Towards Graphical Query Notation for Semantic Databases

Kārlis Čerāns, Jūlija Ovčiņņikova, Mārtiņš Zviedris

karlis.cerans@lumii.lv, julija.ovcinnikova@lumii.lv, martins.zviedris@lumii.lv
Institute of Mathematics and Computer Science, University of Latvia
Raina blvd. 29, Riga, LV-1459, Latvia

Abstract. We describe a notation and a tool for schema-enabled visual/diagrammatic creation of SPARQL queries over RDF databases. The notation and the tool support both the standard basic query pattern comprising a main query class and possibly linked condition classes and means for aggregate query definition and placing conditions over aggregates including also aggregation of aggregate results. We discuss the applicability of the tool for ad-hoc query formulation in practical use cases.

Keywords: Visual query creation, SPARQL, RDF databases, Semantic technologies

1 Introduction

The semantic technologies built around the RDF [1,2], OWL [3] and SPARQL [4] standards are the basis for the Semantic Web [5] and Linked Data [6], as well as they may well be used in enterprise-level use cases. The semantic technologies offer much higher-level view on data than do the classic relational databases (RDB) with their corresponding SQL query language thus enabling more direct involvement of various domain experts in data set definition, exploration and analysis.

The availability of the data in the semantic information landscape is ensured mainly via mappings into RDF/OWL from original data sets in various formats, notably the relational databases, where exists a W3C mapping standard R2RML [7], as well as numerous mapping formalisms (see e.g. Virtuoso RDF Views [8], D2RQ [9], ontop [10] and RDB2OWL [11]). The RDF data stores such as Virtuoso [8] and Stardog [12] provide native RDF data storage possibility.

The classical approach to data access in relational databases involves creating user interface applications for common operations over the data and then asking programmers to create on-demand SQL queries in the case of non-standard information requests. A similar approach can be followed also in the case of RDF databases and SPARQL queries, however, this approach will not respond to the expectation for the direct domain expert involvement in the data access. Therefore a number of approaches such as ViziQuer [13,14] and Optique VQS [15] for visual/diagrammatic creation of SPARQL queries have emerged. The practical experience with the ViziQuer tool with the medical domain experts have confirmed the importance of the visual query creation concept, however, it has also confirmed a further need in support of aggregate query generation that is a very typical custom query kind over the data. The problem addressed in this paper is presenting a visual query notation and tool for aggregate query definition and translation into SPARQL.

We base the work on the availability of the aggregate queries in SPARQL version 1.1. [4]. The task of visual/diagrammatic query creation is quite challenging since the diagrammatic notations for aggregates in the queries is not common even for query creating systems also in the much widely established area of RDB/SQL databases.

In the rest of the paper we introduce first the basic visual query notation, including already the simplest patterns for aggregate query construction, followed by direct grouping and subquery constructs usable for more involved query generation. The practical use aspects of the approach is discussed in the final section of the paper. The illustrations in the paper are given on a mini-University example, however they can be carried over directly to more practical use cases, including the medical domain.

2 Basic Query Notation

This section outlines the basic visual query SPARQL notation, including specification of class information, attribute selection, conditions and links, similar to [13,14], as well as new simple aggregate attribute definition. The visual/diagrammatic query generation is based on the data schema definition as OWL ontology or RDF Schema; for the construction illustration we shall use the following mini-University ontology, presented in Figure 1 in graphical OWLGrEd[1] ontology editor notation [16].

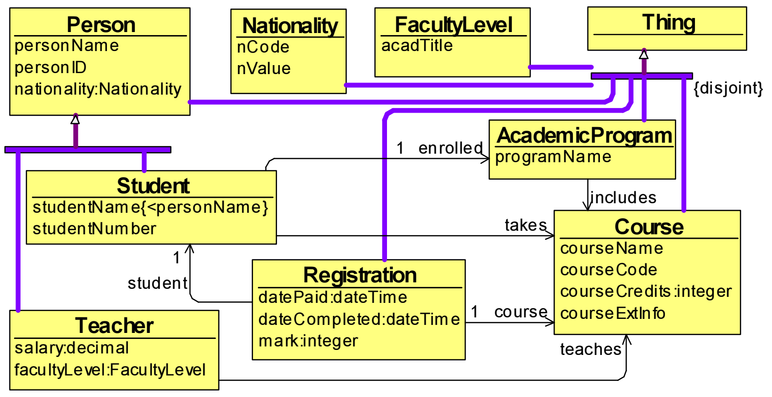
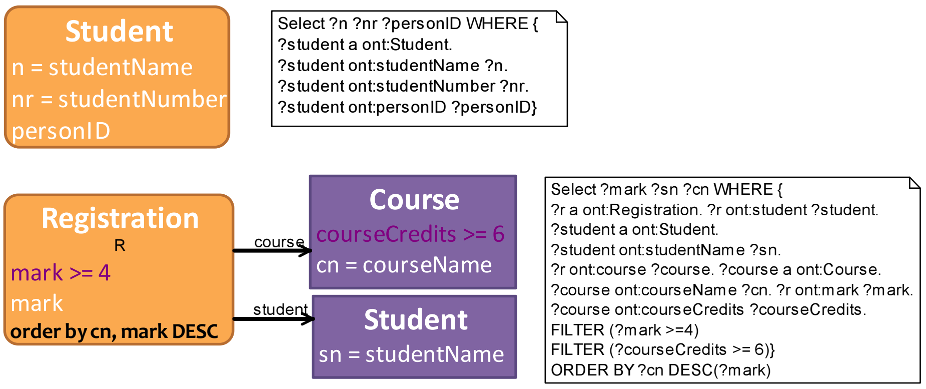


Fig. 1. Example: a mini-University ontology in the OWLGrEd notation

Figure 2 illustrates two basic visual queries and their corresponding SPARQL queries. The first query selects all student names, student ID and student card ID. The second query selects all successful completed registrations (mark >= 4) for large courses (at

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

least 6 credit points) and displays mark, student name and course name[2]. Each query contains a single main query class (shown as orange round rectangle) as well as possibly a number of condition classes (shown as violet rectangles), linked to the query class via links corresponding to the object properties between the classes in the ontology (chains of condition classes and links from condition classes to the main class are allowed, as well). Each of the class boxes in the query represents an instance resource belonging to the class. The class attributes (shown in white letters) define instance properties (attributes) that are to be included in the query output. The conditions (shown in pink/dark) restrict the rows to be returned by the query.



```
Select ?n ?nr ?personID WHERE {
?student a ont:Student.
?student ont:studentName ?n.
?student ont:studentNumber ?nr.
?student ont:personID ?personID}
```

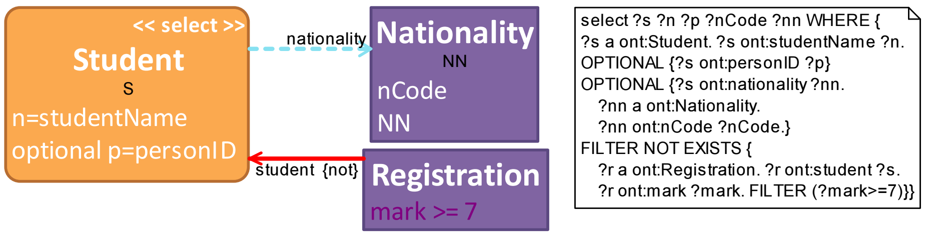
```
Select ?mark ?sn ?cn WHERE {
?r a ont:Registration. ?r ont:student ?student.
?student a ont:Student.
?student ont:studentName ?sn.
?r ont:course ?course. ?course a ont:Course.
?course ont:courseName ?cn. ?r ont:mark ?mark.
?course ont:courseCredits ?courseCredits.
FILTER (?mark >=4)
FILTER (?courseCredits >= 6)}
ORDER BY ?cn DESC(?mark)
```

Fig. 2. Basic query examples

The queries in Figure 2 contain just the attribute values, not the URIs of the involved instance resources. The inclusion of the instance URI in the query output can be specified either with <<select>> stereotype for the class, or by including the class instance name (e.g. *R* for *Registration*) into the attribute list for the class.

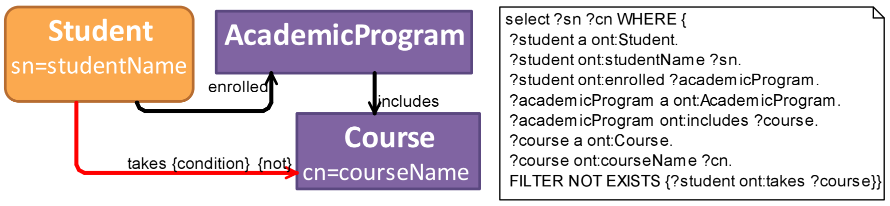
Figure 3 shows the optional attribute (marked by the keyword OPTIONAL) and optional link notation (as the blue dashed line) as well as a negation link (marked in red and the stereotype {not}). The whole query fragments placed behind the optional or negation links (from the viewpoint of the main query class) are in optional or negation group. If a negation link connects class instances that are already connected, as in Figure 4 example, it is to be interpreted as a condition asserting non-existence of the respective link, after the query structure has been created from the other "structural" links. We use the {condition} stereotype on the link to mark its condition semantics; this stereotype can be used also on affirmative/positive edges, asserting the existence of the respective link (the optional link with the {condition} mark makes a void requirement on the query contents). The non-condition loops including negation or optional links are not allowed. There is a requirement for the entire diagram to be connected via non-condition links; if the query logics would require several non-connected components, they are to be connected with un-labelled strict or optional links marking the query structure.

[2] The prefix ont: in the examples stands for http://lumii.lv/ontologies/UnivExample.owl# that is the URI for the mini-University ontology itself.



```
select ?s ?n ?p ?nCode ?nn WHERE {
?s a ont:Student. ?s ont:studentName ?n.
OPTIONAL {?s ont:personID ?p}
OPTIONAL {?s ont:nationality ?nn.
   ?nn a ont:Nationality.
   ?nn ont:nCode ?nCode.}
FILTER NOT EXISTS {
   ?r a ont:Registration. ?r ont:student ?s.
   ?r ont:mark ?mark. FILTER (?mark>=7)}}
```

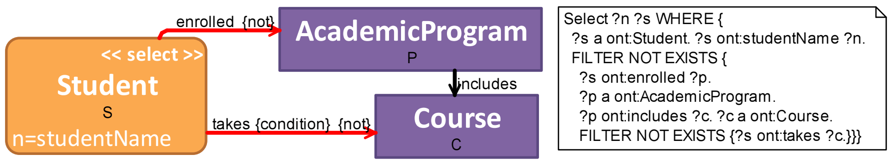
Fig. 3. Explicit instance names, instance URI selection; optional and negation links



```
select ?sn ?cn WHERE {
 ?student a ont:Student.
 ?student ont:studentName ?sn.
 ?student ont:enrolled ?academicProgram.
 ?academicProgram a ont:AcademicProgram.
 ?academicProgram ont:includes ?course.
 ?course a ont:Course.
 ?course ont:courseName ?cn.
 FILTER NOT EXISTS {?student ont:takes ?course}}
```

Fig. 4. Negated condition link: select names of students and courses with the student not taking a course included in academic program (s)he is enrolled in.

One can use the combination of structural and condition links in the query to formulate queries involving universal quantification, expressed using double negations. For instance, the query "find all students taking all courses of the academic program they are enrolled in", is demonstrated in Figure 5 via "find all students not enrolled in academic program including a course the student is not taking." A corresponding query with the structural semantics of both negation links would not be allowed due to the loop with two structural negations emerging.



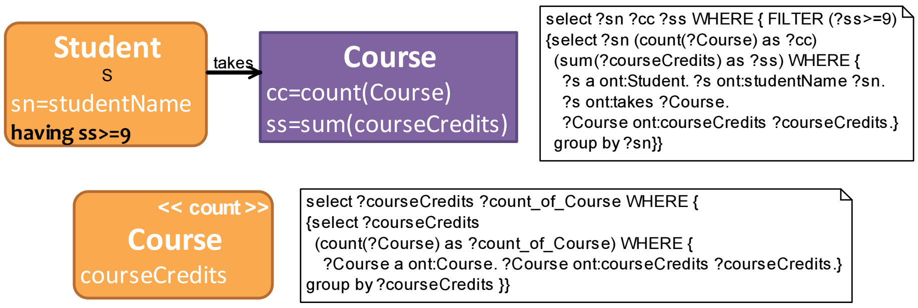
```
Select ?n ?s WHERE {
  ?s a ont:Student. ?s ont:studentName ?n.
  FILTER NOT EXISTS {
   ?s ont:enrolled ?p.
   ?p a ont:AcademicProgram.
   ?p ont:includes ?c. ?c a ont:Course.
   FILTER NOT EXISTS {?s ont:takes ?c.}}}
```

Fig. 5. Double negation: structural and condition negations determine the query structure.

The basic query notation allows for simple aggregate query introduction in the case, if the grouping set for all aggregate function applications coincides with the set of all selected non-aggregate attributes in the query. An aggregate attribute is introduced as an expression where the aggregate function (count, sum, avg, min, max, group_concat) is applied to the attribute name; the aggregation over the instance URI is possible also via the <<count>> or <<count distinct>> stereotype for the class.

The SPARQL query for such a simple aggregated query is generated in two steps: first, for every class instance with aggregate attributes a SPARQL-subquery is generated involving the aggregated attributes from this class instance and the grouping set of all non-aggregated attributes in the query; then all subqueries are joined together into a single SPARQL query (cf. examples in Figure 6 and Figure 7).

Figure 6 contains two examples, where, first, a student name is selected together with the corresponding count of taken courses and sum of credits within those, restricting the result rows to those having course credit sum at least 9. The conditions in the query classes are evaluated before aggregate applications, therefore the row filter depending on the aggregate value is defined in the *having*-compartment of the class box. The other example shows listing the different *courseCredits* values together with the count of courses corresponding to each of these values.

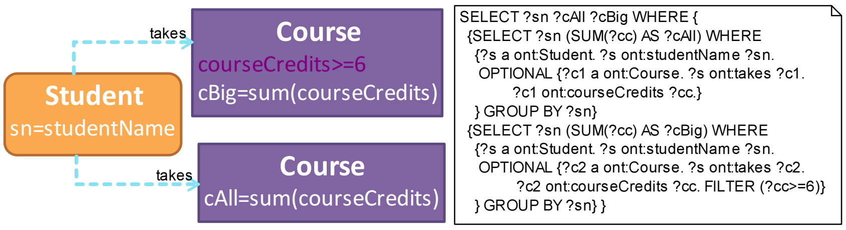


```
select ?sn ?cc ?ss WHERE { FILTER (?ss>=9)
{select ?sn (count(?Course) as ?cc)
 (sum(?courseCredits) as ?ss) WHERE {
  ?s a ont:Student. ?s ont:studentName ?sn.
  ?s ont:takes ?Course.
  ?Course ont:courseCredits ?courseCredits.}
 group by ?sn}}
```

```
select ?courseCredits ?count_of_Course WHERE {
{select ?courseCredits
  (count(?Course) as ?count_of_Course) WHERE {
   ?Course a ont:Course. ?Course ont:courseCredits ?courseCredits.}
group by ?courseCredits }}
```

Fig. 6. Simple aggregation: basic examples

Figure 7 describes a query for finding student names together with both sum of credits for the student in all courses, and in “big” courses with at least 6 credit points each.



```
SELECT ?sn ?cAll ?cBig WHERE {
 {SELECT ?sn (SUM(?cc) AS ?cAll) WHERE
  {?s a ont:Student. ?s ont:studentName ?sn.
   OPTIONAL {?c1 a ont:Course. ?s ont:takes ?c1.
         ?c1 ont:courseCredits ?cc.}
  } GROUP BY ?sn}
 {SELECT ?sn (SUM(?cc) AS ?cBig) WHERE
  {?s a ont:Student. ?s ont:studentName ?sn.
   OPTIONAL {?c2 a ont:Course. ?s ont:takes ?c2.
         ?c2 ont:courseCredits ?cc. FILTER (?cc>=6)}
  } GROUP BY ?sn} }
```

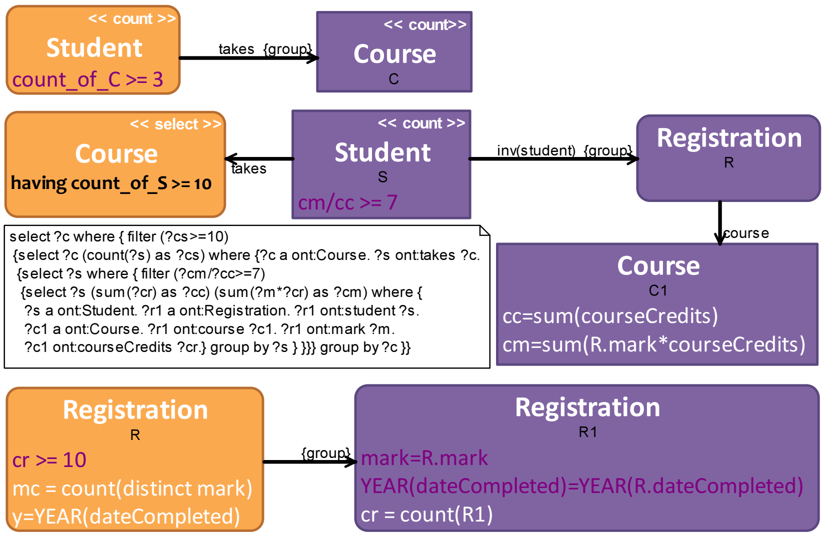
Fig. 7. Simple aggregation: optional links and multiple-class aggregates

3 Explicit Grouping and Subqueries

The outlined basic query constructions cover a large range of practical queries that may arise in the exploration of the mini-University example, as well as in the practical use cases of query formulation over hospital information system and clinical records databases. There are, however, natural queries not fitting naturally the basic query pattern due to involving the aggregate-over-aggregate pattern, e.g.:

A. Count all students taking at least three courses (count-over-counts).
B. Find all courses passed by at least 10 students with mean mark (over all passed courses) at least 7 (filtered counts over filtered aggregate expressions).
C. List all different years for the course completion dates together with the count of different marks received in this year at least 10 times.

We introduce an explicit {group} stereotype for affirmative and optional links. Its semantics is splitting the query diagram into “main” part towards the main query class side of the {group} link and subquery part behind the other end of the link, with the further design assumption that the URI of the instance on the "main" end of the link, as well as the link itself also participates in the subquery, making the "join condition" between the subquery and the main query. The {group} stereotype is not compatible with {condition} stereotype, nor is it to be used for negation links. The general query shape has to be a tree of simple components (i.e. components built with non-condition and non-group links), linked by {group}-links; the {condition}-links are allowed only within a single component, or from within a component to its incoming {group}-link source node. Figure 8 shows the A-C queries in the visual query notation.



```
select ?c where { filter (?cs>=10)
 {select ?c (count(?s) as ?cs) where {?c a ont:Course. ?s ont:takes ?c.
  {select ?s where { filter (?cm/?cc>=7)
   {select ?s (sum(?cr) as ?cc) (sum(?m*?cr) as ?cm) where {
    ?s a ont:Student. ?r1 a ont:Registration. ?r1 ont:student ?s.
    ?c1 a ont:Course. ?r1 ont:course ?c1. ?r1 ont:mark ?m.
    ?c1 ont:courseCredits ?cr.} group by ?s } }}} group by ?c }}
```

Fig. 8. Visual queries with explicit grouping.

There is also the option of introducing and using explicit named subqueries in the visual language. While the {group}-links in the query diagram may often appear to be an easier mechanism of involved query specification, the named subqueries would correspond to "derived concept" introduction and allow easier query reuse. Figure 9 shows the B example query definition and usage in the named subquery notation.

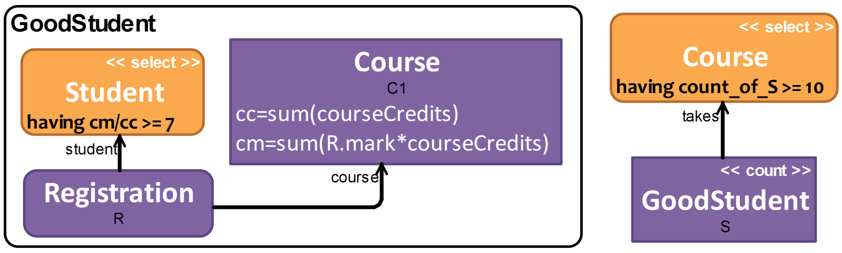


Fig. 9. A named subquery example

The visual query language provides also means for query result set limiting to a number of rows (the SPARQL LIMIT construction). Although one could easily add to the visual query language this notation also on the subquery level (including the inline and named subqueries), the SPARQL language the queries are translated into does not contain at least direct means for expressing this type of constructs. Therefore such an important query pattern as “find all x with their related most common y” (e.g. find all courses with the most often received marks in them) cannot be directly offered for practical usage. The practical workaround to this limitation would be to re-formulate the queries with LIMIT-bound subqueries in a way that they return larger result sets from which the needed results can easily be obtained e.g. in a spreadsheet.

4 Discussion and Conclusions

We aim at practical visual query formulation system creation over data via their conceptual structure view. As the initial experiments show, most of the practically interesting queries both in the hospital information system and clinical records database use cases can be formulated using the provided notation. The notation and tool polishing is certainly worth to be continued including the offering of the query tool to the domain experts. There would be need in a query creation methodology and initial training before the domain experts would be ready to use the query tool themselves.

The notation described in this paper has a prototype implementation in the ViziQuer tool[3]. The ViziQuer tool can be used for SPARQL query creation over any SPARQL endpoints with their data RDF schema available. The work in [14] discusses RDF schema extraction from a SPARQL endpoint. [17] outlines the possibility of using ViziQuer in the context of relational database semantic re-engineering, where a conceptual model of a relational database is created as an OWL ontology, then the mapping from the RDB to the ontology is described, enabling creation of the ontology-structured SPARQL endpoint that can accept queries created in ViziQuer. The limitation of SPARQL not allowing row-level aggregate subqueries with TOP/LIMIT restrictions (such queries would not be problematic e.g. in SQL) may lead to considerations of direct translations from the visual query language to SQL that would be well defined within the RDB semantic re-engineering framework.

A less considered query language aspect in this paper is the language of expressions allowed in conditions and query output definition. The idea of translating expressions into SPARQL is based on supporting SPARQL-like expression syntax in the language, with the extensions allowing instance references and data attribute names to stand for resources and their properties respectively. The practical use cases have shown the possibility of expression creation and translation, however, the date manipulation functions available in SPARQL standard [4] are clearly insufficient for practical queries e.g. concerning duration calculation based on the available date or datetime values. Notably, the Virtuoso RDF data store [8] supports the extensions allowing the necessary date and interval value handling. For the RDB semantic re-engineering use case this entails the necessity of creating and storing RDB-to-RDF dump into a triple store, instead of using an on-the-fly maintained SPARQL endpoint.

[3] http://viziquer.lumii.lv/

Acknowledgements

This work has been supported by European Union within the ERDF project 'Granular ontology tools for data analysis' (Project No. 2DP/2.1.1.1.0/14/APIA/VIAA/072) and Latvian State Research program NexIT project No.1 'Technologies of ontologies, semantic web and security'.

References

1. Resource Description Framework (RDF), http://www.w3.org/RDF/
2. RDF Schema [WWW] http://www.w3.org/TR/rdf-schema/
3. Motik, B; Patel-Schneider P.F; Parsia B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, 2009
4. SPARQL 1.1 Overview. W3C Recommendation 21 March 2013 [WWW] http://www.w3.org/TR/sparql11-overview/
5. Linked Data, http://linkeddata.org
6. Tim Berners-Lee, James Hendler and Ora Lassila, "The Semantic Web", Scientific American, May 2001, p. 29-37.
7. R2RML: RDB to RDF Mapping Language, http://www.w3.org/TR/r2rml/
8. C.Blakeley: "RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping)", OpenLink Software, 2007.
9. D2RQ Platform. Treating Non-RDF Relational Databases as Virtual RDF Graphs. http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/spec/
10. 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.
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. Stardog, http://stardog.com/
13. 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).
14. 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
15. A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks. OptiqueVQS: Towards an Ontology based Visual Query System for Big Data. In: MEDES. 2013.
16. 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.
17. 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.