Extensible Visualizations of Ontologies in OWLGrEd

Kārlis Čerāns, Jūlija Ovčiņņikova, Renārs Liepiņš, Mikus Grasmanis

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

Abstract. OWLGrEd is a visual editor for OWL 2.0 ontologies that combines UML class diagram notation and textual OWL Manchester syntax for expressions. We review the basic OWLGrEd options for ontology presentation customization and consider the framework of OWLGrEd extensions that enables introducing rich use-case specific functionality to the editor. A number of available OWLGrEd extensions offering rich ontology management features to their end-users are described, as well.

Keywords: OWL, OWLGrEd, custom ontology visualization

1 Introduction

Presenting OWL ontologies [1] in a comprehensible form is vital for ontology designers and their users alike. Several approaches and tools, including OWLViz [2], VOWL [3], OntoDia [4], ODM [5], TopBraid Composer [6] and OWLGrEd [7] have been developed to present the ontologies visually so that ontologically related constructs are linked together in the presentation (e.g. an object property can be depicted as a line connecting its domain and range classes, or a sub-class can be linked to its super-class). A recent extensive and in-depth overview of the ontology visualization methods and tools is [8].

The OWLGrEd ontology editor[1] [7] stands out in the ontology visualization tools family by combining the ontology visualization and editing facilities. So, an ontology or its fragment can be adjusted after its initial automatic visualization, or an ontology can be even created from scratch within the editor and then saved into some standard textual serialization format. We describe here the options for and experience with custom/extended ontology presentation in OWLGrEd. These can be viewed also as an initial response to the call for "ontology visualization framework implementing a core set of visual and interactive features that can be extended and customized" in [8].

The OWLGrEd notation [7] comprising extended UML class diagrams [9] combined with OWL Manchester syntax [10] for textual expression encoding allows to express all OWL 2.0 [1] ontology constructs. Should an ontology be used as a data model within some context, it may be convenient to store an important part of the model contents within the ontology entity annotations. To facilitate custom handling of designated annotation properties, ontology visualization profiles extending the diagram element

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

structure have been introduced to OWLGrEd in [11], cf. also [12]; the sharing and development of the profiles is described here for the first time.

The practical usage of visualization profiles in custom-annotated ontology engineering has shown that it is convenient to consider such a profile within a context of a more "heavy-weight" ontology editor extension (an editor "plugin" in terms of [13]) which, besides the profile itself, may contain programmable editor functionality extensions.

We shall describe in the paper and show in the demonstration the following:

1) Review of the OWLGrEd notation and its basic options for presentation tuning,
2) The list of available existing and novel OWLGrEd extensions together with the instructions how to install an extension into a users' OWLGrEd project, and
3) The means (advanced) to create new extensions for OWLGrEd editor (extension architecture, custom fields and views, outline of the programmable data model).

2 Basic Tuning of Presentation in OWLGrEd

Figure 1 shows an OWL ontology example in the OWLGrEd notation, we refer to e.g. [7,13] or the OWLGrEd home page for its more detailed explanation.

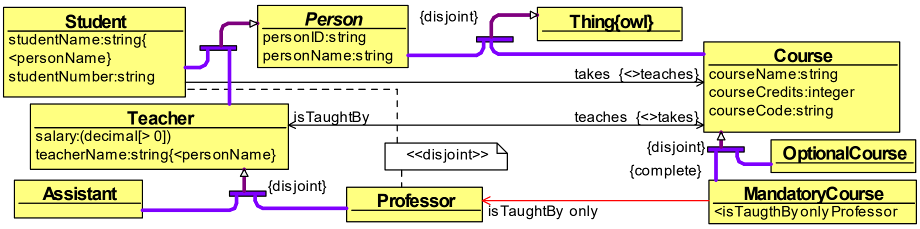


Fig. 1. A simple mini-University ontology in OWLGrEd

The UML notation [9] allows for presenting a property either graphically as an association role, or textually as an attribute. OWLGrEd supports alternative visual ways of the same semantic construct expressing for properties and restrictions, as well as e.g. for subclass, equivalent classes and disjoint classes notation. A line connecting two classes shows visually their connectivity, while the textual form may be preferable to reduce the graph overloading with lines, or eventually may enable splitting a large connected graph into separate fragments. The ***RefactoringServices*** extension (cf. Section 3) provides built-in ontology presentation re-shaping services.

There are options to configure the inclusion and initial shaping (e.g. visual or textual) of ontology constructs in the ontology diagram created during the ontology loading (import) phase by means of ontology import parameters, as described in [13].

There are two options to change the presentation style (including visibility) of an element present in an ontology diagram. The "shallow" (ad-hoc) option is provided by the GRTP platform [14] the OWLGrEd editor is built upon and is available on a per-symbol basis (e.g. it is possible to change the color of a diagram element). The structure-based definition of the presentation style (e.g. make all superclass-to-subclass lines the left-to-right flow lines) is also available in the default OWLGrEd configuration (cf. the "*Style palette*" context menu option and toolbar icon).

3 OWLGrEd Extensions

An OWLGrEd extension can add custom (domain-specific) data structures and functionality to the editor. The principal components of an OWLGrEd extension are:

1) ***custom information fields*** for ontology diagram symbols, each with possible visual appearance and/or semantics (e.g. annotation assertion) specification, and
2) high-level programmable ***functionality extensions*** (tied e.g. to context menu or palette elements, or to explicit extension points in the existing procedures).

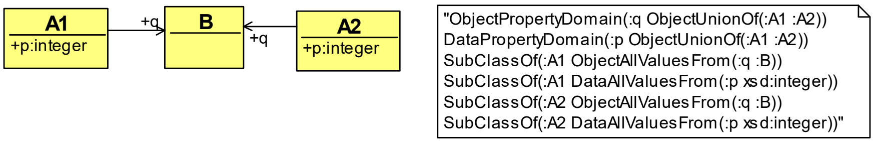
In addition, ***ontology presentation views*** [11] can be specified within each OWLGrEd extension; a view can define the style (including visibility) of visual element types, with an option for conditionally applied styles based on data field values.

Any user of the OWLGrEd editor can download extensions e.g. from the Extensions section of the OWLGrEd home page; then add them (de-compressed first) to the project via "*Extensions*" context menu command within a project diagram.

The currently available extensions include:

1) ***OWLGrEd_UserFields***, historically the first extension [11], part of default OWLGrEd configuration, providing the basic mechanisms of both creation and run-time support for custom information fields and ontology presentation views;
2) ***RefactoringServices***, supports transformations among ontology element visual presentation options (e.g. an object property presentation can be switched between the graphical and textual form); the transformations are added to the context menus of the ontology diagram elements to be transformed;
3) ***OWLGrEd_Schema***, a novel (work in progress) extension supports assertions of a property applicability within a given class context; the property domain is then computed as the union of all classes for which the property is applicable (cf. Fig.2);
4) ***OWLGrEd_OBIS,*** supports the annotation framework [15] for automated ontology-based information system generation [16]. Parts of it have been also refactored into separate extensions, including ***UML_Plus*** (introduction of UML-style elements for modeling notation: a composition, an enumerated class, an abstract class and a derived property) and ***DefaultOrder*** (recording of attribute ordering information within a class node).
5) ***OWLCNL_LanguageFields***, an experimental framework for adding verbal forms to ontology entities to enable contextual verbalization of ontologies [17].

The OWLGrEd extensions currently are actively developed further and applied in practical use cases. For instance, for the ontology development for the existing (legacy) data structure of the Latvian Enterprise Registry registers the UML_Plus, OWLGrEd_Schema and RefactoringServices custom extensions have been important[2].



```
"ObjectPropertyDomain(:q ObjectUnionOf(:A1 :A2))
DataPropertyDomain(:p ObjectUnionOf(:A1 :A2))
SubClassOf(:A1 ObjectAllValuesFrom(:q :B))
SubClassOf(:A1 DataAllValuesFrom(:p xsd:integer))
SubClassOf(:A2 ObjectAllValuesFrom(:q :B))
SubClassOf(:A2 DataAllValuesFrom(:p xsd:integer))"
```

Fig. 2. Outline of property applicability assertions: visual form and OWL Functional Syntax

[2] An image of the ontology is available at owlgred.lumii.lv/success_stories#ur

4 Create Your Own Extension

An OWLGrEd extension is a data folder that can be added to a project by placing it under the Plugins sub-folder within the projects' folder and then re-opening the project. The principal elements of the contents of an OWLGrEd extension folder are:

1) info.lua – a text file stating the extensions' identifier, name and version;
2) load.lua – the code to be executed upon installing the extension (e.g. introducing the custom fields, and other editor configuration updates);
3) unload.lua – the code to be executed upon uninstalling the extension;
4) other data and functionality information, referred to from the loading and unloading programs, including the code to be attached to e.g. newly created menu and toolbar items, or pre-defined extension points within the editor code.

The OWLGrEd editor is built upon the GRTP platform [14], whose data model is best described in [18] and augmented by the custom fields part in [11]. The "live" data model diagram is available also from within the editor itself under the *Show->Metamodel* global menu item. The programming environment is based on Lua programming language and uses library lQuery [19] for data model management. Examples of loading and unloading transformations can be seen e.g. in the code of ***UML_Plus*** extension.

The definitions of custom fields are typically stored in textual form within extension's AutoLoad sub-folder and must be loaded by the code in load.lua. The custom field definition file can be created using the OWLGrEd style palette environment (to be opened from the project diagram): under '*Manage views and profiles*' select '*New profile*', give it a name, add custom fields (views can be added, as well), then save the profile. The abstract profile structure follows [11]; the OWLGrEd style palette allows to "fill in" the instances of the profile metamodel.

Each custom field is defined within editor structure context (a field can be ascribed to e.g. a class, a role or an attribute) and can have basic appearance properties, functional translets and semantic tags defined. A semantic tag typically is a template for OWL Functional Syntax [1] assertion associated with the field, as, for example, AnnotationAssertion(obis:textPattern $subject $value) in ***OWLGrEd_OBIS*** extension is, where the $subject and $value variables refer to the context and the contents of the field respectively. The semantic tags and style effects can be attached to the choice items of check-box or drop-down type editor fields, as well.

5 Conclusions

OWLGrEd offers a wide range of customization options for presentation of an ontology loaded or created within the editor.

The OWLGrEd editor extensions are playing a major role in supporting ontology development and custom presentation in use cases that typically come up with their own requirements for extra add-on functionality to the standard editor features.

The source code of the OWLGrEd extensions is included within OWLGrEd distribution (their code is interpreted during the run-time, so any changes to it are effective immediately) and is free to be amended, modified and used (as is OWLGrEd itself).

The OWLGrEd extension architecture is suitable also for eventual migration into the web environment, as is envisaged for the OWLGrEd editor itself; it admits the user role separation allowing the system administrators to define and install the extensions, while letting the end-users to choose which extensions to activate within a project.

References

1. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (2009)
2. OWLViz, http://www.co-ode.org/downloads/owlviz/
3. Lohmann, S., Negru, S., Haag F., Ertl, T.: Visualizing Ontologies with VOWL. Semantic Web 7(4), 399-419 (2016)
4. Mouromtsev, D., Pavlov, D., Emelyanov, Y., Morozov, A., Razdyakonov, D. & Galkin, M. The simple, web-based tool for visualization and sharing of semantic data and ontologies. In: ISWC P&D 2015, CEUR, vol.1486, http://ceur-ws.org/Vol-1486/paper_77.pdf (2015)
5. ODM UML profile for OWL, http://www.omg.org/spec/ODM/1.0/PDF/
6. TopBraid Composer. http://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/.
7. Bārzdiņš, J., Čerāns, K., Liepiņš, R., Sproģis, A.: UML Style Graphical Notation and Editor for OWL 2. In: Proc. of BIR'2010, LNBIP, Springer 2010, vol. 64, pp. 102-113 (2010)
8. Dudáš, M., Lohmann, S., Svátek, V., Pavlov, D.: Ontology visualization methods and tools: a survey of the state of the art. *The Knowledge Engineering Review*, *33*, (2018)
9. Unified Modeling Language Specification, https://www.omg.org/spec/UML/2.5
10. OWL 2 Manchester Syntax, http://www.w3.org/TR/owl2-manchester-syntax/
11. Čerāns, K., Ovčiņņikova, J., Liepiņš, R., Sproģis, 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)
12. Čerāns, K., Liepiņš, R., Sproģis, A., Ovčiņņikova, J., Bārzdiņš, G.: Domain-Specific OWL Ontology Visualization with OWLGrEd. In: ESWC 2012 Satellite Events, Springer LNCS, pp. 419-424 (2012)
13. Ovčiņņikova, J., Čerāns, K.: Advanced UML Style Visualization of OWL Ontologies. In: Proc. of VOILA 2016. CEUR, vol. 1704, CEUR-WS.org, 2016, pp.136-142 (2016)
14. Bārzdiņš, J., Zariņš, A., Čerāns, K., Kalniņš, A., Rencis, E., Lāce, L., Liepiņš, R., Sproģis, A.: GrTP: Transformation Based Graphical Tool Building Platform. In: Proc. of MDDAUI-2007, CEUR, vol. 297, http://ceur-ws.org/Vol-297/paper6.pdf, (2007).
15. Čerāns, K., Romāne. A.: OBIS: Ontology-Based Information System Framework. In: Proc. CAiSE FORUM 2015, CEUR vol.1367, http://ceur-ws.org/Vol-1367/paper-09.pdf (2015)
16. Zviedris, M., Romāne, A., Bārzdiņš, G., Čerāns, K.: Ontology-Based Information System. In: Proc. of JIST'2013, Springer LNCS, Vol. 8388, pp.33-47 (2014).
17. Liepiņš, R., Bojārs, U., Grūzitis N., Čerāns, K., Celms, E.: Towards Self-explanatory Ontology Visualization with Contextual Verbalization. In: Arnicans, G., Arnicane, V., Borzovs, J., Niedrite, L. (eds.), DBIS, Springer, CCIS, Vol 615, pp.3-17 (2016)
18. Bārzdiņš, J., Čerāns, K., Kozlovics, S., Lace, L., Liepiņš, R., Rencis, E., Sproģis, A., Zariņš, A.: An MDE-based Graphical Tool Building Framework. In Scientific Papers, University of Latvia, 2010, Vol 756, pp.121-138 (2010)
19. Liepiņš, R.: Library for model querying: lQuery. In Proceedings of Workshop on OCL and Textual Modelling, pp.31-36 (2012)