OWLGrEd Ontology Visualizer

Renārs Liepiņš, Mikus Grasmanis, and Uldis Bojārs

Institute of Mathematics and Computer Science, University of Latvia,
Raina bulvaris 29, Riga, LV-1459, Latvia
{renars.liepins,mikus.grasmanis}@lumii.lv,uldis.bojars@gmail.com

Abstract. The OWLGrEd Ontology Visualizer is an online tool for visualizing OWL ontologies using a compact UML-based notation. This paper describes the implementation of the OWLGrEd Ontology Visualizer which consists of a web-based user interface, the graph (visualization) generation component, the layout component and the graph rendering and sharing component. The paper concludes with a list of future development ideas including switching from HTML canvas to vector graphics and extending the visualization with an ontology verbalization layer.

Keywords: Ontologies, Semantic Web, Visualization, OWL

1 Introduction

Visualizations are an important tool for working with ontologies. They can provide a "bird's eye view" of the ontology, enrich the documentation and help debug ontologies by letting developers spot mismatches between what they intended and what is defined in the actual ontology.

OWL ontologies can be represented in RDF and visualized as RDF graphs (e.g., using the W3C RDF validator). However, such visualizations are low-level and work at the conceptual level of triples instead of ontology building blocks such as classes and properties. For ontology graphical representation to be useful it needs to be at the level of ontology language concepts and, in order to provide a good overview, it needs to be as clear and as compact as possible.

This paper describes the OWLGrEd Ontology Visualizer[1] – an online tool for visualizing OWL 2 ontologies. It is based on a desktop application – the OWLGrEd graphical OWL editor [1]. Both applications employ the same compact UML-based notation for representing OWL ontologies described in the next section.

2 Background

The OWLGrEd notation[2] is based on UML class diagrams that software developers may already be familiar with. Most OWL features have a 1:1 mapping

[1] http://owlgred.lumii.lv/online_visualization
[2] http://owlgred.lumii.lv/notation

to UML concepts (OWL classes to UML classes, datatype properties to class attributes, ...). New graphical and text elements are introduced for OWL features that do not have UML equivalents. Classes and other elements have text fields where OWL expressions may be added if needed (e.g. to indicate an equivalent class). The notation is further described in [1].

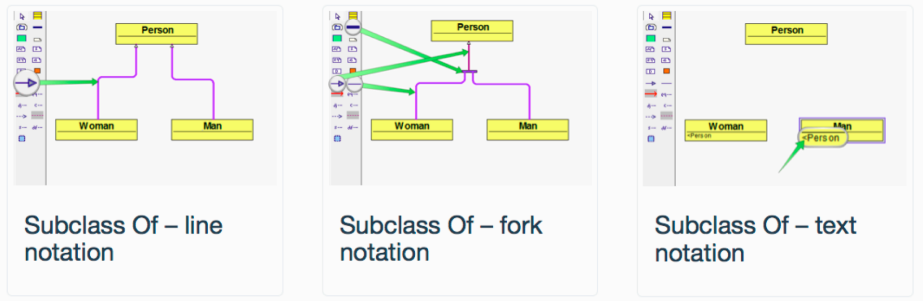

Fig. 1. Options for representing class hierarchy relationships in OWLGrEd.

The notation supports OWLGrEd's aim to make visualizations as compact as possible. For example, Figure 1 shows alternative ways for representing generalization (a subclass-of relation). The naive representation – a line for every generalization relation (shown on the left) – can be expressed more compactly with the fork notation where multiple incoming lines are merged into one (shown in the middle). A text notation is also available for cases where it is more appropriate (e.g. to refer to a superclass that is defined using an OWL class expression and is not referenced anywhere else).

These and other notation features allow us to create cleaner visualizations but make the application more complex because it has to consider the alternative ways for representing OWL axioms. This impacts the graph generation component described in Section 3.2.

OWLGrEd diagrams use the orthogonal layout where the inheritance-defining relations (i.e. `subclass-of` relations between classes and `instance-of` relations between classes and instances) are presented in a hierarchical layout (i.e. they "flow" from one side of the diagram to the opposite side) and all other relations "flow" in the direction perpendicular to it. We have observed that for a typical OWL diagram the horizontal direction (left-to-right) seems to be the more readable and the one which leads to more compact diagrams. The generation of diagram layout is described in Section 3.3.

Figure 2 shows a visualization of the Koala ontology from the Protégé ontology library[3]. Visualizations are assigned custom URLs that allow to share them with others[4].

[3] `http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library`
[4] Koala ontology: `http://owlgred.lumii.lv/online_visualization/koala.owl`

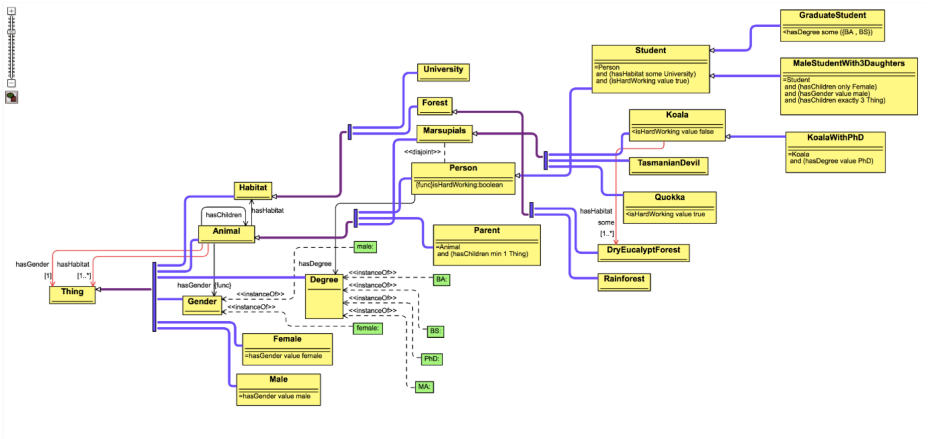

Fig. 2. Visualization of the Koala ontology.

3 System Architecture

This section describes the implementation of the OWLGrEd Ontology Visualizer. It is a client-server type application where the client-side accepts user requests and contains the visualization component while most of the work is done server-side. The server side is implemented as a data transformation pipeline and consists of a file upload service, pipeline manager, and individual pipeline modules described in detail later in the paper. This modular architecture provides separation of concerns, permitting easier application updates and enabling scalable solutions that can run transformations in parallel.

The server side of the application runs inside JVM and is written in Java and Clojure. Communication between the client and the server uses asynchronous HTTP requests with JSON payload.

3.1 Ontology Upload and Parsing

Users can either (a) upload and visualize their ontology; or (b) explore example visualizations. In order to start the process a user uploads the ontology file to be visualized. The application saves the file and passes it to subsequent steps - ontology parsing and graph generation ("graphization").

The application parses the ontology using OWL API[5] [2]. By using an established API for parsing OWL we can avoid having to deal with multiple ways for representing OWL ontologies. As a result users can upload ontologies in any format as long as it is recognized by the OWL API. The "graphization" component that is next in the pipeline makes OWL API calls in order to retrieve all the information necessary for visualization.

[5] `https://github.com/owlcs/OWLAPI`

3.2 Graph Generation ("Graphization")

Graph generation is a core step of the ontology visualization process. It transforms the parsed set of ontology axioms into a graphical form that is laid out and sent to the browser. The OWLGrEd notation attempts to summarize the ontology by showing it in the most compact form depending on the context. Thus there can be more than one way to display the same information (see Section 2) associated with a set of rules that define how to choose the visualization approach in each particular case. For each type of axioms and entities there is a visualization function that transforms them to a graph representation.

The whole transformation is implemented as a pipeline that starts with two sets: (a) items to transform; and (b) item renderings generated. Initially the first set (items to transform) contains all entities and axioms from the ontology; the second set (item renderings) is empty. The result set is a collection of objects: Nodes and Edges. They both have a type (`OWLClassBox`, `SublassOfLine`, etc.) and a list of text labels. Text labels have a text value and type (e.g. `ClassNameLabel`, `PropertyNameLabel`).

The pipeline proceeds through an ordered list of transformation steps each consisting of a selector function and a transformation function. The selector function selects items from the "items to transform" set that are suitable for the given step. The selected items are then passed to the transformation function that renders them. The result is added to the rendered items set and the processed items are removed from the "items to process" set. The remaining items are passed to the next select / transform step. The resulting JSON structure is passed to the layout step described in the next section.

The transformation component is written in Clojure and consists of ∼70 transformation rules (corresponding to the total number of [node, edge, label] types in the notation) and ∼1500 lines of code. We chose a functional programming language because it fits well with the transformation pipeline approach where each transformation can be defined as a function. Other benefits of choosing Closure are the interactive interpreter that makes testing easier and natively access to Java objects which enables interoperation with the OWL API.

The last step performed by the "graphization" component is styling which walks through the set of graph elements (nodes, edges, labels) and applies styling information according to their type and properties. The styling configuration that determines the appearance of visualization elements (colors, label positions, etc.) is stored in a separate file for easy customisation.

3.3 Layout Generation

The layout generation module, written in Java, walks the graph (provided as a JSON structure) generated by the "graphization" component, enriches it with layout information (element coordinates and dimensions) and sends the result to the rendering component. Having a custom layout generation engine allows us to fully support the chosen notation and to fine-tune the visualization as needed. OWLGrEd uses the orthogonal diagram layout described in Section 2.

Initially, the dimensions of all text items (textual inside nodes and edge labels) are calculated taking into account the text and its style. For each node

minimum possible dimensions are calculated that will enclose all text items related to the node. As the next step, we enrich the original graph with geometric constraints (minimum sizes, spacings, edge orientation, label anchors) and calculate a graph layout satisfying these constraints. The resulting layout information is injected into the JSON file describing the graph.

The completed JSON file describing the graph and its layout is sent to a browser-side rendering component which draws the diagram. As a convenience feature the JSON file is also saved under a short name in order to enable diagram sharing.

3.4 Rendering, Presentation & Sharing

The Rendering component displays the resulting visualization in the browser where users can navigate it (zoom, pan, select) and share it with others via custom URIs generated for each new visualization.

The graph is shown in an HTML5 canvas element using a custom rendering library built on top of the KineticJS library[6]. This component draws the visualization according to the diagram structure, element coordinates and styles contained in the supplied JSON structure.

The user interface also contains some visualization examples that can be accessed via their URIs or by pressing the "Enjoy our examples" button which will cycle through the examples.

4 Experience Using the Application

The usage of the application can be characterized by the number of ontologies submitted to it. We analyzed the weekly statistics of the upload button hits from international locations (according to Google Analytics). In the time period from 2014-04-01 till 2014-07-18 there were 512 upload button hits from locations outside Latvia. There were 368 successful file uploads most of which were OWL ontologies that were visualized by the application.

The size of ontology files uploaded varied from 412 bytes (for a small ontology in Turtle RDF) to ∼1.7 Mb (for larger ontologies expressed in RDF/XML) with average size ∼103 Kb and a median of ∼24 Kb.

Ontologies were expressed in multiple formats including various RDF representations and the OWL XML syntax. Thanks to OWL API the application can work with all these formats without requiring us to write additional code. Sometimes "real-life" ontologies uploaded for visualization may be contain "surprises" that prevent us from parsing them. For example, we identified a bug in OWL API which prevents an ontology containing the BOM (byte-order mark that may be present at the start of text files) from being parsed. The bug was reported to OWL API developers who promptly resolved it resulting in OWL API improvement that anyone can benefit from[7].

[6] `http://kineticjs.com/`

[7] `https://github.com/owlcs/OWLAPI/issues/187`

5 Future Work

This section lists future development ideas for the OWLGrEd Ontology Visualizer. The implementation details, a current state of which is recorded in this paper, may change as a result of these developments.

Moving from canvas to vector graphics - we plan to change the ontology presentation component by switching ontology display from HTML5 canvas to SVG. This may also allow users to save resulting visualizations and work on them further using graphic editing software.

Visualization publication and sharing - the sharing feature (where visualizations get their own URIs) was added recently and we intend to further develop it. It would be useful to publish visualizations along with metadata (e.g., schema.org) asserting that this is a visualization of an ontology and providing information about it.

Ontology editing - currently the application provides a read-only view of the ontology. Our medium-term plan is to extend the application with the editing functionality, converting it into an online version of the OWLGrEd editor. This will involve substantial changes to the application described in this paper.

Ontology verbalization - a visualization of an ontology is a good way for getting an overview of the ontology but, if viewers do not know semantics of the notation, they cannot just come to the picture and instantly "understand" it. We plan to mitigate this by integrating controlled natural language verbalizations into the visualization so that user can click on any element and see a natural language representation of this element.

6 Conclusion

This paper described the implementation of the OWLGrEd Ontology Visualizer – an online application for creating and sharing visualizations of OWL ontologies. The main activities performed by the application are ontology parsing, generation of the visualization graph, layout calculations and displaying the visualization to the user. The visualizer benefits from using OWL API that takes care of parsing various OWL ontology formats.

We believe ontology visualizations are a valuable resource with many uses such as ontology exploration and debugging and are looking forward to further developments in this area.

References

1. Bārzdiņš, J., Bārzdiņš, G., Čerāns, K., Liepiņš, R., Sproģis, A. (2010). UML Style Graphical Notation and Editor for OWL 2. Perspectives in Business Informatics Research, Lecture Notes in Business Information Processing, Volume 64, Part 2, pages 102-114, 2010
2. Horridge, M., Bechhofer, S. (2011). The OWL API: A Java API for OWL ontologies. Semantic Web, 2(1), 2011, 11-21

Acknowledgments. This work was partially supported by ESF project 2013/0005/1DP/1.1.1.2.0/13/APIA/VIAA/049.