Advanced OWL 2.0 Ontology Visualization in OWLGrEd

Kārlis ČERĀNS*, Jūlija OVČIŅŅIKOVA*, Renārs LIEPIŅŠ† and Artūrs SPROĢIS†
Institute of Mathematics and Computer Science, University of Latvia
Raiņa bulvāris 29, Riga LV-1459, Latvia
{Karlis.Cerans, Julija.Ovcinnikova, Renars.Liepins, Arturs.Sprogis}@lumii.lv

Abstract. Intuitive ontology visualization is a key for their learning, exchange, as well as their usage in conceptual modeling and semantic database schema design. OWLGrEd is a visual tool for compact graphical UML-style rendering and editing of OWL 2.0 ontologies. We describe here the extensibility features for OWLGrEd that allow tailoring the editor for specific ontology-based modeling needs, including custom entity annotation visualizations and description of integrity constraints for semantic database schemas. We discuss the application of concrete OWLGrEd extensions in the context of ontology-centered information system engineering.

Keywords. OWL, UML/OWL profile, OWLGrEd, graphical ontology visualization, semantic databases, integrity constraints, semantic information systems

Introduction

The semantic technologies, based on RDF [1], RDFS [2] and OWL [3,4] data formats, among the others, offer new perspectives for data organization, management and integration on the basis data conceptual structure that is either explicitly formulated (e.g. in the case of new data designs), or is yet to be recovered via the data semantic re-engineering process. The concept of Semantic Web [5] and, for instance, the publicly available Linked Open Data [6] collection demonstrate the use of semantic technologies on the web scale; these technologies are of not less potential also for enterprise scale where large volumes of heterogeneous and interconnected information has been collected and is intensively used.

The use of RDF/OWL as the backbone of data integration and management infrastructure becomes possible due to a combination of factors such as high-level conceptual modeling constructs, formal semantics and effective reasoning support (cf. e.g. Pellet [7], Fact++ [8] and Hermit [9] reasoners), as well as efficient data stores supporting RDF and OWL data management (see e.g. Virtuoso [10], AllegroGraph [11] and OWLIM [12]). There is a new development of Stardog OWL/RDF database environment [13] that includes use of integrity constraints together with traditionally "open-world" OWL specifications, making it particularly well suited for use in semantic information systems.

A key component in ontology-based information infrastructure is a intuitive ontology presentation and editing notation that allows presenting the existing and well-

* 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.

thought OWL/RDF modeling constructs to the end user. A number of tools and approaches exist for rendering and/or editing OWL ontologies in a graphical form, including UML Profile for OWL DL [14], ODM [15], TopBraid Composer [16], Protégé [17] plug-in OWLViz [18], OWLGrEd [19,20]. Most of these approaches (cf. at least [14,15,19,20]) use some variant or extension of UML class diagrams [21,22] to visualize OWL ontologies. Although there is no one-to-one correspondence between all OWL ontology and UML class diagram concepts, the attempts to adapt UML notation also to OWL ontology management is quite understandable due to the well-thought basis and widespread distribution of UML notation. The benefits of graphical UML-style presentation of ontologies have been observed also in the conceptual study of "Semantic Latvia" information infrastructure [23] and its applications to the semantic data re-engineering task in the medical domain [24,25].

Although the UML-style class diagram notation for basic OWL constructs can be successfully used in OWL ontology rendering and authoring, the use of OWL ontologies in semantic database and information system structure definition introduce the need for further customized or "domain-specific" notations for OWL ontology presentations. Since one of the primary design goals of OWL has been to obtain a decidable logical language, there are rather limited possibilities to introduce extensions into the "logical" part of OWL. There is, however, a rather important construct of annotations in OWL 2.0 [4] that may carry substantial model contents as well as model management information that just does not fit into the "logical" part of the OWL ontology. Our proposal in this paper is to come up with a structural means for defining custom / domain specific ontology annotation assertion visualizations based on specific textual presentations and graphical effects (e.g. diagram symbol appearance change). We base our presentation on the example of OWLGrEd ontology editor, for which we have developed the ontology annotation visual configuration means; however, there should be no principle obstacles to alternative implementations of the notations we are proposing. Some of the principles underlying this work have been already sketched as [26], however, here we are able to present a much detailed design of our ideas, as well as report on an implemented system and give concrete usage examples.

We illustrate our approach by examples of (i) defining "annotation semantics" for advanced UML constructs, such as composition or property derived union (note that these constructs do not have direct counterparts in OWL), (ii) defining database connectivity annotation fields for OWL classes, object and data properties, based on RDB2OWL mapping language for relational database to RDF/OWL format mappings, and (iii) devising notation for entity and axiom level annotations, allowing to incorporate integrity constraints [27,28] that are semantically important in semantic database schema design and that make the extended OWLGrEd editor well suited for schema management in StarDog database environment [13].

1. OWLGrEd Notation and Editor

OWLGrEd[3] provides a complete graphical notation for OWL 2 [4], based on UML class diagrams. It follows the basic principle in UML-style visualization of OWL to visualize an independent hierarchy of ontology classes and then structure the data and object property visualizations along the property domain and range classes.

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

OWLGrEd visualizes OWL classes as UML classes, OWL object properties as association roles and OWL data properties as attributes, as well as OWL individuals as objects[4]. This design decision allows also for easy graphical visualization also of subclass assertions in form of UML generalization (we make use also of UML generalization sets to encode the disjointness or completeness assertions on the subclasses), simple cardinality constraints and inverse-of relations. There is however the need to offer suitable graphical representations also for further OWL ontology constructions (e.g. class expressions, properties with more than one domain assertion, sub-property relations etc.). The design choice of OWLGrEd ontology editor is to use textual OWL Manchester syntax [29] for class expressions where the graphical notation is not available or is not desired.

More precisely, we enrich the UML class diagrams with the new extension notations (cf. [19,20]):

- fields in classes for *equivalent class*, *superclass* and *disjoint class* expressions written in Manchester OWL syntax;
- 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.

OWLGrEd provides option to specify class expressions in compact textual form rather than using separate graphical element for each logical item within class expression. If an expression is referenced in multiple places, it can optionally be shown as an anonymous class. An anonymous class is also used as a base for property domain/range specification, if this domain/range is not a named class.

Figure 1 contains a variant of mini-University ontology, shown in OWLGrEd notation: there are disjoint *Person*, *AcademicProgram* and *Course* classes with their respective subclasses, where the *Teacher* class is specified to be the disjoint union of *Professor*, *AssociateProfessor* and *Assistant* classes. The object properties (e.g. *enrolled*, *belongsTo*, *includes*, *relates*, and *teaches* are ascribed as roles on associations containing their respective domain and range classes. The sub-property relation is depicted using "<" notation, e.g. the *SubProperty(takes relates)* axiom is depicted by *{<relates}* compartment associated with the *takes* property description.

For the notation illustration purpose we have included two depiction forms for axioms *EquivalentClasses(AcademicStaff Teacher)* – the graphical *<<equivalent>>* symbol and the *=Teacher* text in a compartment for *AcademicStaff* class, – and *SubClassOf(PermanentTeachingStaff ObjectSomeValuesFrom(teaches MandatoryCourse))* – the textual form in *PermanentTeachingStaff* class, as well as the red restriction line towards *MandatoryCourse* class symbol. We illustrate also the standard notation for annotations (the comments in the example) that is available in OWLGrEd.

[4] We note that an implementation is underway also for an alternative OWL individual visualization in the list form (this may be essential e.g. for enumerated classes)

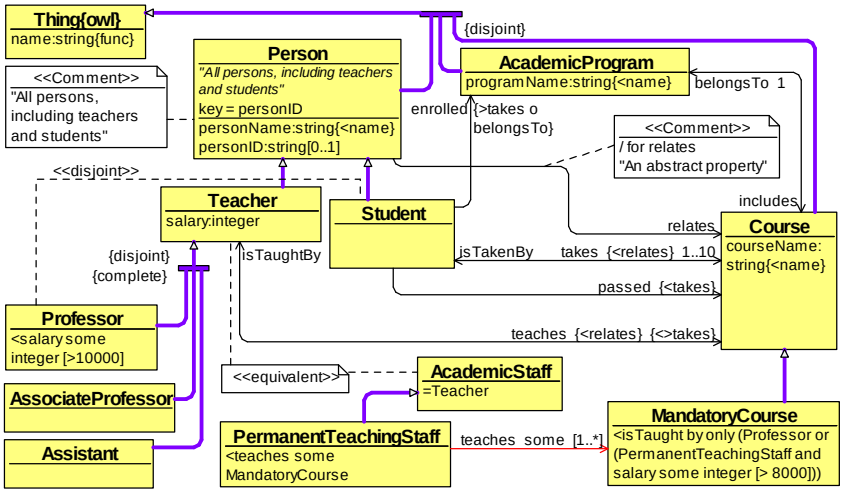

Fig. 1. Example: OWLGrEd notation for a mini-University ontology

We note that the data property *name* that has sub-properties *personName*, *programName* and *courseName* at *Person*, *AcademicProgram* and *Course* classes respectively. A typical UML-style design might have used here the same attribute *name* at all these classes, however, in the case of OWL semantics that would correspond to the domain of *name* being intersection of the classes (not as intended). The single placement of *name* property at *owl:Thing* class provides also a place in the editor for recording its characteristics (e.g. a functional property), annotations and relations to other entities in the ontology.

The OWGrEd editor offers read/write functionality for OWL Functional syntax [4], as well as ontology interoperability (import/export) functionality with Protégé 4.1. ontology editor [17]. The principal OWLGrEd usage tool chains are:

- ontology authoring (create and edit an ontology in OWLGrEd, then save or export it to Protégé and possibly further on to other ontology processing tools)
- ontology visualization (an ontology that is imported from Protégé is displayed graphically to obtain a comprehensible visual view on it); we note also the availability of ontology fragment visualization tools in OWLGrEd.

Any combination of these two OWLGrEd usage patterns, including ontology round-trip engineering between OWLGrEd and Protégé, are possible, as well.

We note that the OWLGrEd editor is implemented within TDA environment [30], on the basis of its Graph Diagram Engine [31], Tool Definition Framework [32] and User Dialogue Engine [33]; the TDA environment itself as well as all its frameworks and engines use data structures formed as MOF-style [34] meta-models that provide for easy configuration means, as well as for extension programming in high level model transformation languages.

2. Custom Ontology Visualizations: The Concepts

Custom ontology visualizations in OWLGrEd ontology editor are defined by means of ontology visualization profiles. Each ontology visualization profile consists of a set of visual item (= abstract field) specifications, where each field comprises:

(i) field type (e.g. textual/boolean(= check box)/combo box field)
(ii) field appearance (e.g. visibility and text font style)
(iii) visual effects on ontology diagram symbols and other fields (e.g. symbol color and shape)
(iv) field semantics (what OWL axioms (e.g. annotation assertions) or axiom annotations a value in the field corresponds to).

For an ontology to be visualized in OWLGrEd in a custom way, the corresponding ontology visualization profile has to be created or imported using OWLGrEd visualization profile plug-in. When ontology created in such domain-specific extension of OWLGrEd is saved in OWL Functional syntax notation or exported to Protégé ontology editor, the ontology diagram node and edge fields that correspond to profile visual items generate the OWL axioms or axiom annotations, as specified in field semantics description.

We recall following [4] that OWL 2 provides both built-in annotation properties (e.g. *rdfs:Label* and *rdfs:Comment* for common *rdfs* namespace, and a number of others), as well as a mechanism for user-defined annotation property introduction (some examples of user defined annotation properties are *A:DBExpr, A:isImportant, A:isComposition* and *A:isDerivedUnion* in the example below in Figure 2). An annotation property can be intuitively thought of as a "type" of individual annotations that ascribe a value to an ontology entity (e.g. an OWL class, object property or data property); these individual annotations relating the annotation property, the annotated OWL entity and the annotation value are called in OWL "annotation assertions". Another use of annotation properties is in annotating not the OWL entities but rather the axioms themselves forming the OWL ontology (we use the term "axiom annotation" in this case).

The most common custom ontology visualization pattern consists in ascribing the specific graphical presentations to the OWL built-in or user-defined annotation properties, with the understanding that the graphical presentation is applied to the ontology entity whenever the entity is annotated by an annotation with the corresponding property[5].

Consider, for example, an ontology *A* fragment that is visualized in a custom way, as in Figure 2. The graphical notation, if compared to the "basic" OWLGrEd ontology editor, has the following "custom" user fields:

- a new class field "DB" rendered textually with prefix "*{DB:*" and suffix "*}*",
- a class field "isImportant" whose value "true" is rendered as orange background and 3D shape of the class symbol,
- association role sub-field "isComposition" whose value "true" is rendered as diamond symbol on opposite association end, and
- association role sub-field "isDerivedUnion" whose value "true" is rendered as prefix "/" to the association role name field.

[5] We note that there can be other axiom patterns that are attached to visual item specifications, for instance, all directly specified sub-classes of some pre-defined class can be marked as red.

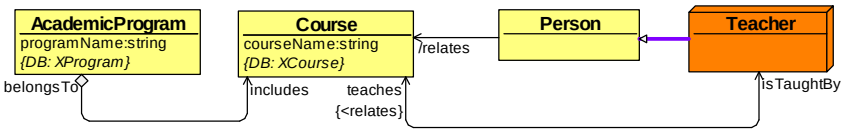

Fig. 2. Simple custom ontology annotation visualization

We desire to have these fields correspond to the following axioms:
AnnotationAssertion(A:DBExpr A:AcademicProgram "XProgram")
AnnotationAssertion(A:DBExpr A:Course "XCourse")
AnnotationAssertion(A:isImportant A:Teacher "true")
AnnotationAssertion(A:isComposition A:includes "true")
AnnotationAssertion(A:isDerivedUnion A:relates "true")

This is achieved by the following semantics declarations, where each declaration contains a pattern for the generation of OWL 2 Functional Syntax axioms on the basis of the concrete field value (the *$value* pattern) and the context (e.g. the diagram element) where the field is placed (the *$subject* pattern):

- *AnnotationAssertion(:DBExpr $subject $value)* for the field "DB",
- *AnnotationAssertion(:isImportant $subject "true")* for the value "true" in the boolean-typed field "isImportant",
- *AnnotationAssertion(:isComposition $subject "true")* for the value "true" in "isComposition", and
- *AnnotationAssertion(:isDerivedUnion $subject "true")* for the value "true" in "isDerivedUnion".

When an ontology that uses the *A:isImportant*, *A:DBExpr* and *A:isComposition* annotations (or other OWL built-in or user defined annotations whose visual image is foreseen in a loaded ontology visualization profile) is imported into OWLGrEd with ontology visualization profile pre-loaded, the editor is able to create the custom visualization (like Figure 2) automatically.

We note that we have demonstrated assigning "annotation semantics" to two typical UML constructs, namely composition and property derived union that are not available within the "logical part" of OWL (the notion of composition does not fit well within the OWL notation framework; the property derived union stating, that a property does not have other subject-to-object relation pairs, than its sub-properties, is not included in the OWL "logics" due to the need to have a decidable reasoning support).

3. Ontology Visualization Profile Specification

In this section we explain in more detail the ontology visualization profile and abstract field concepts outlined in Section 2.

The basic structure and available functionality of the ontology visualization profile is characterized by the meta-model in Figure 3. The core classes of the meta-model are:

- *AA#Profile* – the profile itself,
- *AA#Field* – visual item, understood as visible or invisible diagram element field to be added to the basic editor notation,

- *AA#ChoiceItem* – an item within a fixed drop-down list associated to a field; there can be choice items in both Boolean and textual fields; their typical use is to activate style settings for elements they are placed in, as well as for fields therein, and
- *AA#StyleSetting* – a "style effect" specification either for a newly introduced field itself, or for another diagram item (element, field) on the basis of related choice item or view selection.

Some important additional classes are:

- *AA#ContextType* – the context of the new field in the editor (linked from the field by fieldContext link), i.e. the type of the element (e.g. a class node, or association line) and the place within the element's field structure (e.g. – a top level place in a class box, or a top level place within the association's role description), where the new field is to be added to;
- *AA#Tag* – the tags for special processing of field or choice item values (tags can be ascribed also to visualization profiles themselves); for the custom ontology visualization in OWLGrEd the tags (in their *tagPattern* attribute) contain the field and choice item semantics declarations for ontology import from/export to Protégé ontology editor;
- *AA#View* – a collection of style settings that can be applied both to the standard editor fields and to the ones introduced by the profile; an example use of views is to enable showing/hiding the custom field information in the ontology diagram, moreover, any functionality exposed by style settings can be induced also by views over any element or field type in the diagram (the combination of choice item and view conditions for style settings is possible, as well).

The *AA#Configuration* and *AA#TagType* classes are meant to help the visualization profile designer by pre-defining the context and tag types that can be used to structure the visual profile definition environment (e.g. by providing tag labels and available context type structure). The *AA#Translet* class allows attaching specific procedures for handling the fields during their processing within editor's property dialogues.

We note that the *hasMirror*/*addMirror* notation e.g. in *AA#ContextType* and *AA#ViewStyleSetting* classes is also meant to ease the work of visualization profile designer by allowing specifying the custom fields and style effects for only one of symmetric line ends (e.g. the association ends) in the editor.

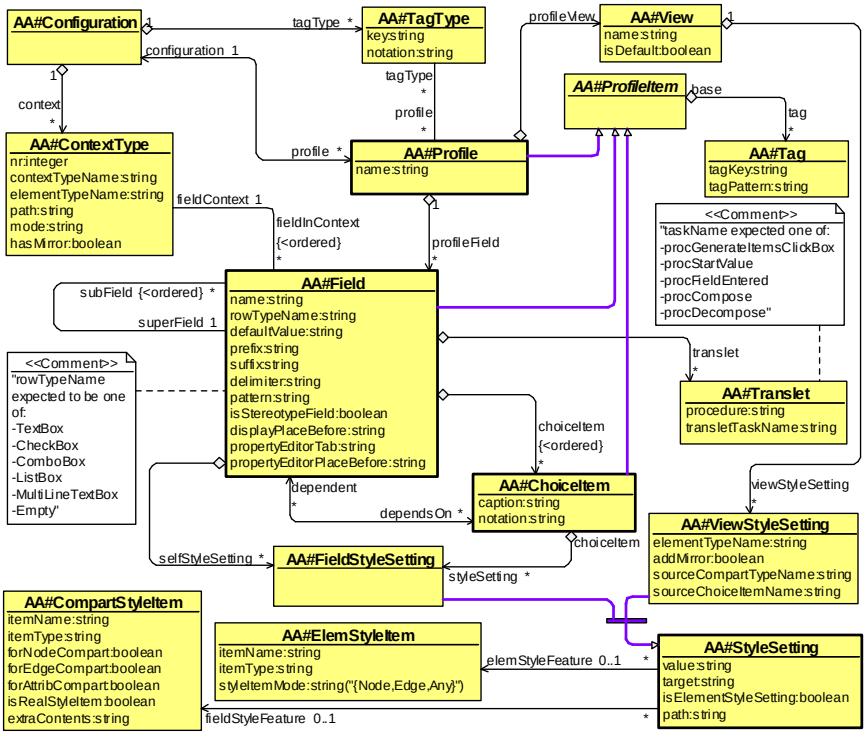

Fig. 3. Visualization profile meta-model

We add some further explanations on *AA#Field* and *AA#StyleSetting* classes in the visualization profile meta-model.

The *rowTypeName* attribute of *AA#Field* indicates the type of the field (visual item) to be added to the diagram elements; we note sub-fields are allowed to *TextBox* and *MultiLineTextBox* fields only, while choice items may correspond only to *CheckBox*, *ComboBox* and *ListBox* type fields.

The field structure specification in *AA#Field* class determines both the fields' placement within the graphical diagram representation (the *displayPlaceBefore* attribute) and within the corresponding element's property dialogue within the editor (the *propertyEditorTab* and *propertyEditorPlaceBefore* attributes). The choice to offer in the field attributes *defaultValue*, *prefix*, *suffix* (relating the field value in the editor and in the graphical presentation), *delimiter* (for multi-line fields) and *pattern* (the symbols allowed in the field) in the visualization profile meta-model is related to the TDA Tool Definition Framework [32] implementation of these attributes.

The *isStereotype* attribute for *AA#Field* marks the field as "stereotype", with the meaning that its choice items are allowed to have dependent fields (the fields that are present in the element only if there is a corresponding choice item activated); the stereotype fields, however, are not allowed to be dependent fields themselves. We note that this construction allows simulating of UML stereotype tagged values.

The style settings in *AA#StyleSetting* class describe the "style effects" brought to the editor by the instance of the visualization profile. Each style setting has the following components:

- the source – a new field added to the diagram element (the field self-setting), a choice item to be activated, or a view applied to the diagram;
- the target (the *target* attribute in *AA#StyleSetting* class) element or field to be affected by the style setting;
- the style item – what style component (e.g. box shape, color, line strength, or field font face, size, etc.) is to be affected;
  the available "real" style items are defined by TDA Graph Diagram Engine [31], and are reproduced also here in Figure 4; furthermore there are "style items" for textual target fields that allow adding an extra prefix or suffix to the field (the concrete prefix/suffix is specified in the *extraContents* attribute of *AA#CompartStyleItem* class);
- the value the concrete style item is to assume (there are different possible value sets for different style items).

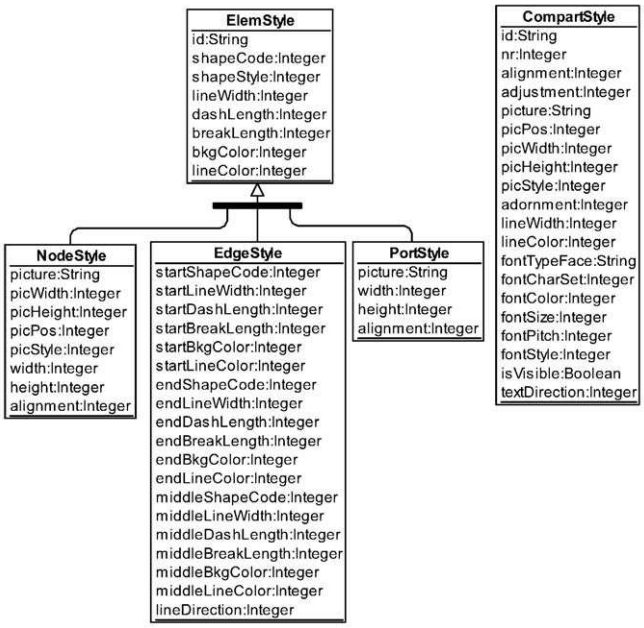

Fig. 4. Element and field style settings in TDA Graph Diagram Engine (reproduced from [31])

We note that the diagram visualization profile mechanism is implemented as a plug-in to OWLGrEd editor[6], and it allows defining, importing and exporting profiles, each profile consisting of field definition set, as well as defining graphical views and applying these to the diagrams (a view may change appearance for certain box, line and field types, including an option of hiding certain field types from a diagram).

[6] See the current version at http://owlgred.lumii.lv/plugin/CustomUserFields/

4. Ontology Visualization Profile Use Cases

We demonstrate the use of the developed notation and technology on the concrete examples arising in the context of semantic database engineering. The possibility to define "annotation semantics" for typical UML constructs such as composition and property derived union has been demonstrated already in Section 2. Here we proceed to examples of database connectivity notation and integrity constraint modeling.

4.1. Database Connectivity Notation for OWL Ontologies

The creation of semantic information infrastructure includes the need to populate it from the existing (legacy) data that are typically stored in the form of relational databases. The task of semantic re-engineering of relational database data involves specification of mappings from the relational database to RDF/OWL format. The custom OWL ontology visualization format allows for creation of simple visualization profile e.g. for ontology entity annotations by RDB2OWL expressions [35,36] that document the relation of OWL ontology entities to their data counterparts in the relational databases[7].

Figure 5 contains a re-engineering of mini-University ontology with RDB2OWL annotations using OWLGrEd with DBExpr-ontology visualization profile (observe a similar figure in [35]).

The implementation of the profile is achieved by adding multi-line *DBExpr*-fields with semantics tags *AnnotationAssertion(owlFields:DBExpr $subject $value)*:

- to the Class symbol,
- as a subfield to Association role field
- as a subfield to Attribute field within the class symbol.

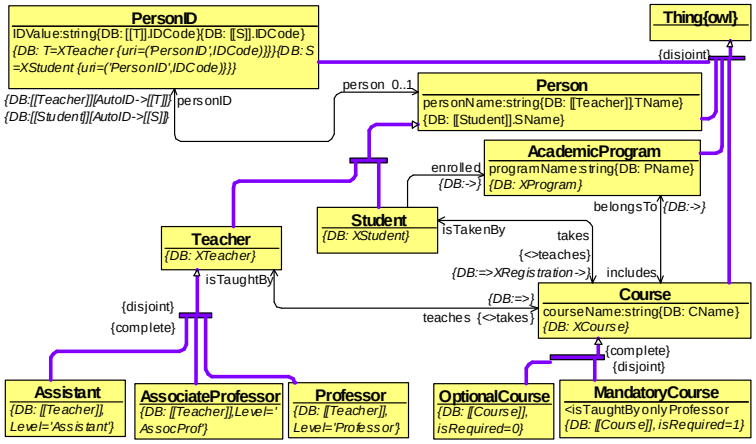

Fig. 5. RDB2OWL database connectivity assertion visualization

[7] We refer the reader to [27] for discussion of different RDB-to-RDF/OWL mapping formalisms of which RDB2OWL is an option; our primary aim here is to illustrate the applicability of our ontology visualization technology.

Some of the annotation assertions, generated from DBExpr-annotations during the ontology export from OWLGrEd to Protégé are, as follows:

```
AnnotationAssertion(owlFields:DBExpr :AcademicProgram "XProgram")
AnnotationAssertion(owlFields:DBExpr :Assistant "[[Teacher]],Level='Assistant'")
AnnotationAssertion(owlFields:DBExpr :AssociateProfessor "[[Teacher]],Level='AssocProf'")
AnnotationAssertion(owlFields:DBExpr :Course "XCourse")
AnnotationAssertion(owlFields:DBExpr :courseName "CName")
AnnotationAssertion(owlFields:DBExpr :enrolled "->")
AnnotationAssertion(owlFields:DBExpr :IDValue "[[S]].IDCode")
AnnotationAssertion(owlFields:DBExpr :IDValue "[[T]].IDCode")
AnnotationAssertion(owlFields:DBExpr :MandatoryCourse "[[Course]], isRequired=1")
AnnotationAssertion(owlFields:DBExpr :OptionalCourse "[[Course]],isRequired=0")
AnnotationAssertion(owlFields:DBExpr :personID "[[Student]][AutoID->[[S]]")
AnnotationAssertion(owlFields:DBExpr :personID "[[Teacher]][AutoID->[[T]]")
AnnotationAssertion(owlFields:DBExpr :PersonID "S=XStudent {uri=('PersonID',IDCode)}}")
AnnotationAssertion(owlFields:DBExpr :PersonID "T=XTeacher {uri=('PersonID',IDCode)}}")
AnnotationAssertion(owlFields:DBExpr :personName "[[Student]].SName")
AnnotationAssertion(owlFields:DBExpr :personName "[[Teacher]].TName")
AnnotationAssertion(owlFields:DBExpr :Professor "[[Teacher]],Level='Professor'")
AnnotationAssertion(owlFields:DBExpr :programName "PName")
AnnotationAssertion(owlFields:DBExpr :Student "XStudent")
AnnotationAssertion(owlFields:DBExpr :takes "=>XRegistration->")
AnnotationAssertion(owlFields:DBExpr :Teacher "XTeacher")
AnnotationAssertion(owlFields:DBExpr :teaches "=>")
```

4.2. Integrity Constraints in RDF/OWL Database Schema Design

The use of standard OWL "open-world assumption" semantics [37] may in certain cases of semantic database schema specification produce un-intended results. For instance, in the case of ontology of Figure 1:

- if a teacher *X* who is not a professor (e.g. an assistant) has registered by an error as taking (*takes*) a course, instead of teaching (*teaches*) it, the system infers that *X* is a student since only students are allowed to take a course;
- if a course belongs to two academic programs (with no names specified yet), these would be inferred to be the same academic program;
- if there is a student not taking any course, the system will not regard this as a problem, since the course might not yet be specified;
- 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 (the OWL ontology statements interpreted in "closed world" sense and not used in the OWL inference but are just checked, if they are satisfied by the present data) [27,28] are nowadays commonly invoked to handle these situations, supported also by the StarDog database environment[8] .

We do not go into details of integrity constraint specification alternatives but just present an ontology visualization profile[9] for OWLGrEd that that foresees a possibility to attach an (i)/(c)-mark ("i" for inference, "c" for constraint) to visual places that can be identified as "holding" the concrete axioms, as in Figure 6 for mini-University.

[8] http://stardog.com/

[9] The extended editor is available as OWLGrEd/S from http://owlgred.lumii.lv/s

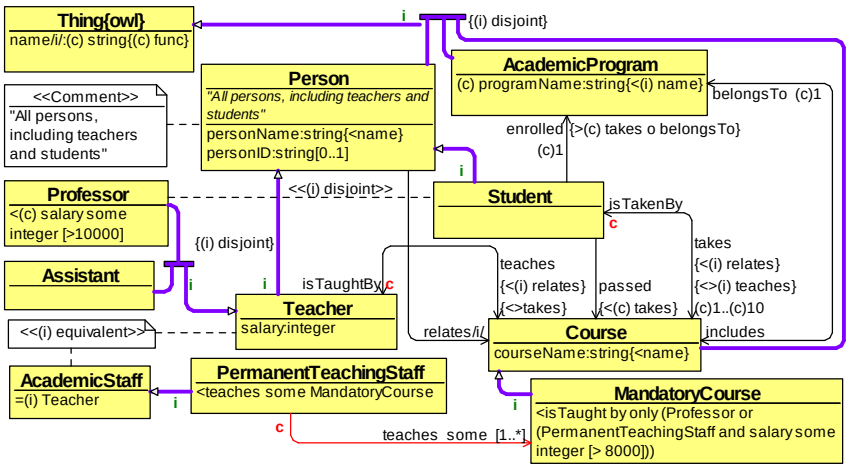

Fig. 6. Integrity constraint specification for mini-University ontology

Some positions for (i)/(c)-markings in the extended editor are, as follows:

- object property line start and end positions, reflecting domain and range assertions for the property (e.g. property *teaches* or *takes* in Figure 6);
- data property name prefix, reflecting data property domain assertion;
- equivalent classes, disjoint classes and superclasses assertions within class nodes (e.g. assertion < *salary some integer [>10000]* in *Professor* class); in a similar way the notation is extended also to equivalent, disjoint and super-properties (e.g. *<(i)name* assertion for *personName* property in *Person* class)
- property chain assertions
- object and data property characteristics (e.g. *{(c)func}* for *name* in class *Thing*)
- cardinality restrictions (e.g. *(c)1..(c)10* cardinality for *takes*)
- generalization lines (*SubClassOf*-markers) in the graphical form (e.g. for subclasses of *Thing*, *Person* and *Teacher* classes)
- disjoint/complete assertions placed at generalization set descriptors (forks), e.g. the fork joining subclasses of *Teacher* class.

In the example, for instance, the axiom *ObjectPropertyDomain(A:takes A:Student)* is annotated to become *ObjectPropertyDomain(Annotation(C:isConstraint "true") A:takes A:Student)* for a suitable namespace *C* holding the *isConstraint* annotation property. The visual **c**-notation placed at the beginning of *takes*-role link is obtained from a "DomainMode" field under the association role *takes*. The corresponding semantics specification for the "DomainMode" field causing the considered *ObjectPropertyDomain*-axiom annotation is *Annotation(C:isConstraint "true")*.

5. Conclusions

The presentation and examples considered here outline the potential of custom ontology visualization profiles in OWLGrEd, as well as their importance in the

situation of data structure specification for semantic information systems. There should be no principal problems to use the developed framework also for annotations to be added to OWL for specifying user interface form generation on the basis of the OWL ontology structure, or different kinds of integrity constraints specification (e.g. the ones written in SPARQL [38] language).

The reader is invited also to come up with specific notation for his/her own favorite or custom OWL annotation properties.

Given the generic nature of the diagram visualization profiles it would not be difficult to apply the constructs developed here also to other editors created within the TDA + TDMM environment (see e.g. [39]). We note also that style setting on style attribute level has been done here for the first time for the TDA environment.

Due to the open and model-based structure of the TDA environment [30] and its Tool Definition Framework [32] where the OWLGrEd editor is implemented in, there has been a possibility to implement the ontology visualization profile mechanism (including profile configuration form description) by MDA-style high level model transformations, written in LuA library lQuery [40].

References

[1] Resource Description Framework (RDF), http://www.w3.org/RDF/
[2] RDF Vocabulary Description Language: RDF Schema, http://www.w3.org/TR/rdf-schema/
[3] Smith, M. K.; Welty, C.; and McGuiness, D.: OWL Web Ontology Language Guide, 2004
[4] Motik, B; Patel-Schneider P.F; Parsia B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, 2009
[5] Tim Berners-Lee, James Hendler and Ora Lassila, "The Semantic Web", Scientific American, May 2001, p. 29-37.
[6] Linked Data, http://linkeddata.org
[7] Pellet, reasoner http://clarkparsia.com/pellet
[8] FaCT++, reasoner, http://owl.man.ac.uk/factplusplus/
[9] Hermit OWL Reasoner, http://hermit-reasoner.com/
[10] http://virtuoso.openlinksw.com/
[11] AllegroGraph, http://www.franz.com/agraph/allegrograph/
[12] OWLIM, http://www.ontotext.com/owlim
[13] StarDog, http://stardog.com
[14] 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.
[15] ODM UML profile for OWL, http://www.omg.org/spec/ODM/1.0/PDF/
[16] TopBraid Composer, http://www.topquadrant.com/products/TB_Composer.html.
[17] Protégé 4, http://protege.stanford.edu/
[18] OWL Viz, http://www.co-ode.org/downloads/owlviz/
[19] 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.
[20] 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.
[21] Unified Modeling Language: Infrastructure, version 2.1. OMG Specification ptc/06-04-03, http://www.omg.org/docs/ptc/06-04-03.pdf
[22] Unified Modeling Language: Superstructure, version 2.1. OMG Specification ptc/06-04-02, http://www.omg.org/docs/ptc/06-04-02.pdf
[23] J.Barzdins, G.Barzdins, R.Balodis, K.Cerans, et.al.: (2006). Towards Semantic Latvia. In Communications of 7th International Baltic Conference on Databases and Information Systems, pp.203-218.
[24] G.Barzdins, E.Liepins, M.Veilande, M.Zviedris: Semantic Latvia Approach in the Medical Domain. Proc. 8th International Baltic Conference on Databases and Information Systems. H.M.Haav, A.Kalja (eds.) Tallinn University of Technology Press, pp. 89-102. (2008).

[25] G.Barzdins, S.Rikacovs, M.Veilande, and M.Zviedris: Ontological Re-engineering of Medical Databases, Proceedings of the Latvian Academy of Sciences. Section B, Vol. 63 (2009), No. 4/5 (663/664), pp. 20–30.
[26] Barzdins, J.; Cerans, K.; Liepins, R.; Sprogis, A.: Advanced ontology visualization with OWLGrEd. In Proc. of OWLED 2011, 2011.
[27] Tao, J.; Sirin, E.; Bao J; McGuinness, D.: Integrity Constraints in OWL. In Proc. of AAAI 2010, 2010.
[28] Sirin, E; Smith, M; Vallace, E: Opening, Closing Worlds – On Integrity Constraints. In Proc. of OWLED 2008, 2008.
[29] OWL 2 Manchester Syntax, http://www.w3.org/TR/owl2-manchester-syntax/
[30] Barzdins J., Rencis E., and Kozlovics S. The Transformation-Driven Architecture, Proc. of 8th OOPSLA Workshop on Domain-Specific Modeling. Nashville, USA, 2008, pp.60-63.
[31] Barzdins J., Cerans K., Kozlovics S., Rencis E., and Zarins, A. A Graph Diagram Engine for the Transformation-Driven Architecture, Proc. of 4th International Workshop of Model-Driven Development of Advanced User Interfaces, Florida, USA, 2009, pp.29-32.
[32] J. Barzdins, K. Cerans, S. Kozlovics, L. Lace, R. Liepins, E. Rencis, A. Sprogis, A. Zarins. An MDE-based Graphical Tool Building Framework. In Scientific Papers, University of Latvia, 2010, Vol 756, ISSN 1407-2157, pp. 121-138
[33] S. Kozlovics, A Dialog Engine Metamodel for the Transformation-Driven Architecture. . In Scientific Papers, University of Latvia, 2010, Vol 756, ISSN 1407-2157, pp. 151-170
[34] OMG's MetaObject Facility, http://www.omg.org/mof/
[35] 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.
[36] 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
[37] Motik, B.; Patel-Schneider, P. F.; and Grau, B. C.: OWL 2 Web Ontology Language Direct Semantics, 2009
[38] SPARQL 1.1 Query Language, http://www.w3.org/TR/2010/WD-sparql11-query-20100601/
[39] J. Barzdins, K. Cerans, A. Kalnins, M. Grasmanis, S. Kozlovics, L. Lace, R.Liepins, E. Rencis, A. Sprogis, A. Zarins. Domain Specific Languages for Business Process Management: a Case Study. Proceedings of DSM'09 Workshop of OOPSLA 2009, Orlando, Florida, USA, pp. 34 – 40, 2009.
[40] R. Liepiņš. Library for model querying – lQuery. In Proceedings of Workshop on OCL and Textual Modelling, 2012.