AxiomVis: An Axiom-Oriented Ontology Visualization Tool

Jian Zhou[1,2], Xin Wang[1,2(✉)], and Zhiyong Feng[1,2]

[1] School of Computer Science and Technology, Tianjin University, Tianjin, China
[2] Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China
{jianzhou,wangx,zyfeng}@tju.edu.cn

Abstract. With the continuous growth of ontologies in size, the amount of ontology axioms is steadily increasing. To help users understand ontologies with a large number of axioms, we devise a novel ontology visualization tool, called *AxiomVis*, which defines a mapping from DL axioms to the *directed acyclic graph* (DAG). We give a recursive procedure to generate the target DAG and then propose a new *axiom-aware* layout algorithm to display the DAG elements using the characteristics of the expression tree. The distinguishing feature of our tool is that it visualizes ontologies from the axiom viewpoint. This paper demonstrates comparison of different layouts, which exhibits the advantages of our layout on ontology visualization.

Keywords: Ontology · Axioms · Visualization · Layout

1 Introduction

With the rapid development of the Semantic Web, more and more ontologies have been released in the Web Ontology Language (OWL) as standardized by W3C [1]. As a cornerstone of OWL, description logics are a family of logic-based knowledge representation formalisms, which support the logic description of concepts, roles, and their relationships, using a variety of operators, to form more complex descriptions. A description logic knowledge base typically consists of two components, i.e., a TBox and an ABox. In a TBox, the concept definition of a new concept is in terms of other defined concepts. For example, the fact that orphan's parents are not alive is expressed by the concept inclusion

$$\texttt{Orphan} \sqsubseteq \texttt{Human} \sqcap \forall \texttt{hasParent}.\neg\texttt{Alive} \tag{1}$$

in which case we say that the concept orphan is subsumed by the composite concept

$$\texttt{Human} \sqcap \forall \texttt{hasParent}.\neg\texttt{Alive} \tag{2}$$

This paper focuses on TBox axioms since they are more complex than ABox axioms and our techniques can be easily adapted to handle ABox axioms (Note that in the following we use the term *axiom* to refer to a TBox axiom).

D. Zhao et al. (Eds.): CSWS 2014, CCIS 480, pp. 156–166, 2014.
DOI: 10.1007/978-3-662-45495-4_14

It is known that axioms are the first-class citizens in ontologies. However, The majority of the existing techniques for ontology visualization [2–12] are not aware of axioms, which cannot represent relationships among concepts, roles, and operators from the axiom perspective. For example, several visualization techniques include OWL-VisMod [14] and Prefuse [15], which use treemapping technology that is a method for displaying hierarchical data by using nested rectangles to visualize certain aspects of ontologies. While these techniques consider the class hierarchy of ontologies, they do not take into account axioms. The approaches that are closely related to AxiomVis include GrOWL [13] and SOVA[1], which use graph-based visualization techniques to represent ontologies. GrOWL and SOVA define more elaborated notations in order to be consistent to ontology semantics according to the custom criteria. However, considering the fact that description logics are decidable subsets of first-order logic, we can utilize the form of the expression tree to represent ontologies. We propose an axiom-aware ontology visualization tool, called AxiomVis, which (1) constructs a DAG according to a set of axioms, (2) draws a DAG according to our axiom-aware layout, and (3) shows a set of demonstration use cases.

2 Algorithms

In this section, we describe the core algorithms used in AxiomVis, which can construct a DAG from axioms and then draw this graph according to our axiom-aware layout.

We present the DAG construction function $dag : \mathcal{P}(TB) \rightarrow DAG$, where $\mathcal{P}(TB)$ is the power set of the set of axioms TB and DAG is the set of DAGs. Let N_C and N_R be infinite sets of concept and role names, respectively. We define the set of concept descriptions on N_C and N_R as $Desc$ that is the smallest set such that (1) for each $A \in N_C$, $A \in Desc$, (2) if $C \in Desc$ and $D \in Desc$, then $\neg C \in Desc$, $C \sqcap D \in Desc$, and $C \sqcup D \in Desc$, and (3) if $r \in N_R$ and $C \in Desc$, then $\exists r.C \in Desc$ and $\forall r.C \in Desc$. A TBox axiom is of the form $A \equiv C$ or $C \sqsubseteq D$, where $A \in N_C$, $C \in Desc$, and $D \in Desc$. We use $O = \{\neg, \sqcap, \sqcup, \exists, \forall, \equiv, \sqsubseteq\}$ to denote the set of operators. TB can be modelled as a DAG $G = (V, E)$, where the node set $V = \{v \mid v \in N_C \cup N_R \cup O\}$, the edge set $E \subseteq V \times V$. Assume the function $root : Desc \cup N_R \rightarrow V$, where this function returns the root node of the expression tree that corresponds to $d \in Desc$. If $d \in N_C \cup N_R$, then $root(d) = d$. For a given set of axioms TB, the function $dag(TB)$ constructs a DAG G in accordance with the following steps: for each $t \in TB$, (1) if $\neg C \in Desc$ and $\neg C$ occurs in T, there exists an edge $e = (\neg, root(C)) \in E$, (2) if $C \in Desc$, $D \in Desc$, $m \in \{\sqcap, \sqcup\ \sqsubseteq, \equiv\}$, and CmD occurs in T, there exists two edges $e_1 = (m, root(C)) \in E$ and $e_2 = (m, root(D)) \in E$, and (3) if $r \in N_R$, $C \in Desc$, $m \in \{\exists, \forall\}$, and $mr.C$ occurs in t, there exists two edges $e_1 = (m, r) \in E$ and $e_2 = (m, root(C)) \in E$. When the number of axioms is only one, the DAG G degenerates into a expression tree. For example, the axiom (1) can be converted

[1] http://protegewiki.stanford.edu/wiki/SOVA

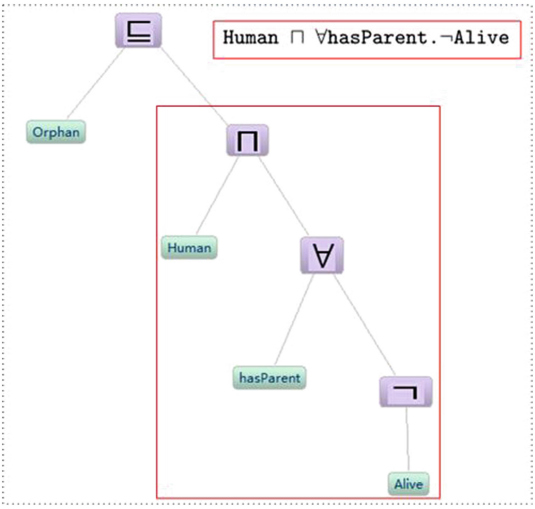


Fig. 1. The subtree representation of the composite concept

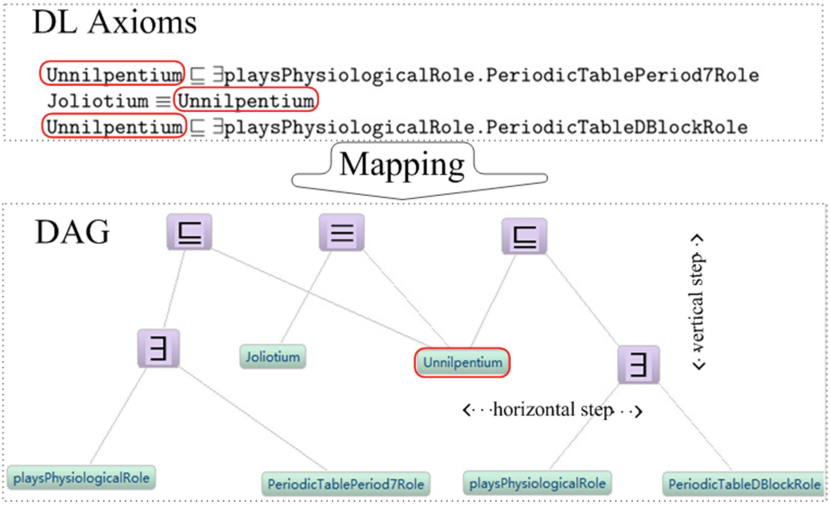


Fig. 2. A mapping from DL axioms to the DAG and axiom-aware layout

to the expression tree, and the composite concept (2) can be converted to the subtree as shown in Fig. 1, whose root node is the operator $\sqsubseteq$.

We design the aware-axiom layout algorithm in order to draw the DAG G. Each axiom is an expression tree in the DAG. By calculating the depth of the expression tree and the number of nodes in each layer, we can easily get the horizontal step for the tree and the vertical step for each layer as shown in Fig. 2. The vertical step and the horizontal step determine the vertical coordinates of

Algorithm 1. Axiom-aware layout

Input: the set of axioms TB
Output: the DAG G, coordinates of each node in the DAG G

for each $t \in TB$ **do**
  $G \leftarrow dag(TB)$
  $wd \leftarrow w/|TB|$ ▷ wd is the horizontal width of a expression tree in the canvas
  $k \leftarrow 1$, $x \leftarrow 0$, $y \leftarrow 0$, $prex \leftarrow 0$ ▷ Coordinates of node (x, y)
  $et \leftarrow G.getTree(t)$
  $vs \leftarrow h/(et.layer + 1)$ ▷ vs is the vertical step of the adjacent layer
  while $k \leqslant et.layer$ **do**
    $Q \leftarrow et.getQueue(k)$
    $hs \leftarrow wd/(Q.size() + 1)$ ▷ hs is the horizontal step of the same layer
    while $!Q.isEmpty()$ **do**
      $node \leftarrow Q.dequeue()$
      $x \leftarrow x + hs$ ▷ Update the horizontal coordinates of nodes
      $node.setLoc(x, y)$
    end while
    $y \leftarrow y + vs$ ▷ Update the vertical coordinates of nodes
    $k \leftarrow k + 1$ ▷ Move down to the next layer
  end while
  $x \leftarrow prex + wd$ ▷ Update the horizontal coordinate of the root node
  $y \leftarrow 0$ ▷ Update the vertical coordinate of the root node
  $prex \leftarrow x$
end for

nodes between the adjacent layers and the horizontal coordinates of nodes in the same layer, respectively. For the sake of managing the position of each node in the tree, multiple queues are created. The number of queues is the depth of the tree. Each queue allows to enqueue the nodes with the same layer in the tree. When drawing the node in the DAG, each queue keeps the horizontal step to dequeue the nodes with the same layer, and then the queues maintain the vertical step between the adjacent layers. Let h be the height of the canvas, w be the width of the canvas, et be a expression tree, $et.layer$ be the layer of et, and Q is a queue that is composed by the nodes with the same layer in et. Algorithm 1 shows the procedure for the axiom-aware layout, in which the function $G.getTree$, $et.getQueue$, and $node.setLoc$ are used to get the expression tree et in G according to the given axiom t, get a queue Q for one layer k in et, and set the location of $node$ in the canvas, respectively.

We establish a mapping from DL axioms to the DAG. For example, we visualize all DL axioms that include the concept `Unnilpentium` in OpenGALEN. It is obvious to observe that the node Unnilpentium in the DAG represents the concept in DL axioms and a tree in the DAG denotes an axiom as shown in Fig. 2.

3 Demonstration

This section presents the demonstration environment and scenarios of AxiomVis, which is published as an open source software[2].

3.1 Demonstration Environment

AxiomVis is written in Java, and the version of JDK is 1.7. The Eclipse Rich Client Platform (RCP) as the basis of AxiomVis. The visualization part of AxiomVis is developed using Zest[3], a set of visualization components built for Eclipse. The main feature of Zest is that it makes graph programming easy. Our computer has Intel 3.30 GHz CPU and 4 GB memory. The operating system is Windows 7. Our method is general enough to be applied to all OWL ontologies. Due to space limitations, we only use OpenGALEN[4] (209,248 axioms, 84 fragments of OWL) as an example to demonstrate the functionalities of our AxiomVis.

3.2 Demonstration Scenarios

In this section, we present the demonstration scenarios by the following use cases.

Demonstration use case 1: Querying the number of the concepts or roles.

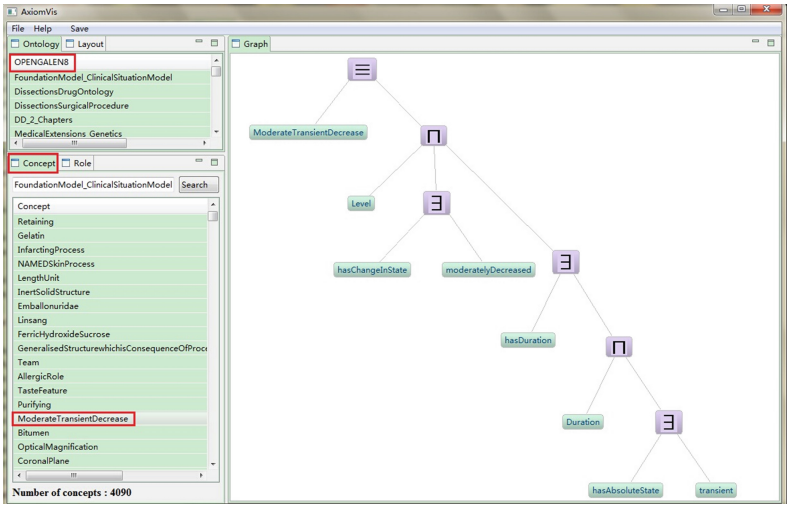
When choosing 8 different ontology fragments in OpenGALEN, we can get the total number of the concepts or roles in the selected fragment. Table 1 shows these query results.

Table 1. Demonstration results of querying the number of the concepts or roles

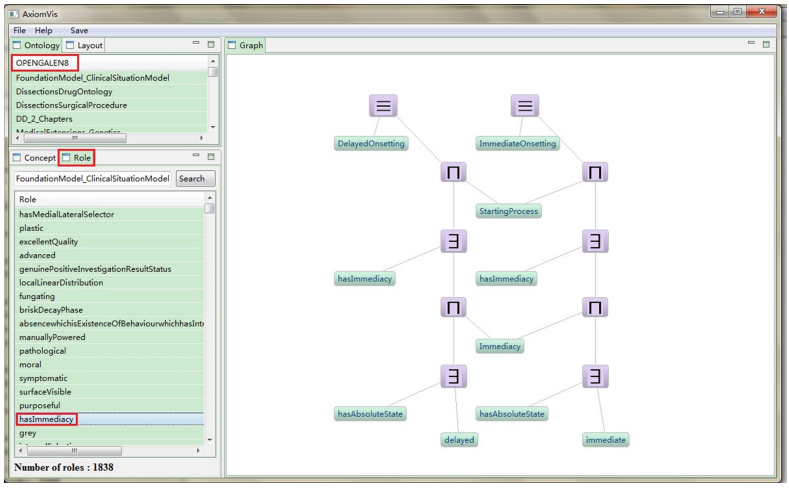
Serial number	Ontology fragment	#concepts	#roles
t1	`FoundationModel_ClinicalSituationModel`	4090	1838
t2	`MedicalExtensions_HumanAnatomy`	11682	1853
t3	`MedicalExtensions_Genetics`	11631	1229
t4	`DissectionsDisease`	34342	5514
t5	`DS_4076_Musculoskeletal`	37148	6436
t6	`DS_85_Diagnostic`	27764	2209
t7	`DS_3227_Cardiothoracic`	34054	5048
t8	`DD_8048_Components`	37667	5054

Demonstration use case 2: Querying a concept or role.

[2] AxiomVis. http://xinwang.tju.edu.cn/drupal/?q=research/demo/axiomvis.
[3] Zest. http://www.eclipse.org/gef/zest/.
[4] OpenGALEN. http://www.opengalen.org/.



(a) Ontology visualization according to the selected concept



(b) Ontology visualization according to the selected role

Fig. 3. Screenshots of demonstration use cases 3

If you want to query a concept or role in the selected fragment of OWL, the query results will be listed after clicking on the search button in the fragment `FoundationModel_ClinicalSituationModel` as shown in Fig. 6(a). We can also obtain the total number of the concept or role in this fragment.

Demonstration use case 3: Ontology visualization according to the selected concept or role.

We can get visualization information, for instance, if we choose the concept `ModerateTransientDecrease` or the role `hasImmediacy` in OpenGALEN

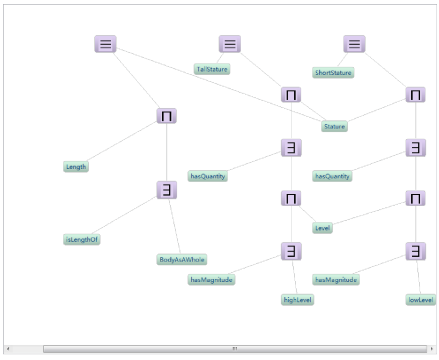
as shown in Fig. 3(a) and (b). The default layout is our axiom-aware layout. It is easy to get the axiom (3) related to the concept `ModerateTransientDecrease` from Fig. 3(a).

$$\begin{aligned}\texttt{ModerateTransientDecrease} \equiv\ & \texttt{Level} \sqcap \\ & \exists\texttt{hasChangeInState.moderatelyDecreased} \sqcap \\ & \exists\texttt{hasDuration.(Duration} \sqcap \exists\texttt{hasAbsoluteState.transient)}\end{aligned} \tag{3}$$

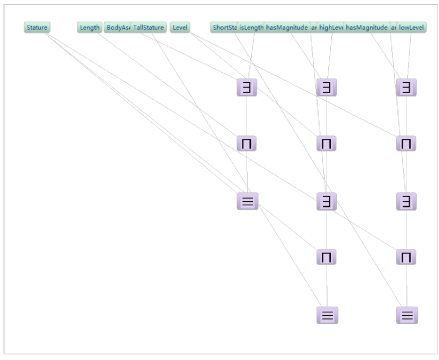
Similarly, we can obtain the following two axioms based on the role `hasImmediacy` from Fig. 3(b).

$$\begin{aligned}\texttt{DelayedOnsetting} \equiv\ & \texttt{StartingProcess} \sqcap \exists\texttt{hasImmediacy.} \\ & \texttt{(Immediacy} \sqcap \exists\texttt{hasAbsoluteState.delayed)}\end{aligned} \tag{4}$$

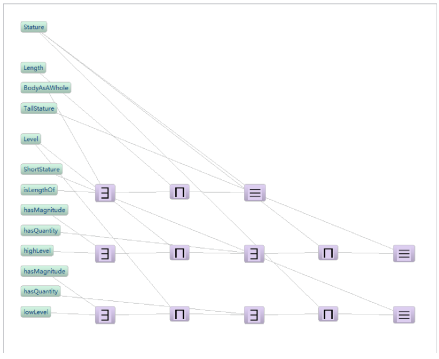
$$\begin{aligned}\texttt{ImmediateOnsetting} \equiv\ & \texttt{StartingProcess} \sqcap \exists\texttt{hasImmediacy.} \\ & \texttt{(Immediacy} \sqcap \exists\texttt{hasAbsoluteState.immediate)}\end{aligned} \tag{5}$$



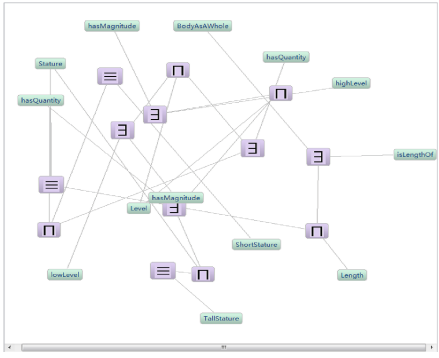
(a) Axiom-aware layout



(b) Vertical tree layout



(c) Horizontal tree layout



(d) Spring layout

Fig. 4. Screenshots of demonstration use case 5

(a) Grid layout

(b) Horizontal shift Layout

(c) Horizontal layout

(d) Vertical layout

(e) Directed graph layout

(f) Radial layout

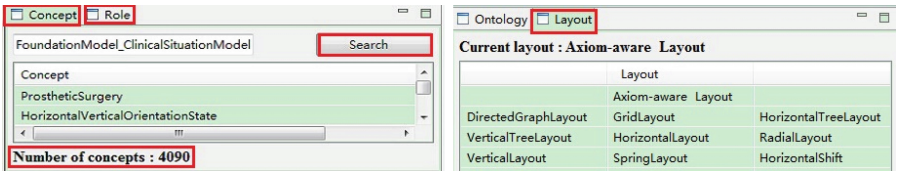
Fig. 5. Screenshots of demonstration use case 5

If necessary, we also can switch to a different Layout, drag the nodes and edges in the canvas, and save this canvas.

Demonstration use case 4: Graph layout options.

If you want to choose a different layout for the current visual graph. AxiomVis provides a number of layout options including our axiom-aware layout and 9 layouts from Zest as shown in Fig. 6(b).

Demonstration use case 5: Comparison of layout algorithms.



(a) Concept or role query (b) Layout options

Fig. 6. Screenshots of demonstration use cases 2, 4

We choose our axiom-aware layout algorithm. As a comparison, We also select 9 layouts from Zest that include VerticalTreeLayout, HorizontalTreeLayout, SpringLayout, GridLayout, HorizontalShiftLayout, HorizontalLayout, VerticalLayout, DirectedGraphLayout, and RadialLayout for ontology visualization. Figures 4(a)–(d) and 5(a)–(f) show the results of clicking the concept `Stature` in OpenGALEN. HorizontalTreeLayout and VerticalTreeLayout cannot clearly distinguish each axiom in the canvas, although they can keep a tree structure from the horizontal or vertical direction. SpringLayout, GridLayout, and HorizontalShiftLayout cannot make full use of the characteristics of the expression tree for each axiom even though they can properly arrange the position of each node in the DAG. HorizontalLayout, VerticalLayout, and DirectedGraphLayout pay too much attention to the overall layout of direction, which lead to the overcrowding of visual elements. RadialLayout selects some elements as the root nodes, and then arranges the other nodes to form a series of circles, which also ignores the characteristics of the expression tree. In summary, it can be observed that our layout has more advantages on dealing with ontology visualization than the state-of-the-art layouts.

4 Conclusion

This paper presents a new axiom-oriented visualization approach, which can facilitate the analysis and understanding of ontologies from the axiom perspective. In the future, many aspects of the approach can be improved including the design of user-friendly interface and further optimization of axiom-aware layout algorithm. We hope to integrate more features into AxiomVis. In particular, we would like to improve the performance of AxiomVis and design a hierarchical structure to visualize large-scale ontologies. We also need to address the representation of individuals and extend our algorithms to handle ABox axioms.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (61100049), the Graduate English Course Construction Project of Tianjin University (S216E003), and the National High-tech R&D Program of China (863 Program) (2013AA013204).

References

1. McGuinness, D.L., Van Harmelen, F.: OWL web ontology language overview. W3C recommendation (2004)
2. Alani, H.: TGVizTab: an ontology visualisation extension for Protégé. In: Knowledge Capture Workshop on Visualization Information in Knowledge Engineering (2003)
3. Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A.: The Protégé OWL plugin: an open development environment for semantic web applications. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 229–243. Springer, Heidelberg (2004)
4. Hussain, A., Latif, K., Rextin, A., Hayat, A., Alam, M.: Scalable visualization of semantic nets using power-law graphs. Appl. Math. Inf. Sci. **8**(1), 355–367 (2014)
5. Negru, S., Haag, F., Lohmann, S.: Towards a unified visual notation for OWL ontologies: insights from a comparative user study. In: 9th International Conference on Semantic Systems, pp. 73–80 (2013)
6. Motta, E., Mulholland, P., Peroni, S., d'Aquin, M., Gomez-Perez, J.M., Mendez, V., Zablith, F.: A novel approach to visualizing and navigating ontologies. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 470–486. Springer, Heidelberg (2011)
7. Heim, P., Lohmann, S., Stegemann, T.: Interactive relationship discovery via the semantic web. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part I. LNCS, vol. 6088, pp. 303–317. Springer, Heidelberg (2010)
8. Lanzenberger, M., Sampson, J.: Alviz - a tool for visual ontology alignment. In: 10th International Conference on Information Visualization, pp. 430–440 (2006)
9. Storey, M.-A., Noy, N.F., Musen, M., Best, C., Fergerson, R., Ernst, N.: Jambalaya: interactive visualization to enhance ontology authoring and knowledge acquisition in protégé. In: Workshop on Interactive Tools for Knowledge Capture (2001)
10. Katifori, A., Torou, E., Halatsis, C., Lepouras, G., Vassilaki, C.: A comparative study of four ontology visualization techniques in protégé: experiment setup and preliminary results. In: Information Visualization, pp. 417–423. IEEE (2006)
11. Parsia, B., Wang, T., Golbeck, J.: Visualizing web ontologies with cropcircles. In: 4th International Semantic Web Conference, pp. 6–10 (2005)
12. Wang, T.D., Parsia, B.: CropCircles: topology sensitive visualization of OWL class hierarchies. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 695–708. Springer, Heidelberg (2006)
13. Krivov, S., Villa, F., Williams, R., Wu, X.: On visualization of OWL ontologies. In: Baker, C.J.O., Cheung, K.-H. (eds.) Semantic Web, pp. 205–221. Springer, New York (2007)

14. García-Peñalvo, F.J., Colomo-Palacios, R., García, J., Therón, R.: Towards an ontology modeling tool. A validation in software engineering scenarios. Expert Syst. Appl. **39**(13), 11468–11478 (2012)
15. Heer, J., Card, S., Landay, J.: Prefuse: a toolkit for interactive information visualization. In: Proceedings of the SIGCHI Conference on Human factors in Computing Systems, pp. 421–430 (2005)