A Method for Automatically Generating Schema Diagrams for OWL Ontologies

Cogan Shimizu[1], Aaron Eberhart[1], Nazifa Karima[1], Quinn Hirt[1], Adila Krisnadhi[2], and Pascal Hitzler[1]

[1] Data Semantics Laboratory, Wright State University, Dayton, OH, USA
[2] Faculty of Computer Science, Universitas Indonesia, Depok, Indonesia

Abstract. Interest in Semantic Web technologies, including knowledge graphs and ontologies, is increasing rapidly in industry and academics. In order to support ontology engineers and domain experts, it is necessary to provide them with robust tools that facilitate the ontology engineering process. Often, the schema diagram of an ontology is the most important tool for quickly conveying the overall purpose of an ontology. In this paper, we present a method for programmatically generating a schema diagram from an OWL file. We evaluate its ability to generate schema diagrams similar to manually drawn schema diagrams and show that it outperforms VOWL and OWLGrEd. In addition, we provide a prototype implementation of this tool.

1 Introduction

Engineering an ontology is a complex and time-consuming process [15]. Providing a broad and sophisticated set of tools and methodologies to support ontology engineers can help mitigate these factors. This method is part of an overall thrust at improving the ontology engineering process. Specifically, we focus on so-called Modular Ontology Modelling

In this paper we describe an algorithm that generates a schema diagram from an OWL file and evaluate a corresponding prototype implementation. The evaluation shows that our approach is superior to the related visualization tools WebVOWL[3][14] and OWLGrEd[4][2] specifically for the type of schema diagrams that we have found to be most useful. Our prototype tool, SDOnt, is publicly available online.[5]

A schema diagram is a commonly used and invaluable tool for both understanding and developing ontologies. A survey we conducted shows that it ranks among the most important components in the documentation of an ontology [11]. Schema diagrams provide a view, albeit limited, of the structure of the relationships between concepts in an ontology. Frequently, a schema diagram is generated manually during the design phase of the engineering process. At that

[3] `http://vowl.visualdataweb.org/`
[4] `http://owlgred.lumii.lv/`
[5] `http://dase.cs.wright.edu/content/sdont`

time the diagram is a conceptual, mutable document. After the schema diagram has been created, an OWL file is created to model the diagram with OWL axioms which precisely capture its semantics. We call this a *diagram-informed OWL file*. As we discuss in Section 5, this method may lead to unforeseen problems, e.g. whether an OWL file faithfully represents its diagram. A tool that generates a schema diagram programmatically allows for ontology engineers to create an *OWL-informed diagram*. Additionally, it would provide a mechanism by which schema diagrams may be easily updated in the case of a newer versioned OWL file.

In this paper, we describe our method for generating schema diagrams from OWL files. The programmatically generated schema diagrams visualize the same information as those that are manually generated following a specific visualization paradigm which we found to be most effective in practice. We evaluate its effectiveness by comparing it to two existing OWL visualization tools, VOWL and OWLGrEd. We would like to note that for now we are ignoring layout questions; we consider only the question of what *content* should be in a graph. We intend to explore layout issues in follow-up work.

The rest of the paper is organized as follows. Section 2 describes existing visualization tools and how they differ from our method and tool. Section 3 describes the process of schema diagram generation. Section 4 gives a very brief description of our implementation and our method. Section 5 evaluates our method, details possible points of improvement, and discusses the results. Finally, in Section 6 we conclude and outline our next steps and future work.

2 Related Work

Visualization is critical to understanding the purpose and content of an ontology [4, 5, 11]. There are many tools that provide visualization capabilities. We are interested in how meaningfully they construct a visualization, rather than the details of their implementations. For instance, many of these tools offer some sort of interactivity, such as drag and drop construction and manipulation or folding for dynamic exploration. This differs from our intent to provide a method for constructing a diagram that portrays the *relationships* between concepts. Our approach also does not provide specific support for visualizing an ABox; our emphasis is on supporting the creation and use of schema diagrams.

Below, we have selected for comparison a few tools that are representative in their functionality. For a more complete survey see [6]. VOWL and OWLGrEd were chosen for comparison to our method. They are described in Section 5.

NavigOWL[6] is a plugin for the popular tool Protégé[7]. NavigOWL provides a graph representation of the loaded ontology such that the representation follows a power-law distribution, which is a type of force-directed graph representation. It also provides a mechanism for filtering out different relational edges while exploring an ontology [1]. This tool is not supported in the current version of

[6] `http://home.deib.polimi.it/hussain/navigowl/`
[7] `https://protege.stanford.edu/`

Protégé.[8] It is particularly well suited to visualizing the ABox, which is outside the scope of our intent and method.

OWLviz is also a Protégé plugin. It generates an IS-A hierarchy for the loaded ontology rooted with the concept owl:Thing. OWLviz displays *only* subclass relations between concepts and does not extract properties from those axioms. Hovering over the nodes in the graph representation provides axioms related to the class represented by that node. This plugin is not supported by the current version of Protégé. The lack of relational specificity per edge is non-ideal for our purposes. Furthermore, information accessible only through interaction is not desired in a reference diagram.

TopBraid Composer is a standalone tool similar in functionality to Protégé with OWLviz; it is developed, maintained, and sold by TopQuadrant, Inc.[9] There is no free version for academic purposes.

OntoTrack is a standalone tool for visualizing the subsumption hierarchy of an ontology rooted at owl:Thing. Properties are not extracted from axioms and used to label edges. The tool only supports ontologies in the deprecated OWL-Lite$^-$ and automatically augments the visualization with subsumptions found with the reasoner RACER[10] [13]. Between the limitations on OWL and the interactivity, this tool is not strictly suitable for creating schema diagrams.

MEMO GRAPH was developed to be a memory prosthesis for users suffering from dementia [6]. As such, it is particularly focused on representing the relations between family members. It is not currently available for public use.

RDF Gravity is a standalone tool that provides a visualization for an ontology via graph metrics. The tool generates a force-directed graph representation of the underlying ontology. We could not find any data on how it handles blank nodes, represents class disjointness, and other non-graph metrics, as, at time of this writing, the tool is unavailable, no publication on its method can be found, and it seems to be survived only by screenshots. We include this entry for the sake of completeness.

OntoGraf[11] is a plugin for Protégé that supports interative navigation through OWL ontologies. In particular, it allows the user to navigate and filter relationships and nodes based on certain criteria (e.g. subclass, individuals, domain/range object properties, and equivalence). OntoGraf is not supported on newer versions of Protégé (5.0+).

In all, we note that there are many tools for visualizing ontologies, however many of them older and no longer supported or provide only limited support past graph-metric data, which in our experience is insufficient for conveying the purpose of an ontology. It also seems that out of this general line or research, VOWL and its associated tools (e.g., WebVOWL) emerged as the prominent paradigm - yet a paradigm that addresses different use cases than those of concern for us.

[8] https://protegewiki.stanford.edu/wiki/NavigOWL
[9] https://www.topquadrant.com/products/
[10] https://www.ifis.uni-luebeck.de/index.php?id=385
[11] https://protegewiki.stanford.edu/wiki/OntoGraf

3 Method

A schema diagram does not necessarily aim to represent all information encoded in an ontology. As mentioned in Section 2, there are several tools that attempt to do so, in particular, VOWL and OWLGrEd. However our experiences with ontology modeling with domain experts from many different fields shows us that it is necessary to strike a good balance between complexity and understandability. In fact, in these collaborations we gravitate towards diagrams that merely capture classes and the relationships between them. This omits most semantic aspects such as whether a relationship between classes does or does not indicate domain or range restrictions or even more complex logical axioms. We find that the exact semantics are better conveyed using either natural language sentences or logical axioms (preferably in the form of rules [17]) in conjunction with a very simplified diagram.

Typically, we create ontologies by first drawing schema diagrams with domain experts and then capturing the exact logical axioms that constitute the ontology. In this paper we reverse the process: start with the logical axioms and automatically derive the schema diagrams.[12] We do this to help us to deal with ontologies constructed by others for which no suitable schema diagrams are provided. As we will see later in Section 5, our visualization approach can also be helpful in finding errors in OWL files or in manually drawn schema diagrams.

To maximize information and minimize clutter we define some guidelines:

- All classes inherit from owl:Thing, so it is unhelpful to clutter a diagram with subclass edges from concepts to owl:Thing.
- We do not represent any logical connectives (e.g. disjunction) or complex axioms (e.g. a domain restriction that contains an intersection) since, in our experience, this type of information is better conveyed non-visually.
- Disjointness of classes does not need explicit graphical representation. In most cases, disjointness is immediately clear for a human with some knowledge about the domain.
- Inverse relations are not represented, as they are syntactic sugar for any relation.
- The ABox [9] is disregarded; instances of classes are not represented.

With these assumptions in mind, we detail our method using the rules below:

Steps 3 and 4 may be omitted if there are no direct domain or range restrictions given. However, because there are multiple ways of expressing the same information in OWL, domain and range may appear in the declarations of the Object or the Datatype Properties.

In Step 5 it is important to note the differences between logical and schematic equivalence. *Schematic equivalence* between two ontologies means they share the

[12] At this time, we do not consider `owl:Imports` as the tool is completely offline. Additionally, we wish to only generate a schema diagram for those axioms directly in the OWL file of interest.

1. Create a node for each class in the ontology's signature.
2. Create a node for each datatype in the ontology's signature.
3. Generate a directed edge for each Object Property based on its domain and range restrictions, if such are given. The source of the edge is the Property's domain and the target of the edge is the Property's range.
4. Generate a directed edge for each Datatype Property, in the same manner as for an Object Property, if domain and range restrictions are present.
5. For each other axiom in the TBox:
   Case 1: if the subclass and superclass are atomic, generate a subclass edge between them.
   Case 2: if the axiom is of the forms presented in (1) and (2) below, generate the associated directed edge.
   Case 3: apply rules (3) to (6), as listed below, recursively until the resulting axiom sets can be handled by Cases 1 and 2.
6. Display.

Fig. 1: The algorithm for generating a schema diagram.

same graphical representation. Consider, for example, the definitions for scoped domain and range restrictions:

$$\exists R.B \sqsubseteq A \tag{1}$$
$$A \sqsubseteq \forall R.B \tag{2}$$

Logically, (1) and (2) convey two different meanings. Schematically, though, they may be represented by the same artifact in a graph: $A \xrightarrow{R} B$. Thus we consider them schematically equivalent. We may also break down more complex axioms using the rules defined in (3) through (6). These rules hold for both intersection ($\sqcap$) and union ($\sqcup$). We list only the union versions. Note that not all of these are logically equivalent transformations.

$$A \sqsubseteq \forall R.(B \sqcup C \sqcup \cdots) \Rightarrow \begin{cases} A \sqsubseteq \forall R.B \\ A \sqsubseteq \forall R.C \\ \quad \vdots \end{cases} \tag{3}$$

and

$$\exists R.(B \sqcup C \sqcup \cdots) \sqsubseteq A \Rightarrow \begin{cases} \exists R.B \sqsubseteq A \\ \exists R.C \sqsubseteq A \\ \quad \vdots \end{cases} \tag{4}$$

(5) and (6) are used only in the union case as shown here. (5) is not a logically equivalent transformation.

$$B \sqcup C \sqcup \cdots \sqsubseteq A \Rightarrow \begin{cases} B \sqsubseteq A \\ C \sqsubseteq A \\ \quad \vdots \end{cases} \tag{5}$$

$$A \sqsubseteq B \sqcup C \sqcup \cdots \Rightarrow \begin{cases} A \sqsubseteq B \\ A \sqsubseteq C \\ \quad \vdots \end{cases} \tag{6}$$

We may recursively apply (3) through (6) for non-atomic concepts $A, B, \cdots$ until we have reached axioms of the form (1) and (2) or atomic subclass relationships.

The time complexity for this process is minimal. If c is the maximum number of concepts and datatypes in any axiom in the ontology then there are at most $\binom{c}{2}$ so-called "simple" axioms that together are schematically equivalent to the "complex axiom." Thus, there are at most $\binom{c}{2} \cdot n$ edges to parse per ontology, where n is the number of axioms in the TBox, giving our method a time complexity of $O(n)$. This calculation ignores algorithms for the graph layout.

The goal of our approach is to generate a static schema diagram for a particular OWL file. However, there are practical limitations. For sufficiently large number of classes and properties (and connectivity, thereof), nearly any graph visualization will become essentially unreadable. Indeed, our approach is primarily meant for smaller (i.e. limited number of classes and properties) OWL files, such as ontology design patterns [7] or ontology modules [12]. For use with this approach, ontologies of the former variety (many classes, properties, etc.) should be organized into modules and the tool applied to each module in order to create a collection of interrelated schema diagrams. In fact, this would likely result in a better engineered ontology [10].

4 Implementation

Our prototype implementation, SDOnt, is a pipeline consisting of three parts: a GUI, a parser module, and a rendering module. SDOnt is developed in Java and provided as an executable JAR file. Ontology manipulations are done using the OWLAPI. Online we provide the source code, test set, evaluation results, and a tutorial for the tool's use.[13]

The GUI is implemented using Java Swing and serves as an interface for navigating and loading ontologies into the program. The Ontology Parser is an implementation of our algorithm as described in Section 3. The parser provides a set of nodes to the rendering module that represent the classes and datatypes in the ontology's signature and the node-edge-node artifacts that represent their properties, domains, and ranges for the visualization.

[13] `http://dase.cs.wright.edu/content/sdont`

The rendering module takes those node-edge-node artifacts and the node set and combines them to create the visualization and render it to the screen. The rendering module utilizes the library JGraphX[14] for generating, laying out, and displaying the schema diagram. JGraphX is an open source library written in Java for displaying and manipulating graphs. However, the SDOnt code-base has been written in such a way that any visualization library may be used. That is, the algorithm is implemented in a modular fashion; an external developer may code against the SDOnt code-base with no changes necessary to the method.

5 Evaluation

In this evaluation we describe the closest alternatives to SDOnt and their methods in Section 5.1, the method by which we conduct our evaluation in Section 5.2, our choice of test set in Section 5.3, and discuss the results of our evaluation in Section 5.4.

5.1 Compared Tools

We evaluate SDOnt by comparison with author supplied visualizations, WebVOWL, and OWLGrEd.

Each of the ontologies in our test set, which is outlined in Section 5.3, has a general visualization provided by the authors. We use these diagrams as a baseline against which the tools can be judged and assume that these represent the authors' best attempt to generalize the semantics of the ontology. However, there may be some irregularities in the methodologies different authors use to produce these visualizations. Indeed, some of the methodologies are very distinct–some are very minimal; others take inspiration from UML. Some of the diagrams appear to be created automatically from Protégé or some other automated tool, while others are manually draw using a variety of graphing utilities. The variations in these sources may be partially responsible for suboptimal results, especially as none of the three compared tools use a UML-style visualization.

VOWL is a graphical notation tool for OWL. The specification can be viewed in detail in [14]. VOWL represents ontologies using detailed force-directed graphs. We use the web implementation WebVOWL[15] to generate visualizations of our ontologies. The website application has a high degree of potential customization. In the settings, we choose to filter out only class disjointness axioms and set the degree of collapsing to 0 since we use smaller ontologies.

WebVOWL is able to quickly produce a visualization for every valid OWL file that we analyze. Usually the output animation clearly represents the intent of the ontology. Occasionally, however, the result contains incomplete or missing information. In our experience WebVOWL is often ambiguous when it tries to display complex statements, which could be the cause of this.

[14] `https://github.com/jgraph/jgraphx`
[15] `http://www.visualdataweb.de/webvowl/`

OWLGrEd is a Graphical Ontology Editor that allows for interactive, drag-and-drop creation of ontologies [2]. It utilizes UML-like visualizations for displaying axioms associated to a class. In addition, it provides Manchester Syntax translations of axioms. OWLGrEd displays *all* axioms, sometimes as additional nodes. The visualization is hierarchical, so there is a subClass edge between an owl:Thing node and every un-subsumed concept in the ontology's signature.

Figure 2 shows examples of what each tool outputs in comparison to a manually created schema diagram.[16]

5.2 Comparison Scheme

The method for constructing schema diagrams for ontology patterns and modules, as introduced in the previous section, results in a diagram similar to published reference diagrams which follow the visualization paradigm that we found most useful in interactive modeling sessions with domain experts. In order to provide a meaningful evaluation, we use as gold-standard reference the manually drawn diagrams that have been published in papers or on websites by the ontologies' authors. These diagrams have been designed with human understandability in mind and their creation pre-dates our automated diagram generation method.

We compare the diagrams generated by SDOnt, WebVOWL and OWLGrEd with the gold-standard diagrams taken from the respective publications or .

To define some terminology, we say a *node* represents a class or concept. An *edge* represents a relationship or role, where the source of the edge is the relationship's domain and the head of the edge represents the relationship's codomain – domain and codomain are not meant to be formal technical terms in the sense of OWL restrictions or RDFS domain/range declarations, but rather intuitive notions that are ambiguous like a schema diagram. An edge from class A to class B in the diagram indicates that A is in the domain of the relation and B is in the codomain of the relation. There may also be an edge with the same role label between two different classes C and D elsewhere in the diagram, without making the classes A and C (or B and D) identical, as would happen if these were formal domain or range declarations.

All three visualization tools generate directed edges. To conduct this comparison, we evaluate the following criterion for node-edge-node artifacts:

> For every node-edge-node artifact in the generated diagram, does it appear in the reference diagram, and vice-versa?

However, there are a few different cases, as we outline below. By using this vocabulary it becomes natural to calculate an F_1–measure for each diagram. We report the average F_1–measure for each tool in Table 1.

[16] For this diagram, SDOnt utilized an "orthagonal layout" that is pre-defined in the associated visualization library. Our method for generating a schema diagram, at this time, makes no assertions about the placement of nodes in the graph.

– True Positive: the artifact appears in both generated and reference diagrams
– False Positive: the artifact appears in the generated diagram, but not the reference diagram.
– False Negative: the artifact does not appear in the generated diagram, but does appear in the reference diagram.

Of course, we cannot calculate True Negatives as there are an infinite number of items that do not occur in the generated diagram nor in the reference diagram.

5.3 Test Set

In order to test our process, we constructed a test set of ontology design patterns. Our process for selecting the patterns was simply searching the main publishing outlets and `ontologydesignpatterns.org` and choosing those patterns that had published diagrams, as well as unbroken links to their OWL files. All test data and the complete set of ontologies we used for our evaluation can be found on our tool's website.[17]

In some cases, it was necessary to make minor changes to the OWL files during this evaluation. Both OWLGrEd and WebVOWL produced errors on importing certain external resources.[18] Whenever removing an import allowed us to continue with the analysis we did so, using the new OWL file for both tools, even if the file worked for the others tools initially. However, there were still some patterns that failed to work for any tool. Our results report only on those OWL files that could be successfully processed by each of the compared tools. After these criteria were met, our test set contains 63 ontology design patterns. These 63 patterns are also available in the SDOnt portal.

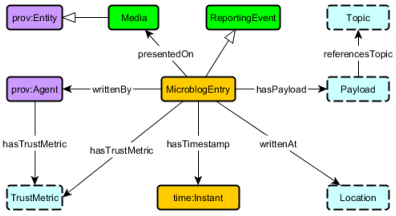
5.4 Results & Discussion

The results of our evaluation are summarized in Tables 1 and 2. The node-edge-node triple sets are determined manually. During this process we take care to use a consistent naming format. This allows us to conduct our comparison programmatically. The code utilized for this comparison, with some documentation for its use, is also available in the online portal.

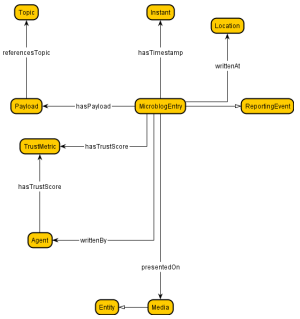
We see that in Table 1, the F_1–measures are very low. SDOnt has an F_1–measure of 0.465, OWLGrEd has an F_1–measure of 0.269, and WebVOWL has an F_1–measure of 0.163. However, we also note that there are many stylistic differences in the generation of diagrams. Many of the reference diagrams are presented in UML-esque manner. Comparing these to a force directed graph, such as those produced by WebVOWL would, of course, perform very poorly.

[17] `http://www.dase.cs.wright.edu/content/sdont`

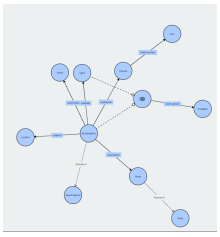
[18] We note that there has been an update to both of these web application since we conducted our evaluation. However, due to time constraints, we were unable to go back and re-run the evaluation. Fortunately, we do not punish any tool for being unable to render a document; we only compare performance against the subset of patterns that *every* tool successfully processed.



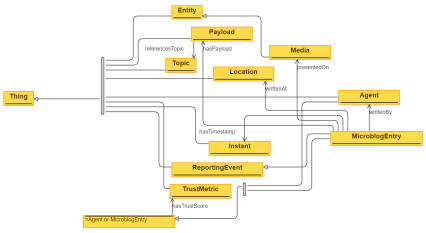
(a) The Reference Diagram.



(b) Schema Diagram generated by SDOnt.



(c) Schema Diagram generated by VOWL.



(d) Schema Diagram generated by OWLGrEd.

Fig. 2: The manually curated reference diagram (a) followed by the generated schema diagrams by SDOnt, VOWL, and OWLGrEd, for the MicroblogEntry ontology design pattern.

In this evaluation we do not encounter any false positives that are a misrepresentation of an axiom. Instead, false positives are strictly caused by the OWL file containing more information than expected. The exact reasons for this seem to vary from case to case. We speculate that in some cases the reason may be that the diagram may look more elegant or the name of a concept may imply its natural superclass. For example, in the Hazardous Event pattern [3], HazardousEvent is a subclass to Event, but this is not indicated in the reference diagram, leading to a false positive. In other cases, the OWL file could be malformed.

To formally compare the performances we run three Wilcoxon signed rank tests, with null hypothesis that there are no difference in performance. Our tests show that SDOnt performs significantly better than OWLGrEd ($p < 0.001$), and WebVOWL ($p < 0.001$). As well as that OWLGrEd performs significantly better than WebVOWL ($0.05 < p < 0.01$)

There are also motivating cases for using schema diagrams as error checkers during ontology development. The tools do poorly on many of the diagrams simply because the respective OWL files do not actually contain the information

	F_1–Measure
SDOnt	0.465
OWLGrEd	0.269
WebVOWL	0.163

Table 1: F_1–Measure for each of the tool's performance.

Table 2: Significance of Pairwise comparison of the tools using the Wilcoxon signed rank test.

SDOnt vs OWLGrEd	SDOnt vs WebVOWL	OWLGrEd vs WebVOWL
$p < 0.001$	$p < 0.001$	$0.05 < p < 0.01$

the diagram implies. This may be that they contain *more* information (e.g. alignments to different patterns) or lack information (e.g. errors).

VOWL and OWLGrEd consistently performed worse than SDOnt for two reasons. First, VOWL had many duplicated edges for different functional properties, even after adjusting settings in an attempt to prevent them. Secondly, both OWLGrEd and VOWL had trouble extracting properties from complex axioms. For OWLGrEd these axioms were represented as anonymous nodes, leading to false positive artifacts. Set operation nodes in WebVOWL also lead to additional false positive artifacts.

6 Conclusions

Our results are promising even if they do not present as such. A lack of a consistent visual notation for diagrams in our test set and poor quality control in the OWL files definitely contribute to a poor, raw showing. However, we note that even given our low F_1–measures the results are consistent. And we see that SDOnt performs significantly better than both OWLGrEd and WebVOWL for generating schema diagrams that are most similar to the reference diagrams.

To be fair, the test set against which we evaluated contained many UML-esque diagrams, to which none of the evaluated tools are well-suited. VOWL and OWLGrEd were used for comparison simply because they are the current state-of-the-art for generalized ontology visualization and they are the tools which produced the most similar diagrams to the desired ones. Our results do not invalidate VOWL or OWLGrEd: they simply serve other purposes.

There are still many ways to improve our method and its implementation. First, we see in many diagrams that namespaces are frequently color coded, as well as providing different node styles for external patterns. As ontology engineering practices mature, we expect to see these distinctions to be formally encoded in the ontology, e.g., according to the Ontology Design Pattern Representation Language (OPLa) as described in [8]. As such, once the necessary

tooling support for OPLa has been realized, SDOnt will be able to leverage the annotations and inform style and placement of nodes for increased clarity in the schema diagram. We will also explore different styles of incorporating UML-like visualizations for datatypes. The manually created reference diagrams are fallible or simply unclear from the perspective of the OWL file which information is necessary to convey. We believe incorporating OPLa and augmenting SDOnt to account for these annotations will also help in this regard.

Secondly, we intend to investigate the most effective ways of creating a good layout beyond force-directed graphs and will explore the option of providing our work as an additional rendering capability for the OWLAPI.

Finally, we will integrate SDOnt with other existing Protégé plugins developed in our lab, including ROWL,[19] OWLax,[20] and OWL2DL[21] [16–18], in order to work towards a well-rounded ontology engineering suite which supports the Modular Ontology Modeling paradigm.

Acknowledgement. Cogan Shimizu acknowledges support by the Dayton Area Graduate Studies Institute (DAGSI). This work was partially supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0386.

References

1. Scalable visualization of semantic nets using power-law graphs. In *Applied Math*, volume 8, pages 355–367, 01 2014.
2. J. Barzdins, G. Barzdins, K. Cerans, R. Liepins, and A. Sprogis. OWLGrEd: a UML style graphical notation and editor for OWL 2. In *Proceedings of the 7th International Workshop on OWL: Experiences and Directions (OWLED 2010), San Francisco, California, USA, June 21-22, 2010*, volume 614 of *CEUR-WS.org*, 2010.
3. M. Cheatham, H. Ferguson, I. Charles Vardeman, and C. Shimizu. A modification to the hazardous situtation ODP to support risk assessment and mitigation. In *Proceedings of WOP*, volume 16, 2016.
4. A.-S. Dadzie and M. Rowe. Approaches to visualising linked data: A survey. *Semantic Web*, 2(2):89–124, Apr. 2011.
5. V. Geroimenko and C. Chen. *Visualizing the Semantic Web: XML-based Internet and Information Visualization*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
6. F. Ghorbel, N. Ellouze, E. Métais, F. Hamdi, F. Gargouri, and N. Herradi. MEMO GRAPH: An ontology visualization tool for everyone. *Procedia Computer Science*, 96(Supplement C):265–274, 2016.
7. P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors. *Ontology Engineering with Ontology Design Patterns – Foundations and Applications*, volume 25 of *Studies on the Semantic Web*. IOS Press, 2016.
8. P. Hitzler, A. Gangemi, K. Janowicz, A. A. Krisnadhi, and V. Presutti. Towards a simple but useful ontology design pattern representation language. *Proceedings WOP 2017*, October 2017. To appear.

[19] `http://dase.cs.wright.edu/content/modeling-owl-rules`
[20] `http://dase.cs.wright.edu/content/ontology-axiomatization-support`
[21] `http://dase.cs.wright.edu/content/owl2dl-rendering`

9. P. Hitzler, M. Krötzsch, and S. Rudolph. *Foundations of Semantic Web Technologies*. Chapman and Hall/CRC Press, 2010.
10. P. Hitzler and C. Shimizu. Modular ontologies as a bridge between human conceptualization and data. In P. Chapman, D. Endres, and N. Pernelle, editors, *Graph-Based Representation and Reasoning - 23rd International Conference on Conceptual Structures, ICCS 2018, Edinburgh, UK, June 20-22, 2018, Proceedings*, volume 10872 of *Lecture Notes in Computer Science*, pages 3–6. Springer, 2018.
11. N. Karima, K. Hammar, and P. Hitzler. How to document ontology design patterns. In K. Hammar, P. Hitzler, A. Lawrynowicz, A. Krisnadhi, A. Nuzzolese, and M. Solanki, editors, *Advances in Ontology Design and Patterns*, volume 32 of *Studies on the Semantic Web*, pages 15–28. IOS Press, Amsterdam, 2017.
12. A. Krisnadhi and P. Hitzler. Modeling with ontology design patterns: Chess games as a worked example. In P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns – Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 3–21. IOS Press, 2016.
13. T. Liebig and O. Noppens. Ontotrack: Combining browsing and editing with reasoning and explaining for OWL lite ontologies. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, volume 3298 of *Lecture Notes in Computer Science*, pages 244–258. Springer, 2004.
14. S. Lohmann, S. Negru, F. Haag, and T. Ertl. Visualizing ontologies with VOWL. *Semantic Web*, 7(4):399–419, 2016.
15. A. D. Nicola, M. Missikoff, and R. Navigli. A software engineering approach to ontology building. *Information Systems*, 34(2):258–275, 2009.
16. M. K. Sarker, A. Krisnadhi, D. Carral, and P. Hitzler. Rule-based OWL modeling with ROWLTab Protégé plugin. In E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler, and O. Hartig, editors, *The Semantic Web – 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 – June 1, 2017, Proceedings, Part I*, volume 10249 of *Lecture Notes in Computer Science*, pages 419–433, 2017.
17. M. K. Sarker, A. Krisnadhi, and P. Hitzler. OWLAx: A Protégé plugin to support ontology axiomatization through diagramming. In T. Kawamura and H. Paulheim, editors, *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016.*, volume 1690 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.
18. C. Shimizu, P. Hitzler, and M. Horridge. Rendering OWL in description logic syntax. In E. Blomqvist, K. Hose, H. Paulheim, A. Lawrynowicz, F. Ciravegna, and O. Hartig, editors, *The Semantic Web: ESWC 2017 Satellite Events - ESWC 2017 Satellite Events, Portorož, Slovenia, May 28 - June 1, 2017, Revised Selected Papers*, volume 10577 of *Lecture Notes in Computer Science*, pages 109–113. Springer, 2017.