

Visualizing Web Ontologies with CropCircles

Bijan Parsia, Taowei Wang, and Jennifer Golbeck
bparsia@isr.umd.edu, {tw7, golbeck} @cs.umd.edu

Dept. of Computer Science,
University of Maryland,
College Park MD 20742

Abstract. We apply a new visualization for complex hierarchies, CropCircles, to the interactive visualization of Web Ontologies and \mathcal{E} -Connections of Web Ontologies.

1 Introduction

Ontologies are often interesting merely because of their large size. Even a pure taxonomy without significant axiomatization can be extremely valuable. Just having an established set of specific terms regarding a domain, some initial indication of the relations between the terms, and a locus for a body of practice, discussion, and even code which can drive more sophisticated modeling. With richer theories, the size provides both impetus and impediment to use and reuse: large, sophisticated ontologies contain a lot of information but are difficult to process and difficult to understand.

In this paper, we describe preliminary investigations into a novel class graph visualization technique, CropCircle, that seems to work for large class graphs with complex structure of certain sorts. For tree like class graphs we believe that it can present more and more complex branches of the tree (both deep and variably wide) than current alternative methods. We also show how CropCircles can help in the understanding of a novel formalism for combining ontologies, \mathcal{E} -Connections, and conversely, how first factoring a Web Ontology into an \mathcal{E} -Connection can improve the effectiveness of the CropCircle display.

2 Visualizing Graph Hierarchies

Visualizing class hierarchies of modern ontologies is a difficult task for several reasons: they tend toward the very large (ranging from hundreds to tens of thousands of classes); multiple inheritance precludes purely tree-like structures; inheritance chains can be deep, broad and irregular; nodes are complex; and the details of the relationship between nodes is important. Most Web Ontology browsers such as Protégé, Swoop, and Triple20 use standard table-based tree widgets¹ which have a number of obvious problems.

¹ One notable exception to this trend is OntoTrack[6].

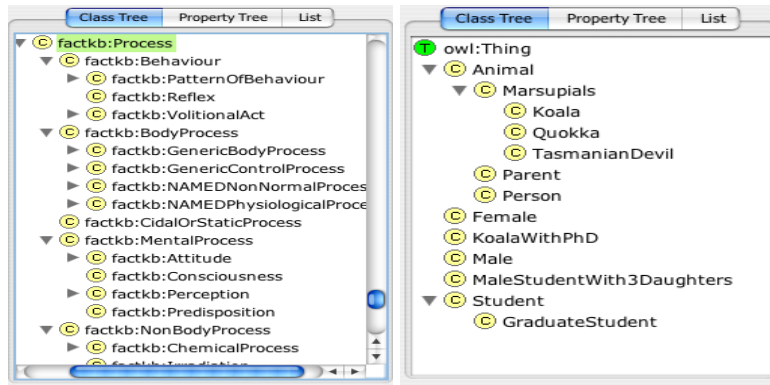


Fig. 1. The left is part of the Galen ontology visualized by a tree widget. The right is a similar visualization of a simplified version of the much smaller example ontology `koala.owl`

Consider the two tree widget displays in Figure 1. The version of the Galen ontology² shown has 2749 classes, while the tree widget with these dimensions can only show about 18, not even enough to show the entire branch under the Process class. The other widget displays the entire class tree of that ontology, and, in this case, the indentation is reasonably effective, although it can be difficult to see that Student is on the same level as Animal, given their separation. Also, given that the remaining classes on that level have no children, it can be hard to realize that they are in fact siblings. Of course the user can hide the subtrees Animal and Student, but then the sibling view is overemphasized.

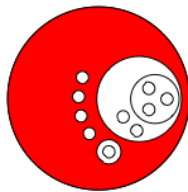


Fig. 2. A CropCircle of the simplified version of `koala.owl`

In contrast, the CropCircle view of the same tree presents all the information at once, both siblingness and subclass structure, without significant bias toward either. Of course, the tree widget approach has the advantage of incorporating

² Available from <http://protege.stanford.edu/plugins/owl/owl-library/not-galen.owl> with the comment “A selective adaptation made in 1995 of an early prototype GALEN model; content is not related to or representative of any current or historical OpenGALEN release.”

the class names into the visualization itself, thus, given meaningful and readable names, gives the user some clue of the meaning of the classes. Such clues are, perhaps, somewhat easily misinterpreted, causing the reader to believe there are relations where they are only implied by the text. Thus, it is unclear whether embedding such information into the structural view is unambiguously useful to understanding that structure. In our OWL editor, Swoop, the user may see the names of classes while browsing, either by hovering over the class's circle, or by selection. (See section 4 for more details.)

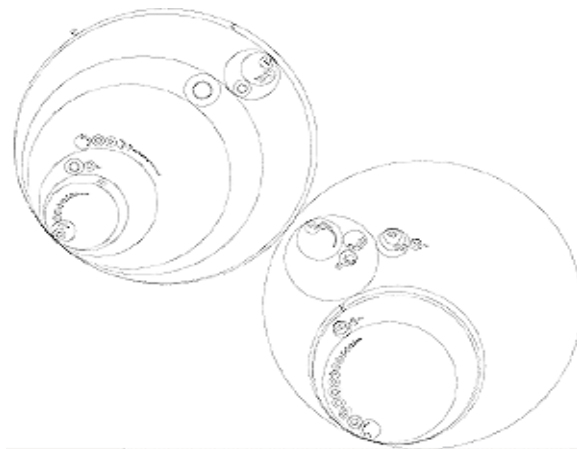


Fig. 3. Two branches of the Galen class tree.

Furthermore, it seems that even our naïve implementation of CropCircles scales to large, branchy, and deep branches of class trees. Figure 3 hints at this. Even though the top-down layout results in rather obviously poor space filling, the layout does a surprisingly good job of showing the structure of two complex class branches in a compact space. Several more circles could be added to the display without interfering with the existing ones. This fact helps in dealing with the problem that the same class can appear in many distinct CropCircles due to multiple inheritance. When a node is selected, the system highlights everywhere that node appears in the glass graph. Obviously, this is only effective if every branch in which the node appears can appear in the display.

3 Visualization of and with \mathcal{E} -Connections

\mathcal{E} -Connections[5] are a robust framework for combining several families of decidable logics including description logics, modal logics, and many logics of time and space. \mathcal{E} -Connections have also proved to be useful for supporting modular, distributed modeling such as is becoming common on the Semantic Web[2]: distinct subject domains can be represented as distinct components of an \mathcal{E} -Connection

while the \mathcal{E} -Connection constructors permit the definitions of classes in one component in terms of classes from other components. In this work, we restrict our attention to \mathcal{E} -Connections of OWL-DL ontologies.

\mathcal{E} -Connections are a very recent and novel formalism. Thus, there is no experience at all with modeling with them or understanding them. This situation entails two problems: 1) a lack of \mathcal{E} -Connected ontologies to study and 2) a lack of tools to help understand any such acquired \mathcal{E} -Connections. To address the first lack, in a related project in our lab, we developed a deterministic algorithm (see [3][1]) for automatically partitioning an OWL ontology into a corresponding \mathcal{E} -Connection. We now can easily generate interesting and interestingly structured \mathcal{E} -Connections. To address the second lack, in addition to implementing our decision procedure for \mathcal{E} -Connections of OWL ontologies in our OWL reasoner, Pellet³, we extended our Web Ontology editor, Swoop,⁴ with \mathcal{E} -Connection support. Figure 4 shows the basic Swoop interface for \mathcal{E} -Connections.

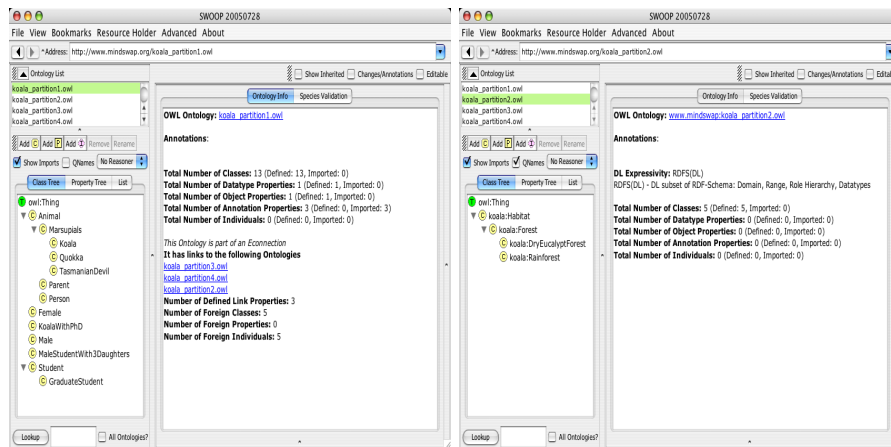


Fig. 4. The ontology editor Swoop browsing a generated \mathcal{E} -Connection corresponding to the sample ontology `koala.owl`. The figure on the left displays in detail the first component which corresponds to Animals. This component has links to all the other three components, as is shown in the right pane by the list of hyperlinks. The right figure shows the display after following one of those links.

Moving counterclockwise around a Swoop window from the upper left, first we see the list of components⁵ in the \mathcal{E} -Connections, in this case, the four components resulting from converting the example ontology⁶ from the Protégé tuto-

³ <http://www.mindswap.org/2003/pellet/>

⁴ <http://www.mindswap.org/2004/SWOOP/>

⁵ In general, the component list can also contain standard OWL ontologies.

⁶ <http://protege.stanford.edu/plugins/owl/owl-library/koala.owl>

rial.⁷ The selected component’s class tree (or property tree, or list of individuals) is displayed in a standard tree widget below the component list. Finally, in the right pane there is a view of the current focal entity, which, in these screenshots, is the selected component.

The view of the first component shows further useful information about the overall structure of the \mathcal{E} -Connection, namely that it has or can have link properties into all the other components. If we follow one of those hyperlinks, we still see the possible links coming that component; it is just that in all the other cases there are none. Thus, the display of the *overall* structure of the \mathcal{E} -Connection is in the dynamics of the browser: to understand the structure of the \mathcal{E} -Connection, one has to cycle through views of all the components. While perhaps feasible for this small \mathcal{E} -Connection, many large ontologies we have converted contain tens of components with hundreds or thousands of classes. Navigation just will not handle that well. Even with this very small \mathcal{E} -Connection, a simple graph layout (see Figure 5) is helpful. When the number of components and links gets large, the display gets less effective (at least, with the layout techniques we’ve tried thus far), but two features of the display retain their utility: first, nodes are proportional to the size of the component, that is, how many classes, properties, and individuals “belong” to that component. Second, nodes are colored according to whether they have no link properties (green), only incoming link properties (blue), and at least one outgoing link property (red). Intuitively, red nodes represent components that are more tightly coupled with other components. If there are a lot of small green nodes, then it is clear that the ontology has a lot of classes which are essentially placeholders for “future work.”

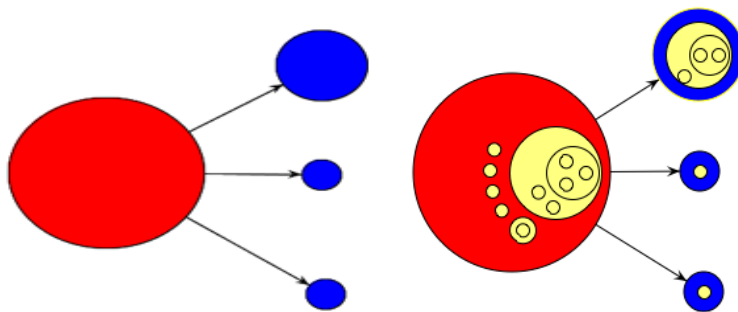


Fig. 5. On the left, a simple visualization of the Koala \mathcal{E} -Connection. On the right, the display has been augmented with CropCircles of the class trees of each component.

However, we can augment this display with CropCircles representing the class graph of each component. When combined with various interaction features, it seems superior to the normal Swoop browser for getting a sense of how

⁷ http://protege.stanford.edu/doc/tutorial/get_started/

the disjoint domains depend on one another. While CropCircles improve the \mathcal{E} -Connection visualization, the partitioning into an \mathcal{E} -Connection improves the efficacy of the CropCircles. Since each component circle represents a disjoint set of classes, it is impossible that any node in a component's CropCircle appears in any other component. The constraints of \mathcal{E} -Connections limit the scope of possible inheritance. Conversely, given that generated \mathcal{E} -Connections result from an algorithm that tries to break down the ontology as finely as possible, any classes that end up in the same component have a good chance of being related. In Figure 5, the right graph's largest component shows the asserted (that is, unclassified) class graph for that component. There is no single top level class, thus in a CropCircle of the unpartitioned ontology there would be nothing to connect these many classes, even though in the classified version all those nodes would appear inside the larger circle.

4 Implementation

Our CropCircle renderer is implemented in Java, using the the Java Universal Network/Graph Framework (JUNG)⁸. The layout of the top-level ontology vertices leverages on JUNG's layout package.

There is a minimum size for each circle that represents a class in CropCircle. Every leaf node has this minimum size. The sizes of the other node circles are computed to guarantee enclosure of all of its subtrees. Because the size of the leaf nodes is known, the layout algorithm can work from top down. At each node, the algorithm looks at the distribution of its subtrees to decide how to best lay its children out. Several strategies are employed. If the subtrees are of the same size, then they are laid out equidistant from the center of their parents. If there is only one single subtree, it is placed as a concentric circle to its parent. If the sizes vary, the algorithm places the subtree circles from the largest to the smallest, in one of two eye-pleasing spiral layouts.

In Swoop, CropCircles are used to support browsing. For example, a user may mouseover each circle to see the name of the class it represents. A user may also click on the circle to highlight and see a list of its immediate children on a selection pane. The selection pane can let user drill down the class hierarchy level-by-level, and it also supports the user's browsing history. Finally, Swoop's CropCircle browser allows user to selectively choose which interested top level ontology nodes to show in the visualization, so user can filter out ontologies that are not interesting (e.g. not connected to other ontologies).

5 Related Work

Visualizing hierarchies is an important task on the Semantic Web, and as a result there are several tools available that approach the problem. While there are many excellent visualizations for simple tree navigation that show the hierarchy

⁸ <http://jung.sourceforge.net/>

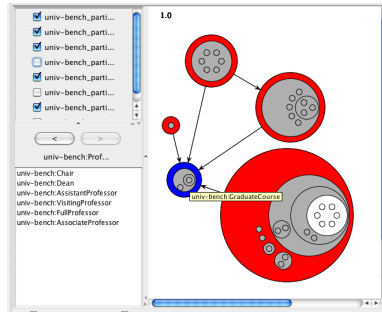


Fig. 6. The CropCircle based browser in Swoop browsing an \mathcal{E} -Connection generated from the Lehigh University Benchmark ontology.

in a list format, the number of classes, the depth of these ontologies, and the connections between them require a different approach if the user is to get a full visual grasp of the tree structure and interconnections.

Jambalaya[9] is a visualization tool integrated into the Protégé development environment that uses SHriMP [8] to visualize regular knowledge-bases. It does not focus on any one visualization technique, but rather supports several views of ontologies. It includes a nested view and a treemap view for visualizing hierarchies.

Treemaps are a space-filling visualization method used to represent large hierarchical collections of quantitative data [7]. A treemap works by dividing the display area into a nested sequence of rectangles. The size and color of the areas are connected to specific features of the dataset.

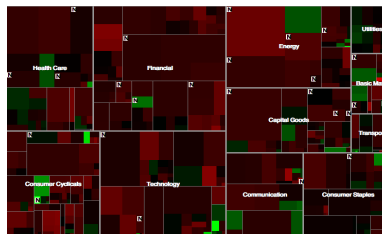


Fig. 7. A sample tree map from the SmartMoney Market Map

In one respect, the Jambalaya nested and treemap views are closest to the approach we take here. However, there are differences that are important for our particular task of visualizing Semantic Web ontologies and \mathcal{E} -Connections. Most importantly for our tasks, clearly seeing the distinctions between subtrees is central. With treemaps, the borders between subtrees are necessarily subtle in order to allow for the space-filling features.

The Cluster Map visualization [4] has aspects that are similar to ours in terms of showing the connections between subgroups. Their focus is on clustering instances according to class, and showing clusters of instances that overlap several categories. We employ similar techniques to show connections between \mathcal{E} -Connected ontologies. However, since our tool is one for hierarchies of classes, the techniques are fundamentally different.

6 Conclusion

An interesting aspect of CropCircles is their aesthetic quality. People find them both striking and intriguing, which, if combined with effective interaction mechanisms, we expect will increase users' willingness and interest in exploring large ontologies. Future work includes minimizing the space used by the layout without sacrificing the visual appeal, exploring more interaction features such as additional zooming and other animation, and incorporating other sorts of information such as instances and property-based relations between classes.

References

1. B. Cuenca-Grau. *Combination and Integration of Ontologies on the Semantic Web*. PhD thesis, Universidad de Valencia, 2005. Available at <http://www.mindswap.org/2004/multipleOnt/papers/Dissertation.pdf>.
2. B. Cuenca-Grau, B. Parsia, and E. Sirin. Working with multiple ontologies on the semantic web. In *Proc. of ISWC 2004*, 2004.
3. Bernardo Cuenca-Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Automatic partitioning of owl ontologies using \mathcal{E} -Connections. Technical report, UMIACS, 2005. Available at <http://www.mindswap.org/2004/multipleOnt/papers/Partition.pdf>.
4. Christiaan Fluit, Marta Sabou, and Frank van Harmelen. Ontology-based information visualisation: Towards semantic web applications. *Proceedings of Visualizing the Semantic Web*, 2005.
5. O. Kutz, C. Lutz, F. Wolter, and M. Zakharyashev. \mathcal{E} -Connections of abstract description systems. *Artificial Intelligence* 156(1):1-73, 2004.
6. Thorsten Liebig and Olaf Noppens. OntoTrack: Combining browsing and editing with reasoning and explaining for OWL Lite ontologies. *Proceedings of the 3rd International International Semantic Web Conference*, 2004.
7. Ben Shneiderman. Tree visualization with treemaps: A 2-D space-filling approach. *ACM Transactions on Graphics*, 1(11):92-99, 1992.
8. M.-A. Storey, C. Best, J. Michaud, D. Rayside, M. Litoiu, and M. Musen. Proceedings of CHI 2002. conference extended abstracts on human factors in computer systems. *SHriMP views: an interactive environment for information visualization and navigation*, 2002.
9. M.-A. D. Storey, M. A. Musen, J. Silva, C. Best, N. Ernst, R. Ferguson, and N. F. Noy. Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé. *Workshop on Interactive Tools for Knowledge Capture (K-CAP-2001)*, 2001.