Graphical Schema Editing for StarDog OWL/RDF Databases using OWLGrEd/S

Karlis Cerans[*], Guntis Barzdins[*], Renars Liepins[**], Julija Ovcinnikova[*], Sergejs Rikacovs[*], Arturs Sprogis[**]

Institute of Mathematics and Computer Science, University of Latvia
{Karlis.Cerans, Guntis.Barzdins, Renars.Liepins, Julija.Ovcinnikova, Sergejs.Rikacovs, Arturs.Sprogis}@lumii.lv

Abstract. The developers of StarDog OWL/RDF DBMS have pioneered a new use of OWL as a schema language for RDF databases. This is achieved by adding integrity constraints (IC), also expressed in OWL syntax, to the traditional “open-world” OWL axioms. The new database paradigm requires a suitable visual schema editor. We propose here a two-level approach for integrated visual UML-style editing of extended OWL+IC ontologies: (i) introduce the notion of ontology splitter that can be used in conjunction with any OWL editor, and (ii) offer a custom graphical notation for axiom level annotations on the basis of compact UML-style OWL ontology editor OWLGrEd.

Keywords: OWL, integrity constraints, STARDOG, OWL/RDF databases, graphical database schema editor, OWLGrEd, UML class diagrams

1 Introduction

Web ontology language OWL [1,2,3] follows “open world assumption” (OWA) semantics that implies every statement whose truth is not known to be *undefined* rather than *false* in the contrasting “closed world assumption” (CWA) semantics (cf [4]). While OWA semantics is appropriate for many traditional OWL uses, in the recent years there have been also efforts to introduce “integrity constraints” (see e.g. [5,6,7]) over OWL ontology models through CWA semantics. The integrity constraint (IC) assertions may appear natural in e.g. information system specifications, where, for example, a missing phone number for a person x under the assertion that every person has a phone number would be naturally interpreted as a data error rather than inferring existence of some unknown phone number for x.

The IC specification in [6,7], implemented in Stardog [8] OWL/RDF database, reuse the OWL syntax itself also for IC thus materializing the idea of using “the full expressivity of OWL and OWL 2 ... as a schema language for RDF”[1]. This opens a possibility for a wide range of applications of (extended) OWL in information base structure (schema) specification. This, however, raises an issue of suitable graphical

[*] Partially supported by European Union via European Regional Development Fund project 2011/0009/2DP/2.1.1.1.0/10/APIA/VIAA/112.
[**] Partially supported by Latvian 2010.-2013. National Research Program Nr.2 project Nr.5.
[1] cf. http://clarkparsia.com/pellet/icv/

notation for extended OWL notation rendering and editing, as it is common e.g. for MOF-style model repository schemas in the form of UML [9,10] class diagrams, or for relational databases.

There are a number of approaches and tools including UML/OWL profile [11], ODM [12], Top Braid Composer [13] and OWLGrEd [14,15] implementing (some variant/extension of) UML class diagram notation as visual notation for OWL ontologies, however, none of these have been explicitly intended for visual management of extended OWL+IC ontologies. Note that any of the said OWL ontology editors can be used to graphically edit the OWA (="proper OWL") part of the extended ontology, leaving the IC specification to be done by some other means.

Our aim here is to offer extended ontology editor and framework that are able to cope with both OWA-axioms and IC within a single notational space (single ontology or UML-style class diagram). The OWA vs. IC ontology separation then is left to an ontology post-processing step to be performed by some "ontology splitter" that can be defined as a procedure receiving as an input any OWL ontology (as a syntactic unit) and producing as the output its "partitioning" into OWA and IC parts.

For many practical use cases it might be sufficient for such ontology splitting procedure to rely just on the structure of input ontology axioms. An example splitter could, e.g. send all cardinality restrictions into the IC-part of the ontology, while all *subClassOf(A,B)* axioms with named *A* and *B* could go into the OWA-part. Another "splitter" could send all ontology into its OWA part, leaving the IC-part empty.

We note that even the basic functionality of any OWL editor (including all said UML/OWL editors and e.g. Protégé [16]), in combination with such ontology splitter would be sufficient for extended OWA+IC ontology authoring in these use cases.

The full generality of ontology splitters is easily obtained by allowing them to resort not only to axiom structure, but also to ontology entity and axiom annotations. For instance, a splitter may send to the OWA-part only those *subPropertyOf(:p,:q)* axioms, where *:q* is annotated by *AnnotationAssertion(a:isInferred :q "true")* for a suitable annotation property *a:isInferred.*

The use of such "general" ontology splitter requires, however, availability of suitable entity and axiom annotation notation within the editor, that is a non-trivial task for UML-style OWL ontology editors. Although there are generic means for entity annotation in OWLGrEd, we offer and describe here its extended version OWLGrEd/S supporting a custom notation for entity and axiom annotations that is suitable for extended OWA+IC ontology specification.

From the methodological viewpoint we note that the ontology splitter to be applied to the resulting OWA+IC ontology should be viewed as belonging to the *semantics* of the editor used in the ontology authoring. There could be a number of concrete well established ontology splitters suitable for different application areas and modeling tasks based on OWA+IC ontologies that could be applied in appropriate situations. We briefly sketch here some principles of ontology splitter construction for semantic database schema definition and offer one possible candidate splitter definition.

In the following sections we briefly review the UML-style OWL ontology editor OWLGrEd, comment on the integrity constraints and schema semantics of extended ontologies and then move to ontology splitter and OWLGrEd/S notation description.

2 Visual Ontology Modeling with OWLGrEd

OWLGrEd (http://owlgred.lumii.lv/) provides a complete graphical notation for OWL 2, based on UML class diagrams. We visualize OWL classes as UML classes, data properties as class attributes, object properties as association roles, individuals as objects, cardinality restrictions on property domain class as UML cardinalities, etc. We enrich the UML class diagrams with the new extension notations, e.g. (cf. [14,15]):

- fields in classes for *equivalent class*, *superclass* and *disjoint class* expressions written in Manchester OWL syntax [17];
- fields in associations and attributes for *equivalent*, *disjoint* and *super* properties and fields for property characteristics, e.g., *functional*, *transitive*, etc.;
- anonymous classes containing *equivalent class expression* but no name (we show graphically only those anonymous classes that need to have graphic representation in order to be able to describe other ontology concepts in the diagram);
- connectors (as lines) for visualizing binary *disjoint*, *equivalent*, etc. axioms;
- boxes with connectors for n-ary *disjoint*, *equivalent*, etc. axioms;
- connectors (lines) for visualizing object property restrictions *some*, *only*, *exactly*, as well as cardinality restrictions.

Figure 1 contains example mini-University ontology, shown in OWLGrEd notation [14,15]. We note also that the OWLGrEd editor offers ontology interoperability (import/export) functionality with Protégé 4.1. ontology editor [16].

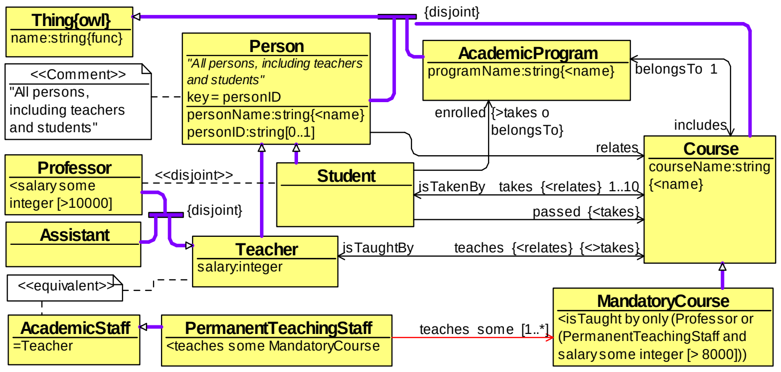

Fig.1. Mini-University ontology in OWLGrEd

3 Integrity Constraints in Semantic Database Schemas

Regarding mini-University ontology of Figure 1 as a database schema would lead to certain un-intended consequences due to OWL standard "open-world" semantics, e.g.:

- if an assistant X has registered, by an error, as taking (*takes*) a course, the system infers that X is a student since only students are allowed to take a course;
- existence of a student with no taken courses specified does not rise an error;
- if a course belongs to two academic programs (with no names specified yet), these would be inferred to be the same academic program;

- if a student *X* *takes* a course *Y* belonging to (*belongsTo*) academic program *W* that is other than *V*, where *X* is *enrolled*, *X* is inferred to be *enrolled* also in *W*;
- if a professor has a recorded salary of 9500, the system would infer that there is also another salary for the professor that is > 10000.

The integrity constraints (IC) [5,6,7] are nowadays commonly invoked to handle these situations and the StarDog database environment [8] supports the approach of [6]. Following [6], we let an extended ontology be a pair <*K,C*>, where *K* is an ontology (interpreted according to OWA) and *C* is IC specification (interpreted according to CWA over *K*), both expressed in OWL syntax. In [6] a constraint $\alpha \in C$ is said to be *satisfied* by *K*, written $K \models_{IC} \alpha$, if and only if all minimal equality (ME) models [6] [2] of *K* satisfy α (one can informally say that α has to be satisfied on all "intersections" of non-contradictory ME-models) and the extended ontology <*K,C*> to be *valid* if and only if $K \models_{IC} \alpha$ for all $\alpha \in C$. We extend this definition to call <*K,C*> *consistent* if and only if *K* is consistent (i.e. it has a model) and <*K,C*> is valid.

Our interest here is to offer graphical editors for database schemas, defined as extended ontologies. Regarding <*K,C*> as a database schema means that there is some data expected to be "filled in" to it, and that the actual consistency checking and constraint validation tasks are to be performed in a <*K+DK,C+DC*> situation for a data ontology *DK* (typically consisting of A-Box axioms) and some (possibly empty) data-level constraint set *DC*. We say that the extended data ontology <*DK,DC*> *conforms to* the schema ontology <*K,C*> whenever the combined extended ontology <*K+DK,C+DC*> is consistent. The schema-semantics of the extended ontology <*K,C*> can then be defined as the set of all <*DK,DC*> conforming to <*K,C*>.

Note that within the "schema-semantics" of <*K,C*> the "satisfaction by all ME-models" (= by ME-model "intersections") for the constraint validity is considered for any *K+DK*, where *DK* is arbitrary "data ontology", thus covering a large part of *K* models satisfying *C* as the representative ME-model "intersections" for suitable *DK*.

4 Ontology Splitters

An ontology splitter is a function that, given ontology (a set of OWL 2.0 axioms) *X*, produces two sets of axioms *O(X)* and *C(X)*, whose union has the same logical meaning, as *X* (i.e. *O(X)+C(X)* with both *O(X)* and *C(X)* viewed in OWA-sense is valid on a model *M* if and only if *X* is valid). In the context of separating IC-part out of the ontology, the application of such an ontology splitter would allow producing an extended ontology <*O(X),C(X)*> with *O(X)* interpreted in OWA-sense and *C(X)* interpreted in CWA-sense from the ontology *X*. A simple ontology splitter would just partition the ontology axiom set into two subsets, however, there may be cases when an axiom re-factoring is needed (e.g. an *EquivalentClasses*-axiom may be split into two *SubClassOf*-axioms). We note that for different application areas and different system modeling paradigms there might be different ontology splitters applied (e.g. there can be a "trivial" ontology splitter having $O(X)=X$ and $C(X)=\varnothing$, or there can be a splitter doing some ontology axiom differentiation).

[2] In essence, a ME-model has no unnecessary equalities between named individuals.

An ontology splitter can be defined in terms of rules that determine for each source ontology *X* axiom *A* the action to be taken: (i) move *A* into *C(X)*, (ii) move *A* into *O(X)*, or (iii) re-factor *A* into parts to be further processed by the ontology splitter (i.e. moved into *C(X)*, *O(X)* or re-factored further).

The action taken by the splitter on axiom *A* can be determined on the basis of:

- axiom *A* structure (e.g. by pattern matching over some *A* syntactical presentation, we use here OWL Functional Syntax [2] (OFS)),
- annotation assertions (or other axioms) in *X* on entities involved in *A*,
- axiom *A* annotations (with pre-defined annotation properties and values).

We summarize the possible re-factoring actions for translating an axiom into the set of its parts in Figure 2 using an intuitive pattern matching notation over OFS, where the variable placeholder, such as X?, stands for arbitrary OFS term and X1? .. Xn? notation is used to denote a list of OFS terms. These rules do not change the OWA-semantics of the ontology, as required by the ontology splitter definition.

```
a. EquivalentClasses(X? Y?) -> {SubClassOf(X? Y?), SubClassOf(Y? X?)}
b. EquivalentClasses(X1? .. Xn?) -> {EquivalentClasses(Xi? Xj?) | 1≤i<j≤n}
c. DisjointClasses(X1? .. Xn?) -> {DisjointClasses(Xi? Xj?) | 1≤i<j≤n}
d. SameIndividual(X1? .. Xn?) -> {SameIndividual(Xi? Xj?) | 1≤i<j≤n}
e. DifferentIndividuals(X1? .. Xn?) -> {DifferentIndividuals(Xi? Xj?) | 1≤i<j≤n}
f. SubClassOf(X? ObjectIntersectionOf(Y1? .. Yn?)) ->{ SubClassOf(X? Yi?) | 1≤i≤n}
g. SubClassOf(X? ObjectExactCardinality(Y? Z? W?))->{ SubClassOf(X? ObjectMinCardinality(Y? Z? W?)),
                                                      SubClassOf(X? ObjectMaxCardinality(Y? Z? W?))}
h. SubClassOf(X? DataExactCardinality(Y? Z? W?)) -> { SubClassOf(X? DataMinCardinality(Y? Z? W?)),
                                                      SubClassOf(X? DataMaxCardinality(Y? Z? W?))}
i. DisjointUnion(X? Y1?..Yn?)->{DisjointClasses(Y1?..Yn?),EquivalentClasses(ObjectUnionOf(Y1? .. Yn?) X?)}
j. EquivalentObjectProperties(X1? .. Xn?) -> {EquivalentObjectProperties(Xi? Xj?) | 1≤i<j≤n}
k. EquivalentObjectProperties(X? Y?) -> {SubObjectPropertyOf(X? Y?), SubObjectPropertyOf(Y? X?)}
l. EquivalentDataProperties(X1? .. Xn?) -> {EquivalentDataProperties(Xi? Xj?) | 1≤i<j≤n}
m.EquivalentDataProperties(X? Y?) -> {SubDataPropertyOf(X? Y?), SubDataPropertyOf(Y? X?)}
n. ClassAssertion(ObjectIntersectionOf(X1?..Xn?) Y?)->{ClassAssertion(ObjectIntersectionOf(Xi? Y?)|1≤i≤n}
o. f(ObjectComplementOf(ObjectComplementOf(Y?))) -> {f(Y?))} for any context f
p. f(ObjectComplementOf(ObjectUnionOf(X1? .. Xn?))) -> {f(ObjectIntersectionOf(X1? .. Xn?))}  for any f
```

Fig.2. Ontology axiom re-factoring rules

Figure 3 contains an example "database-style" ontology splitter in an intuitive rule notation, where for each axiom in *X* the first succeeding rule is applied (the isAsserted(A?) predicate is fulfilled by the existence of axiom matching *A?* in *X*). Note that the last CWA(_) line of the example ontology splitter would send all axioms not matching any of the stated OWA-patterns into the IC-part of the ontology).

```
1. refactorRules( a, b, c, f, i, j, k, l, m, n).
2. OWA(SubClassOf(X? Y?) :- isEntity(X?) or ?X-:-ObjectOneOf() , isEntity(Y?) or Y? -:- DataHasValue()).
3. OWA(DisjointClasses(X?, Y?) :- isEntity(X?) or X?-:-ObjectOneOf(), isEntity(Y?) or Y?-:-ObjectOneOf()).
4. OWA(SubObjectPropertyOf(_ X?) :- isAsserted(AnnotationAssertion(X? owlgred_s:isInferred "True"))).
5. OWA(SubDataPropertyOf(_ X?) :- isAsserted(AnnotationAssertion(X? owlgred_s:isInferred "True"))).
6. OWA(InverseObjectProperties()).                 12. OWA(SameIndividuals()).
7. OWA(SymmetricObjectProperty()).                 13. OWA(DifferentIndividuals()).
8. OWA(TransitiveObjectProperty()).                14. OWA(ObjectPropertyAssertion()).
9. OWA(AsymmetricObjectProperty()).                15. OWA(DataPropertyAssertion()).
10. OWA(IrreflexiveObjectProperty()).              16. OWA(DataTypeDefinition()).
11. OWA(ClassAssertion(X? _) :- isEntity(X?)).     17. CWA(_).
```

Fig.3. An example "database-style" ontology splitter

The example “database-style” ontology splitter of Figure 3 restricts the reasoner from inferring the existence of new individuals in the knowledge base or un-stated co-incidence of two differently named individuals. We follow here also principle of *minimal model determinism* (existence of a single “smallest” ME-model in the sense of [6]) for schema+data ontologies (for this reason the disjunctive *SubClassOf* (in superclass position) and *ClassAssertion* axioms are excluded from the OWA part of the ontology). The example ontology splitter is well suited for use together with Stardog OWL/RDF data store with OWL2 RL or OWL 2 DL [18] reasoning enabled.

We note that the axiom-level annotations have not been necessary in the example ontology splitter and that there are only two rules resorting to entity-level annotations; simpler ontology splitters marking all sub-property assertions either as OWA or IC might also be perfectly sensible for database-style use of extended ontologies (the use of sub-property assertions in different senses is up to the used database modeling discipline; a similar situation is also with property domain/range assertions used either for open-world classification, or for closed-world constraint checking).

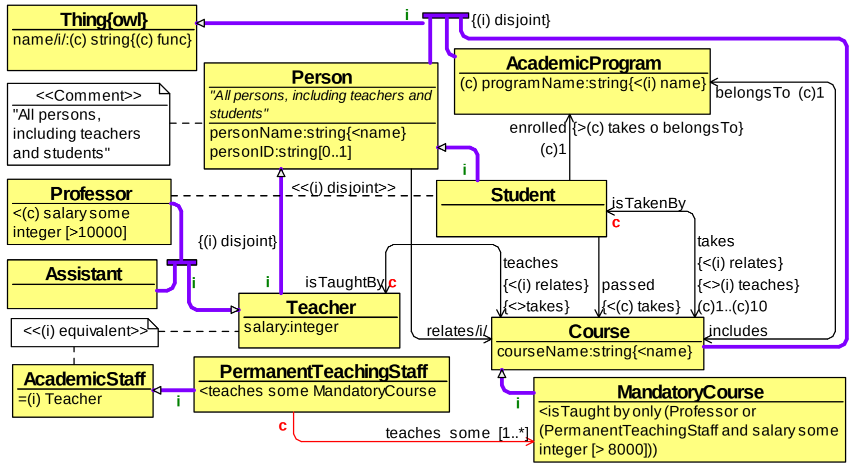

Fig.4. Mini-University ontology, with some axiom level OWA/CWA-specifications

5 Graphical Interface for Entity and Axiom Annotations

Section 4 provides a conceptual base for integrated extended OWA+IC ontology management within single ontology space or UML-style class diagram.

The OWLGrEd editor [14,15] provides a “standard” notation for entity-level annotation (such as comments to the *Person* class in Figure 1 and Figure 4). A convenient custom graphical notation for *isInferred*-annotations (and any other that may be needed for ontology splitters) can be introduced into OWLGrEd along the lines of [19, 20]: we may denote the existence of *owlgred_s:isInferred* annotation for a data or object property by an /i/-suffix added to the property name, as for object property *relates* and data property *name* in Figure 4.

Figure 4 illustrates also a further OWLGrEd notation and editor extension, called OWLGrEd/S (http://owlgred.lumii.lv/s), with specific axiom-level notation for marking concrete axioms as belonging to the OWA or IC part of the extended ontology. The notation allows attaching the mark ‘(i)’ (standing for “inference”, meaning axiom inclusion in OWA-part of the ontology) or ‘(c)’ (standing for “constraint”, meaning axiom inclusion in the IC part) to the visual representations of ontology axioms. We offer explicit means for both marking an axiom to be OWA, or IC, since this axiom-level mark-up can be used in conjunction with other rules in ontology splitter that may be setting different “default” actions for the axiom not having an explicit “semantics markup” attached to it. The explicit (i)/(c) marking is not possible, however, on axiom “part” levels, as it may become possible in ontology splitter via axiom re-factoring. We note also that in the case, if an axiom is reflected in several parts of the diagram, the (i)/(c) marking of any single place of the axiom representation suffices to have the entire axiom marked as OWA/IC, respectively.

The conceptual tool chain for working with OWLGrEd/S editor involves defining or referencing an ontology splitter, then editing the ontology in the editor, possibly assigning the individual axiom markers. Further on two ontologies, say, *open.owl* and *ic.owl* are exported from the editor and can be used in Stardog database environment as ontology and integrity constraints files. Currently we are working with an alternative implementation with an independent ontology splitter, where the ontology is created in OWLGrEd or OWLGrEd/S, exported to Protégé [16], and then split afterwards.

6 Conclusions

The introduction of high-level integrity constraints [6] for RDF/OWL databases may well enhance the use of RDF/OWL technology in real information base development. A suitable visual database schema management environment, like the one offered in this paper, is a critical companion to the new database technology to ensure its widespread use. A key observation presented in this paper is that any existing OWL editor, including the UML-style OWL editors, can be largely used “as is” also for extended OWL+IC knowledge base schema editing by introducing an ontology post-processing step for splitting the ontology into its OWA and IC parts. This adds a new way of OWL+IC ontology management by existing OWL editors, if compared to the approaches studied in [7].

We believe that the UML-style compact OWL editor OWLGrEd, whose one of the main strengths is use of textual OWL Manchester Syntax [17] notation in combination with its UML-style graphics, and its customization OWLGrEd/S for more convenient management of specific entity-level and axiom-level annotations may be well suited for visual UML-style schema editing of extended OWL+IC knowledge bases.

The introduced ontology splitter notion provides also a base for further discussions on natural semantics variants for joint OWA+IC assertion specification within a single ontology schema. It allows also the “power users” of ontology editors to define ontology splitters fitting their specification purposes.

The creation of the OWLGrEd/S editor has been possible do to an open-architecture, model-based and highly customizable OWLGrEd implementation based on TDA platform [21] and the tool definition meta-model [22]. The same architecture allows also expert users of OWLGrEd/S to tailor the appearance and to some extent the functionality of the editor to the user's specific needs. As a possible future work we consider including the support in OWLGrEd and OWLGrEd/S tools for custom integrity constraints specified in some extended OWL notation, or SPARQL [23].

References

1. Smith, M. K.; Welty, C.; and McGuiness, D.: OWL Web Ontology Language Guide, 2004
2. Motik, B; Patel-Schneider P.F; Parsia B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, 2009
3. Motik, B.; Patel-Schneider, P. F.; and Grau, B. C.: OWL 2 Web Ontology Language Direct Semantics, 2009
4. Mazzocchi, S.: Closed World vs. Open World: the First Semantic Web Battle, http://www.betaversion.org/~stefano/linotype/news/91/
5. Motik, B.; Horrocks, I.; and Sattler, U. Bridging the Gap between OWL and Relational Databases. In Proc. of WWW 2007, 807–816, 2007.
6. Tao, J.; Sirin, E.; Bao J; McGuinness, D.: Integrity Constraints in OWL. In Proc. of AAAI 2010, 2010.
7. Sirin, E; Smith, M; Vallace, E: Opening, Closing Worlds – On Integrity Constraints. In Proc. of OWLED 2008, 2008.
8. Stardog, http://stardog.com/
9. Unified Modeling Language: Infrastructure, version 2.1. OMG Specification ptc/06-04-03, http://www.omg.org/docs/ptc/06-04-03.pdf
10. Unified Modeling Language: Superstructure, version 2.1. OMG Specification ptc/06-04-02, http://www.omg.org/docs/ptc/06-04-02.pdf
11. Brockmans, S., Volz, R., Eberhart, A., Löffler, P. Visual Modeling of OWL DL Ontologies Using UML, Proc. of ISWC 2004, LNCS 3298, pp. 198-213, 2004.
12. ODM UML profile for OWL, http://www.omg.org/spec/ODM/1.0/PDF/
13. TopBraid Composer, http://www.topquadrant.com/products/TB_Composer.html.
14. 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.
15. 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, 2010.
16. Protégé 4, http://protege.stanford.edu/
17. OWL 2 Manchester Syntax, http://www.w3.org/TR/owl2-manchester-syntax/
18. Motik, B.; Grau, B. C.; Horrocks, I.; Wu, Z.; Fokoue, A.; Lutz, C.: OWL 2 Web Ontology Language Profiles, 2009
19. Barzdins, J.; Cerans, K.; Liepins, R.; Sprogis, A.: Advanced ontology visualization with OWLGrEd. In Proc. of OWLED 2011, 2011.
20. Cerans, K.; Liepins, R.; Sprogis, A.; Ovcinnikova, J.; Barzdins G.: Domain-Specific Ontology Visualization with OWLGrEd. In Proc. of ESWC'2012, 2012.
21. Barzdins J., Rencis E., and Kozlovics S. The Transformation-Driven Architecture, Proc. of 8th OOPSLA Workshop on Domain-Specific Modeling. Nashville, USA, pp.60-63, 2008.
22. Barzdins, J.; Cerans, K.; Kozlovics, S.; Lace, L.; Liepins, R.; Rencis, E.; Sprogis, A; Zarins, A.: An MDE-based Graphical Tool Building Framework. In Scientific Papers, University of Latvia, Vol 756, ISSN 1407-2157, pp. 121-138, 2010.
23. SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/, 2008.