A Method for Automatically Generating Schema Diagrams for OWL Ontologies

Cogan Shimizu[1], Aaron Eberhart[1], Nazifa Karima[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 their underlying schemas expressed as ontologies, is increasing rapidly in industry and academic application areas. In order to support ontology engineers and domain experts, it is necessary to provide to them robust tools for facilitating the ontology engineering process. In many cases, the schema diagram of an ontology is the single most important tool for quickly conveying the overall purpose of (part 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, for this purpose. In addition, we provide a prototype implementation of this tool.

1 Introduction

Engineering an ontology is a complex and time-consuming process [21]. As such, it is necessary to provide a broad and sophisticated set of tools to support ontology engineers. To this end, this paper describes a method for generating a schema diagram from an OWL file, and evaluates a corresponding prototype implementation. The evaluation shows that our approach is superior to the related visualization tools VOWL and OWLGrEd, for the types of schema diagrams which we have found to be most useful. Our prototype tool, SDOnt, is publicly available.[3]

A schema diagram is a widespread and invaluable tool for both understanding and developing ontologies. A survey conducted by us has shown that it ranks among the most important components in a documentation of an ontology [15]. Schema Diagrams provide a view, albeit limited, of the structure of the relationships between concepts of an ontology. Frequently, a schema diagram is generated manually during the design phase of the engineering process. At that time, the diagram is a mutable, living document. After the schema diagram has been created, the OWL file is created in the likeness of the diagram by means of OWL axioms which precisely capture the underlying intention of the possibly ambiguous diagram. We may call this a *diagram-informed OWL file*. As we see

[3] `http://dase.cs.wright.edu/content/sd4odp`

in Section 5, this can lead to unforeseen problems, e.g. whether the OWL file truly represents the diagram. Thus, one possible, beneficial side effect of having a tool that can generate a schema diagram programmatically is that it allows 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, such that 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[4] [20] and OWLGrEd[5] [2]. We would like to note up front, though, that for now we are ignoring layout questions, i.e. we consider only the question 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, in detail, the process by which we generate a schema diagram. Section 4 gives a very brief description of our implementation of our method. Section 5 evaluates the efficacy of 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 a critical aspect to understanding the purpose (and content) of an ontology [7, 8, 15]. There are many tools that offer visualization capabilities. Specifically, we are interested in the method by which they construct a visualization rather than its implementation. For example, 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 relations between concepts. Also, our approach does not provide specific support for visualizing an ABox, as our emphasis is on supporting the creation and use of schemas.

Below, we have selected for comparison a few tools that are representative in their functionality. For a more complete survey, see [9]. As we have chosen VOWL and OWLGrEd for direct evaluation against our method, we describe them 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.

[4] `http://vowl.visualdataweb.org/`
[5] `http://owlgred.lumii.lv/`
[6] `http://home.deib.polimi.it/hussain/navigowl/`
[7] `https://protege.stanford.edu/`

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 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. That is, 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 interactivity is non-ideal for a reference diagram.

TopBraid Composer is a standalone tool similar in functionality to Protégé augmented 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. Further, the tool only supports ontologies in the deprecated OWL-Lite$^-$ and automatically augments the visualization with subsumptions found with the reasoner RACER[10] [19]. 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 [9]. 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.

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 in collaboration with domain experts from many different fields led us to understand that it is necessary to strike a good balance between complexity and understandability. In fact, due to interactions with domain experts we have gravitated rather quickly towards diagrams which capture hardly more

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

than classes and possible relationships between them, thus omitting most semantic aspects, like, whether a relationship between classes does or does not indicate domain or range restrictions or even more complex logical axioms. We found that the exact semantics is then better conveyed using either natural language sentences or logical axioms (preferably in the form of rules [**?**]) in conjunction with a very simplified diagram.

After several years of creating ontologies by first drawing schema diagrams with domain experts and subsequent capturing of the exact logial axioms which constitute the ontology, we now, in this paper, reverse the process: We want to start with the logical axioms and derive from them, automatically, the schema diagrams which follow the paradigm which we found most helpful. 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.

In a sense, we attempt to maximize information while minimizing clutter. To do so, we follow a number of principles:

- All classes inherit from owl:Thing, so it is not necessary to clutter a diagram with a corresponding subclass edge for every concept.
- We do not represent any logical connectives, or complex axioms, other than direct subClass relationships between named classes, since in our experience this type of information is better conveyed by non-visual means.
- Disjointness of classes does not need explicit graphical representation. In most cases, disjointness (or not) is immediately clear for a human with some knowledge about the domain.
- Inverse relations are not represented, as they are technically syntactic sugar for any relation.
- The ABox is disregarded; instances of classes are not represented.

With these assumptions in mind, we detail our method, with references to rules below:

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.

We note that Steps 3 and 4 may be omitted if there are no direct domain or range restrictions given. However, due to multiple ways of expressing the same information in OWL, domain and range may or may not appear in the declarations of the Object or Datatype Properties.

Steps 1-4 are straightforward. However, for Step 5, it is important to note the differences between logical and schematic equivalence. We define *schematic equivalence*: two statements are schematically equivalent if we can represent each statement with the same graphical representation following our approach. 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, we see that both may be represented by the same artifact in the graph, namely an edge from A to B, labelled R. 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 logical equivalence 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

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

The following two are used only in the union case as displayed – the first is again *not* a logical equivalence 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.

Let us briefly look at the time complexity for our method. Consider c to be the maximum number of concepts and datatypes in any one axiom in the ontology. Then, at most, there are $\binom{c}{2}$ 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 of course ignores algorithms for the graph layout, from which we abstract in this paper.

As the goal of our approach is to generate static schema diagrams, it of course has practical limitations as to the size of the ontology which can be dealt with: Any schema diagram becomes essentially unreadable if it gets too large. Indeed, our approach is primarily meant for smaller OWL files, such as those constituting ontology design patterns [10] or ontology modules [16]. Larger projects would first have to be broken down into modules before creating separate schema diagrams for each.

4 Implementation

Our prototype implementation, SDOnt, is a pipeline consisting of three parts: a GUI, the parser module, and the rendering module. SDOnt is developed in Java and provided as an executable JAR file; all manipulations of the ontology are done using the OWLAPI. We provide the source code, test set, evaluation results, and a tutorial for the tool's use online.[11]

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

The rendering module utilizes the library yFiles[12] for generating, laying out, and displaying the schema diagram. yFiles is a closed source library maintained

[11] `http://dase.cs.wright.edu/content/sd4odp`

[12] `http://www.yworks.com/products/yfiles-for-java`

and sold by yWorks.[13] We chose to use yFiles as our base visualization library as it also powers the popular diagramming tool yED.[14] However, our codebase provides a largely modular and extensible framework. In principle, any visualization library can be used to render the schema diagram.

5 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 individual results for selected patterns in Section 5.4, and finally, we discuss the overall performance of these methods with respect to the reference diagrams in Section 5.5.

5.1 Compared Tools

Here we briefly describe the two tools against which we evaluated SDOnt.

VOWL is a graphical notation for OWL. The specification can be viewed in detail in [20]. For the purposes of this evaluation, we used WebVOWL[15] in order to generate visualizations of ontologies. These visualizations are very detailed and include an immense amount of information. The representation of the ontology is laid out using a force-directed graph.

OWLGrEd is a Graphical Ontology Editor. It 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 the axioms. All axioms are displayed, sometimes as individual nodes. Additionally, the visualization is intended to be hierarchical, thus there is a subClass edge between an owl:Thing node and every concept in the ontology's signature.

For our evaluation of OWLGrEd, we needed to utilize both the web and desktop applications. During our evaluation, we encountered several OWL files that did not work with the web application. While we did not receive any meaningful error data, we surmise that the web application would visualize the loaded ontology as well as its imports. However, it would only display the imported ontologies and not the loaded ontology. In order to evaluate these ontologies, we used the desktop application to render the visualizations. To the best of our knowledge and judgement, the generated graphs are structurally identical although they differ significantly in layout. We ignore layout issues in our evaluation, i.e., the use of the two different versions of OWLGrEd for different ontologies does not impact the evaluation.

Even utilizing both applications, two ontologies could not be visualized using OWLGrEd. The ComputationalEnvironment ontology failed, likely due to embedded controlled vocabularies. The Event pattern failed but did not give

[13] https://www.yworks.com/
[14] https://www.yworks.com/products/yed
[15] http://www.visualdataweb.de/webvowl/

any descriptive error messages for debugging. The Event ontology hung during processing with no discernible error message.

5.2 Comparison Scheme

We seek to show that the method for constructing schema diagrams for ontology patterns and modules, as introduced in the previous section, results in a diagram most similar to published reference diagrams which follow the visualization paradigm which 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 which have been published in the papers or on the websites where the corresponding ontologies have been discussed by their authors. I.e., these diagrams have been designed with human understandability in mind, and their creation pre-dates our automated diagram generation method.

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

In order to have some useful 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 here not meant to be formal technical terms in the sense of OWL restrictions or RDFS domain/range declarations, but rather intuitive notions which are as ambiguous as a schema diagram: An edge from class A to class B in the diagram indicates that A is (informally) in the domain of the relation, and B is in the codomain of the relation. However, 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?

We state the results of each comparison as an F_1-score in Table 1 using the criteria below.

- 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.

5.3 Test Set

For our evaluation, we are concerned with a very specific visualization paradigm. We have worked closely with domain experts in different fields. In our experiences, we have fine-tuned a visualization paradigm that maximally conveys information to domain experts using a minimalist style of representation.

For this reason, for our evaluation we have chosen ontologies with published and manually drawn schema diagrams that are representative of this visualization paradigm. As the generation of schema diagrams is targeted at small ontologies or parts of ontologies, we have furthermore selected corresponding ontologies and in particular ontology design patterns. Many of these ontologies and patterns are available on ontologydesignpatterns.org. Our process for selecting the patterns was simply that we searched the main publishing outlets and ontologydesignpatterns.org for diagrams drawn in the style which we find most useful.

The complete set of ontologies we used for our evaluation is: Activity [26], AgentRole [18], ComputationalEnvironment [6], DetectorFinalState [3], Event [17], LifeCycleAssessment [14], MaterialTransformation [13], MicroblogEntry [24], ModifiedHazardousSituation [5], Trajectory [12], and Tree [4]. We also provide all test data on our tool's website.[16] We have made no changes to any of the OWL files during this evaluation, and we are reporting results on all OWL files which we tried and could successfully process with at least two of the compared tools.

5.4 Results

The data from our evaluation is given in Table 1 – TP, FP and FN values are hand-counted from the diagrams. To formally ecompare the performances, we ran three Wilcoxon signed rank tests, with null hypothesis that there were no difference in performance. The test shows that SDOnt is significantly better than VOWL ($p < 0.005$), that SDOnt is significantly better than OWLGrEd ($p < 0.001$) and that VOWL is significantly better than OWLGrEd ($p < 0.05$).

In this evaluation, we did not encounter any false positives that were a misrepresentation of an axiom. Instead, false positives were 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 [5], 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, which we assume to to be the case with the LifeCycleAnalysis [14] and Event [17] patterns.

In general, the reasons for the lower performances by VOWL and OWLGrEd remained mostly the same across the different test ontologies. Both frequently

[16] `http://www.dase.cs.wright.edu/content/sd4ont`

Table 1: Results for the Trajectory Design Pattern.

	SDOnt				VOWL				OWLGrEd			
	TP	FP	FN	F_1	TP	FP	FN	F_1	TP	FP	FN	F_1
Activity	9	7	0	0.720	5	11	4	0.400	5	11	3	0.417
AgentRole	1	3	3	0.250	2	8	2	0.286	2	14	2	0.200
ComputationalEnvironment	31	3	0	0.954	30	14	1	0.800	x	x	x	x
DetectorFinalState	22	4	0	0.917	11	11	10	0.512	11	37	11	0.314
Event	2	5	2	0.364	2	3	2	0.444	x	x	x	x
LifeCycleAssessment	4	0	9	0.471	4	5	9	0.364	4	22	9	0.205
MaterialTransformation	8	1	0	0.941	8	12	0	0.571	6	12	2	0.462
MicroblogEntry	10	0	0	1.000	8	3	2	0.762	8	4	2	0.727
ModifiedHazardousSituation	14	1	1	0.933	12	7	3	0.706	14	8	1	0.757
Trajectory	12	0	0	1.000	12	18	0	0.571	12	26	0	0.480
Tree	8	4	0	0.800	8	8	0	0.667	7	11	1	0.538
Total	121	28	15	0.849	102	100	33	0.605	69	145	31	0.439

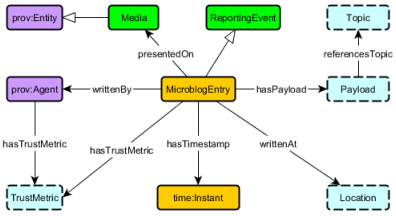
rendered more information than was strictly necessary for conveying information about the structure or purpose of an ontology. They attempt to provide a visualization for every axiom and relation, but as mentioned, this is undesirable for a schema diagram in our opinion. In Figure 1, we have presented example output for all the tools.

We see this when VOWL displays multiple edges with the same label and different subscripts based on different functional properties, such as invertibility and maximum or minimum cardinality. Therefore, if an Object Property had multiple functional properties, there was an additional edge for each. This problematically lead to increased clutter of the visualization and decreased performance in the evaluation. Additionally, blank nodes representing the intersection or union of classes are generating, such as in Figure 1c.

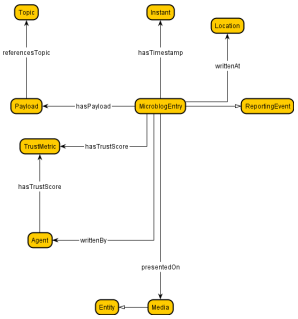
As for OWLGrEd, it casts every visualization into a hierarchical view. Thus, it would draw a subclass edge from a class to owl:Thing, which we can see in Figure 1d. In order to provide a more realistic evaluation given this unwanted behavior, these subclass edges were counted only once per visualization. Furthermore, OWLGrEd rendered an edge between two distinct owl:Thing nodes for every distinct property in the ontology's signature for no discernible reason and this behavior could not be turned off.

We now discuss the results for some selected ontologies in more detail.

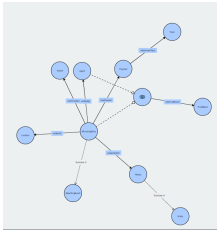
Activity The OWL file [26] contains more information than the diagram depicts, which is the source of the false positives for all three methods. The false negatives in VOWL and OWLGrEd stem from non-representation of datatypes. The evaluation for OWLGrEd on this pattern utilized the visualization from the desktop application, as the web application failed to display the correct pattern, possibly due to an import error.



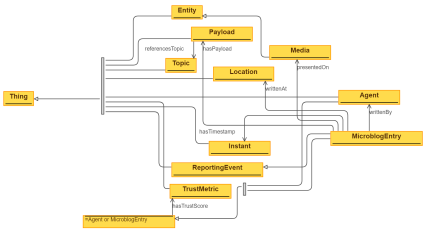
(a) The Reference Diagram.



(b) Schema Diagram generated by SDOnt.



(c) Schema Diagram generated by VOWL.



(d) Schema Diagram generated by OWLGrEd.

Fig. 1: The manually curated reference diagram (a) followed by the generated schema diagrams by SDOnt, VOWL, and OWLGrEd, for the MicroblogEngry ontology design pattern.

AgentRole Based on each of the renderings, we may guess that the diagram is out of date. The reference diagram was obtained from [18]. The original reference diagram has two properties: startsAtTime and endsAtTime with domain TimeInstant. However, the generated diagrams utilize TemporalExtent instead, suggesting a design change. We can use this case in order to help motivate our point that published ontologies can only benefit from having an easy way to update the schema diagram.

The evaluation for OWLGrEd on this pattern utilized the visualization from the desktop application, as the web application failed to display the correct pattern, possibly due to an import error. OWLGrEd also rendered artifacts pertaining to disjointness that were counted as false positives.

DetectorFinalState SDOnt performed suitably well on this ontology [3] and only displayed extra information pertaining to the SelectionCriteria class. The poor performance by VOWL is due to a large amount of doubly rendered edges. Both VOWL and OWLGrEd were unable to identify properties embedded in some axioms and therefore failed to render many of the artifacts and exacerbated

the issue by rendering blank nodes representing entire axioms. Additionally, OWLGrEd rendered artifacts pertaining to disjointness that were counted as false positives.

Event This pattern has a large error for each tool, in that it provides a number of axioms related to owl:Thing, rather than to the core concept Event. This is likely an error with the tool that generated the OWL file. We note that this case may serve as motivation that a proper tool for generating schema diagrams may help match an OWL file under development to the reference diagram.

LifeCycleAssessment This OWL file is malformed [14]. It contains only declarations and a few axioms; it does not seem like the OWL file was saved properly before being uploaded. As such the performance for each rendering is very low. We note that this case is another example where having a schema diagram generated periodically for the OWL file would help with preventing these issues. Finally, we note that OWLGrEd also rendered artifacts pertaining to disjointness that were counted as false positives further impacting its performance.

ModifiedHazardousSituation SDOnt missed only one artifact for this pattern [5]. In fact, all threetools missed the same axiom – and it turns out that the property is completely missing from the OWL file. Aside from this, VOWL and OWLGrEd were subject to the same errors: inability to extract properties from complex axioms and duplication of inverse edges.

Tree The reference diagram [4] did not contain nodes for the non-leaf and non-root classes, likely for clarity and simplicity, but were included in the OWL file. All three tools depicted these classes and their relations leading to an overall lower score for each.

5.5 Discussion

In summary, we see that SDOnt produces schema diagrams that are much closer to the reference diagrams than VOWL or OWLGrEd. In fact, SDOnt is outperformed in only two cases: AgentRole and Event, both of which we assume to be the result of malformed OWL files.

There are also motivating cases for using schema diagrams as error checkers during ontology development. AgentRole's schema diagram, for example, contains classes that are not present in the reference diagram. This is a very obvious sign of a problem with the reference diagram or the OWL file. In other cases, such as in LifeCycleAssessment, there are only three edges in the entire generated diagram, leaving most classes isolated.

VOWL and OWLGrEd consistently performed worse than SDOnt for two reasons. First, VOWL had many duplicated edges for different functional properties. Secondly, both OWLGrEd and VOWL had trouble extracting properties

from complex axioms. In fact, for OWLGrEd, these axioms were represented as anonymous nodes, leading to false positive artifacts.

6 Conclusions

Our results were promising. Even with malformed or unexpected information contained in a few of the OWL files, SDOnt had an $F_1 = 0.849$, performing roughly 25% and 50% better than VOWL and OWLGrEd, respectively. The differences were statistically significant with $p < 0.005$ or better. To be fair, we have evaluated against a very specific visualization paradigm, to which neither VOWL nor OWLGrEd 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 [11]. 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. In addition, we note that the manually created reference diagrams are fallible or that it is simply unclear from the perspective of the OWL file, which information is strictly 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 further increase the modularity of the implementation so that it is 100% independent of any visualization library. This will, in fact, allow us to provide SDOnt as a library, so that any developer may utilize it to generate visualizations. At this point in time, we expect to use GraphML as a middle layer. In addition, we will investigate the most effective ways of creating a good layout, 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,[17] OWLax,[18] and OWL2DL[19] [22, 23, 25], 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).

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

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. D. Carral, M. Cheatham, S. Dallmeir-Tiessen, P. Herterich, M. D. Hildreth, P. Hitzler, A. Krisnadhi, K. Lassila-Perini, E. Sexton-Kennedy, C. Vardeman, and G. Watts. An ontology design pattern for particle physics analysis. In E. Blomqvist, P. Hitzler, A. Krisnadhi, T. Narock, and M. Solanki, editors, *Proceedings of the 6th Workshop on Ontology and Semantic Web Patterns (WOP 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pensylvania, USA, October 11, 2015*, volume 1461. CEUR-WS.org, October 2015.
4. D. Carral, P. Hitzler, H. Lapp, and S. Rudolph. On the ontological modeling of trees. In *8th Workshop on Ontology Design and Patterns – WOP2017*, October 2017. To appear.
5. 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.
6. M. Cheatham, C. Vardeman, N. Karima, and P. Hitzler. Computational environment: An ODP to support finding and recreating computational analyses. In *8th Workshop on Ontology Design and Patterns – WOP2017*, October 2017. To appear.
7. A.-S. Dadzie and M. Rowe. Approaches to visualising linked data: A survey. *Semantic Web*, 2(2):89–124, Apr. 2011.
8. V. Geroimenko and C. Chen. *Visualizing the Semantic Web: XML-based Internet and Information Visualization*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
9. 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.
10. 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.
11. 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.
12. Y. Hu, K. Janowicz, D. Carral, S. Scheider, W. Kuhn, G. Berg-Cross, P. Hitzler, M. Dean, and D. Kolas. A geo-ontology design pattern for semantic trajectories. In T. Tenbrink, J. Stell, A. Galton, and Z. Wood, editors, *International Conference on Spatial Information Theory*, pages 438–456. Springer, 2013.
13. C. F. V. II, A. A. Krisnadhi, M. Cheatham, K. Janowicz, H. Ferguson, P. Hitzler, and A. P. C. Buccellato. An ontology design pattern and its use case for modeling material transformation. *Semantic Web*, 8(5):719–731, 2017.
14. K. Janowicz, A. Krisnadhi, Y. Hu, S. Suh, B. P. Weidema, B. Rivela, J. Tivander, D. E. Meyer, G. Berg-Cross, P. Hitzler, W. Ingwersen, B. Kuczenski, C. Vardeman, Y. Ju, and M. Cheatham. A minimal ontology pattern for life cycle assessment

data. In E. Blomqvist, P. Hitzler, A. Krisnadhi, T. Narock, and M. Solanki, editors, *Proceedings of the 6th Workshop on Ontology and Semantic Web Patterns (WOP 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pensylvania, USA, October 11, 2015.*, volume 1461 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

15. 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.
16. 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.
17. A. Krisnadhi and P. Hitzler. A core pattern for events. In *Proceedings of the Workshop on Ontology and Semantic Web Patterns (7th edition), Kobe, Japan*, 2017. To appear.
18. A. A. Krisnadhi. *Ontology pattern-based data integration*. PhD thesis, Wright State University, 2015.
19. 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.
20. S. Lohmann, S. Negru, F. Haag, and T. Ertl. Visualizing ontologies with VOWL. *Semantic Web*, 7(4):399–419, 2016.
21. A. D. Nicola, M. Missikoff, and R. Navigli. A software engineering approach to ontology building. *Information Systems*, 34(2):258–275, 2009.
22. 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.
23. 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.
24. C. Shimizu and M. Cheatham. An ontology design pattern for microblog entries. In *8th Workshop on Ontology Design and Patterns – WOP2017*, October 2017. To appear.
25. 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.

26. B. Yan, Y. Hu, B. Kuczenski, K. Janowicz, A. Ballatore, A. A. Krisnadhi, Y. Ju, P. Hitzler, S. Suh, and W. Ingwersen. An ontology for specifying spatiotemporal scopes in life cycle assessment. In C. d'Amato, F. Lécué, R. Mutharaju, T. Narock, and F. Wirth, editors, *Proceedings of the 1st International Diversity++ Workshop co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 12, 2015.*, volume 1501 of *CEUR Workshop Proceedings*, pages 25–30. CEUR-WS.org, 2015.