Visualizing Ontologies with VOWL

Steffen Lohmann\textsuperscript{a,\ast}, Stefan Negru\textsuperscript{b,\ast\ast}, Florian Haag\textsuperscript{a}, Thomas Ertl\textsuperscript{a}
\textsuperscript{a} Institute for Visualization and Interactive Systems (VIS), University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
E-Mail: \{steffen.lohmann, florian.haag, thomas.ertl\}@vis.uni-stuttgart.de
\textsuperscript{b} Faculty of Computer Science, Alexandru Ioan Cuza University, Strada General Henri Mathias Berthelot 16, 700483 Iasi, Romania
E-Mail: stefan.negru@info.uaic.ro

Abstract. Visualizations can be very useful when working with ontologies. The Visual Notation for OWL Ontologies (VOWL) is a comprehensive and well-specified visual language for the user-oriented representation of ontologies. It defines graphical depictions for most elements of the OWL Web Ontology Language that are combined to a force-directed graph layout visualizing the ontology. In contrast to related work, VOWL aims for an intuitive representation that is also understandable to users less familiar with ontologies. This article presents VOWL in detail and describes its implementation in two different tools: ProtégéVOWL and WebVOWL. The first is a plugin for the ontology editor Protégé, the second a standalone application entirely based on open web standards. Both tools demonstrate the applicability of VOWL by means of various ontologies. In addition, the results of three user studies conducted to evaluate the comprehensibility and usability of VOWL are summarized. They are complemented by latest insights gained from an expert interview and from testing the visual scope and completeness of VOWL with a benchmark ontology. The evaluations helped to improve VOWL and confirm that it creates comparatively intuitive and usable ontology visualizations.

Keywords: OWL, VOWL, ontology, visualization, user-orientation

1. Introduction

Ontologies have received a lot of attention with the rise of the Semantic Web as a way to give information well-defined meaning \cite{12}. They are nowadays used in many different contexts to structure and organize information \cite{60}. As a consequence, an increasing number of people in modern knowledge societies get in contact with ontologies. They are no longer exclusively used by ontology experts but also by non-expert users. However, especially these casual users often have difficulties in understanding ontologies.

Visualizations can help in this regard by assisting in the development, exploration, verification, and sense-making of ontologies \cite{29,40,44}. They are particularly useful to casual users, but can also provide a new perspective on ontologies for experts. Although several visualizations for ontologies have been developed in the last couple of years, they either focus on specific aspects of ontologies or are hard to read for casual users. Furthermore, many visualizations are tailored for a specific task or use special types of diagrams that must first be learned to understand the visualization. A review of existing ontology visualizations is given in Section 2 of this article.

The Visual Notation for OWL Ontologies (VOWL) aims to fill this gap by defining a comprehensive visual language for the representation of ontologies that can also be understood by non-expert users with only little training. It is designed for the OWL Web Ontology Language \cite{63}, which has become the de facto standard to describe ontologies. VOWL defines graphical depictions for OWL elements that are combined to a graph visualization representing the ontology. It can be

\textsuperscript{1}Invited extended version of a paper submitted to EKAW 2014.
\textsuperscript{\ast}Corresponding author.
\textsuperscript{\ast\ast}Stefan Negru is now with MSD IT Global Innovation Center.
used to generate static ontology visualizations, while it also supports the interactive exploration of ontologies and the customization of the visual layout.

An early version of VOWL has been presented in [56] and compared to the UML-based visualization of ontologies in [55]. Based on insights from that comparison and further feedback, the notation was completely reworked and VOWL 2 was developed, with significant improvements and more precise mappings to OWL. VOWL 2 will be presented in Section 3, followed by a description of two tools that implement it in Section 4: ProtégéVOWL, a plugin for the ontology editor Protégé [3], and WebVOWL, a responsive web application entirely based on open standards.

VOWL and its implementations have been evaluated in user studies that are summarized in Section 5, followed by an expert interview providing additional insights. Finally, an evaluation of the visual scope and completeness of VOWL based on a benchmark ontology is provided.

2. Related Work

A number of visualizations for ontologies have been presented in the last couple of years [21,29,40,44] while some of them are implemented as standalone applications, most visualizations are provided as plugins for ontology editors like Protégé [4].

2.1. Graph Visualizations of Ontologies

Many approaches visualize ontologies as graphs, which is a natural way to depict the structure of the concepts and relationships in a domain of knowledge. The graphs are often rendered in force-directed or hierarchical layouts, resulting in appealing visualizations. However, only few visualizations show complete ontologies, but most focus on certain aspects. For instance, OWLViz [37], OntoTrack [45], and KC-Viz [52] depict only the class hierarchy of ontologies. GLOW [36] also depicts the class hierarchy but uses a radial tree layout and hierarchical edge bundles to display additional property relations. OWLPropViz [64], OntoGraf [24], and FlexViz [25] represent different types of property relations, but do not show datatype properties and property characteristics required to fully understand ontologies.

A smaller number of works provide more comprehensive graph visualizations that represent all key elements of ontologies. Unfortunately, the different ontology elements are often hard to distinguish in the visualizations. For instance, TGViz [8] and NavigOWL [39] use very simple graph visualizations where all nodes and links look the same except for their color. This is different in GrOWL [43] and SOVA [13], which define more elaborated notations using different symbols, colors, and node shapes. Furthermore, as the notations of both GrOWL and SOVA rely on symbols from Description Logic [9] and abbreviations, they are not perfectly suited for casual users. Furthermore, the visualizations created with GrOWL and SOVA are characterized by a large number of crossing edges which has a negative impact on the readability.

There are also 3D graph visualizations for ontologies, such as OntoSphere [15], or tools that use hyperbolic trees to visualize ontologies, such as OntoRama [22] or Ontobroker [26]. However, these works again focus only on specific aspects of ontologies, such as the class hierarchy or the relationships of certain elements.

2.2. Specific Diagram Types

While the reported graph visualizations use common node-link diagrams to represent ontologies, there are also a number of works that apply other diagram types. For instance, Jambalaya [61] and OWL-VisMod [28] use treemaps to depict the class hierarchy of ontologies. Jambalaya additionally provides a nested graph visualization called SHRiMP that allows to split up the class hierarchy into different views [61]. CropCircles is a related visualization technique that visualizes the class hierarchy of ontologies with the goal to support the identification of “undermodeled” ontology parts [65]. All these approaches visualize once again mainly the class hierarchy, without considering other property relations.

Cluster Maps use a visualization technique that is based on nested circles and has also been successfully applied to ontologies [27]. Instead of showing the class hierarchy, Cluster Maps visualize individuals grouped by the classes they are instances of. Each individual is represented by a small circle that is shown inside a larger circle representing the class. Similar techniques are used in VisCover [46] and OOBIAN Insight [2] that additionally provide a number of interactive filtering capabilities. While offering appealing visualizations, these tools show only a selection of classes along with their instances but do not provide complete visualizations of ontologies.
Another related approach is OntoTrix [10], which represents ontologies with the NodeTrix visualization technique, a combination of node-link diagrams and adjacency matrices [34]. While OntoTrix also focuses on the visualization of individuals and their connections within ontologies, it provides a more complete image of the ontology. However, the ontology is converted into the NodeTrix structure for the visualization, making it difficult to get an impression of its global structure and topology.

2.3. UML-like Ontology Visualizations

A powerful type of diagram related to OWL and often reused to visualize ontologies is the class diagram of the Unified Modeling Language (UML) [6]. Precise mappings between OWL and UML class diagrams are specified in the Ontology Definition Metamodel (ODM) [1], among others. There are numerous editors for UML diagrams, but as conventional UML editors cannot read and visualize OWL files, special ontology editors or plugins for UML editors have been developed. Examples are the Visual Ontology Modeler (VOM) [42], TopBraid Composer [5], and OWL-GrEd [11].

A major drawback of these attempts is that they require knowledge about UML class diagrams. Although many people with an IT background are familiar with these types of diagrams, people from other domains have difficulties interpreting them correctly, as found in the aforementioned user study [55]. Since UML has been designed for the representation of software rather than knowledge, there are also some conceptual limitations and incompatibilities when using UML class diagrams for the visualization of ontologies.

A related type of diagram for ontology visualization is used by OntoViz. It groups classes and datatypes in boxes that can be linked by different properties. However, the resulting diagram quickly grows in size and is not very intuitive. This is different in Graffoo [23] which aims for an easy-to-understand notation for OWL diagrams similar to VOWL. However, it is intended to be used in diagram editors and therefore rather related to the idea of UML-based modeling than to the visualization approach that is followed by VOWL.

2.4. Other Visualization Approaches

Since any OWL ontology can be represented as RDF graph, it can also be visualized using the common RDF notation [41]. RDF visualizations are, for instance, provided by the RDF validator of W3C [7] or tools like RDF Gravity [62]. However, the visualizations quickly become large with plenty of nodes and edges, as the OWL constructs are represented by multiple triples in RDF. Such RDF visualizations of OWL are not only hard to read but also fail to adequately reflect the semantics of the OWL constructs.

This is similar for Linked Data visualizations that are also RDF-focused and usually do not consider and visualize complete ontologies [19]. One such tool is RelFinder [32] that visualizes relationships between individuals described by ontologies and makes these relationships interactively explorable. Another tool is gFacet [33] where individuals are grouped by their classes and filtered by selecting linked individuals or data values. Both tools provide some insight into the relationships between a limited set of classes and/or individuals, but they do not visualize complete ontologies. This is similar in the tool LodLive [18] that enables the visual exploration of Linked Data using a graph visualization.

2.5. Discussion of Related Work

Looking at the related work, some common characteristics stand out: Most visualizations utilize a well-known type of diagram for ontology visualization (graph visualization, treemap, UML), are in 2D, and focus on specific aspects of ontologies. Only few attempts aim for a comprehensive ontology visualization. Even less approaches provide an explicit description of the visual notation, i.e., a specification that precisely defines the semantics and mappings of the graphical elements. Often, there is no clear visual distinction between different property types or between classes, properties, and individuals.

Furthermore, many works implement a stepwise approach of ontology exploration, where only a root class is shown at the beginning and the user has to navigate through the visualization (e.g., by expanding or collapsing visual elements). VOWL rather aims for an approach that provides users with an overview of the complete ontology and let them subsequently explore parts of it in depth, following the popular Visual Information Seeking Mantra of “overview first, zoom and filter, then details-on-demand” [58]. This approach was chosen, as it was considered important to give users a visual impression of the size and topology of the ontology before they start to explore it any further.
Most importantly, VOWL aims for an intuitive visualization that is also understandable to users less familiar with ontologies, while most of the related work has rather been designed for ontology experts. It uses a graph visualization, as this is considered a natural and intuitive way to represent the structure of ontologies, which is confirmed by many of the related work reported above.

3. Visual Notation for OWL Ontologies (VOWL)

So far, two versions of VOWL have been specified at http://purl.org/vowl/spec/. While the first version emerged from the idea of providing an integrated representation of classes and individuals [56], VOWL 2 focuses on the visual representation of classes, properties, and datatypes. This information is known as the Terminological Box (TBox) in Description Logic and distinguished from the individuals and data values making up the Assertional Box (ABox). The TBox is usually most important to understand the conceptualization described in an ontology and therefore considered ‘first-class citizen’ in most ontology visualizations.

VOWL 2 addresses primarily the TBox and only optionally integrates some ABox information in the visualization. It recommends to display detailed information about individuals in another part of the user interface (e.g., a sidebar) that is linked with the visualization. This design decision is supported by comments from a user study on VOWL 1 [55], expressing concerns about the scalability of an integrated TBox and ABox visualization. Even with few individuals per class, additional information, such as property values of individuals, would be difficult to include in the visualization without creating lots of clutter.

3.1. Basic Building Blocks of VOWL

VOWL 2 is based on the graphical representations specified in VOWL 1 but uses a more systematic and modular approach for the visual language. The basic building blocks of VOWL 2 are a clearly defined set of graphical primitives and a color scheme. Both express specific aspects of the OWL elements (e.g., datatype or object property, different class and property characteristics, etc.), while considering possible combinations thereof. In addition, VOWL 2 provides explicit splitting rules that define which elements are multiplied in the graph visualization in which way.

This systematic and modular approach does not only improve the semantics of the visual language but also facilitates its implementation. For instance, the small set of graphical primitives that are consistently varied and combined fit well with object-oriented programming. The same holds for the style information that is specified in a modular way in the stylesheet deployed with the VOWL specification.

Finally, the graph structure was refined in a way in VOWL 2 that it can be more easily visualized (e.g., by avoiding edges between property labels), and developers were given more freedom in the parametrization of VOWL (e.g., by defining an abstract color scheme that can be adapted as needed).

3.1.1. Graphical Primitives

Table 1 lists the small set of graphical primitives that VOWL is based on, along with the ontology elements they are applied to. Classes are depicted as circles that are connected by lines representing the properties with their domain and range axioms. Property labels and datatypes are displayed in rectangles, and text is used for the labels and for cardinality constraints.

Where available and desired, the number of instances that belong to a class can be visually expressed by the circle size. VOWL does not specify a particular scaling method for the circle radius, but proper results will likely be achieved with a logarithmic or square-root scaling in most cases. If instance information is not considered in the visualization, the circles of all classes have the same predefined size. The only exception are the circles of anonymous classes and of classes that represent owl:Thing: They are shown in a smaller size, as they usually do not carry relevant domain information. Even though all individuals in an ontology are instances of owl:Thing according to the OWL specification, this is irrelevant for the visualization so that owl:Thing has always a fixed size.

Most connecting lines in VOWL have an arrowhead that points to the class or datatype that is defined as the range of the property the line represents. If no domain and/or range axiom is defined for a property, owl:Thing is used as domain and/or range.
An exception are datatype properties without a defined range, where rdfs:Literal is used as range instead. Lines of inverse properties have arrowheads at both ends, while the direction of the respective property can be interactively highlighted by hovering over the label. If the same pair of classes is connected by multiple lines, it is recommended to curve the lines to avoid visual overlapping. Subproperty axioms are also indicated by interactive highlighting instead of explicit connections between the property labels (as in VOWL 1) to reduce the number of edge crossings and to facilitate the implementation and interpretation of VOWL.

The rectangles representing property labels have no border to distinguish them from those representing datatypes. The labels depict usually the text for the element given with rdfs:label in the language selected by the user. If no label is given for an element, the last part of the URI is taken, i.e., the string that follows the last slash (/) or hash (#) character. Long labels are abbreviated, but the full text is always available on demand (e.g., displayed in a sidebar or tooltip).

### 3.1.2. Color Scheme

In addition, VOWL defines a color scheme for a better distinction of the different elements (see excerpt in Table 2). The colors are defined by their function in an abstract way and relative to the canvas color to leave room for customization. While concrete colors are recommended, developers may choose different colors as long as they comply with the abstract descriptions. The color scheme also defines how the colors should relate to each other in order to encode the VOWL semantics. For example, the external color is defined to be a “dark version of the general color”, and the datatype color is described as a “light color that is clearly different from the other colors”. Where several of the color mappings may apply, priority rules are specified. One example for such a rule is that the deprecated color has always priority over the external color, as this information is considered to be more important in most cases.

The recommended color scheme has been designed in accordance with the general guidelines: For instance, and inline with the above example, the recommended external color (dark blue) is similar to the general color (light blue), whereas the color for datatype properties (light green) is clearly different. Some of the other colors were picked based on their global characteristics, such as the signal aspect of the highlighting colors (red and orange).

However, colors are not required in order to use the visualization—it is also comprehensible when printed in black and white or viewed by color-blind people. Details that rely on color, such as the subtle distinction of owl:Class and rdfs:Class, may be added as text information in these cases. Other information is by default provided as text so that it remains available in the absence of colors. The text labels also help to make the visualization more self-explanatory, as was found in a user study [51]. Similar to the conflicting colors, the VOWL specification contains guidelines on how to combine conflicting text labels (in most cases, they are displayed as a comma-separated list (e.g., “deprecated, external”).

### 3.2. Visual Elements

VOWL provides visual elements for most language constructs of OWL 1 and 2, including those of RDF and RDFS reused in OWL. The representations are based on the graphical primitives and the color schema described above; a selection is shown in Figure 1. As already mentioned, information is redundantly encoded in several cases. One such example is the visual element for external classes, i.e. classes whose base URI differs from that of the visualized ontology. It has both the external color, as defined in the color scheme, and the hint “external” beneath its label. Other examples are the visual elements for class disjointness and logical disjunction that use either a mathematical symbol or text label in combination with an illustration reminiscent of Venn diagrams. These illustrations help to communicate the underlying set operations to users who are not familiar with the mathematical symbols and names of the concepts.

Some of the visual elements were inspired by UML class diagrams, such as the notations for subclass relations and cardinality constraints. The representations of these elements were considered intuitive by many participants of the user study that compared VOWL 1 to the UML-based visualization of ontologies [55]. The notations of object and datatype properties dif-

<table>
<thead>
<tr>
<th>Table 2: Excerpt of the VOWL color scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>General</td>
</tr>
<tr>
<td>External</td>
</tr>
<tr>
<td>Deprecated</td>
</tr>
<tr>
<td>Datatype</td>
</tr>
<tr>
<td>Datatype property</td>
</tr>
<tr>
<td>Highlighting</td>
</tr>
</tbody>
</table>
fer only in color, which is sufficient, as object properties usually point to classes and datatype properties to datatypes, so that they remain distinguishable even without colors.

It is important to note that certain elements are merged in the visualization. Equivalent classes are one example, which is visually indicated by a double circle border in the graphical representation. The idea behind the merging is that equivalent classes share the same properties, among others, and can therefore be represented by only one element in the visualization. The labels of merged elements are shown in brackets so that they are still available to the viewer.

The graphical representations for further OWL elements are defined in the VOWL specification. Most of the remaining representations are variations of the ones presented in Figure 1 and visualize specific property characteristics (e.g. functional, transitive) or other set operators (e.g. intersection, complement), among others.

3.3. Graph Visualization

The visual elements are combined to a graph representing the entire ontology. By default, VOWL graphs are visualized using a force-directed algorithm. Such an algorithm tends to arrange the nodes in a way that highly connected classes are placed more to the center of the visualization, while less connected ones are rather placed in the periphery. This helps to reflect the relative importance of the classes in the resulting graph layout, as the number of connections of a class is often an indication of its importance in the ontology [57]. Moreover, graph layouts generated with force-directed algorithms are usually perceived as aesthetically pleasant, since all edges have roughly the same length and since they tend to avoid edge crossings, which increases the readability of the visualization.

In order to relax the energy of the force-directed layout and reduce the visual importance of generic ontology concepts, certain elements are multiplied so that they may appear more than once in a VOWL graph. Multiplication is realized with the aforementioned splitting rules that determine whether there is no multiplication for elements, multiplication across the entire graph, or multiplication for each connected class. For instance, the generic class owl:Thing is multiplied in a way that it appears once for every class it is connected to, while datatype nodes appear once for every datatype property.
Figure 2 shows the visualization that results for a small ontology if the visual elements are rendered and combined according to VOWL. The visualization represents version 1.0 of the Modular Unified Tagging Ontology (MUTO) [47,48]. It has been created with the WebVOWL tool that is presented in the next section.

4. Implementations

VOWL has been implemented in two different tools that demonstrate its applicability: ProtégéVOWL has been realized as a Java-based plugin for the ontology editor Protégé [3], whereas WebVOWL is a standalone application based on web technologies. Both tools are released under the MIT license and are publicly available at http://vowl.visualdataweb.org. A demo of ProtégéVOWL has been presented at ESWC 2014 [50]; a demo of WebVOWL will be presented at EKAW 2014 [49].

4.1. ProtégéVOWL: VOWL Plugin for Protégé

Protégé is currently the most widely used ontology editor [3,35]. It is an open source project developed at Stanford University and can be used free of charge. Although a lightweight version of Protégé is available as web application (WebProtégé), the more popular and powerful tool is the Java-based Desktop version of the editor (Protégé Desktop) at the moment. It has an open plugin structure that allows for the integration of special-purpose components, such as ontology visualizations [4].

Like VOWL 2, ProtégéVOWL focuses on the visualization of the ontology schema, i.e., the classes, properties, and datatypes (TBox), while it does not consider individuals and data values (the ABox) for the time being. ProtégéVOWL is deployed as a JAR file that must be copied to the plugins folder of the Protégé installation. Figure 3 shows a screenshot of ProtégéVOWL depicting version 1.35 of the SIOC Core Ontology [14]. The user interface consists of three parts: The VOWL Viewer displaying the ontology visualization, the VOWL Sidebar listing details about the selected element, and the VOWL Controls allowing to adapt the force-directed graph layout.

4.1.1. VOWL Viewer

ProtégéVOWL makes use of the visualization toolkit Prefuse [31] to render the visual elements and to arrange them in a force-directed graph layout. It accesses the internal ontology representation provided by the OWL API of Protégé and transforms it into the data model required by Prefuse. The OWL elements are mapped to the graphical representations as specified by VOWL and combined to a graph. Prefuse uses a physics simulation to generate the force-directed graph layout consisting of three different forces: Edges act as springs, while nodes repel each other, and drag forces ensure that nodes settle. The forces are iteratively applied, resulting in an animation that dynamically positions the nodes.

Users can smoothly zoom in to explore certain ontology parts in detail or zoom out to analyze the global structure of the ontology. They can pan the background and move elements around via drag and drop, to further optimize the graph visualization and adapt it to their needs. Whenever a node is dragged, the rest of the nodes are repositioned with animated transitions by the force-directed algorithm.

4.1.2. VOWL Sidebar

Whenever an element is selected in the visualization, details about it are shown in the sidebar, such as for the selected class “User Account” in Figure 3. The details are provided by the OWL API and include the type and URI of the element as well as ontology comments, among others. URIs are displayed as hyperlinks that can be opened with a Web browser or other tools for further exploration.

4.1.3. VOWL Controls

The repelling forces of classes and datatypes are separately adaptable with the two sliders in the VOWL Controls. This allows to fine-tune the force-directed layout in accordance with the size and structure of the visualized ontology. Since datatypes have their own repelling force, they can be positioned in close proximity to the classes they are connected with—to emphasize their radial arrangement and increase the readability of the visualization. In addition, the force-directed algorithm can be paused via the VOWL Controls. This does not only reduce processor load but also allows to rearrange selected elements without an immediate adaptation of the whole layout.

4.2. WebVOWL: Web Implementation of VOWL

Implementing visualizations as plugins for ontology editors like Protégé has the advantage that one does not have to deal with ontology processing. The parsing of the OWL files and their transformation into an efficient
data structure is performed by the ontology editor, or the library it uses for this purpose, such as the OWL API in case of Protégé.

4.2.1. Ontology Preprocessing

This is different in WebVOWL, which is not a plug-in like ProtégéVOWL but a standalone application. It is based on open web standards (HTML, JavaScript, CSS, SVG) and runs completely on the client-side, i.e. does in principle not require any server-side processing. Instead of being tied to a particular OWL parser, WebVOWL defines a JSON schema that ontologies need to be converted into.

The JSON schema is optimized with regard to VOWL, i.e., its format differs from typical OWL serializations in order to enable an efficient generation of the interactive graph visualization. Due to this fact, it is also different from other JSON schemas that emerged in the context of the Semantic Web, such as RDF/JSON [20] and JSON-LD [59]. The JSON-VOWL file contains the classes, properties, and datatypes of the ontology along with type information (owl:Class, owl:ObjectProperty, xsd:dateTime, etc.). Additional characteristics (inverse, functional, deprecated, etc.) as well as header information (ontology title, version, etc.) and optional ontology metrics (number of classes, properties, etc.) are separately listed. If no ontology metrics are provided in the JSON file, they are computed in WebVOWL at runtime. Instance information is currently not included in the JSON-VOWL file but planned to be added in the future.

Even though WebVOWL is based on JavaScript, the transformation of the OWL ontology into the JSON file can also be done with other programming languages. By default, WebVOWL is deployed with a Java-based OWL2VOWL converter that uses the well-tested OWL API of the University of Manchester [38]. The converter accesses the ontology representation provided by the OWL API and transforms it into the JSON format required by WebVOWL. It already applies the rules for node splitting and node aggregation specified by VOWL to allow for annotated JSON files and a more efficient generation of the graph visualization.

4.2.2. User Interface and Visualization

Figure 4 shows a screenshot of WebVOWL (version 0.2.13) visualizing version 1.5 of the Personas Ontology [53,54]. The user interface of WebVOWL is similar to that of ProtégéVOWL, consisting of the ontology visualization in the main view, a sidebar with details about the selected element, and a menu of controls and options.
Fig. 4. WebVOWL has a similar user interface as ProtégéVOWL, consisting of the ontology visualization, a sidebar, and a menu with controls.

The SVG-based visualization is generated from the JSON file at runtime. WebVOWL renders the graphical elements according to the VOWL specification, i.e., it takes the SVG code and CSS styling information provided by the specification. The force-directed graph layout is created with the JavaScript library D3 [16]. It is based on a physics simulation similar to the one of Prefuse used in ProtégéVOWL. The forces cool down in each iteration and the layout animation stops automatically after some time to remove load from the processor and provide a stable graph visualization.

Users can interact with the graph visualization in a similar way as in ProtégéVOWL. They can zoom in and out, pan the background, and move elements around to adapt the force-directed layout. Certain elements (e.g., subproperties) are interactively highlighted according to the VOWL specification. Details about selected elements are once again shown in the sidebar (partly as hyperlinks), such as for the selected class “AffectiveState” in Figure 4. In addition, the sidebar displays metadata about the ontology, such as its title, namespace, author(s), and version, as well as the ontology description and the aforementioned metrics provided by the JSON file or computed at runtime. The information in the sidebar is displayed in an accordion widget to save screen space.

Like ProtégéVOWL, WebVOWL provides two gravity sliders that allow to adapt the repelling forces of classes and datatypes, respectively. It also comes with a pause button to suspend the automatic layout in favor of a manual positioning of the nodes. Furthermore, WebVOWL implements some features that complement the ones provided by ProtégéVOWL. One such feature is a special “pick-and-pin” mode inspired by the RelFinder tool [32]: It allows to decouple selected nodes from the automatic layout and pin them at fixed positions on the canvas. Pinned nodes are indicated by a needle symbol (cf. classes “Persona” and “Person-aType” in Figure 4) that can be removed to recouple the nodes with the force-directed layout.

Another class of interactive features provided by WebVOWL are filters that help to reduce the size of the VOWL graph and increase its visual scalability. Currently, two filters are implemented, one for datatypes and another for “solitary subclasses”, i.e., subclasses that are only connected to their superclass but no other classes. Finally, WebVOWL allows to export the complete graph visualization or a portion of it as SVG image that can be opened in other programs, scaled without loss of quality, edited, shared, and printed.

4.2.3. Discussion of WebVOWL

To the best of our knowledge, WebVOWL is the first tool for comprehensive ontology visualization that is completely based on open web standards. Like ProtégéVOWL, most other ontology visualizations are implemented in Java (see Section 2). Related tools running in web browsers, such as FlexViz [25] or OOBIAN Insight [2], are based on technologies like Adobe Flex or Microsoft Silverlight that require proprietary browser plugins. While the tool LodLive [18] is also based on open web standards and technically related to WebVOWL, it focuses on the visual explo-
ration of Linked Data and not on the visualization of ontologies.

WebVOWL works in all major browsers, except for the current version of Internet Explorer, which lacks support for several SVG features. However, cross-browser compatibility is an issue of any application based on open web standards and a general drawback compared to Java-based tools like ProtégéVOWL. Moreover, WebVOWL has been designed with different interaction contexts in mind, including settings with touch interfaces. For instance, zooming can either be performed with the mouse wheel, a double click/tap, or a two fingers zooming gesture. As some features may not be available in all situations (e.g., mouseover effects), care has been taken that these features are not crucial for the interaction and for understanding the ontology.

5. Evaluations

In order to evaluate the comprehensibility and usability of VOWL, a number of experiments of different types have been conducted.

5.1. Comparisons with Other Ontology Visualizations

Previous work has compared VOWL to other approaches of ontology visualization in three qualitative user studies. None of the participants in these studies had extensive prior knowledge about ontologies or other Semantic Web technologies, but they could rather be regarded as lay users.

In the studies, participants were first provided with a brief explanation of the visual notations and the underlying ontology concepts. Then, they were presented various ontologies visualized on a screen. Questions about these ontologies were asked, which had to be solved by looking at the visualizations. Among those questions, some referred to the general structure of the visualized ontologies (e.g., “What is the approximate number of classes visible?”), while others focused on particular ontology elements (e.g., “Which property is the inverse of the property P?”). Moreover, participants were asked for comments on the visualizations, with respect to aspects such as general overview, choice of shapes and colors, the level of intuitive comprehensibility or confusion when looking at the visualization, perceived completeness of displayed information and suggestions for improvement.

Ontology visualizations aiming at an overview over the entire ontology and at approximating feature-completeness typically follow a node-link diagram approach (cf. Section 2). Hence, all of the alternative visualizations that VOWL was compared to were based on some kind of node-link diagram: The first version of VOWL was compared to an OWL visualization using UML class diagrams [55]. The second comparison [51] focused the WebVOWL implementation of VOWL 2, the GrOWL visualization [43], and the SOVA plugin for Protégé [13]. Finally, the SOVA plugin was also featured in the third evaluation, this time in comparison with the ProtégéVOWL plugin [50]. In the first study, static images of the visualizations were used, while the second and third studies were conducted with implementations and therefore included interactive features.

Overall, participants considered all presented visualizations generally comprehensible. The inherent possibility to distinguish basic elements such as properties and classes was stated to be important, and a reduced number of crossing edges was confirmed to support the overview.

Users who were already familiar with similar visual notations, as was the case for some users with the UML-based ontology visualization, considered that similarity to be beneficial. This knowledge also helped some users interpret unlabeled subclass arrows in VOWL 1. As subclass arrows are labeled in VOWL 2, they could be correctly understood by all users, though one user remarked that with his prior knowledge of UML class diagrams, the subclass label was redundant. Generally, short descriptive labels on elements, as they were included in VOWL, were seen as helpful. In the case of special property characteristics and modifiers, such as functional or transitive, the unabbreviated textual description was considered indispensable for interpreting the visualizations correctly.

The multiplication of nodes, as featured in VOWL, was met with mixed reactions. Remembering the aforementioned explanations on the visual notation, some of the participants quickly understood that generic elements, such as owl:Thing, would appear several times, and that groups of equivalent classes were combined into a single visual node, while others had expected a different behavior—only one representation of the unique class owl:Thing and one separate node per equivalent class—and thus required a moment to adjust to the visualization.

Some of the interactive features requested in the evaluation of VOWL 1 had been incorporated into
VOWL 2 and its implementations. Participants welcomed the continuous zooming option, but asked for additional highlighting features (e.g., a highlighting of all nodes that are directly related to the selected one) as well as a feature to search and highlight specific elements.

In summary, users could solve most tasks well when using VOWL. When a preference for one of the visualizations was stated, it leaned toward VOWL for the majority of participants, based on aspects such as clarity and distinguishability of elements, ease of use with interactive highlighting and layouting features, as well as aesthetics.

5.2. Evaluation by Experienced Ontology Users

To get more insight into how users perceive VOWL, and to become more aware of the differences between users experienced with ontologies and lay users, VOWL has been presented to five users who had previously worked with ontology editors and formal OWL syntax. Their expertise varied between being very familiar with the specifics of OWL and able to define custom ontology elements, and knowing the major OWL features from working with existing ontologies.

The recruited users could interact with the Web-VOWL implementation themselves, while they were gradually told about the interactive features. A quick presentation of the VOWL 2 specification and the Protégé-VOWL implementation [50] were provided for comparison. However, as opposed to the other user studies, no systematic introduction to the VOWL notation was given. Instead, participants started out without any prior knowledge on the meaning of shapes, colors and symbols found in VOWL and were supposed to familiarize themselves with the visualization while adhering to the “think-aloud” method. Hence, they would state everything they were thinking, feeling, considering, or doing with respect to the exploration of various ontologies (FOAF [17], MUTO [47], PersonasOnto [53], SIOC [14]) displayed with Web-VOWL. When participants could not figure out the meaning of a visual element based on the information provided in the visualization or the sidebar, they could ask for an explanation. Moreover, four of the five participants explored the ontologies in groups of two, meaning that they could discuss their thoughts with each other during the interview.

To provide some rough guidance through the features of VOWL and ensure touching upon a wide range of visualization aspects, a number of questions were prepared. Participants were asked these questions to give them an incentive to look at WebVOWL features and aspects of the ontologies they might not notice on their own, and to induce suggestions on missing or desired features. Where appropriate, additional hints were provided during the interview.

The results showed that the participants had no problems in distinguishing classes, properties, and datatypes.

5.2.1. Navigation and Layout

All study participants spoke favorably about the force-directed layout, as it helped them to recognize clusters and the general inheritance relations between classes defined in the ontology. They stated that central elements could be easily recognized due to the layout, and that the gravity options, which control how closely elements are attracted to each other, were helpful to further understand the ontology. While the pick-and-pin feature was generally thought of as useful, one participant even asked for such a feature on his own. Several participants asked for more automated layouts beside the force-directed algorithm and for the ability to focus single elements that the remaining elements should align around. Irrespective of the particular layout chosen, one participant suggested the addition of a minimap to ease navigation in large graphs.

5.2.2. Multiplication

The multiplication of nodes was considered beneficial for the simplicity of the graph structure. In particular, the multiplication of datatypes was seen as desirable by all participants except one, and most participants agreed that they normally would not want to answer questions such as “What datatype properties with range X exist in the ontology?” The one remaining participant did not see the necessity for datatype multiplication, but was not confused by it, either.

5.2.3. Equivalent Classes

The meaning of double-ringed classes as groups of equivalent classes was not inherently clear to any of the participants. Once they had been explained their meaning, they viewed the visual representation as appropriate. One participant wondered why one of the equivalent class names is bigger than the others. Also, one participant did not see the necessity for a distinct visual notation, as he would prefer groups of equivalent classes to look the same as a single normal class instead. All participants were content with the combination of all equivalent classes into a single visual element, though one participant remarked that for edit-
ing and determining the provenance of property definitions, the equivalent class groups might optionally have to be splittable.

5.2.4. Properties

Both object and datatype properties were generally instantly found and recognized for what they were. Three participants asked for options to hide either property labels or property datatypes, as they deemed that information to be relevant only in certain scenarios.

Inverse properties were correctly identified by most participants; one of them explicitly praised the way how pairs of inverse properties were displayed on a single graph edge (i.e., as bidirectional links with arrows at both sides). However, users generally had to be pointed to the highlighting feature indicating which property is associated with which direction. Likewise, the highlighting feature of subproperties was not noticed unless explicitly pointed out. With that explanation, however, the visual representation was comprehensible to users who had some experience with the ontology concept of subproperties.

In summary, while highlighting features of properties are considered helpful, users would not discover them as flawlessly as the permanently displayed elements of the visualization.

5.2.5. Set Operators

Users were not sure they could recognize the visual notation for anonymous classes based upon set operations—union, intersection, and complement—on their own. Three of the users quickly figured out the meaning based on the symbols (∪, ∩, ¬), but did not even notice the Venn diagrams at first. Two users thought the combination of a node border, the Venn diagrams, and the logical symbols was a visual overload that should be tackled, while another one remarked that, were the symbol replaced with a descriptive text, the notation would be more consistent with the rest of VOWL.

5.2.6. Filtering

Several users asked for additional ways of filtering elements in the graph. For example, one user wished for advanced filter criteria such as ranges of properties to hide the respective property edges, while two others thought an option for selectively collapsing and expanding subtrees or subgraphs was missing. The possibility to hide datatype properties and weakly connected subclasses were praised as promising steps into this direction.

5.2.7. Colors

Most colors were not commented on a lot. Two users stated the color coding was not clear to them, especially with respect to external classes. One of them pointed out that the dark color of external elements emphasizes these elements, but did not see a particular reason why external classes should visually call for attention. Two users would have wished for colors that express the class hierarchy—for example, the specialization level in the inheritance tree—in some way, though they admitted such a color coding would probably only be conceivable for strictly hierarchical class relationships rather than the inheritance graphs supported by OWL.

5.2.8. Completeness

When asked whether VOWL lacks any information, three users answered that the amount of information currently displayed is rather almost too much, for which some overlapping labels that occurred during the user study were thought of as proof. Users agreed that disjoint relationship between classes should not be displayed by default, but only for particularly pointing out disjointedness among a given set of classes. One user would have preferred some more explicit information in the sidebar, such as an indication of namespaces along with resource labels. The same user did, however, ask for a way to display other class restrictions that are not currently supported by VOWL, such as restricting the instances of a class to a specific set of resources. Beside these omissions, the visualization was stated to be “quite complete” in terms of OWL 2 features considered relevant by the users.

5.2.9. Possible Use Cases

Lastly, participants of the expert interview were asked how they could imagine to use VOWL. All users agreed with the basic idea of finding out about the structure and content of an ontology, and some mentioned techniques such as navigating through an ontology starting with some core elements. One participant could imagine to use VOWL to align several ontologies. Usage for various editing tasks—related to adding, modifying, and removing ontology elements—was suggested by all users, which also includes using VOWL for the debugging of ontologies.

One user also proposed to show VOWL in a teaching context such as lectures on the Semantic Web, in order to explain OWL concepts to learners. Another user pondered about navigating through ontologies displayed with VOWL as some form of a “decision graph” to choose a certain result.
5.3. Benchmark of the VOWL Visualization

VOWL has additionally been tested with OntoViBe, the Ontology Visualization Benchmark [30], available at http://ontovibe.visualdataweb.org. OntoViBe is an ontology comprising of a comprehensive set of OWL 1 and 2 language constructs and systematic combinations thereof. It has been designed to support the testing of feature-completeness of ontology visualizations.

Figure 5 shows the VOWL visualization of version 1.0 of OntoViBe. It has been created with WebVOWL (version 0.2.13), which provided a nearly complete implementation of VOWL 2 in contrast to ProtégéVOWL at the time the benchmark was performed. The visualization has been annotated to point to some of the aspects discussed in the following.

By viewing OntoViBe in VOWL, the effects of the force-directed layout are noticeable right away. A strongly connected class, PropertyOwner, appears to be very central on the first glance, based on the large number of other nodes it is connected to (annotation ① in Figure 5). The convenient radial placement of outbound and inbound edges, which mostly avoids any overlapping arrowheads, is immediately visible, as was already noted during the user studies. OntoVibe includes a small class hierarchy, which is sufficiently arranged in VOWL (annotation ②). Further visualization features that are directly visible are the merging of equivalent classes to single nodes (also seen in annotation ③), and the use of human-readable labels, where available.

Several kinds of possible conflicts in the definitions are consistently resolved by VOWL. In the case of conflicting colors to express different characteristics of elements, the guidelines from the specification are applied: Deprecated elements appear with light gray background, while the dark blue back-
ground color of external elements overrides the gray background in deprecated external elements such as DeprecatedImportedClass (in annotation 2). The color of the deprecated properties, on the other hand, overrides the standard colors of light blue and green for object and datatype properties, respectively. Another kind of avoided conflict concerns the geometry, as multiple properties between the same pair of classes, such as classToClassProperty1 and classToClassProperty2 do not visually obstruct one another, but are rendered as curved multi-edges. The same applies to sets of cyclic properties whose domain and range axioms point to the same class. An example of this can be seen around the class MultiPropertyOwner (annotation 3).

As OntoViBe features a few classes that are not directly connected to the rest of the graph, the fact that these tend to drift away from the central ontology part up to a certain distance in VOWL is noticeable. While some measures will have to be found for supporting users to find disconnected ontology parts, especially when viewport size is limited, at the same time, this behavior emphasizes the disjointedness of the respective subgraphs. Beside the lack of support for custom OWL 2 data ranges in VOWL—which is intentionally not included for the time being, as VOWL focuses on classes and properties—, only two omissions in the current VOWL syntax became apparent that might be crucial to the class structure of some ontologies: On the one hand, unions, intersections, and complements of classes have only been defined for anonymous classes in VOWL. So far, no notation for named classes that are defined by these set operations exists. On the other hand, in the case of set operation classes that refer to other set operation classes, the nesting direction is not yet displayed in VOWL: A union of an intersection and an intersection of unions may look the same (annotation 4).

Based on these observations, it can be concluded that VOWL supports a large portion of constructs featured in OntoViBe, and hence in OWL 1 and 2. Therefore, the notation can be expected to consistently visualize a wide variety of ontologies.

6. Conclusion and Future Work

In an earlier effort to create a uniform visual notation for OWL ontologies, a first version of VOWL has been developed [56]. Based upon that work, related endeavors, as well as findings from a user study [55], large parts of the notation have been redesigned to create VOWL 2, a visual language that can also be understood by casual ontology users. This article described the key considerations, features, and capabilities of VOWL 2, as well as two implementations, a plugin for the widely used ontology editor Protégé and a responsive web application. Both tools demonstrate the applicability and usability of VOWL by means of several ontologies. The comprehensibility of VOWL is confirmed by several user studies conducted with different user groups, while its visual expressiveness and completeness have been tested with a benchmark ontology.

The ontologies used in the evaluations were of relatively moderate size. However, there is no upper limit for the size of ontologies, both because a vast number of topics can be covered in one ontology, and because an ontology can be modeled down to an arbitrary level of detail. Graph visualizations are, on the other hand, viable only up to a certain size, at which the overview is lost and the graph is not easily usable any longer. The visualization of large-scale ontologies with VOWL is still an issue that needs to be tackled. First steps have been done by designing interactive features to filter elements and reduce the graph size. Future solutions might consider both automatic and manual methods to detect ontology components that carry context-specific importance, so that parts of the visualization can be hidden or bundled where appropriate.

A related issue is that ontology elements have no inherent location information. Therefore, all elements are initially placed in a random manner in the force-directed layout. While this does not influence a single session of work, it prevents users from creating a "mental map" of the visualization that is valid for several sessions, since the elements are at different locations every time the VOWL graph is rendered. Future work will have to develop reasonable guidelines on how to best place ontology elements so their positioning follows a reproducible pattern.

Future work will also be concerned with incorporating further OWL elements into VOWL. Although VOWL 2 already considers a large portion of the language constructs of OWL 1 and 2, it is not complete. The ultimate goal is to further extend VOWL in order to make it a visual language that represents OWL ontologies as completely as possible. This would also call for a web service where users can upload ontologies and get them visualized (e.g., with WebVOWL).

Other open issues with regard to the VOWL implementations are multi-language support, improved
search functionality, and the inclusion of instance data in the JSON schema, converter, and tools. Apart from that, it is hoped that VOWL and its implementations will be useful to others and in related areas, such as ontology alignment or Linked Data exploration, as well as in teaching and training contexts.

Acknowledgements

The authors would like to thank David Bold, Vincent Link, and Eduard Marbach for their contributions to the implementation of VOWL and their valuable feedback on the notation. Likewise, the authors would like to thank all participants of the user studies, and all experts who took part in the expert interview.

References


