for creating information systems powered with a SPARQL endpoint back-end and is re-used here for browsing of data stores corresponding to RDB data. We include in the RDB semantic re-engineering tool

chain also ViziQuer tool (Barzdins et al., 2008b; Zviedris et al., 2013; Zviedris and Barzdins, 2011) providing a graphical interface of SPARQL query generation.

Most of the tools considered here have been described in earlier papers. The contribution of this paper is to describe the OWLGrEd, RDB2OWL, OBIS and ViziQuer tools as parts of an integrated tool chain, oriented towards relational database semantic re-engineering task. The availability of implementation for the RDB2OWL mapping specification language and ViziQuer and OBIS tools at a level that is sufficient for practical data mapping examples also is novel in this paper.

In the rest of the paper Section 2 reviews the OWLGrEd ontology editor together with its means for custom annotation specification; Section 3 describes RDB2OWL, together with principles of introducing inferred knowledge into mapped databases. Further on, Section 4 describes OBIS and Section 5 outlines ViziQuer. Finally, Section 6 describes the re-engineering of the Latvian medical database (Barzdins et al., 2008b) and Section 7 concludes the paper.

2. Data ontology creation: OWLGrEd

The first step in database semantic re-engineering is to create an ontology that reflects, on a conceptual level, the data that are contained in the relational database. The data ontology creator may choose either to create the ontology corresponding to all data that are in the database, or just to cover a part of them – the data that are interesting or meaningful in the context of a given usage/application/demonstration. It could be possible to offer also conceptualizations of the given data set corresponding to the different viewpoints levels of detail that the users may want to use when looking at the data. The conceptual data ontology itself is defined in Web ontology language OWL (Motik et al., 2009) that is a nowadays widely accepted standard for data semantic representation.

There is a variety of editors that allow defining OWL ontologies, including Protégé (WEB, d), TopBraid Composer (WEB, l), or even a structured text editor (since OWL ontologies do have a variety of ways of textual representation). We consider here the OWLGrEd³ ontology editor (Barzdins et al., 2010a; Barzdins et al., 2010b) that

- (i) allows rendering and editing ontologies in an intuitive yet compact graphical notation that combines UML (WEB, m; WEB, n) class diagram style graphics and textual OWL Manchester syntax (Horridge and Peter, 2012), and
- (ii) provides a plugin mechanism that can be used to add specific support for data connection and user interface generation annotation manipulation (cf. (Cerans et al., 2013)).

Fig. 1 contains an example mini-University ontology presentation in standard OWLGrEd notation extended by means for enumerated class (classifier) denotation.

OWLGrEd visualizes and allows editing of OWL classes as UML classes (e.g. *Student*, *Person*, *Course*), OWL object properties as roles on associations between the domain and range classes of the property (e.g. *teaches*, *takes*, *enrolled*) and OWL data properties as attributes of property domain classes. The UML notation is also re-used to model sub-class and inverse properties notions, as well as cardinality constraints.

³ <http://owlgred.lumii.lv/>

There are more advanced OWL 2.0 constructs expressed in OWLGrEd using the textual OWL Manchester Syntax form, e.g. class expressions in axioms (e.g. for *Professor* class), sub-properties (e.g. *studentName*<*personName*), disjoint properties (e.g. *teaches*<>*takes*), property chains (e.g. *takes*> inverse(*student*) *ocourse*) and datatype facet restrictions (e.g. for properties *salary* and *mark*).

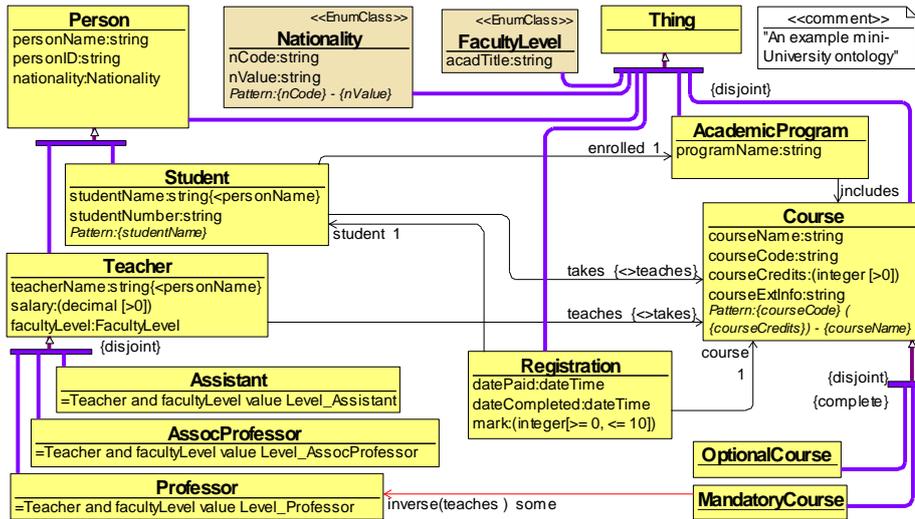


Fig. 1. Example: the OWLGrEd notation for a mini-university ontology

We refer the reader to (Barzdins et al., 2010a; Barzdins et al., 2010b) for a more detailed and thorough explanation of the OWLGrEd notation and editor functionality, just note here that OWLGrEd allows saving the created ontology in accordance to standard OWL ontology notations, as well as it is able to read a textual ontology notation and produce automatically a visual diagram for it.

In order to accommodate the use of OWLGrEd in the RDB semantic re-engineering process two custom annotation profiles have been created that include both the custom user fields and their support scripts:

- *DBExpr* profile for database correspondence annotations (including code-completion functionality), and
- *OBIS* profile for user interface annotations.

Each OWLGrEd annotation profile offers notational and editing services for assertions corresponding to specific annotation properties, still the created (exported) ontology will be in full conformance with OWL 2.0 syntax that foresees the possibility of user defined annotation properties⁴. So, for example, two of the annotations in the OBIS profile are *obis:isEnumerated*⁵ and *obis:textPattern*; the example in Fig. 1 shows the visual effects of the corresponding annotation assertions for the *Nationality* and *FacultyLevel* classes. The concrete ontology annotations that are available in the *DBExpr*

⁴The OWLGrEd editor contains also generic means for ontology annotation handling.

⁵The prefix *obis*: corresponds to the namespace <http://obis.lumii.lv/2013/01/obis#>.

and *OBIS* profiles are described, together with their semantics, in Section 3 and Section 4, respectively. Note that the *OBIS* profile induces a semantic change into the OWLGrEd class attribute presentations by asserting the 0..1 default cardinality for all attributes whose cardinality is not explicitly specified (in OWLGrEd no implicit cardinality restrictions are imposed what, in effect, means 0..* default cardinality also for attributes).

3. Database correspondence: RDB2OWL

Regarding a conceptual-level OWL ontology as a “semantically re-engineered” view on a RDB data that are organized along the structure defined by its schema requires specifying a mapping between the database schema and the corresponding OWL ontology (or, RDF schema). Following the general discussion in the Introduction, we concentrate here on the RDB2OWL mapping specification language for describing the RDB-to-ontology correspondence.

3.1. RDB2OWL mapping definition and implementation

A RDB2OWL mapping maps ontology classes to database tables (or rowset-valued expressions), data properties to table columns (or column expressions) and object properties to table relations (e.g. foreign-to-primary key relations). What makes RDB2OWL distinctive among other RDB-to-RDF/OWL correspondence approaches is (i) the concrete expression language for mapping specification, as well as (ii) the principle of placing RDB2OWL mapping expressions within the annotations of classes, properties and individuals of the ontology.

Fig. 2 contains an example database schema for a mini-University and Fig. 3 contains a slight variation of the mini-University ontology from Fig. 1, together with the correspondence annotations in RDB2OWL notation. Visually, the Fig. 3 uses the OWLGrEd extension with the DBExpr profile, announced in Section 2; the DB correspondence annotations for classes, attributes and roles are placed within ‘*{DB: <annotation_text>}*’ notation. Semantically, *rdb2owl:DBExpr*⁶ annotation property is used for these annotations in the OWL notation.

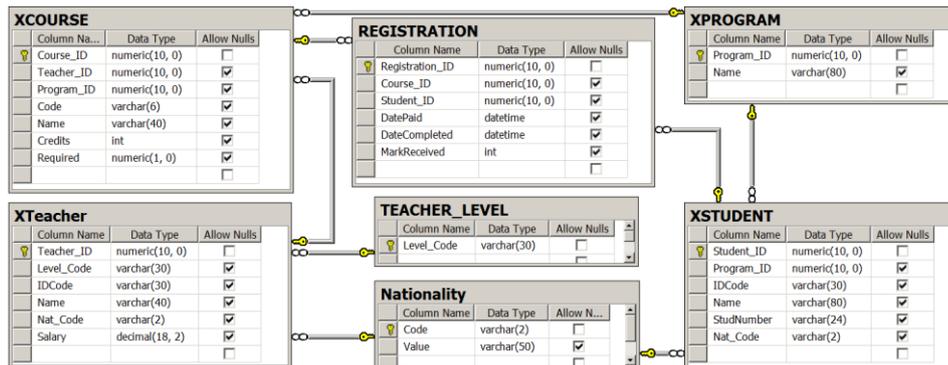


Fig. 2. A mini-University RDB schema

⁶ *rdb2owl*:=<<http://rdb2owl.lumii.lv/2012/1.0/rdb2owl#>>

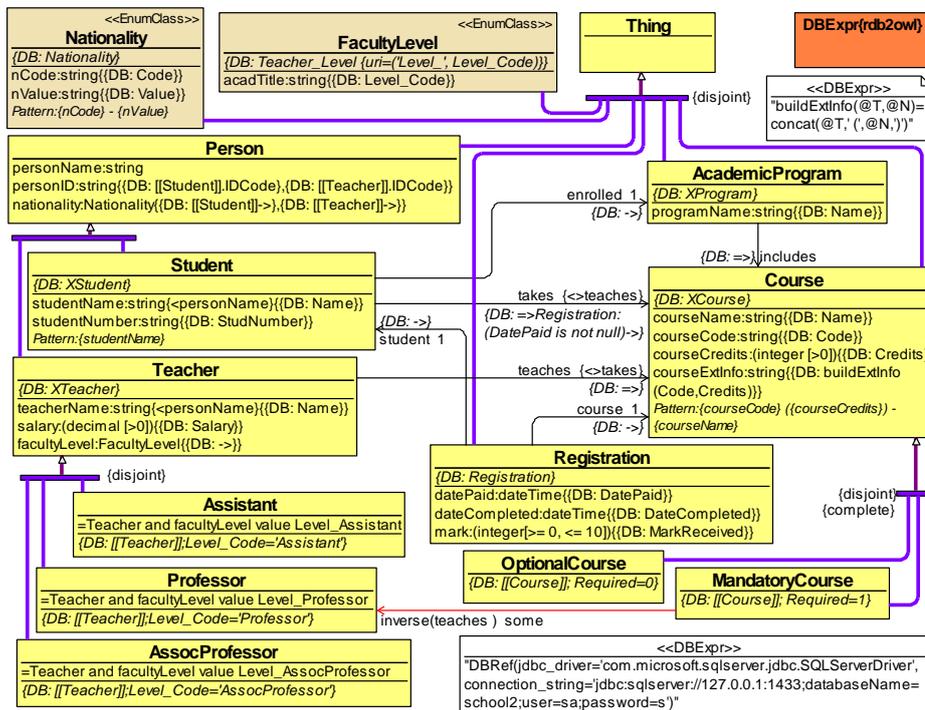


Fig. 3. Mini-university ontology with RDB2OWL annotations

As it can be observed in Fig. 3, the DBExpr-annotations can be attached to:

- ontology classes (e.g. `{DB:XStudent}`, `{DB:Registration}`),
- attributes (e.g. `{{DB: Name}}`, `{{DB:DatePaid}}`) and
- roles (e.g. `{DB: ->}` meaning a link between classes on the basis of foreign-to-primary key correspondence between the tables these classes are based on, or `{DB: =>Registration:(DatePaid is not null)->}` that involves an intermediary table within the corresponding class table link together with a filter placed on the rows contained therein).

The annotation `[[Course]];Required=0` attached to the *OptionalCourse* class specifies a reference to the class-to-table correspondence, as described in the class *Course*, with an additional filter introduced on top of it. The annotation for the *FacultyLevel* class illustrates the possibility of creating explicit instance URI patterns for the class' entities based on the data in DB table rows. More detailed description of RDB2OWL mapping construction is available in (Cerans and Bumans, 2011; Bumans and Cerans, 2011).

The current implementation of RDB2OWL⁷ is via its mapping translation into the mapping definitions for D2RQ server (WEB, a) that is further on capable either to generate the dump of the source RDB in a form of RDF triples, or to offer a SPARQL endpoint for RDB access via SPARQL queries, using on-the-fly SPARQL-to-SQL rewriting. There is also available an experimental direct generation of SQL-to-RDF

⁷<http://rdb2owl.lumii.lv/>

transformation, along the lines of (Bumans and Cerans, 2010). It is assumed that the RDF triples, generated either by D2RQs RDF dump or by the direct method, would be stored into a RDF triple store, such as e.g. Virtuoso (Blakeley, 2007) that would then provide a SPARQL endpoint for accessing these data. Note that, as most of RDB-to-RDF/OWL mapping solutions, RDB2OWL both in the “offline” and “on-the-fly” modes offers read-only access to the source RDB data. This does not preclude, however, creating the data stored on other media that link to the RDF triples containing the exposed RDB data.

3.2. Advanced ontology assertions in mapped databases

OWL 2.0 ontology language admits a rich set of ontology specification features including restrictions and derived class specifications, as well as cardinalities and class/property disjointness assertions (most of these constructs are present in the ontology of Fig. 1, some less are also in Fig. 3). These advanced ontology axiom forms cannot be directly taken into account in the RDB-to-ontology mapping process and have to be accounted for separately.

What a RDB2OWL mapping allows is direct RDF triple generation in the form of class assertions or data or object property assertions of concrete instances. The assertions (axiom forms) in the ontology that go beyond that could be used both/either:

- to complement the created RDF triple set with the knowledge that can be *inferred* from them (this is the standard “open world” semantics for OWL; e.g. the subclass assertions in the ontology would be naturally viewed this way), and
- to view them as *integrity constraints* that are expected to hold on the generated RDF triple set, extended with the inferred knowledge (the “closed world”, or integrity constraint semantics of OWL axioms, following (Tao et al., 2010); e.g. the cardinality constraints would naturally have this semantics⁸).

As shown in (Cerans et al., 2012), it would be reasonable to split the set of all ontology axioms into the ones interpreted in accordance to the “open world”/“inference” semantics, and the ones considered just to have the “closed world”/“constraint” semantics by not participating in the inference process. Furthermore, such an ontology split can be reasonably defined in a generic way just on the basis of the ontology axiom structure, without regarding the specifics of the concrete ontology.

A practical implementation of RDB2OWL mappings (this applies to RDB-to-RDF/OWL mappings in any other language, as well) could provide the inferred knowledge from the mapped database only via the means of the triple store serving the created SPARQL endpoint with the database data, or via explicit extensions of the mapping implementation. The Virtuoso RDF Graphs (Blakeley, 2007) allow for a built-in sub-class, sub-property and inverse properties inference. The D2RQ Server does not have any built-in inference capabilities, so to include these basic inferences also in on-the-fly access of RDB from a SPARQL endpoint, these have been built into the D2RQ mapping file generation from RDB2OWL annotations.

In the case of more advanced mappings, involving e.g. OWL class expressions, there is an option of building the RDB2OWL mapping in an extended way that covers direct introducing of facts that could, in principle, be introduced by an advanced reasoning

⁸Any assertion used for the inferred knowledge generation would automatically also hold as an integrity constraint on the model that includes this inferred knowledge

process. So, if there were a property chain assertion '*>inverse(student) o course*' introduced for the object property *takes* introduced in Fig. 1 it could be simulated by the RDB2OWL assertion $\{DB: =>Registration-\>\}$ in RDB2OWL (the current assertion $\{DB: =>Registration:(DatePaid \text{ is not null})-\>\}$ in Fig. 3 specifies even a finer-grained integrity constraint). The equivalent class assertion '*=Teacher and facultyLevel value Level_Professor*' for the *Professor* class can be simulated by the class map $\{DB: [[Teacher]]; Level_Code='Professor'\}$.

The RDB2OWL mapping process itself is not in a position to ensure validity of particular integrity constraints over the obtained RDF triple set, if their validity has not been built into the mapping specification, or in the inference process, associated with the mapping. Checking these constraints might be the task of the software exploring the created SPARQL endpoint and their (in-)validity at least in certain cases can be interpreted as (in-)validity of the original RDB data.

4. Data exploring interface: OBIS

There is a variety of tools that support SPARQL (WEB, i; WEB, j) standard and that can be used for the exploration of the RDF triple store that is created as a result of the RDB2OWL annotation implementation over a relational database. We consider here the OBIS⁹ tool (Zviedris et al., 2013) designated originally for generating RDF/SPARQL-based information systems from the system data ontology.

Fig. 4 and Fig. 5 present example screen shots from an OBIS application that is generated automatically from the mini-University ontology of Fig. 1 and whose data are served by a SPARQL endpoint containing the RDF triple data corresponding to the source database contents. Fig. 4 presents a class table view of the class *Student*, including the columns corresponding both to the *Student* class itself and to its superclass *Person*. Fig. 5 shows details for an instance of the *Student* class, including its data properties and links to other instances via direct and inverse object properties.

ID	personName	personID	nationality	studentName	studentID
.../Student1	Dave	123456789	LV - Latvia	Dave	SX43376
.../Student2	Eve	987654321	EE - Estonia	Eve	SX45675
.../Student3	Charlie	555555555	UK - United Kingdom	Charlie	SX43212
.../Student4	Ivan	345453432		Ivan	SY44333

Fig. 4. OBIS user interface for mini-University system: class table view.

⁹ <http://obis.lumii.lv/>

The screenshot displays the OBIS user interface for a mini-University system, specifically the 'Student Details' view. On the left, a class hierarchy is shown under 'Classes', with 'Person' expanded to show 'Teacher' (Assistant, Professor, AssocProfessor) and 'Course' (OptionalCourse, MandatoryCourse). The main area is titled 'Student Details' and contains a form with the following fields:

- ID: http://localhost:2020/XStudent1
- personName: Dave
- personID: 123456789
- nationality: LV - Latvia
- studentName: Dave
- studentID: SX43376

Below the form, there are two tables:

Student - enrolled - AcademicProgram

ID	programName
...Program1	Computer Science

Student - inv(student) - Registration

ID	courseName	courseCode	credits
...XCourse2	Semantic Web	CC0207	3
...XCourse4	Quantum Computations	CX5504	3

Fig. 5. OBIS user interface for mini-University system: class details view.

Technically, OBIS can work with a SPARQL endpoint that is served by OpenLink Virtuoso RDF triple store, or by the D2RQ server.

In the Virtuoso triple store scenario the OBIS application requires a link to the server, an application name and the data ontology; it is up to the user to load the RDB corresponding RDF data into Virtuoso using its interface (the RDF triple data can be generated e.g. by the `rdf-dump` facility of the D2RQ server, as described in Section 3). The D2RQ server scenario furthermore requires a D2RQ mapping file for the application generation and there is no need to load the data manually into the RDF triple store since the D2RQ server connects directly to the RDB using the mapping file.

The generated OBIS application backbone is the named class and object and data property structure specified within the ontology. The application structure takes into account the cardinality restrictions placed on object properties within the ontology, as well as data type information for the respective data properties.

In addition to the data browsing capabilities of OBIS in terms of the class and object property structure of the ontology, OBIS provides also via its report mechanism an integrated SPARQL endpoint to support both the integrity constraint checking and *ad hoc* query evaluation.

Regarding the OBIS visual interface note that the student list form in Fig. 4 contains the attributes (data properties) for the class *Student* itself, as well as for its superclass *Person*. Furthermore, the object property *nationality* is textually represented in this table as well, using the *obis:textPattern* notation attached to the *Nationality* class in the data ontology.

For classes with larger number of data and object properties it would be important also to have a specific order of columns/fields for a class table/form in OBIS. Since

OWL itself does not have a notion of ordering between data and object properties, the attribute order information is recorded in OWLGrEd/OBIS ontology editor into an annotation *obis:defaultOrder* for the direct attributes of any class (without any explicit participation of the user), and the OBIS system is ready to interpret these annotations for the user interface configuration. There is also an option of defining explicit column order for a data component corresponding to a specific class using the *obis:view* annotation, however, creation of such annotation would require a manual information entry in the ontology editor.

The OBIS application creation technology, as well as OBIS application meta-model has been described in (Zviedris et al., 2013); we note here that this application structure allows for both adding further configuration annotations in the input data ontologies, as well as introduce manual configuration/fine tuning of applications after the initial application generation completion.

5. SPARQL query generation: ViziQuer

One of important business information system's aspects is data overview. An overview is based on a specific data query that usually incorporates data from more than one data ontology class.

Such data queries are usually written by an IT specialist that translates it from natural language to a specific query language. As part of our proposed approach we create a diagrammatic query language with the support tool ViziQuer (Barzdins et al., 2008b; Zviedris et al., 2013; Zviedris and Barzdins, 2011) to make at least part of data queries easy-to-formulate for non-IT specialists.

We introduce as simple query example in Fig. 6. The query is depicted via the ViziQuer diagrammatic query language and we want to select all student names and ID's that takes a course that is thought by a professor. Also it contains the SPARQL notation that is generated from the query for execution on a SPARQL endpoint.



```

PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
SELECT ?studentName ?studentID ?courseName ?teacherName WHERE
{ ?Student a :Student. ?Student :takes ?Course.
?Course a :Course. ?Professor :teaches ?Course.
?Professor a :Professor. ?Student :studentName ?studentName.
?Student :studentID ?studentID.
?Course :courseName ?courseName.
?Professor :teacherName ?teacherName. }

```

Fig. 6. A ViziQuer notation for a SPARQL query

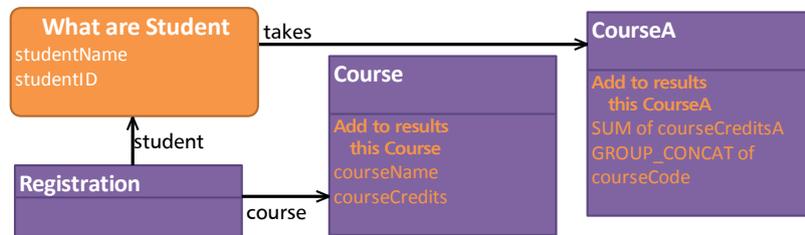
During our research we have found out that diagrammatic query language should have at least some features that would bring data querying and non-IT specialist more closer as nowadays most queries is formulated by an IT specialist.

First aspect is central element in query formulation. We call it central class element. This element is central point for query construction and query reading process. We can see this in Fig. 6, where the central query element is the student class. The class is depicted in different color and shape as other classes.

Second aspect is that we try to use explicit words in diagrammatic graphic that the query is easier to read. Central class has additional prefix “What are”, while other classes has specific sections “Add to results” that implicates that those data elements should be added to central class data.

Central idea behind this query language notation is that easy-to-read becomes into easy-to-write language. One can mention the paper (Juel, 1988) that describes research that children that are better readers later become better writers.

It is possible to formulate also more advanced queries that contain aggregation functions, e.g. one depicted in Fig. 7 that describes students and their courses, as well as their total taken credit points and taken course code lists.



```

PREFIX : <http://lumii.lv/ontologies/UnivExample.owl#>
SELECT ?studentName ?studentID ?courseName ?courseCredits
(SUM (?courseCreditsA) as ?SUM_of_courseCreditsA)
(GROUP_CONCAT (?courseCode) as ?GROUP_CONCAT_of_courseCode)WHERE {
?Student a :Student. ?Student :takes ?CourseA.
?CourseA a :Course. ?Registration :student ?Student.
?Registration a :Registration. ?Registration :course ?Course.
?Course a :Course. ?Student :studentName ?studentName.
?Student :studentID ?studentID.
?CourseA :courseCredits ?courseCreditsA.
?CourseA :courseCode ?courseCode. ?Course :courseName ?courseName.
?Course :courseCredits ?courseCredits. }
GROUP BY ?studentName ?studentID ?courseName ?courseCredits
    
```

	A	B	C	D	E	F	G	H
1	studentName	studentID	courseName	courseCredits	SUM_of_cour	GROUP_CONCAT_of_courseCode		
2	Charlie	SX43212	Semantic Web	3	3	CC0207		
3	Dave	SX43376	Quantum Computations	3	6	CC0207 CX5504		
4	Dave	SX43376	Semantic Web	3	6	CC0207 CX5504		
5	Eve	SX45675	Computer Networks	3	6	CC0140		
6	Eve	SX45675	Programming Basics	6	6	CC0140		

Fig. 7. An aggregate query example in ViziQuer, together with results from sample database, exported to a spreadsheet (note that *Eve* takes only the *Programming Basics* course)

Still it is not yet possible to formulate fully graphically more advanced queries that require sub query constructions. For example, if we want to calculate for a student credit point that he takes and credit points that he have paid for as it requires two different sub queries for each calculation as each aggregation should be done in separate sub query.

The SPARQL queries that are generated by ViziQuer or some other tool or that are handwritten can be executed in the OBIS environment that provides both further exploring within the browser framework of the instances found in the query result and export of the query results to a spreadsheet for further analysis (cf. Fig.7).

Alternatively, there are also SPARQL endpoints provided by the Virtuoso or D2RQ server environments. Execution of the SPARQL queries within the OBIS environment, however, allows for more direct integration of the query results within the data browsing framework provided by OBIS.

Writing SPARQL queries manually or using some other tool for the query generation would be an alternative to the usage of the ViziQuer tool within the database semantic reengineering tool chain. Our observation is that the technical-oriented textual SPARQL syntax would most likely make direct creating of SPARQL queries of limited use for non-IT specialists. The ViziQuer tool provides a graphical environment for SPARQL query composition by selecting the classes involved in the query as nodes (one of the query classes has to be marked as the query's main class) and specifying the class' links and attributes reflecting the instance connections and returned results for the query; it is expected that the ViziQuer query composition would be an activity much more friendlier to non-IT specialists than textual SPARQL query composition.

6. A medical database example

The semantic re-engineering of the Latvian medical data bases had been designed and reported in (Barzdins et al., 2008b), as well as implemented already in 2008, well before established technologies of OWLGrEd, RDB2OWL and OBIS (an early version of ViziQuer have been described in (Barzdins et al., 2008b)). This implementation used a graphical ontology notation that has been manually mapped into OWL, as well as using manual work for the concrete database-to-ontology mapping creation.

The technology chain for the RDB semantic re-engineering, involving OWLGrEd, RDB2OWL, OBIS and ViziQuer provides means for creating a maintainable solution of the Latvian medicine database semantic re-engineering that is able to accommodate not only new or updated database contents, but also the data structure changes. In essence, all the essential data regarding the semantic re-engineering solution specification are placed within the annotated conceptual data ontology from which both the process of RDB-to-RDF data mapping and the data browsing interface generation are controlled (note that the initial 2008 RDB semantic re-engineering technology stack did not include the data browsing interface).

The current state of the technology allows for RDB-to-RDF mapping specification in RDB2OWL as well as its implementation to generate the RDF triples using the D2RQ rdf-dump facility for the full medicine data set consisting of 6 registries, 106 tables, 1353 columns and about 3 million rows; around 40M triples are generated without the basic inference option and about 48M triples with the basic inference option turned on. The use of OBIS browser on the medicine system has been enabled by the introduction of JENA database back-end support for the browsing application configuration

management that has enabled browsing of larger sized data ontologies. Fig.8 shows an example screen shot from the OBIS application with the Latvian medical database.

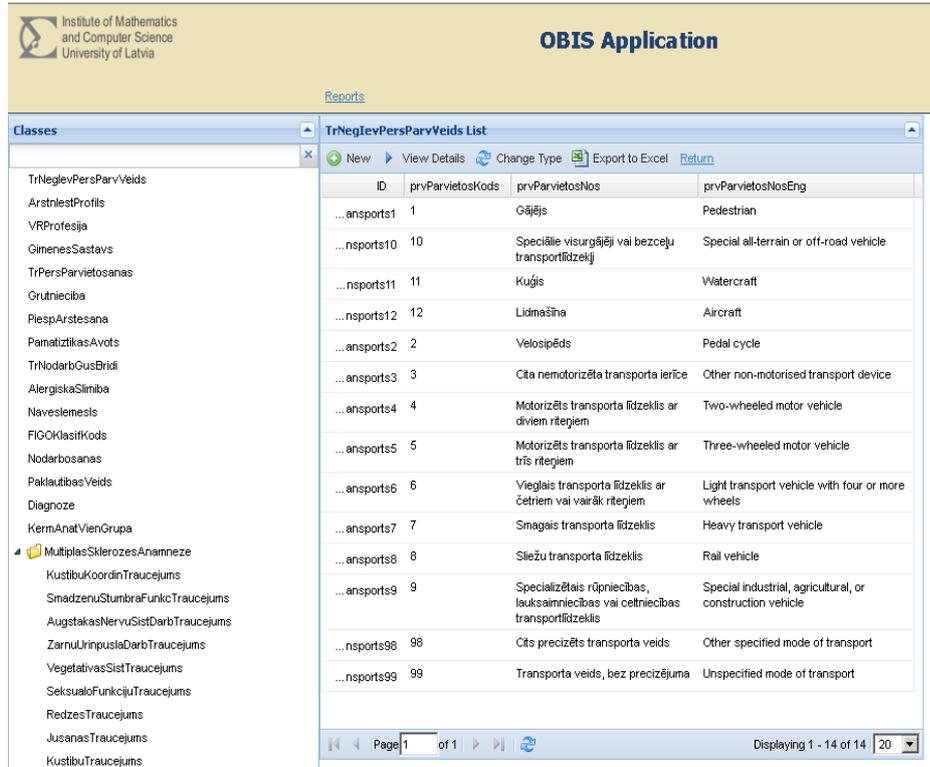


Fig. 8. A Latvian medical database browsing screen shot in OBIS

The ViziQuer tool has been also successfully tested on this application. Fig.9 and Fig.10 show a simple query of injury mechanisms together with the counts of respective injury instances related to these mechanisms. We use the facility of OBIS to show the reports corresponding to SPARQL queries that are generated by ViziQuer.

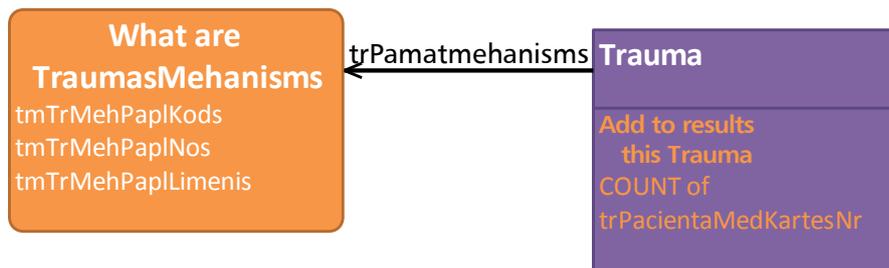


Fig. 9. A ViziQuer query over the medical example

```

PREFIX :
<http://www.lumii.lv/ontologies/2008/med_registries_01.owl#>
SELECT ?tmTrMehPaplKods ?tmTrMehPaplNos ?tmTrMehPaplLimenis
(COUNT (?trPacientaMedKartesNr) as
?COUNT_of_trPacientaMedKartesNr) WHERE {
?TraumasMehanisms a :TraumasMehanisms.
?Trauma :trPamatmehanisms ?TraumasMehanisms.
?Trauma a :Trauma.
?TraumasMehanisms :tmTrMehPaplKods ?tmTrMehPaplKods.
?TraumasMehanisms :tmTrMehPaplNos ?tmTrMehPaplNos.
?TraumasMehanisms :tmTrMehPaplLimenis ?tmTrMehPaplLimenis.
?Trauma :trPacientaMedKartesNr ?trPacientaMedKartesNr.
}
GROUP BY ?tmTrMehPaplKods ?tmTrMehPaplNos ?tmTrMehPaplLimenis

```

Report Results			
▶ View Details  Save to Excel			
COUNT_of_t...	tmTrMehPapl...	tmTrMehPaplLimenis	tmTrMehPaplNos
11193	01.59	3	Kritiens, klupiens, lēciens, grūdiens, bez precizējuma
10333	01.31	3	Cita cilvēka trieciens vai spēriens
10110	98.11	3	Svešķermenis acī vai tā iekļūšana acī vai caur aci
9233	01.51	3	Kritiens un klupiens, aizķeroties tai pašā līmenī
9033	99.99	3	Ievainojuma mehānisms nav precizēts
7663	01.90	3	Saskare ar trulu spēku, bez precizējuma
6546	01.52	3	Kritiens un klupiens, paslīdot tai pašā līmenī
6508	02.13	3	Iegriezums, iešņāpums, iecirtums
4566	01.53	3	Cita veida kritiens, klupiens, lēciens un grūdiens tai pašā līmenī
4519	01.22	3	Sadursme ar nekustīgu priekšmetu

Fig. 10. SPARQL notation and top 10 result rows for the Fig.9 query

7. Conclusions

There is an integrated technology tool chain available for relational database semantic reengineering, consisting of OWL ontology editor OWLGrEd, RDB-to-RDF/OWL mapping language and tool RDB2OWL, followed by an ontology-aware SPARQL endpoint browser OBIS and ad hoc query tool ViziQuer. The tools are based on the open OWL, RDF and SPARQL standards, so usage of other tools for similar tasks can be possible instead of any particular tool from the offered tool chain.

The presented RDB semantic re-engineering process that involves creation of data ontology and mapping of the RDB schema onto it, followed by the obtained SPARQL endpoint exploration via ontology-aware browser and custom SPARQL queries, outlines the feasibility of OWL ontology usage for data modeling. We observe the need to consider the “closed world” or integrity constraint semantics of OWL constructs in parallel with its standard “open world” semantics in defining ontologies as conceptual-level models of data coming from existing databases.

The full size Latvian medical data example implementation in the offered technology tool chain shows its maturity for practically sized databases and corresponding data ontologies. There is a further work in enhancing and fine-tuning the applications to make them more accessible and enjoyable by domain experts that are not IT-professionals; this is an important step to increase the usability of the technology as well as to make a contribution towards spreading the semantic and database access technologies beyond the circles of IT-professionals.

References

- Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J. (2012). *A Direct Mapping of Relational Data to RDF*. Available at <http://www.w3.org/TR/rdb-direct-mapping/>
- Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web, *Scientific American*, May 2001, p. 29-37
- Barzdins, J., Barzdins, G., Balodis, R., Cerans, K. (2006) et.al. Towards Semantic Latvia. In *Communications of 7th International Baltic Conference on Databases and Information Systems*, pp.203-218, 2006.
- Barzdins, J., Barzdins, G., Cerans, K. (2008a). From Databases to Ontologies. In Cardoso, J., Lytras, M. (eds.) *Semantic Web Engineering in the Knowledge Society*. IGI Global, pp. 247-271.
- Barzdins, G., Liepins, E., Veilande, M., Zviedris, M. (2008b). 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).
- Barzdins, J., Barzdins, G., Cerans, K., Liepins, R., Sprogis, A. (2010a). OWLGrEd: a UML Style Graphical Notation and Editor for OWL 2. In *Proc. of OWLED 2010*, 2010.
- Barzdins, J., Cerans, K., Liepins, R., Sprogis, A. (2010b). UML Style Graphical Notation and Editor for OWL 2. In *Proc. of BIR 2010*, LNBIP, Springer 2010, vol. 64, p. 102-113.
- Blakeley, C. (2007). *RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping)*, OpenLink Software, 2007.
- Bumans, G., Cerans, K. (2011). 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
- Bumans, G., Cerans, K. (2010). RDB2OWL: a Practical Approach for Transforming RDB Data into RDF/OWL, in *Proc. of I-Semantics 2010*, ACM International Conference Proceeding Series Article No.: 25 year:2010 ISBN:978-1-4503-0014-8 (3 p.)
- Cerans, K., Bumans, G. (2011). RDB2OWL: a RDB-to-RDF/OWL Mapping Specification Language // Barzdins, J., Kirikova, M. (eds.), *Databases and Information Systems VI*, IOS Press 2011, p.139-152.
- Cerans, K., Barzdins, G., Liepins, R., Ovcinnikova, J., Rikacovs, S., Sprogis, A. (2012). Graphical Schema Editing for Stardog OWL/RDF Databases using OWLGrEd/S // *Proc. of OWLED 2012*, Heraklion, Greece, May 27-28, 2012. CEUR-WS Vol. 849, 8pp.
- Cerans, K., Ovcinnikova, J., Liepins, R., Sprogis, A. (2013). Advanced OWL 2.0 Ontology Visualization in OWLGrEd // Caplinskas, A., Dzemyda, G., Lupeikiene, A., Vasilecas, O. (eds.) *Databases and Information Systems VII*, IOS Press, Frontiers in Artificial Intelligence and Applications, Vol 249, pp.41-54, 2013
- De Laborda, C.P., Conrad, S. (2006). Bringing Relational Data into the Semantic Web using SPARQL and Relational. OWL Semantic Web and Databases. In Third International Workshop, SWDB 2006, Co-located with ICDE, Atlanta, USA, April 2006
- Horridge, M., Peter, F. (2012). *OWL 2 Web Ontology Language Manchester Syntax (Second Edition)*. Available at <http://www.w3.org/TR/owl2-manchester-syntax/>
- Juel, C. (1988). Learning to read and write: A longitudinal study of 54 children from first through fourth grades. *Journal of educational Psychology*, Vol. 80, N. 4, 1988, pp. 437-447.

- Motik, B., Patel-Schneider, P.F., Parsia, B. (2009). *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*. Available at <http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/>
- Sequeda, J.F., Cunningham, C., Depena, R., Miranker, D.P. (2009). Ultrawrap: Using SQL Views for RDB2RDF. In *Poster Proceedings of the 8th International Semantic Web Conference (ISWC2009)*, Chantilly, VA, USA.
- Tao, J., Sirin, E., Bao, J. (2010). McGuinness D. Integrity Constraints in OWL. In *Proc. of AAAI 2010*.
- Zviedris, M., Barzdins, G. (2011). ViziQuer: A Tool to Explore and Query SPARQL Endpoints, *The Semantic Web: Research and Applications*, LNCS, 2011, Volume 6644/2011, pp. 441-445
- Zviedris, M., Romane, A., Barzdins, G., Cerans, K. (2013). Ontology-Based Information System, to appear in *Proc. of JIST'2013*, Lecture Notes in Computer Science.
- WEB (a) *D2R Server: Accessing databases with SPARQL and as Linked Data*. <http://d2rq.org/d2r-server>
- WEB (b) *D2RQ Platform. Treating Non-RDF Relational Databases as Virtual RDF Graphs*. <http://www4.wiwiiss.fu-berlin.de/bizer/D2RQ/spec/>
- WEB (c) *Linked Data*. <http://linkeddata.org>
- WEB (d) *Protégé 4*. <http://protege.stanford.edu/>
- WEB (e) *RDF Schema*. <http://www.w3.org/TR/rdf-schema/>
- WEB (f) *Resource Description Framework (RDF)*. <http://www.w3.org/RDF/>
- WEB (g) *R2RML: RDB to RDF Mapping Language*. <http://www.w3.org/TR/r2rml/>
- WEB (h) *Semantic Web Solutions at Work in the Enterprise*. http://www.topquadrant.com/docs/marcom/TopQuadrant_Whitepaper_online.pdf
- WEB (i) *SPARQL Query Language for RDF*. W3C Recommendation 15 January 2008. <http://www.w3.org/TR/rdf-sparql-query/>
- WEB (j) *SPARQL 1.1 Overview*. W3C Recommendation 21 March 2013. <http://www.w3.org/TR/sparql11-overview/>
- WEB (k) *Spyder tool*. <http://www.revelytix.com/content/spyder>
- WEB (l) *TopBraid Composer*. http://www.topquadrant.com/products/TB_Composer.html
- WEB (m) *Unified Modeling Language: Infrastructure, version 2.1*. OMG Specification ptc/06-04-03. <http://www.omg.org/docs/ptc/06-04-03.pdf>
- WEB (n) *Unified Modeling Language: Superstructure, version 2.1*. OMG Specification ptc/06-04-02. <http://www.omg.org/docs/ptc/06-04-02.pdf>

Received September 17, 2014, accepted September 18, 2014.