Knoocks - Ontology Visualization Plug-in for Protégé

Mgr. Adam Jurčík*
Supervised by: doc. Ing. Jiří Sochor, CSc.†

Department of Computer Graphics and Design
Faculty of Informatics Masaryk University, Brno

Abstract

When searching the World Wide Web for information we use search engines such as Google, Yahoo! or Bing. Full text search works well when finding some articles or documents, but it will not directly answer the questions as "How much something costs?" or "Where can I go for lunch?". Wanted information can be available on the Web as web pages or data (from services) but both without semantics. Ontologies are proposed by W3C to enable inclusion of semantics in web pages. This paper presents Knoocks visualization plug-in for Protégé-OWL editor. The Knoocks approach combines the traditional ontology visualization (e.g., node-link or space-filling) strategies to enhance understandability of OWL Lite ontologies. The re-implementation of Knoocks as a plug-in for Protégé-OWL extends the editor with a new ontology visualization technique and makes the Knoocks visualization publicly available for users. The Knoocks plug-in is good at visualizing of non-trivial ontologies.

1 Introduction

Automated processing of data contained in web pages is impossible without having knowledge of its semantics. In computer science, ontologies represent knowledge of a domain as a set of concepts and relationships between them. Building an OWL (*Web Ontology Language*) ontology and marking up a web page will enable web page's content to be understood by machines [1]. To build an OWL ontology, Protégé-OWL editor[1] (publicly available as freeware) can be used. The editor allows people to manipulate OWL entities such as classes, object/data properties and individuals. The ontology is itself a set of axioms that define entities and relationships between them, the axioms are more easily understood when they are visualized.

The Protégé-OWL editor has basic tree views that show hierarchical relationships between entities. Currently available visualization plug-ins for Protégé-OWL, e.g., OntoGraf, NavigOWL or SOVA visualize ontologies as interlinked nodes. The node-link visualization approach [3] draws classes and individuals as nodes, relationships between entities (hierarchical or property) are drawn as links. On the contrary, space-filling visualizations group entities together by classes or by properties. Groups are drawn as areas that can be included by other areas that represent more general concepts.

Node-link visualizations are optimal in presenting parts of ontologies in detail, while when visualizing overview of an ontology the resulting visualization can become confusing, hard to understand. Finally, it is up to the user to lay out nodes to create understandable presentation of chosen concept. Space-filling visualizations give users very clear idea of hierarchical relationships between entities, i.e. subclasses, or general concepts in ontologies – subdivision of an ontology into smaller logically connected parts. Lack of links in pure space-filling visualizations (see CropCircles[2] [11]) does not allow to express dependencies (typically object property instances) across entities. Therefore, combination of both node-link and space-filling approaches are examined to create more universal visualizations (see Jambalaya[3] [10]).

A Protégé-OWL visualization plug-in implementing Knoocks visualization approach was developed as a part of master's thesis at our department. The Knoocks (stands for Knowledge Blocks) approach was designed by Kriglstein and Motschnig-Pitrik as a combined visualization approach [7]. This paper will present only key concepts of suggested visualization approach. The plug-in was implemented in Java as an OSGi (*Open Services Gateway initiative*) bundle for Protégé framework[4] and uses OWL API and Protégé-OWL API to introspect the visualized ontology. Architecture of the plug-in and some chosen implementation details are presented by this paper. Also an example of usage is included which demonstrates how to benefit from Knoocks when getting familiar with an unknown ontology.

2 Related work

In 2008 Kriglstein showed advantages of representing university curricula as ontologies. The curricula that were rolled out at University of Vienna in 2006 were organized into modules forming a clear structure. The structure of

*xjurc@fi.muni.cz
†sochor@fi.muni.cz

[1] http://protege.stanford.edu/overview/protege-owl.html

[2] http://www.mindswap.org/2005/cropcircles/

[3] http://www.thechiselgroup.org/jambalaya

[4] http://protege.stanford.edu/

a curriculum can be represented by hierarchy of classes and contained courses are in ontologies included as individuals [4]. When representing curricula as ontology it is easy to express (using object properties) arbitrary relationships between courses, modules and other objects. Visualization of curriculum's ontology helps teachers and students to understand it, e.g., it is easier to see dependencies among courses. Ontologies also enable to share or analyze the knowledge contained in a curriculum. In her paper, Kriglstein compared several ontology visualization techniques in the context of curricula visualization and concluded that most requirements were met by Jambalaya.

However, Kriglstein and Motschnig-Pitrik identified disadvantages in existing ontology visualizations and proposed Knoocks as a new visualization approach [7]. Newly proposed approach targeted on three main points.

- **Visualization of ontology overview** – an overview [9] of structure should help user to understand and navigate large ontologies that contain multiple subhierarchies of classes.
- **Visualization of classes** – every class should be easily seen and also names of classes should be visible to distinguish among them.
- **Visualization of individuals** – individuals represent concrete objects and therefore it is important to show them. Also names and classification of individuals should be visualized to make understanding of ontologies easier.

Knoocks uses space-filling visualization to show hierarchy of classes as a block. Each class is represented as a rectangle contained in the block. Furthermore, classes contain individuals to emphasize their classification. First version of Knoocks was implemented in Java and was able to visualize only one class hierarchy. Therefore the requirement of ontology's overview visualization was not met.

Further development of Knoocks was based on user requirement analysis that was conducted by University of Vienna and focused on ontology experts and semi-experts [5]. Interviews and online survey were organized to gather users' requirements on ontology visualizations. The analysis showed that users do not clearly prefer one visualization tool over another. On the other hand, users' expectations about general ontology visualization were similar in few points. Users stated that a good visualization should have both overview and detail views, browsing and updating should be possible and relationships between entities (subclass or object properties) should be visible.

Ontology overview and data/object properties visualization was provided by a new prototype of Knoocks [8]. The prototype was written using C# and OpenGL as a standalone application. This version of Knoocks added switching between overview, which shows an ontology as a set of blocks, and detail view, which shows more details about a block. In overview, object properties between individuals are visualized as edges (node-link approach is used) between their classes. Details about object properties, e.g., which individual is related to which, are available in detail view of a block [6]. Also data properties of an individual can be made visible in detail view. Layout of visualized individual details was chosen after evaluation of its usability by users. Knoocks prototype was further extended to include searching and filtering of entities, and also navigation among detail views based on class/individual selection was added. The Knoocks prototype was taken as a specification for Knoocks plug-in implementation for Protégé-OWL editor. Therefore, Knoocks visualization concepts that were implemented by the plug-in will be described in more details in the next section.

3 Features of Knoocks plug-in

Knoocks plug-in is available in Protégé-OWL as a tab plug-in. Tab plug-ins can be included or excluded from editor's view – configuration is done my using *Window/Tabs* menu. Newly installed tab plug-in is hidden by default.

The Knoocks tab is vertically split into two panels (see Figure 1), where left panel contains user controls for ontology searching and filtering, and right panel contains interactive visualization of an active ontology – the editor allows to manage more ontologies at a time.

The Knoocks prototype provides also a preview view that is in our plug-in missing yet.

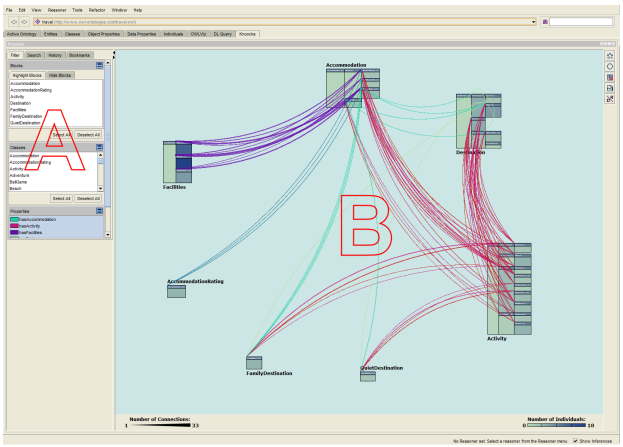
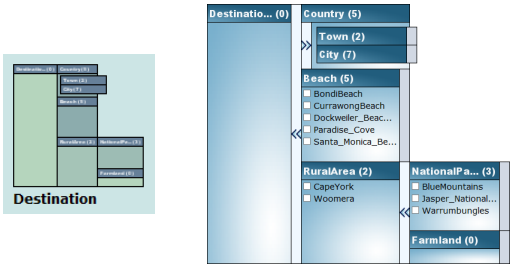


Figure 1: Knoocks tab plug-in in Protégé-OWL. The plug-in's view contains the control panel (A) and the visualization slot (B).

3.1 Knowledge block

Arrangement of class hierarchies into blocks is a key concept of Knoocks approach. Each class that is a direct subclass of the `OWL:Thing` class becomes a root of a block. The block is named after its root class. A block is laid out (see Figure 2) in a way such that subclasses are placed on the right side of their parents. Classes are drawn

as rectangles with captions that show class names (visible only in detail view). Individuals are displayed within their classes. Therefore, the amount of space occupied by a block depends on width of its hierarchy branching and also on count of individuals contained in classes. Classes in a block can be folded to occupy less space in the visualization.



(a) Overview (b) Detail view

Figure 2: Example of a block.

3.2 Overview

Laying out all blocks from an ontology gives users a good view on its structure. In overview, blocks are drawn with less detail, e.g., class names are provided rather as tooltips and individual names are not drawn in class rectangles. At first, blocks are automatically arranged into circular shape (see Figure 3), but they can be moved, so that users can control the visualization themselves. Blocks can be rearranged to circular layout when needed. Bodies of classes are drawn with different shades to visualize cardinality of classes. The darker the class is the more individuals it contains. Navigation to detail view of a block is done by double-clicking it.

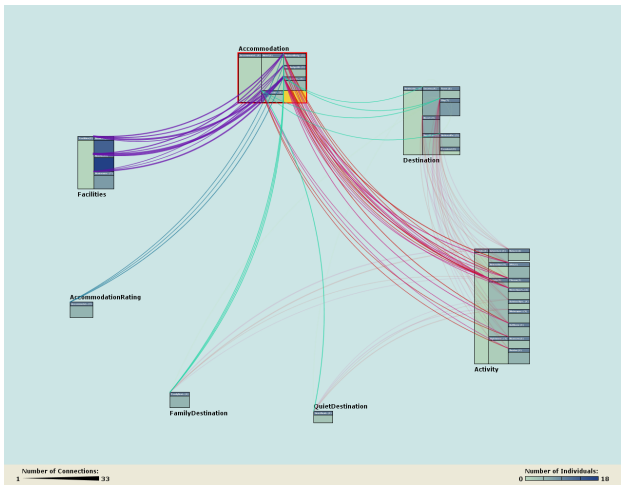


Figure 3: Example of overview of an ontology, blocks were laid out automatically.

Colored connections are drawn to visualize relationships between blocks/classes. Different colors are chosen to distinguish among object properties. Edges are not drawn directly between individuals (where instances of object properties are defined), they are grouped to connections by their class paths (domain and range) instead. Thickness of a connection depends on count of object property instances that were grouped, so that users can quickly infer how much are the two classes interconnected. Visualization of object properties is interactive, a connection can be clicked to display a table which lists concrete relations that were grouped. Classes and individuals in the connection table are also clickable and allow navigating to them, i.e., they are shown in detail view when clicked.

When a block is selected by clicking on it, connections that are not related to selected block are faded out, so that block's connections are emphasized.

Overview provides also useful statistics about ontology visualization. The statistics are located at the bottom panel in the view and shows maximum connection and class cardinalities among the ontology.

3.3 Detail view

Detail view allows users to focus on a block that is of their interest. Class names are visible in class captions and listings of individuals are shown for non-empty classes. A class can contain many individuals, and therefore paging of listing is used – the listing could become very long and would require classes to be too high. A bar button with arrow is added between each parent class and its children to control folding of classes. Class captions and individuals are clickable to allow users to get details about them.

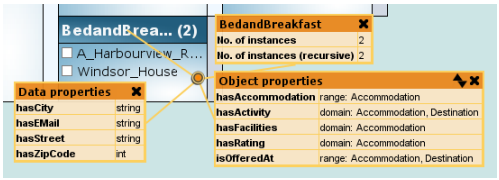


Figure 4: Details of `BedandBreakfast` class.

Class details include statistics about its individuals, data types of related data properties and domains or ranges of related object properties. All three types of information are shown in separate pop-up tables (see Figure 4) that can be moved (independently or together). When an individual is clicked, values of its data properties and its relations to other individuals are visualized. Values of data properties are shown in a pop-up table – in the same way as data types of a class. Relations to other individuals are grouped by their class paths and shown as listings – one for each different class path (see Figure 5). The table and the listings can be moved (same as class details), closing of shown details is done by closing data property values table. In addition, classes and individuals that are displayed in a relation listing are clickable, so that they allow navigation in the visualization.

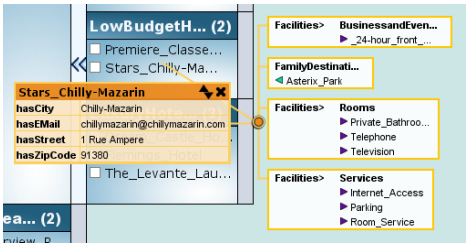


Figure 5: Details of `Stars_Chilly-Mazarin` individual.

3.4 Filtering and Search

Knoocks visualization usability is improved by filtering of ontology entities. In overview, visibility of blocks and connections can be controlled. A block can be hidden by selecting a row with its name in *Hide Blocks* list. Visibility of connections depends on visibility of object properties and is set in *Properties* list. Blocks and classes can be highlighted to emphasize them in the visualization. The highlight is visible in both overview and detail view. For users' convenience, visibility of object properties and block/class highlights can be controlled from context menus too.

The plug-in allows for basic search in the visualized ontology. *Search* panel is used to retrieve matching classes and individuals. Classes are typeset bold in the results list. Clicking on a class or an individual that were matched navigates in the ontology to it and shows it in detail view. Almost every navigation (by clicking) to a class or an individual is recorded and shown in *History* panel. The *History* panel allows users to go back where they came from, this is often useful when inspecting relationships in an ontology. Jumps into history are not recorded again. Furthermore, to make navigation in big ontologies easier, individuals can be bookmarked. Bookmarked individuals can be found in *Bookmarks* panel. When a bookmark is clicked, the view is navigated to particular individual.

Highlights of blocks and classes are visible in all lists that show their names.

4 Implementation

Protégé is modular Java framework for knowledge-base management. The framework is based on OSGi service platform specification[5]. Protégé-OWL editor for OWL ontologies is implemented as a module (bundle in OSGi) which uses and extends functionality of core Protégé. Two versions of Protégé are maintained, because they provide different features. Protégé 4 supports OWL 2.0[6], therefore it was chosen as a target for Knoocks plug-in. The plug-in's compatibility was tested against Protégé 4.1.

[5] http://www.osgi.org/Specifications/HomePage
[6] http://www.w3.org/TR/owl2-overview/

4.1 Architecture

Knoocks visualization plug-in is implemented in Java 6 SE using mainly Swing and Java 2D. The plug-in is distributed as an OSGi bundle and depends on Protégé, Protégé-OWL[7] and OWL API[8] bundles. Protégé-OWL classes are more convenient to use for ontology inspection than using OWL API directly. Furthermore, the plug-in utilizes Commons-Collections with Generics[9] library which is a Java 1.5 port of popular Commons Collections project. Architecture of the plug-in is shown in Figure 6 using UML (*Unified Modeling Language*).

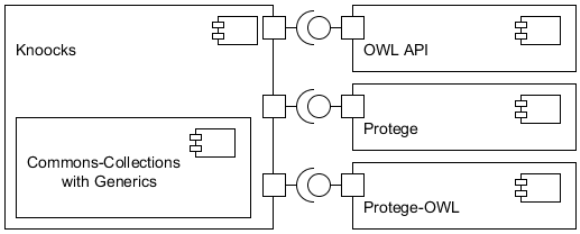


Figure 6: Architecture of Knoocks plug-in (*Component Diagram*, UML).

4.2 Design

Knoocks ontology visualization is interactive and driven by user events. Therefore, the plug-in's code was designed using the MVC (*Model-View-Controller*) pattern[10]. The UI (*User Interface*) part of plug-in's code is tightly coupled with Swing which implements its own modified MVC. In the process of development, the IoC (*Inversion of Control*) principle[11] was identified to be beneficial to use. It was decided that the plug-in is too small to utilize an IoC framework, so that a simple mechanism of handling dependcies was designed and implemented.

4.2.1 Model

The OWL API already models entities of an OWL ontology, but there were reasons to design an overlaying model. The Knoocks approach contains blocks and connections that are not included in the OWL API. Blocks are hierarchies of classes that have to be inspected and remembered, because users are allowed to fold/unfold classes which requires a knowledge of class' hierarchy. Connections group relations among instances by their class path. It is beneficial to extend the new model with these specialized entities. All other entities from OWL API, namely class, individual, data property and object property, have

[7] http://protege.stanford.edu/plugins/owl/api/
[8] http://owlapi.sourceforge.net/
[9] https://github.com/megamattron/collections-generic
[10] http://ootips.org/mvc-pattern.html
[11] http://www.oodesign.com/dependency-inversion-principle.html

their counterparts in plug-in's model. Entities from OWL API model do not allow to get their names directly (using entity's method), OWL entity rendering must be used instead. Knoocks uses same name rendering throughout the visualization, therefore it was easy to include methods that provide entity names in our model. In addition, a few properties regarding visualization were added to some entities, e.g., the new class entitiy has a property named *highlighted* which tells the view whether user wants that class to be rendered as highlighted or not.

4.2.2 Controller

Controllers are classes where logic is contained in MVC applications. In Knoocks plug-in, controllers take care of switching view between overview and detail view, provide searching and navigation and modify the model that is used by views. Every action that is initiated by a user through the view part of Knoocks plug-in is executed by some controller. Controllers are cooperating together to improve reuse of the code and distribution of logic among appropriate classes. Each block in the plug-in has its own controller object which is instanced from appropriate class.

4.2.3 View

Knoocks' views are implemented using Swing and Java 2D for custom drawing. Each plug-in's view is a Swing component. The top level view contains control view as a subview and defines a slot that allows switching between overview and detail view (see Figure 1). The definition of subviews is supported by Swing's laying out of child components. Four new UI components were developed for visualization of blocks. Two components represent whole blocks (in detail view and in overview) while another two components take care of class rendering in appropriate views. Custom layout manager (feature of Swing UI framework) was developed for supporting of automatic layout of classes in a block. The layout manager takes into account folded or unfolded state of clasess in a block.

5 Usage example

Google and Yahoo SearchMonkey support crawling of e-commerce data enriched with concepts from GoodRelations vocabulary [2]. The GoodRelations semantic vocabulary (in OWL 2.0) allows to add business information into web pages. Semantics is marked up using RDFa (*Resource Description Framework in attributes*) or Microformat specifications. Semantically enriched data is machine understandable, therefore it is not necessary to implement specialized e-commerce APIs (such as Google's Content API) to make offered products reachable by search engines. Although the GoodRelations ontology has quite shallow hiearchy it is an example of OWL ontology that is widely used.

Protégé with Knoocks plug-in can be used to quickly become familiar with the ontology. The overview (shown in Figure 7) gives view on general structure of a part of the ontology. Chosen part visualizes all classes that are available for product description. Block arrangement of class hierarchies points out existence of subclasses – specializations of basic concepts. Classes painted with different shades make distribution of ontology's individuals visible at first glance. Definitions of object properties are rendered instead of connections when the view is set to draw arrows between domains and ranges. Rendering of properties is controlled by a button at Knoocks' toolbar (see Figure 8).

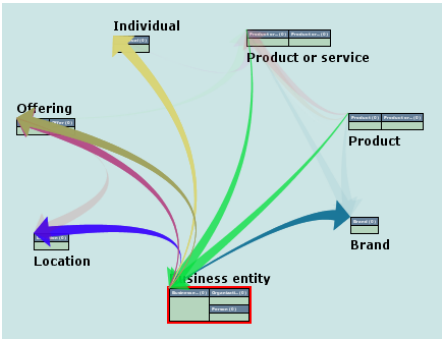


Figure 7: Object properties related to `Business entity` class.

Figure 8: Toolbar.

Detail view can be used to inspect a block. Figure 9 shows detail view visualization of `Payment method` class from the GoodRelations ontology. Names of individuals are drawn in class bodies regarding their classification. Information about data and object properties related to a class will be shown when clicking its header. Furthermore, object property assertions between individuals from `Day of week` class are not visualized in overview (as connections) because the domain and the range of both `has previous` and `has next` properties are same. All assertions related to an individual are shown when the individual is clicked.

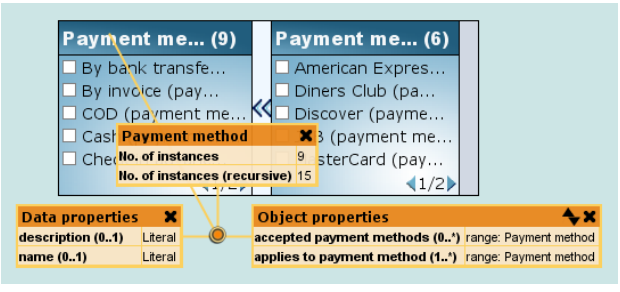


Figure 9: Visualization of `Payment method` block. The block is shown in detail view.

6 Distribution

Plug-ins for Protégé are distributed as OSGi bundles, i.e., JAR archives that provide OSGi metadata. A plug-in is installed by copying it into *plugins* directory that is located at root of Protégé's distribution. Plug-ins are automatically loaded when Protégé-OWL is started.

```
ProtegeInstallDir
└─plugins
  └─cz.muni.fi.knoocks.jar
```

At the time of writing, Knoocks plug-in is in process of publishing at Protégé's *Plugin Library*[12]. The library is considered as a standard distribution platform for Protégé plug-ins. Our plug-in will be labeled with *Visualization* category and annotated with compatibility and dependency information. Each plug-in has its wiki page that can be used by authors to provide additional information such as basic *User's Guide* or link to repository with source code.

7 Conclusion

In this paper, we described Knoocks visualization plug-in for Protégé-OWL editor. We commented briefly on Knoocks visualization approach which combines both node-link and space-filling approaches to visualize general structure of an ontology and also its details, i.e., individuals with linked values and object property assertions. By included example, we showed that our plug-in is really helpful in visualizing new ontologies that a user wants to become familiar with.
Main part of the plug-in was implemented in my master's thesis. Currently, the resulting plug-in is being made publicly available at Protégé's plug-in library, so that wide community of Protégé-OWL users will get a new visualization tool. It is expected that the use of the plug-in will lead to new requirements on Knoocks visualization approach.

References

[1] G. Antoniou and F.V. Harmelen. *A semantic Web primer*. Cooperative information systems. MIT Press, 2004.

[2] M. Hepp. GoodRelations : An Ontology for Describing Products and Services Offers on the Web. In *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW2008)*, volume 5268, pages 332–347, 2008.

[3] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. Ontology visualization methods—a survey. *ACM Computing Surveys*, 39(4):10–es, November 2007.

[4] S. Kriglstein. Analysis of Ontology Visualization Techniques for Modular Curricula. In *Proceedings of 4th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society on HCI and Usability for Education and Work*, pages 299–312, 2008.

[5] S. Kriglstein. User Requirements Analysis on Ontology Visualization. In *2009 International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 694–699. IEEE Comput. Soc. Press, March 2009.

[6] S. Kriglstein. OWL Ontology Visualization: Graphical Representations of Properties on the Instance Level. In *2010 14th International Conference Information Visualisation*, pages 92–97. IEEE Comput. Soc. Press, 2010.

[7] S. Kriglstein and R. Motschnig-Pitrik. Knoocks: New Visualization Approach for Ontologies. In *Proceedings of 12th International Conference Information Visualisation*, pages 163–168. IEEE Comput. Soc. Press, 2008.

[8] S. Kriglstein and G. Wallner. Knoocks - A Visualization Approach for OWL Lite Ontologies. *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 950–955, February 2010.

[9] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. *Visual Languages, 1996. Proceedings., IEEE*, pages 336–343, 1996.

[10] MA. Storey, N.F. Noy, M. Musen, C. Best, R. Fergerson, and N. Ernst. Jambalaya: An Interactive Environment for Exploring Ontologies. In *IUI '02 Proceedings of the 7th international conference on Intelligent user interfaces*, page 239, 2002.

[11] T. D. Wang and B. Parsia. CropCircles : Topology Sensitive Visualization of OWL Class Hierarchies. In *Proceedings of the 5th International Semantic Web Conference ISWC 2006*, pages 695–708, 2006.

[12] http://protegewiki.stanford.edu/wiki/Protege_Plugin_Library