Advanced UML Style Visualization of OWL Ontologies

Jūlija Ovčiņņikova*, Kārlis Čerāns*

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

Abstract. The OWLGrEd ontology editor allows graphical visualization and authoring of OWL 2.0 ontologies using a compact yet intuitive presentation that combines UML class diagram notation with textual OWL Manchester syntax for expressions. We describe here the approaches available for ontology presentation fine tuning within the OWLGrEd editor, namely the ontology visualization option framework and the editor plug-in mechanism together with concrete plug-ins aimed to enhance the ontology presenting and editing experience.

Keywords: OWL, OWLGrEd, UML-style ontology visualization, ontology visualization options

1 Introduction

Presenting OWL ontologies [1] in a comprehensible form is important for ontology designers and their users alike. The graphical form in general and UML class diagram notation in particular offers an option of basic visual ontology construct presentation that allows linking together constructs that are related in the ontology (e.g. an object property can be depicted as a line connecting its domain and range class boxes). UML class diagrams need to be extended to cope with full OWL 2.0 construct modeling. This is solved in different graphical notations in different ways. So, ODM [2], defines a UML profile for ontology presentation and OWLGrEd [3] and TopBraid Composer [4] integrate OWL Manchester Syntax [5] for presenting advanced OWL constructs in textual form. The uniqueness of OWLGrEd lies in its combined ability to lay out an ontology that is imported or created in the editor in a compact graphical UML-style form, and make further manual ontology editing/adjustment. VOWL [6] visualizes ontologies by another approach using graphical primitives both for object and data property presentation, so obtaining a more uniform ontology presentation in a dynamic, yet read-only, graph-like form. The VOWL presentation of the same ontology will also typically require more graphical elements, than OWLGrEd.

The real strength of ontology presentation in an editor like OWLGrEd comes from the user's ability to fine-tune the ontology diagram after its automated rendering to obtain documentation-ready ontology presentations. Such a fine tuning may involve the

* Supported, in part, by Latvian State Research program NexIT project No.1 "Technologies of ontologies, semantic web and security".

diagram object repositioning, as well as its re-structuring up to full ontology editing facilities available in the tool (including e.g. manual deletion of irrelevant information).

In order to achieve a high quality ontology presentation in the tool, even in the presence of manual fine tuning options available, the quality of first ontology diagram created upon the import of the ontology into the tool still remains very important. Since the UML diagrams, as well as the OWLGrEd notation allow for different presentations of the same semantic elements (e.g. a graphical and textual one), and different ontologies may correspond to different desirable ontology presentation options, we describe here a re-factored ontology visualization option framework offering a number of choices that the user can make already before importing the ontology into the tool.

We describe here also a number of OWLGrEd plugins that can be used for diagram refactoring services, as well as its structural and semantics extensions. This part of work extends the earlier authors' work on domain specific ontology visualizations [7,8] (this paper describes new re-factoring services plug-in, as well as reviews the plug-in mechanism and concrete plugin architecture from the ontology visual presentation perspective). The editor plugins described here are included in the OWLGrEd tool download and can be activated on-demand for concrete projects. The OWLGrEd editor with pre-installed configuration, as described here, is available at http://owlgred.lumii.lv/pp.

2 OWLGrEd Notation and Editor

OWLGrEd (http://owlgred.lumii.lv/) provides a graphical notation for OWL 2 [1], based on UML class diagrams. OWL classes are typically visualized as UML classes, data properties as class attributes, object properties as association roles, individuals as objects, cardinality restrictions on association domain class as UML cardinalities, etc. The UML class diagrams are enriched with new extension notations, e.g. (cf. [3,9]):

- fields in classes for *equivalent class*, *superclass* and *disjoint class* expressions written in Manchester OWL syntax [5];
- fields in association roles 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;
- 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.

Fig. 1 contains a simple demonstration fragment of Latvian Medicine Registries ontology [10] in OWLGrEd notation, illustrating the class, data and object property, as well as subclass, sub-property and object property restriction notation, different ways of disjoint classes notations, class-level inline comments and ontology level annotations.

The OWLGrEd tool allows both for ontology authoring (with option to save the ontology in a standard textual format) and ontology visualization that includes automated ontology diagram formation and layouting step, followed by optional manual diagram fine tuning to obtain the highest quality rendering of the ontology.

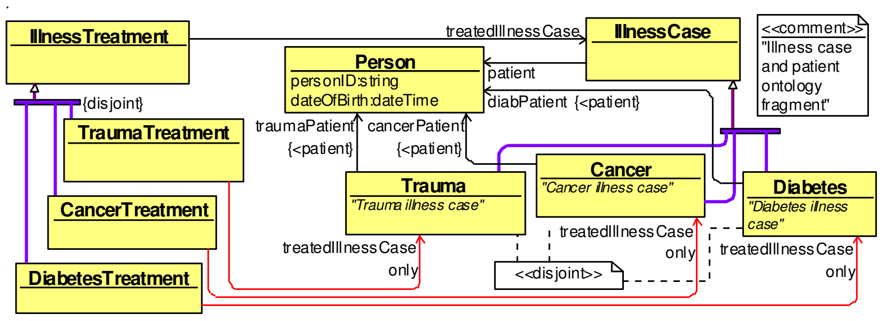


Fig. 1. Demo fragment of Latvian Medicine Registries Ontology

3 Ontology Visualization Parameters

The UML notation provides several options for presenting its semantic elements in different visual ways. This principle is kept also in OWLGrEd by including both graphical and textual notations for such semantic elements as subclass relations, object property relations, annotations and object properties as association roles or as attributes.

The automatic ontology visualization in OWLGrEd by default shall use the graphical notation, if there is no clear reason for switching to textual one. Fig. 2 shows a larger fragment of Medicine Registries Ontology in a graphical notation for all object properties and object property restrictions, and with separate disjoint classes axiom rendering.

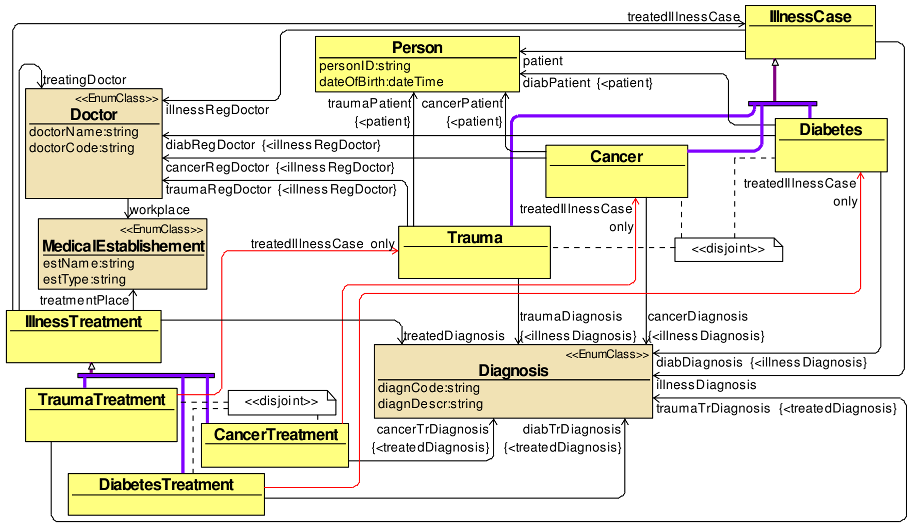
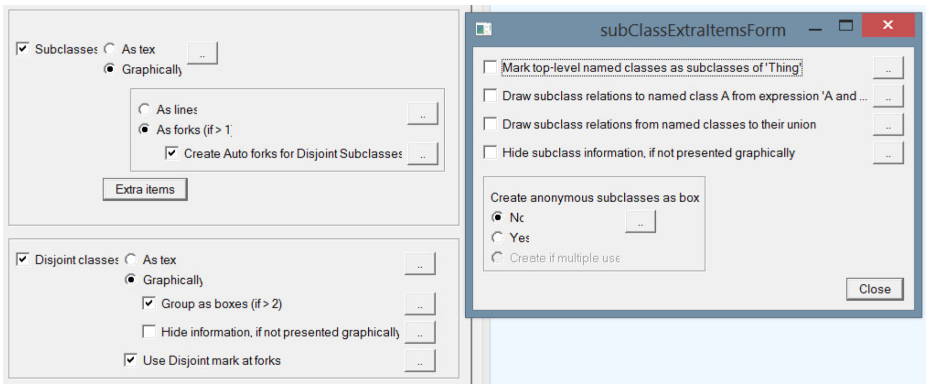


Fig. 2. Medicine Registries Ontology: a larger fragment in a graphical notation

For different ontologies and their parts, the appropriate element visualization methods may differ depending on the ontology content, e.g. if the ontology contains a small number of object property restrictions, a better way to display object property restrictions would be graphical. However, it would not carry much information to display graphically object property restrictions that are based on owl:Thing as the target class. A textual form object property restriction visualization may be favorable also in the presence of a large number of these restrictions in the ontology.

The ontology visualization parameter framework offers parameters for inclusion / not inclusion of different object types in ontology visualization (e.g. one can choose to visualize ontology without data property annotations or without disjoint class axioms). Most of the framework parameters, however, refer to different ways (e.g. a graphical, or a textual form) of different type entity and axiom information visualization.

Figure 3 illustrates parameter setting interface including the option to represent subclass relations as text or graphically. In graphical mode there is a possibility to draw subclass relations as lines or combine them through forks, if there are more than one subclass for a given class.


✓ Subclasses: As tex [✓ Graphically]
- As lines [✓ As forks (if > 1]
- ✓ Create Auto forks for Disjoint Subclasses

Extra items

✓ Disjoint classes: As tex [✓ Graphically]
- ✓ Group as boxes (if > 2)
- Hide information, if not presented graphically
- ✓ Use Disjoint mark at forks

subClassExtraItemsForm
- Mark top-level named classes as subclasses of 'Thing'
- Draw subclass relations to named class A from expression 'A and ...
- Draw subclass relations from named classes to their union
- Hide subclass information, if not presented graphically

Create anonymous subclasses as box: [✓ No] Yes Create if multiple use

Close

Fig. 3. Ontology Visualization Parameter form illustration

There are also parameters for disjoint class axiom rendering in Fig. 3. Along with disjoint classes representation as text or graphically (in a separate graphical node), there is a possibility to group disjointness relations, into boxes if there are more than two disjoint classes, and it is possible to hide information if it is not presented graphically.

An important option in UML is a possibility to mark the generalization sets as disjoint. This notation is brought also over to OWLGrEd since it can greatly simplify the graphical diagram appearance, if the disjointness axiom matches a level within the class hierarchy. If the 'Use Disjoint mark at forks' box is checked in the import parameter form, the ontology importer is able to identify the cases when all class subclasses are disjoint and add specific {disjoint} mark to the generalization element (cf. the generalization set notation for the subclasses of the *IllnessTreatment* class in Fig. 1).

Figure 3 also shows possible extra choices that can be made for the subclass relations, such as "Mark top-level named classes as subclasses of 'Thing'", "Draw subclass relations to named class A from expression 'A and ...'", "Draw subclass relations from named classes to their union", "Hide subclass information, if not presented graphically" and an option to choose whether to create anonymous subclasses as boxes, or not. The extra parameters are left unchecked by default to obtain less loaded graphical presentation, however they can be turned on to show more clear class hierarchy, if desired.

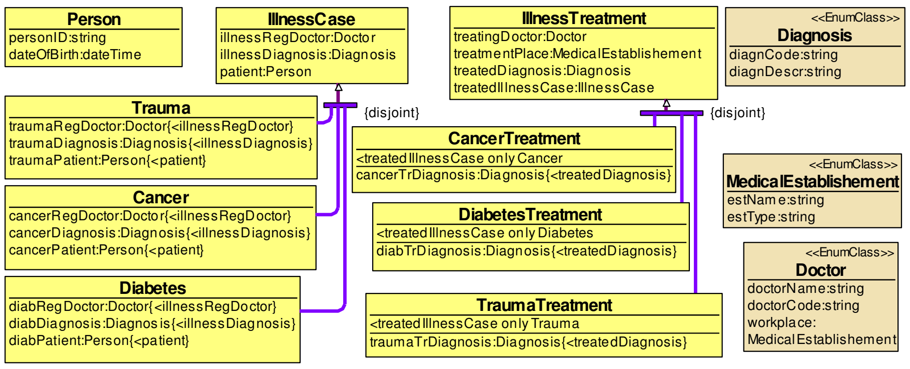

Person
personID:string
dateOfBirth:dateTime
IllnessCase
illnessRegDoctor:Doctor
illnessDiagnosis:Diagnosis
patient:Person
IllnessTreatment
treatingDoctor:Doctor
treatmentPlace:MedicalEstablishement
treatedDiagnosis:Diagnosis
treatedIllnessCase:IllnessCase
<<EnumClass>>
Diagnosis
diagnCode:string
diagnDescr:string
Trauma
traumaRegDoctor:Doctor{<illnessRegDoctor}
traumaDiagnosis:Diagnosis{<illnessDiagnosis}
traumaPatient:Person{<patient}
{disjoint}
CancerTreatment
<treatedIllnessCase only Cancer
cancerTrDiagnosis:Diagnosis{<treatedDiagnosis}
{disjoint}
<<EnumClass>>
MedicalEstablishement
estName:string
estType:string
Cancer
cancerRegDoctor:Doctor{<illnessRegDoctor}
cancerDiagnosis:Diagnosis{<illnessDiagnosis}
cancerPatient:Person{<patient}
DiabetesTreatment
<treatedIllnessCase only Diabetes
diabTrDiagnosis:Diagnosis{<treatedDiagnosis}
<<EnumClass>>
Doctor
doctorName:string
doctorCode:string
workplace:
MedicalEstablishement
Diabetes
diabRegDoctor:Doctor{<illnessRegDoctor}
diabDiagnosis:Diagnosis{<illnessDiagnosis}
diabPatient:Person{<patient}
TraumaTreatment
<treatedIllnessCase only Trauma
traumaTrDiagnosis:Diagnosis{<treatedDiagnosis}

Fig. 4. Demo fragment of Latvian Medicine Registries Ontology: textual notation

Figure 4 illustrates the same Medicine Registries Ontology of Fig.2 with subclass relations presented graphically and disjoint marks at forks, while presenting textually the object properties and object property restrictions. The obtained ontology visualization has less elements that can be a benefit, however, the textual representation of object properties and object property restrictions somewhat hides the ontology structure.

4 Ontology Editor Plugins

To enhance the ontology visualization fine tuning experience after the ontology initial loading into the tool, as well as to support different custom ontology visualization means and the ontology editing work in general, the OWLGrEd ontology editor can be extended by means of plugins. The ontology editor plugins offer editor notation and environment extension means, in a similar manner, as profiles do for UML class diagrams [11,12]. A plugin to the ontology editor, may include structural editor symbol extensions with fields and visual effects, as well as editor behavior extensions.

The OLWGrEd plugin for ontology re-structuring includes transformations for adjusting the ontology visual presentation after its automatic visual rendering. It allows switching between the textual and graphical presentation form for concrete object property and object property restriction presentations within the ontology diagram, as well as re-factoring individual disjoint classes axiom presentation (e.g. from a separate disjoint-box to the 'disjoint' mark at the sub-class form symbol). The plugin can be used to transform e.g. the ontology diagram of Fig. 2 or Fig. 4 into the one of Fig. 5, where certain object properties (in the concrete example – the ones not having super-properties) are displayed in a graphical form to balance the structure presentation and compactness requirements. For an experienced OWLGrEd user it should take about 5 minutes to change manually the ontology diagram of e.g. Fig. 2 into the one of Fig. 5.

The ontology presentation in Fig. 5, as well as the presentation in Fig. 2 and Fig. 4 uses also a custom "enumerated class" boolean user field presentation.

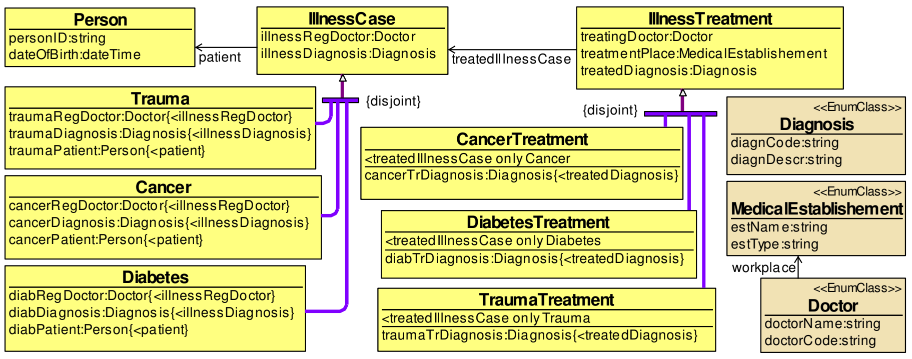

Person
personID:string
dateOfBirth:dateTime
patient
IllnessCase
illnessRegDoctor:Doctor
illnessDiagnosis:Diagnosis
treatedIllnessCase
IllnessTreatment
treatingDoctor:Doctor
treatmentPlace:MedicalEstablishement
treatedDiagnosis:Diagnosis
{disjoint}
{disjoint}
Trauma
traumaRegDoctor:Doctor{<illnessRegDoctor}
traumaDiagnosis:Diagnosis{<illnessDiagnosis}
traumaPatient:Person{<patient}
Cancer
cancerRegDoctor:Doctor{<illnessRegDoctor}
cancerDiagnosis:Diagnosis{<illnessDiagnosis}
cancerPatient:Person{<patient}
Diabetes
diabRegDoctor:Doctor{<illnessRegDoctor}
diabDiagnosis:Diagnosis{<illnessDiagnosis}
diabPatient:Person{<patient}
CancerTreatment
<treatedIllnessCase only Cancer
cancerTrDiagnosis:Diagnosis{<treatedDiagnosis}
DiabetesTreatment
<treatedIllnessCase only Diabetes
diabTrDiagnosis:Diagnosis{<treatedDiagnosis}
TraumaTreatment
<treatedIllnessCase only Trauma
traumaTrDiagnosis:Diagnosis{<treatedDiagnosis}
<<EnumClass>>
Diagnosis
diagnCode:string
diagnDescr:string
<<EnumClass>>
MedicalEstablishement
estName:string
estType:string
workplace
<<EnumClass>>
Doctor
doctorName:string
doctorCode:string

Fig. 5. Demo fragment of Latvian Medicine Registries Ontology: custom notation with plugins

In general, the editor extensions with symbol fields and graphical effects enhance the ontology presentation options by introducing domain-specific notations into the ontology presentation (the extensions can be configured to take into account the domain specific notations also during the ontology import); they are handled by a generic User Fields plugin [8] that is currently part of the default OWLGrEd editor configuration.

The available editor enhancements, supported by the User Fields plugin [8] include:

- custom fields, together with their semantics mappings (e.g. “enumerated class”);
- custom visual effects for text and choice fields and symbols dependent on concrete text or choice field values (e.g. a brown/darker enumerated class color);
- views applying certain visual effects to the entire diagram (e.g. hiding certain information from the presentation). The default OWLGrEd configuration includes views for horizontal and vertical alignment, and hiding all annotations.

An example of ontology editor plugin that involves both the user fields and custom functionality definition is the data ontology support plugin [13]; it contains the above considered “enumerated class” field definition. It also adds extra maximum cardinality 1 axioms to all data and object properties that are rendered in the editor in the textual form and do not have explicit cardinality specification. The plugin also records the class attribute ordering into an annotation so that this ordering could be further used in automated user interface generation for ontology-conformant data browsing (cf. [13]).

The other currently maintained OWLGrEd editor plugins are: (i) Controlled Natural Language (CNL) interface support plugin (used to augment the ontology with lexical information in [14]) and (ii) database to ontology mapping support plugin DBExpr [13].

5 Conclusions

The UML style visualization and authoring of OWL ontologies has been proven to be a successful alternative to other ontology visualization and authoring means. The OWLGrEd editor is able to offer a compact ontology notation by using OWL Manchester syntax [5] for textual presentation of advanced ontology constructs.

We have described three mechanisms that can be applied to enhance the ontology visual presentation and editing experience: (i) ontology visualization parameters; (ii) ontology profiles that may enhance the user experience in creating comprehensible and visually appealing ontology presentations; (iii) ontology visualization transformation means. Each of them can be further developed and enhanced to reflect new ontology rendering patterns, as well as to respond to the needs of concreate use cases.

A future work direction is also to move the OWLGrEd editor to the web environment, this depends on suitable tool definition framework availability providing the necessary editor definition infrastructure; this is work in progress in itself, as well.

References

1. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (2009)
2. ODM UML profile for OWL, `http://www.omg.org/spec/ODM/1.0/PDF/`
3. 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)
4. TopBraid Composer. `http://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/.`
5. OWL 2 Manchester Syntax, `http://www.w3.org/TR/owl2-manchester-syntax/`
6. Lohmann, S., Negru, S., Haag F., Ertl, T.: Visualizing Ontologies with VOWL. Semantic Web 7(4), 399-419 (2016)
7. Cerans, K., Liepins, R., Sprogis, A., Ovcinnikova, J., Barzdins, G.: Domain-Specific OWL Ontology Visualization with OWLGrEd. In: ESWC 2012 Satellite Events, Springer LNCS, pp. 419-424, (2012)
8. Cerans, K., Ovcinnikova, J., Liepins, R., Sprogis, A.: Advanced OWL 2.0 Ontology Visualization in OWLGrEd. In: 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)
9. 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, pp. 102-113 (2010)
10. Barzdins, G., Liepins, E., Veilande, M., Zviedris, M.: Semantic Latvia Approach in the Medical Domain. Proc. DB&IS2008. Haav, H.M., Kalja, A. (eds.), TUT Press, pp. 89-102, (2008).
11. Unified Modeling Language: Infrastructure, version 2.1. OMG Specification ptc/06-04-03, `http://www.omg.org/docs/ptc/06-04-03.pdf`
12. Unified Modeling Language: Superstructure, version 2.1. OMG Specification ptc/06-04-02, `http://www.omg.org/docs/ptc/06-04-02.pdf`
13. Cerans, K., Barzdins, G., Bumans, G., Ovcinnikova, J., Rikacovs, S., Romane, A. and Zviedris, M.: A Relational Database Semantic Re-Engineering Technology and Tools // Baltic Journal of Modern Computing (BJMC), Vol. 3 (2014), No. 3, pp. 183-198.
14. Liepins, R., Bojars, U., Gruzitis N., Cerans, K., Celms, E.: Towards Self-explanatory Ontology Visualization with Contextual Verbalization. In Arnicans, G., Arnicane, V., Borzovs, J., Niedrite, L. (eds.), Databases and Information Systems, Springer, Communications in Computer and Information Science, Vol 615, pp.3-17 (2016)