Using Hierarchical Edge Bundles to Visualize Complex Ontologies in GLOW

Walter Hop
walter@lifeforms.nl

Sven de Ridder
sven@glowvis.org

Frederik Hogenboom
fhogenboom@ese.eur.nl

Flavius Frasincar
frasincar@ese.eur.nl

Erasmus University Rotterdam
PO Box 1738, NL-3000 DR
Rotterdam, the Netherlands

ABSTRACT

In the past decade, much effort has been put into the visual representation of ontologies. However, present visualization strategies are not equipped to handle complex ontologies with many relations, leading to visual clutter and inefficient use of space. In this paper, we propose GLOW, a method for ontology visualization based on Hierarchical Edge Bundles. Hierarchical Edge Bundles is a new visually attractive technique for displaying relations in hierarchical data, such as concept structures formed by 'subclass-of' and 'type-of' relations. We have developed a visualization library based on OWL API, as well as a plug-in for Protégé, a well-known ontology editor. The displayed adjacency relations can be selected from an ontology using a set of common configurations, allowing for intuitive discovery of information. Our evaluation demonstrates that the GLOW visualization provides better visual clarity, and displays relations and complex ontologies better than the existing Protégé visualization plug-in Jambalaya.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*User interfaces*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Graphical user interfaces (GUI)*; I.3.8 [**Computer Graphics**]: Applications

General Terms

Languages, design, management

Keywords

Ontologies, visualization, hierarchical edge bundles, GLOW, Semantic Web

SAC'12 March 25-29, 2012, Riva del Garda, Italy.

1. INTRODUCTION

An ontology formally represents knowledge about a domain. It contains a summary of all the concepts from the domain, and the relations between these concepts. This formal definition of domain knowledge enables computers to answer complex questions, and derive new knowledge about the domain, using automatic reasoning.

The data contained within ontologies can also be used by humans for exploration or information extraction. Since the knowledge in itself is abstract, human-computer interfaces are necessary for this purpose. A frequently used technique is visualization. Ontology visualizations can present a viewer with immediate insight in the structure and properties of the knowledge contained within the ontology.

An analogy can be made between ontologies and graph topologies. When concepts are mapped to nodes, and their relations are mapped to edges, a graph emerges. Using a graph diagram is therefore an obvious choice for ontology visualization.

The layout of the graph should comply with the way users perceive the concepts depicted; for instance, elements should be placed close to major elements they depend on, and the hierarchical nature of concepts should be implied by their representation [17]. A problem with many real-life ontologies, in this regard, is their size: as an ontology grows, it becomes harder to satisfy these constraints in a diagram. 'Pan-and-zoom' functions may partially resolve graph size problems, but can obscure the large-scale structure.

Many ontologies contain hierarchical concept trees, such as those formed by 'subclass-of' or 'type-of' relations. There are many strategies to visualize such tree structures. An example of this is the *inverted radial layout*, for which an instance is displayed in Fig. 1.

The leftmost diagram in Fig. 1 shows that this layout compresses a large amount of information about the hierarchical structure of items into a relatively small space. Also, the center of the diagram leaves room for the drawing of additional edges that may represent a different (in regards to the hierarchical) kind of relation. Mixing these edges with the 'hierarchical' edges would lead to visual clutter [9]. However, even when spatially separated, a high number of inner edges can still lead to a cluttered mesh of edges that is hard to comprehend. The *Hierarchical Edge Bundles* technique, as described by Holten [9], allows to 'bundle' adjacency edges into sets of related curves. As shown in the rightmost di-

agram, this results in a less complex and less cluttered appearance, which enables one to visualize larger graphs. As demonstrated in [9], this does not only lead to aesthetically pleasing images, but also to graphs that help experts, students, and local companies to quickly gain insight in the adjacency relations present in hierarchically organized systems. A *bundling coefficient* can be modified to change the amount to which the adjacency curves are affected by the hierarchy.

In this paper, we explore the use of Hierarchical Edge Bundles for the interactive visualization of complex ontologies, i.e., ontologies that have a large number of classes, individuals, and non-type relations, as well as a large number of distinct properties or rules. We propose GLOW, a method for ontology visualization, that incorporates this technique. The method includes a data selection phase that consists of several configurations useful for analyzing real-life ontologies. Various graph layouts, among which the inverted radial layout, are evaluated.

We construct a reference implementation of the GLOW visualization and data preparation methods in Java. The package can read any ontology via the OWL API [11], a Java API and implementation for working with OWL ontologies. OWL (Web Ontology Language) is the standard format for ontologies on the Semantic Web [2]. Developers of Semantic Web applications can use the package to embed GLOW visualizations into their own programs by making use of the GLOW API.

We also make available a visualization plug-in for Protégé [15], a well-known ontology editor and knowledge acquisition system. The plug-in can be used alongside Protégé's standard tools and visualizations. The full software package is available on-line at `http://www.glowvis.org/`.

The remainder of this paper is organized as follows. First, in Sect. 2 we discuss related work on visualizing relations in complex ontologies. In Sect. 3 we introduce the GLOW framework, followed by details on its reference implementation in Sect. 4. In Sect. 5, we evaluate the performance of our Protégé plug-in with respect to the current ontology visualizer *Jambalaya* [16], incorporating various exploratory use cases based on an existing ontology from the Semantic Web. Finally, we draw conclusions and discuss directions for future work in Sect. 6.

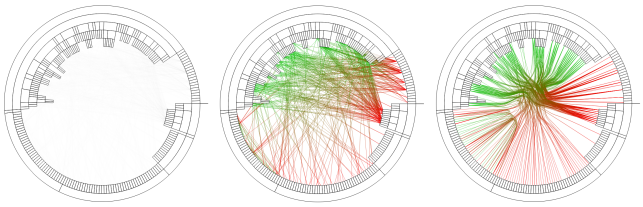


Figure 1: Inverted radial layout visualizations of hierarchical data and adjacency relations. Left: hierarchical relations form a radial bar shell; descendants are positioned inward relative to their parent. Middle: a separate set of adjacency edges is added. Right: the adjacency edges are bundled to create a less cluttered diagram. The graphs were generated using GLOW on the Wine ontology [14], displaying classes, individuals and relations.

2. RELATED WORK

2.1 Ontology Visualization Methods

The OWL language, which has become a W3C standard for describing ontologies in the Semantic Web [2], has serialization formats (e.g., XML) that do not lend themselves to direct viewing by humans. For this purpose, one needs tool support. Many methods and tools for ontology visualization currently exist. Katifori et al. [12] have provided taxonomies of visualizations. We briefly touch upon the major techniques in use today, and discuss their merits and drawbacks as they relate to visualization of complex ontologies (e.g., containing hundreds of classes and individuals, and thousands of relations).

2.1.1 Node–link

Node–link techniques display an ontology as a structure of interconnected nodes. The resulting graph diagrams are intuitive to users, and can provide a good preservation of the structure, even for large ontologies. Different relation types can be represented, for instance by line colors or spatial separation.

Node–link visualizations are generally not very efficient in use of screen space, although this is helped by choosing a specialized layout [9]. The *force-directed* layout determines node positions dynamically to optimize the use of screen space, causing an animation effect. It prevents visual clutter by minimizing the number of edge crossings, and attempts to fill the screen space optimally by equalizing distances between parent and child nodes.

Trees are a special form of node–link graphs. They are commonly used as models in ontology visualization, even if multiple inheritance is not applicable to tree structures, and therefore the true ontology structure cannot always be preserved. Most tree-based visualizations cope with multiple inheritance by duplicating child nodes and placing them under each parent node [12], which introduces an inaccuracy. An example of a tree-based visualization is the well-known Protégé plug-in *OWLViz* [10].

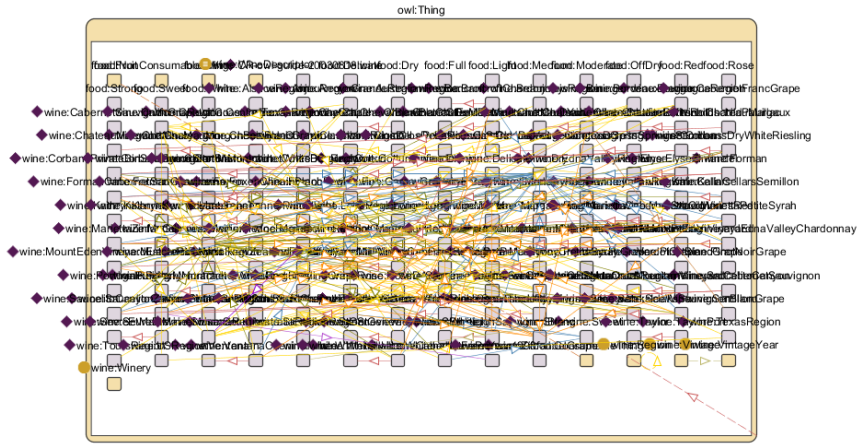
3D visualizations use an extra dimension to represent extra information or increase visual clarity. A 3D implementation can be found in *GViz* [5], where instances of classes are drawn in separate parallel planes connected by edges that represent 'type-of' relations.

2.1.2 Zoomable

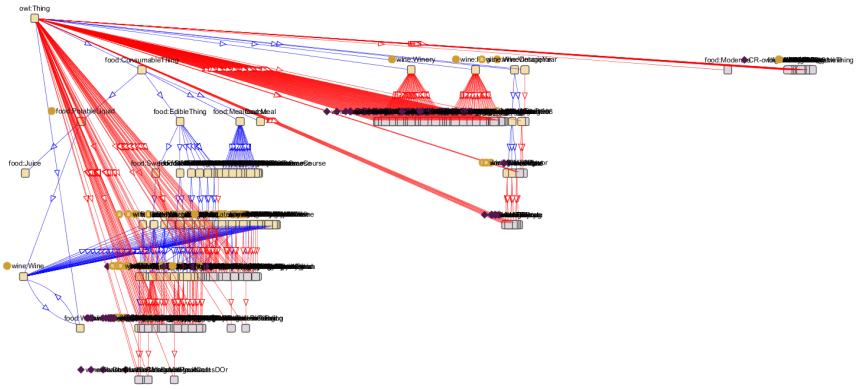
These methods embed consecutive layers of lower-level nodes within a higher-level node. They hold potential for large ontologies, as they can show the large-scale structure first, then allow the user to 'drill down'. On the other hand, the lack of global context inhibits preservation of structure. Also, problems appear when relation edges cross zoom boundaries: in this case, it is not clear where the end point of a relation lies. The Protégé visualization plug-in *Jambalaya* [16] uses zooming techniques in its 'Nested View', shown in Fig. 2.

2.1.3 Focus + context or distortion

These methods can be used in conjunction with other methods, to give more detail to an area of interest by distorting the view of the graph in reaction to user input. Arguments for and against these methods are similar to those of zooming. An implementation of this method can be



(a) *Nested view* visualization



(b) *Tree* visualization

Figure 2: Visualizations of the Wine ontology [14] with individuals and relations in Jambalaya.

found in *TGVizTab* [1], a Protégé plug-in that uses a spring-layout technique, displaying selected nodes in the center of the graph and dynamically grouping related nodes around it. The focus + context approach on selection has received mixed reviews in an experimental setting, with some users finding it 'playful' and 'nice' and others calling it 'dizzying' and 'chaotic' [13].

2.2 Visualization of Adjacency Relations

Holten [9] identifies the need for visualizations that allow for relations of interest to be overlayed on a 'structural' background layout, such as the inclusion (type) hierarchy. Few visualization implementations currently allow for such a configuration.

A notable counterexample is *Jambalaya* [16]. It uses various layouts, allowing for adjacency relations to be mapped on top of a zoomable layout of the structural graph. The visualizer contains many useful predefined scenarios, and is easy to configure. The provided versatility makes it an attractive choice for ontology visualization.

However, its layout options are limited and are not optimized for large ontologies. Also, displaying individuals and relations is still imperfect, as Jambalaya's arc rendering process does not use any algorithm to prevent visual clutter. In an experimental comparison of ontology visualizers [13], almost all users complained that after browsing concepts, the relation edges became so many that they obstructed the visualization. In Fig. 2, we show examples where the visualization techniques fail to provide a clear picture. Here, the classes, individuals and relations in the Wine ontology [14] are displayed.

Another example of a visualizer that allows for displaying of relations over a structural background is *TGVizTab* [1]. It uses a focus + context technique, dynamically arranging related nodes around the selected node. In an experiment, it was observed that there was much clutter when displaying role relations and instances [13]. Furthermore, TGVizTab can only display small numbers of nodes efficiently at one time, as the layout suffers from screen clutter. It prevents this situation by using focus and thresholds (e.g., nodes with many edges are not displayed), which is less fit for visualizing large ontologies as it does not preserve the structure.

Exploring an ontology visualization technique that fully takes into account a compound graph of structural and relation edges, focused on large ontologies, is the primary motivation for this research. Such a visualization should allow for a clear display of ontologies containing hundreds of nodes and thousands of edges.

3. THE GLOW FRAMEWORK

3.1 Reference Model

The information visualization reference model [7] forms a template for common functions, components and interfaces inherent to the task of visualization. In the context of the GLOW framework, the components of the reference model are mapped as follows. The *data source* component loads the data sets to be visualized. The input of this phase is a set of OWL ontologies, as well as a choice of rule sets for extracting hierarchies and relations, and a choice of reasoner. The selection of axioms from the ontology takes place here. The output is the data set to be visualized.

Our *data set* is a compound graph [9]. The first part of the compound graph is the *inclusion hierarchy*, which takes the form of nodes (concepts) and edges (the 'structural' relations between the nodes). In our case, the inclusion hierarchy is generated from OWL classes, optionally OWL individuals, and the type relations between them. The second part of the compound graph is a set of adjacency edges (representing a possibly more 'dynamic' set of relations between the objects). The adjacency edges are generated from non-type relations between the concepts. Which nodes and edges are selected depends on the chosen rule sets, which are described in more detail in Subsect. 3.2.

In the *visualization* phase, the abstract compound graph is transformed into a collection of 'visual items' which contain information such as position, size, and shape. The world coordinates of the compound graph elements are calculated by applying one of several layouts. The adjacency edges, later rendered as splines, receive their end points and control points here. We give more details on this process in Subsect. 3.3.

The *view* is responsible for managing the visual items on the screen. The visual items (e.g., nodes, inclusion edges, adjacency edges) are rendered by object-specific renderers, which transform the items to screen coordinates. The coordinates of the Hierarchical Edge Bundles [9] are also calculated here. Other responsibilities of the view are panning, zooming, and rotation.

Control components allow the user to change configuration parameters, such as layout type, bundling coefficient, colors, selected root node, et cetera. Also, events from the environment are caught; for instance, in Protégé, selecting an OWL class in the class browser limits the GLOW view to only that class and its descendants.

Methodologically, the most notable parts of the framework are the data selection phase, which maps OWL objects and relations to a compound graph, and the visualization, which combines several layouts with Hierarchical Edge Bundling to enable viewing of large ontologies. These parts are described in the next subsections.

3.2 Data Selection

3.2.1 Rule sets

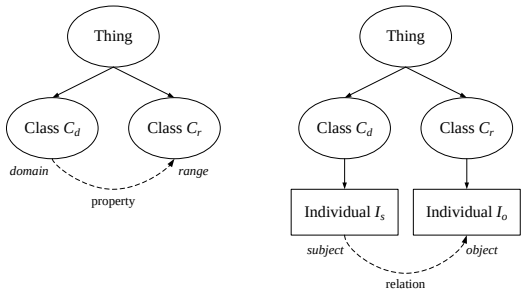
The data selection converts OWL objects and their relations to a compound graph, consisting of a tree part (the inclusion hierarchy) and a set of adjacency edges. The ontology reader depends on a set of OWL ontologies (for instance, a main ontology and its imports), and a choice of rule sets to generate the inclusion hierarchy and the adjacency edges. The rule sets determine which OWL axioms or objects are used. If no sub-selection of classes is made, the `Thing` class becomes the root of the tree, but the user can choose to focus on a sub-tree by selecting a specific class. Fig. 3 shows how a small example graph is created by each rule set.

The *Classes* rule set uses 'subclass-of' relations to build the inclusion hierarchy. Each node in the graph corresponds to an OWL class in the provided ontologies. One adjacency edge is created from a class C_d to a class C_r, for every property that has these classes as its domain and range. This rule set allows us to quickly assess the class hierarchy, and the coverage of property ranges and domains. For instance, in the Pizza ontology [3], we find that the property `hasBase` connects the classes `Pizza` (domain) and `PizzaBase` (range).

The *Classes + Individuals* rule set uses 'subclass-of' as well as 'type-of' relations to build the inclusion hierarchy. Interior nodes are always OWL classes; leaves may be either OWL individuals or OWL classes. Relations between individuals map to adjacency edges. One adjacency edge is drawn from individual I_s to individual I_o for every relation that has these individuals as subject and object. The Hierarchical Edge Bundles approach does not consider adjacency edges from classes (domain) to other classes (range), as it requires the instances (in this case, individuals) to be part of a hierarchical (tree) structure. Furthermore, it assumes adjacency relations only between the leafs of the tree (relations involving non-leafs are interpreted as hierarchical and non-adjacent). The *Classes + Individuals* rule set is an interpretation of the ontology with the classes as tree nodes (non-leafs), individuals as leafs, and adjacency relations as relations between instances. The rule set gives us a complete view of classes, individuals, and their relations. The user can also choose to filter the relations on their property. For instance, in the Wine ontology [14], one who is interested in geographical relations between objects might choose to view only the relations defined by property `locatedIn`.

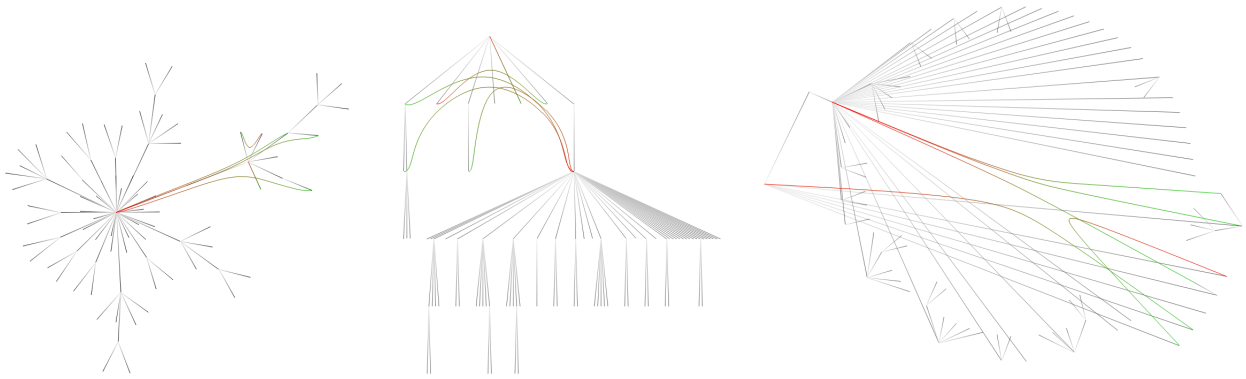
3.2.2 Reasoner support

An optional reasoner can be used through the OWL API. If the user selects a reasoner, subclasses and individuals will be queried via the reasoner, instead of through the ontology's axioms. Also, retrieved classes are checked for satisfiability.



(a) *Classes* rule set (b) *Classes + Individuals* rule set

Figure 3: A small ontology demonstrates how OWL objects are mapped to the compound graph under the two rule sets. Continuous lines represent edges in the inclusion hierarchy; a dotted line represents an adjacency edge.



(a) Force-directed graph (b) Node-link tree (c) Inverted radial tree

Figure 4: Available layout algorithms in GLOW.

If fully semantically correct results are desired, a reasoner must always be used. Using a reasoner will generally produce a more extensive graph, as the inferred knowledge gives rise to extra inclusion and adjacency edges.

If no reasoner is available or selected, objects are selected by only using syntactic rules on the asserted facts. This might yield results that are semantically incorrect. For instance, inconsistent classes may be present in the output, and relations which are not explicitly stated will be missing. Nevertheless, it can be useful to visualize ontologies without a reasoner. For instance, it is generally faster and generates more compact output, while still preserving the ontology's structure to a large degree. Also, it allows the user to visualize inconsistent ontologies that would be rejected by a reasoner's checker, for instance during ontology development.

3.2.3 Class hierarchy limitations

There are some limitations to our current data selection implementation. The Hierarchical Edge Bundles technique necessarily depends on a hierarchical inclusion structure. Multiple inheritance cannot easily be modeled using this algorithm. We handle multiple inheritance by duplicating child nodes and their adjacencies. This is also the approach taken by most visualizations [12]. We feel that this solution is sufficient for most purposes, as completeness of representation is not compromised, e.g., it does not hide information that is present in the ontology. Also, when considering only a sub-tree restricted by one parent, it gives a correct representation of the objects below it.

3.3 Visualization

3.3.1 Compound graph

After the data selection phase has completed, the resultant compound graph $G = \langle V,\ \langle E_i,\ E_a\rangle\rangle$ is used to generate an abstract visual description of the data. The generation is a two-step process, where V represents the vertices of the graph, and E_i and E_a denote the inclusion and adjacent edges, respectively. First, the inclusion hierarchy $G_i = \langle V, E_i\rangle$ is used to perform the tree layout for the visual representation. Then, the tree layout is combined with the adjacency graph $G_a = \langle V, E_a\rangle$ to generate a representation of the edges to be drawn.

3.3.2 Inclusion hierarchy layouts

Various tree layouts are available in GLOW. These are the *force-directed graph*, which positions the nodes such that edge crossings are minimized and a node's child edges are of comparable length; the *inverted radial tree*, where the top concept is placed in the outer ring having sub-concepts placed in consecutive inner circles; and the *node–link tree*, which places every hierarchical level of the inclusion tree on a vertically separate layer.

The chosen layout prescribes the coordinates of the inclusion hierarchy elements, i.e., the placing of the nodes in the diagram. The layouts and their properties are depicted in Fig. 4, which was generated using GLOW on the Wine ontology [14].

3.3.3 Hierarchical Edge Bundles

Next, a visual description for the adjacency edges is generated. We choose the Hierarchical Edge Bundles approach [9] to describe these edges, as this approach significantly reduces visual clutter in graphs with large numbers (e.g., thousands) of edges. Furthermore, the bundling coefficient β can be manipulated in real-time, which provides the user with variable levels of detail over a continuous interval.

Essentially, the Hierarchical Edge Bundles approach describes edges as curves, whose paths are defined by control points generated from the inclusion hierarchy: the control points correspond to the inclusion hierarchy nodes found along the shortest path from the edge source node to the edge target node. In effect, the generation of control points from the inclusion hierarchy curves the edges according to semantic relatedness.

The degree of curvature is determined by the bundling coefficient β. Fig. 5 shows the effects of changing the coefficient. We use Holten's formula for control point straightening [9]:

$$P'_i = \beta \cdot P_i + (1-\beta)(P_0 + \frac{i}{N-1}(P_{N-1} - P_0))\ , \qquad (1)$$

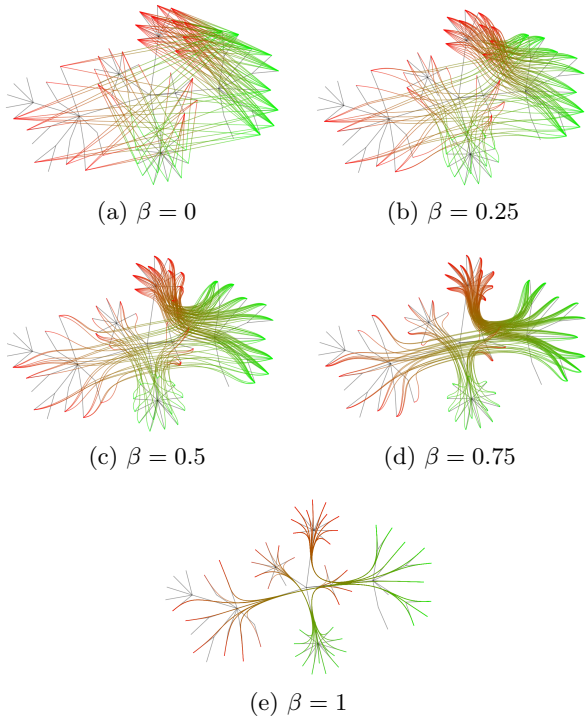
where N is the number of control points, $i \in \{0, ..., N-1\}$ is the control point index, and $\beta \in [0, 1]$ the bundling coefficient. The bundling coefficient β effectively determines the amount of interpolation between the original control point P_i and a point on the line from the edge source node P_0 to the edge target node P_{N-1}.

We use Uniform Non-Rational Basis Splines (UNRBS) [4] to draw the adjacency edges. Computationally, UNRBS are relatively cheap to generate, enabling us to display a greater number of edges while still allowing real-time animation. UNRBS, however, come at the cost of reduced control over the curve pathing, but this proves only to be an issue at the edge endpoints. To force the curve to interpolate through the endpoints, we duplicate the endpoints twice, $P_{-2} = P_{-1} = P_0$ and $P_{N-1} = P_N = P_{N+1}$, to arrive at the control point series $(P_{-2}, ..., P_{N+1})$.

3.3.4 *Control point mirroring*

For the inverted radial tree layout, if we let the control points coincide with the inclusion nodes, we find that adjacency edges occlude the inclusion graph, while much of the screen space in the center of the visualization goes unused. Therefore we perform a radial mirroring, reflecting the original control points into the circumference of the circle defined by the lowest level in our tree structure to arrive at a new set of control points. These new control points preserve the concept of semantic relatedness, but curve the adjacency edges towards the unused central space. This approach is a simple variant on Holten's [9] radial layout construction.

The mirroring strategy, however, presents a problem when one or both of the adjacency endpoints are not at the lowest level in the hierarchy. In such cases, common in OWL ontology hierarchies, the mirrored endpoints do not coincide with the nodes in the inclusion hierarchy, and the adjacency edge is disconnected from the inclusion graph. We solve the issue by applying the mirroring only to intermediate control points, and leaving the endpoints untouched.



(a) $\beta = 0$ (b) $\beta = 0.25$ (c) $\beta = 0.5$ (d) $\beta = 0.75$ (e) $\beta = 1$

Figure 5: Effect of β on adjacency edges. The figure was created using GLOW to display classes and instances of the 'IPDfull' ontology, limited to objects under class `TCMSelectionSystem`.

Similar applications of reflection symmetry, such as reflection into an axis, may aid the legibility of a number of other graphs, such as the Node-Link Tree graph. This layout is unclear because large parts of the adjacency edges are collinear, obscuring individual relations.

4. IMPLEMENTATION OF GLOW

GLOW has been implemented as a Java application, as Java is the *de facto* standard platform for Semantic Web applications, with many tools and libraries available, such as the OWL API and Protégé. Due to their open-source nature and a lively community, these tools are thought to be the most 'future-proof' choices for OWL-related development. The GLOW software, as shown in Fig. 6, has been designed as a set of loosely-coupled components that allow for efficient reuse in various scenarios, such as a Protégé plug-in, stand-alone OWL API applications, or other applications requiring visualization of compound graphs or Hierarchical Edge Bundles. The implementation distinguishes between visualization, bridging OWL API and visualization, and a Protégé 4.0 plug-in wrapper. Two common libraries, i.e., the prefuse visualization toolkit [8] and JOGL [6] were bundled, in order to allow for easy installation under Protégé.

Data selection is done by means of user defined rule sets. When the rule sets are configured, the compound graph is built. The rule sets add inclusions and adjacency edges to the graph, which fully described the data set by means of abstract data (nodes and edges). For visualization, we make use of OpenGL (through the JOGL interfaces [6]), as this provides access to advanced rendering techniques, such as blending algorithms, color interpolation, and spline modeling. In addition, OpenGL is highly performant, enabling real-time animation of complex graphics. Finally, the choice for a 3D graphics API provides later opportunities for 3D visualization. The design of the GLOW visualization components is largely modeled after the prefuse Visualization Toolkit [8], which itself is based on the Information Visualization Reference Model [7]. Extensions to the prefuse components were made to support OpenGL graphics; specifically, we provided an extension to prefuse's `Display` to handle input and animation events and delegate rendering to a set of specialized `GLRenderer` objects. The Inverted Radial Tree layout was added to the set of layout algorithms already provided by prefuse.

5. EVALUATION

To assess the user acceptance of the GLOW visualization, we have carried out an initial small-scale user-based evaluation, comparing GLOW to *Jambalaya* [16], arguably the most well-known ontology visualizer that can handle relations (an example of its display of complex ontology relations is seen in Fig. 2). Experiments only focused on the visualization aspect of both GLOW and Jambalaya. The participants ($n = 7$) were selected based on their scientific profiles from a group of students and faculty staff, having a broad scientific background in the fields of computer science and informatics, with at least a basic understanding of the Semantic Web. Participants' OWL experience ranged from basic experience (e.g., followed classes on OWL) to advanced experience (e.g., published papers on OWL). Participants had no previous experience with GLOW and Jambalaya, and hence were explained the basics of both visualizers.

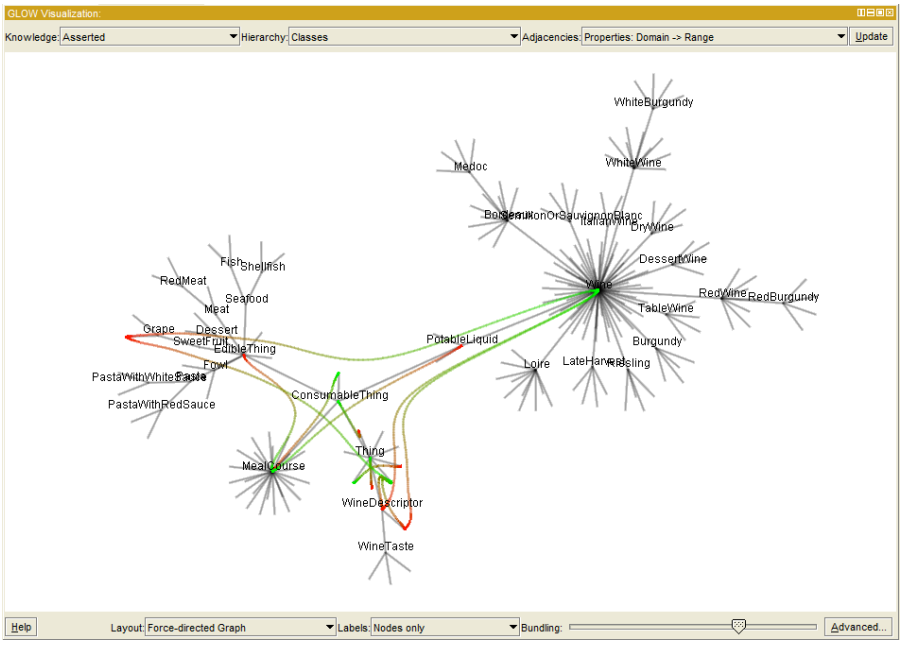


Figure 6: Screen shot of a GLOW view inside Protégé, visualizing the classes and object properties of the Wine ontology [14] using a force-directed graph layout. Colored adjacency edges represent properties, such as `madeFromFruit` and `hasWineDescriptor`. The endpoints of the adjacency edges correspond with the domains and ranges of the properties.

Participants were explained the main goals of GLOW and given instructions on how to install both visualizers. The Wine ontology [14] was provided as an example of a reasonably complex ontology (345 classes and individuals, 181 non-type relations). No mandatory assignments were handed out, but participants were provided a list of sample tasks to assess the two visualizations:

Class-related tasks

- How 'deep' is the ontology in terms of subclass relations?
- Which are the classes of wines that comprise the `Loire` class?

Individual-related tasks

- How many wine grapes (individuals of the `WineGrape` class) are there?
- For which wines is defined where they come from (the `locatedIn` property) and for which wines is this not yet defined?
- There is one wine which has the vintage year 1998 (`hasVintageYear` property); which wine is this?

After allowing the participants to spend some time using the visualizations (e.g., updating β values, changing display types, etc.), the participants were asked to complete questionnaires containing forced ranking questions and open questions. In the ranking questions, participants appeared to prefer GLOW on all criteria (see Table 1), although due to the low number of participants, statistical significance could only be reached on the 'displaying relations' criterion (two-tailed binomial test; H_0: users have no preference, i.e., the preference probability for GLOW equals 50%; $\alpha = 0.05$).

In open questions, participants provided rationales for their rankings. GLOW was generally preferred, with descriptions such as 'nicely organized', 'pretty', 'immediately clear what is shown' and 'clearer trees', while Jambalaya's layouts were overall judged as 'messy', 'unclear', and 'easily cluttered'. Additionally, five out of seven participants felt that GLOW's hierarchical edge bundling feature increased the visual clarity of the diagrams. The adjacency edge color gradient, which shows arc direction without the necessity for arrow symbols, was also well rated.

When comparing GLOW's various layouts for the purpose of ontology analysis, five participants preferred the force-directed graph layout, while one participant preferred the node–link tree layout and one stated no preference. The force-directed feature was not universally praised: two participants felt that the animation of the force-directed layout was too 'bouncy' and never stabilized. One user, for this reason, preferred navigating the ontology in Jambalaya.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we identified a need for visualization methods that preserve the large-scale structure of complex ontologies, while still enabling the representation of classes and individuals, as well as the relations between them. Therefore, we have explored techniques that enable visualization of these complex ontologies and their relations, and demonstrate the utility of the Hierarchical Edge Bundles technique. Furthermore, we have proposed the GLOW framework, a flexible visualization engine that uses Hierarchical Edge Bundles to prevent visual clutter. Its reusable

Table 1: Numbers of participants who preferred either GLOW or Jambalaya on various evaluation criteria. Left column: evaluation criterion. Middle columns: numbers of participants choosing a preferred visualization. Right column: 95% confidence interval for the preference probability for GLOW.

Which visualization do you prefer...	GLOW	Jambalaya	95% CI
... in terms of diagram clarity?	6	1	42.0–99.6%
... for displaying relations?	7	0	59.0–100.0%
... in terms of visualization performance on large ontologies?	5	2	29.0–96.0%
... to answer the example queries about *classes*?	5	2	29.0–96.0%
... to answer the example queries about *individuals*?	5	2	29.0–96.0%

component-based implementation ensures that Semantic Web developers will be able to experiment with GLOW and add it to their own programs, as well as provide extensions of their own by adding new rule sets and layouts.

GLOW's inverted radial tree layout allows for drawing of large numbers of adjacency relations within an ontology, without causing visual clutter. However, in our evaluation using the reasonably complex Wine ontology, most participants chose the force-directed graph layout as most appropriate for analysis of ontologies instead. The evaluation participants largely preferred GLOW's diagrams to those of the well-known relation visualizer Jambalaya. On all criteria, participants generally ranked GLOW above Jambalaya. On the evaluation criterion 'visualization of relations', all participants preferred GLOW. This indicates that the methods employed by GLOW hold promise for improving the visualization of complex ontologies and their relations.

Essentially, the Hierarchical Edge Bundles technique is the generation of spline control polygons based on shortest paths along inclusion/inheritance relationships. The requirement for the inclusion graph to have a proper tree form, however, can be prohibitive. The constraint requires duplication of nodes exhibiting multiple inheritance, and limits the development of new visualization rule sets. Therefore, for future work, we suggest to explore whether the Hierarchical Edge Bundles technique can be extended to handle arbitrary graphs. An adapted algorithm could gather information from the shortest path between nodes in the graph, or a minimum spanning tree could be used. Furthermore, additional rule sets could be useful for visual analysis of ontologies. One example is a hierarchy rule set that extracts 'part-whole' relations to create the inclusion hierarchy. Another example is an edge rule set that creates adjacency edges from N-ary relations; N-ary relations are currently very hard to visualize as there are no OWL primitives to describe these, resulting in the use of 'cross classes' that have no domain counterparts and obfuscate diagrams heavily.

7. REFERENCES

[1] H. Alani. TGVizTab: An Ontology Visualisation Extension for Protégé. In: Workshop on Visualization Information in Knowledge Engineering, Knowledge Capture (KCAP 2003), 2003.

[2] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneijder, and L. A. Stein. OWL Web Ontology Language Reference. W3C Recommendation, 10 February 2004. `http://www.w3.org/TR/owl-ref/`, 2004.

[3] N. Drummond, M. Horridge, R. Stevens, C. Wroe, and S. Sampaio. Pizza Ontology, Pizza Tutorial. `http://www.co-ode.org/ontologies/pizza/`, 2007.

[4] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice in C*. Addison-Wesley Professional, 2nd edition, 1995.

[5] F. Frasincar, A. Telea, and G.-J. Houben. Adapting Graph Visualization Techniques for the Visualization of RDF Data. In *Visualizing the Semantic Web*, chapter 9, pages 154–171. Springer, 2006.

[6] S. Gothel. JOGL: Java Binding for the OpenGL API. `http://kenai.com/projects/jogl/pages/Home`, 2009.

[7] J. Heer and M. Agrawala. Software Design Patterns for Information Visualization. *IEEE Trans. Vis. & Comp. Graphics*, 12(5):853–860, 2006.

[8] J. Heer, S. K. Card, and J. A. Landay. prefuse: A Toolkit for Interactive Information Visualization. In *15th Conf. on Human Factors in Computing Systems (CHI 2005)*, pages 421–430. ACM, 2005.

[9] D. Holten. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Trans. Vis. & Comp. Graphics*, 12(5):741–748, 2006.

[10] M. Horridge. OWLViz. `http://www.co-ode.org/downloads/owlviz/`, 2004.

[11] M. Horridge and S. Bechhofer. The OWL API: A Java API for Working with OWL 2 Ontologies. In *6th Int. Workshop on OWL: Experiences and Directions (OWLED 2009)*. CEUR-WS.org, 2009.

[12] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. Ontology Visualization Methods – A Survey. *ACM Computing Surveys*, 39(4):1–43, 2007.

[13] A. Katifori, E. Torou, C. Halatsis, G. Lepouras, and C. Vassilakis. A Comparative Study of Four Ontology Visualization Techniques in Protege: Experiment Setup and Preliminary Results. In *10th Int. Conf. on Information Visualization (IV 2006)*, pages 417–423. IEEE Computer Society, 2006.

[14] M. K. Smith, C. Welty, and D. L. McGuinness. Wine Ontology, OWL Web Ontology Language Guide. `http://www.w3.org/TR/owl-guide/wine.rdf`, 2004.

[15] Stanford Center for Biomedical Informatics Research. Protégé. `http://protege.stanford.edu/`, 2009.

[16] M.-A. Storey, M. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. Noy. Jambalaya: Interactive Visualization to Enhance Ontology Authoring and Knowledge Acquisition in Protégé. In: Workshop on Interactive Tools for Knowledge Capture (KCAP 2001), 2001.

[17] Y. Tzitzikas and J.-L. Hainaut. On the Visualization of Large-sized Ontologies. In *8th Working Conf. on Advanced Visual Interfaces (AVI 2006)*, pages 99–102. ACM, 2006.