Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé

Margaret-Anne Storey[1,4] Mark Musen[2] John Silva[3]
Casey Best[1] Neil Ernst[1] Ray Fergerson[2] Natasha Noy[2]

Abstract

This paper describes the integration of an interactive visualization user interface with a knowledge management tool called Protégé. Protégé is a general-purpose tool that allows domain experts to build knowledge-based systems by creating and modifying reusable ontologies and problem-solving methods, and by instantiating ontologies to construct knowledge bases. The SHriMP (Simple Hierarchical Multi-Perspective) visualization technique was designed to enhance how people browse, explore and interact with complex information spaces. Although SHriMP is information independent, its primary use to date has been for visualizing and documenting software programs. The paper describes how we have applied software visualization techniques to more general knowledge domains. It is hoped that the integrated environment (called Jambalaya) will result in an easier to use and more powerful environment to support ontology evolution and knowledge acquisition. An example scenario of how Jambalaya can be applied to knowledge acquisition is provided.

1 Introduction

This paper describes how we have integrated an interactive visualization user interface with a knowledge management tool called Protégé. The Protégé environment has been developed at Stanford University over the past 16 years [1, 2, 3]. It supports the modeling of ontologies and use of ontologies to guide acquisition of content knowledge from subject-matter experts. Moreover it allows developers to easily "plug-in" components to add new functionalities to the Protégé tool. Protégé is being actively used by hundreds of users world-wide in many knowledge domains.

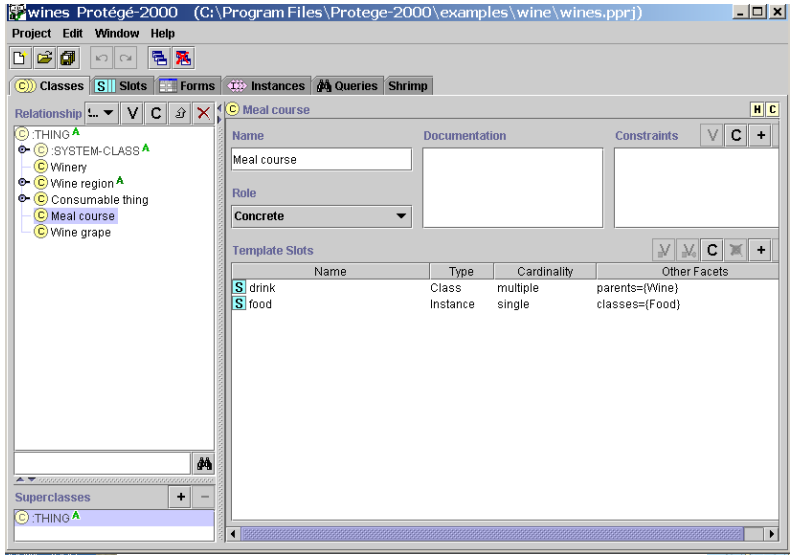

Figure 1: The ontology editing tab in Protege

An ontology defines a common vocabulary and structure of an information space for researchers and domain experts to exchange and share knowledge. A domain expert defines classes to represent concepts in a domain of discourse, with slots to represent properties and relationships between the concepts. An ontology together with a set of instances constitutes a knowledge base. A class can have subclasses to represent more specific concepts.

Figure 1 shows the ontology editing environment in Protégé. The class hierarchy is shown on the left side of the pane, with the form for editing properties of the class depicted on the right hand side. The ontology example used throughout this paper is a food and wine example as described in [4]. In Fig. 1, the class highlighted on the left pane is the "Meal Course" class. The form on the right shows the properties (slots) depicting relationships between this class and instances of this class type

[1] Department of Computer Science, University of Victoria.
[2.] Stanford Medical Informatics, Stanford University.
[3.] National Cancer Institute.
[4.] Department of Aeronautics and Astronautics, MIT.
Email contacts: mstorey@csr.uvic.ca; musen@smi.stanford.edu

have with other instances and classes in the knowledge base.

Figure 2 shows the instances tab in Protégé. In this tab, the class hierarchy along with the instances of a selected class are shown in the two leftmost columns. The column on the right side contains the form for that instance revealing the values for the slots of the instance's class. Here we see an instance of the "Meal Course" class called "Grilled Chicken". The "Food" and "Drink" slots (defined by the instantiating class as shown in Fig. 1) have values corresponding to "Grilled Chicken" and "Beaujolais" respectively. Instance forms are used to enable knowledge acquisition. Forms can be customized by the developer when classes are defined.

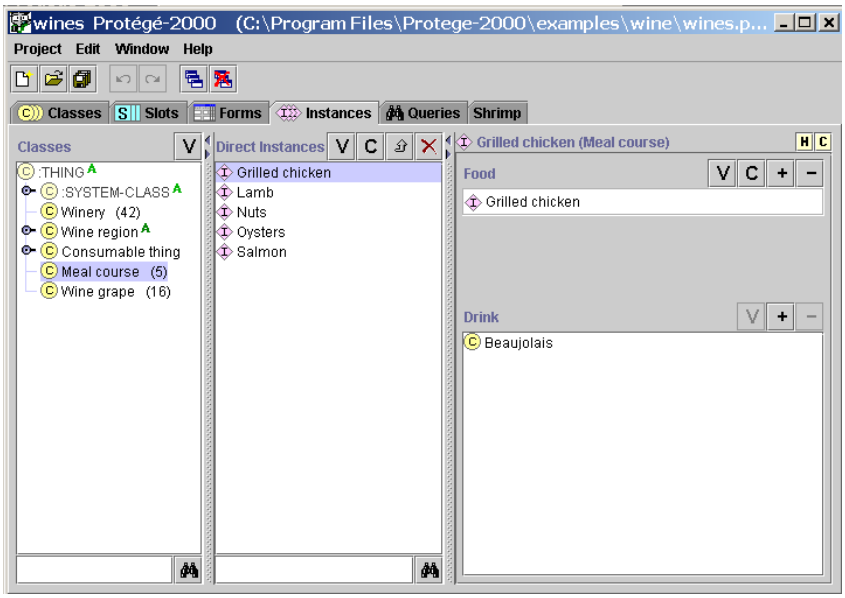

Figure 2: This figure shows the instance tab in Protégé.

Understanding and maintaining the structure of large knowledge bases has been a problem since the days of MYCIN [5]. Many knowledge bases contain very complex structures that need to be viewed at various levels of abstraction and in different contexts to be truly understood. Indeed, the underlying model of any knowledge base is just one interpretation of the knowledge to be modeled and may not reflect another individual's or organization's perception of the information space. Consequently, knowledge acquisition can be a cognitively challenging task as the user has to have a clear understanding of the ontology as well as manage detailed information.

Several Protégé users (most recently at [6]) expressed the need for tools that provide visualizations of the coarser grained structures in the knowledge base as well as provide mechanisms to more easily and more efficiently navigate the information. Some knowledge base tools (including Protégé) do provide some primitive or restrictive visualization capabilities. However, they tend to be either domain or task specific views that are either difficult to customize or do not scale to large, complex knowledge bases. An early visualization tool called SemNET [7], although promising, was not adopted by the general community. SemNET, a proprietary tool, required (at that time) fairly advanced hardware to produce interactive views.

Some of the problems reported to us by Protégé users include disorientation as they navigate through large, complex ontologies. Unwieldy screen clutter often occurs due to many separate window frames being opened. These kinds of difficulties are also shared by users of other tools that deal with large amounts of information. In this paper we describe and discuss the integration of a visualization technique, called SHriMP Views, using Protégé's plug-in facility. SHriMP's primary benefits include generic visualization techniques and navigation facilities to make better use of people's cognitive abilities when using a screen of fixed size.

The rest of the paper is organized as follows. Section 2 provides some brief background on the SHriMP visualization technique. Section 3 describes how we have integrated SHriMP with Protégé, we refer to this integration as Jambalaya. A scenario is used to help describe how Jambalaya can be used in ontology authoring and knowledge acquisition in Section 4. Section 5 discusses the visualization capabilities provided by this integration and describes our plans for future work.

2 SHriMP Views

The SHriMP (Simple Hierarchical Multi-Perspective) visualization technique was designed to enhance how people browse, explore, model and interact with complex information spaces [8]. SHriMP uses a nested graph view to present information that is hierarchically structured. It introduces the concept of *nested interchangeable views* to allow a user to explore multiple perspectives of information at different levels of abstraction. SHriMP combines a hypertext following metaphor with animated panning and zooming motions over the nested graph to provide continuous orientation and contextual cues for the user. These features result in an environment where the user can interact directly with the information space enhancing their understanding of the information structures, thus promoting further exploration.

The SHriMP visualization technique was originally designed to enhance how programmers understand programs [9]. Within the context of software visualization, SHriMP presents an interactive nested graph view of a software architecture. Program source code and documentation are presented by embedding marked up text fragments within the nodes of the nested graph. Finer connections among these fragments are represented by a network that is navigated using a hypertext link-following metaphor.

SHriMP employs a fully zoomable user interface for exploring information. This interface supports three zooming approaches: *geometric*, *semantic* and *fisheye* zooming [8]. A user browsing an information space may combine these approaches to magnify nodes of interest. Geometric zooming is the simplest type of zooming. A part of the nested view is simply scaled around a specific point in the view. Geometric zooming causes other information to be hidden from view. Fisheye zooming allows the user to zoom on a particular piece of the information, while preserving contextual information. Information that is of interest appears larger than other information which is reduced in size accordingly.

SHriMP also provides a semantic zooming method. When magnified, a selected node will display a particular view depending on the task at hand. For example, when using SHriMP to visualize a Java program, zooming on a node representing a Java package will cause the node to display its children (packages, classes, and interfaces). Alternatively, the zooming action may reveal a Javadoc file, if it exists.

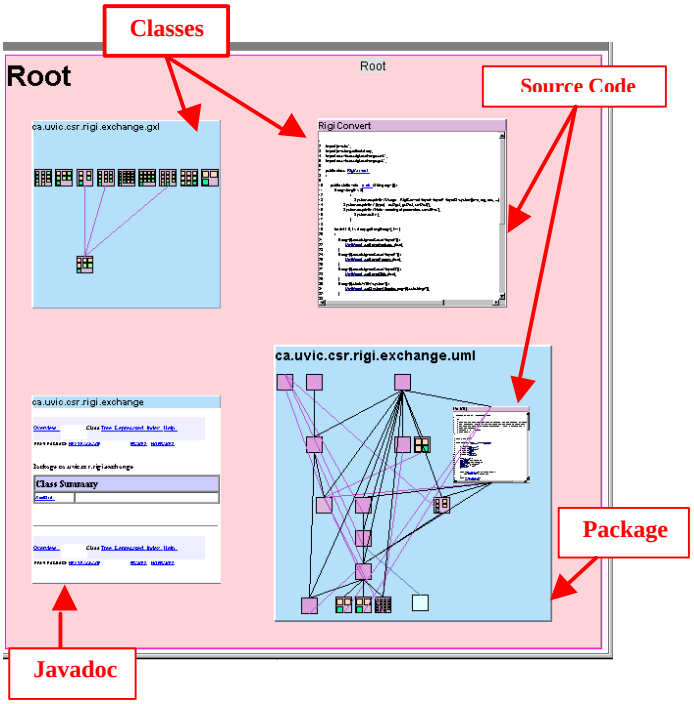

Figure 3: Visualizing a Java program using SHriMP.

Other possible views may include annotation information, code editors or other graphical displays. A node representing a class or interface may display its children (attributes and operations) or it may display the corresponding source code. SHriMP determines which view to show according to the action that initiated the zoom action. For example, if a user clicks on a link within a source code view, SHriMP will zoom to the appropriate destination node and display the source code within that node. Figure 3 shows how SHriMP's interchangeable nested views allow different views for the Java domain.

Although SHriMP has been primarily used for software visualization to date, the SHriMP tool is language independent and can be used for browsing and editing any information space. In the next section of this paper we describe how SHriMP has been recently integrated with the Protégé tool.

3 Jambalaya: Integrating SHriMP with Protégé

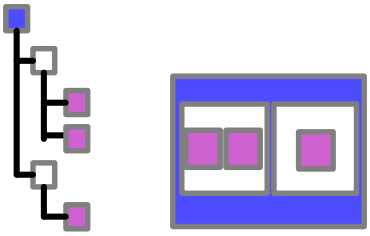

Figure 4: This figure shows two alternatives for displaying a hierarchy. The view on the left shows the kind of tree view typically shown in Protégé to display class hierarchies. The view on the right is the nested view featured in SHriMP.

In this section, we briefly describe how we have integrated the SHriMP plug-in with Protégé. Further technical details are available on the website at [10].

A nested directed graph is used to display the knowledge base visually. Classes and instances in Protégé are represented as nodes in a graph. Nested nodes (see Fig. 4) are used to depict subclass relationships between classes. That is, subclasses are nested (drawn) inside their superclass node.

Instances are also represented by nodes in the graph, with the instance-of relationship represented by nesting instance nodes within the corresponding class node. Instance nodes and class nodes are distinguished from each other using different colours.

Slot dependencies between classes and instances in the knowledge base are represented in the graph using directed arcs

between nodes. For example, when an instance has a slot value referring to another instance, an arc will be drawn between those two instances. Template slots relating classes are also represented using directed arcs between classes. A variety of colours are used to differentiate between different slot types.

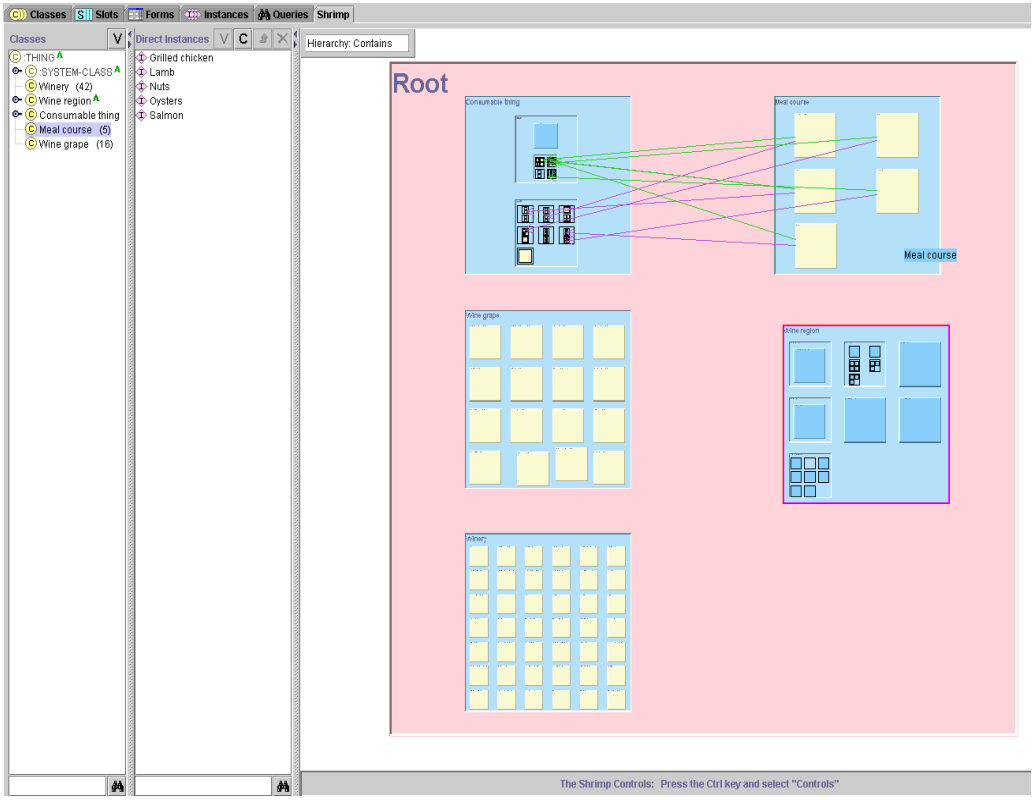

Figure 5 shows the Jambalaya tab in Protégé.

A user can navigate the knowledge base using the Jambalaya tab in several ways. When a user selects either a class or instance in the leftmost columns, the SHriMP view zooms (using an animated zooming technique [11]) to the selected class or instance node. Alternatively, a user can interact directly with the SHriMP view by zooming in and out of nodes in the nested graph. Forms for both instance and class nodes can be embedded directly inside a node. The user can interact with these forms (i.e. edit) in the same fashion as is available in Protégé. In addition, the user can zoom in (bring into focus) other class and instance nodes by double-clicking on the slot values in the forms. Previously in Protégé, double-clicking on a slot value in a form referring to an instance or class would open a separate window containing the class or instance form. Such actions invariably result in complex displays. The SHriMP view contains browser-like buttons to go "back", "forward" and to "search" for nodes. These features greatly assist in navigation.

In the first view in Fig. 6 (top left), the user selects the "Meal Course" node to focus on it. This node is enlarged using the fisheye view mechanism in SHriMP in the top right view. In the bottom left view, one of the children of "Meal Course" is enlarged (the "Lamb" instance). The "Lamb" instance node, when enlarged, reveals its form by default. In the final view (shown bottom right), the "Meal Course" children (i.e. its instances) are hidden and instead the form for this class is shown.

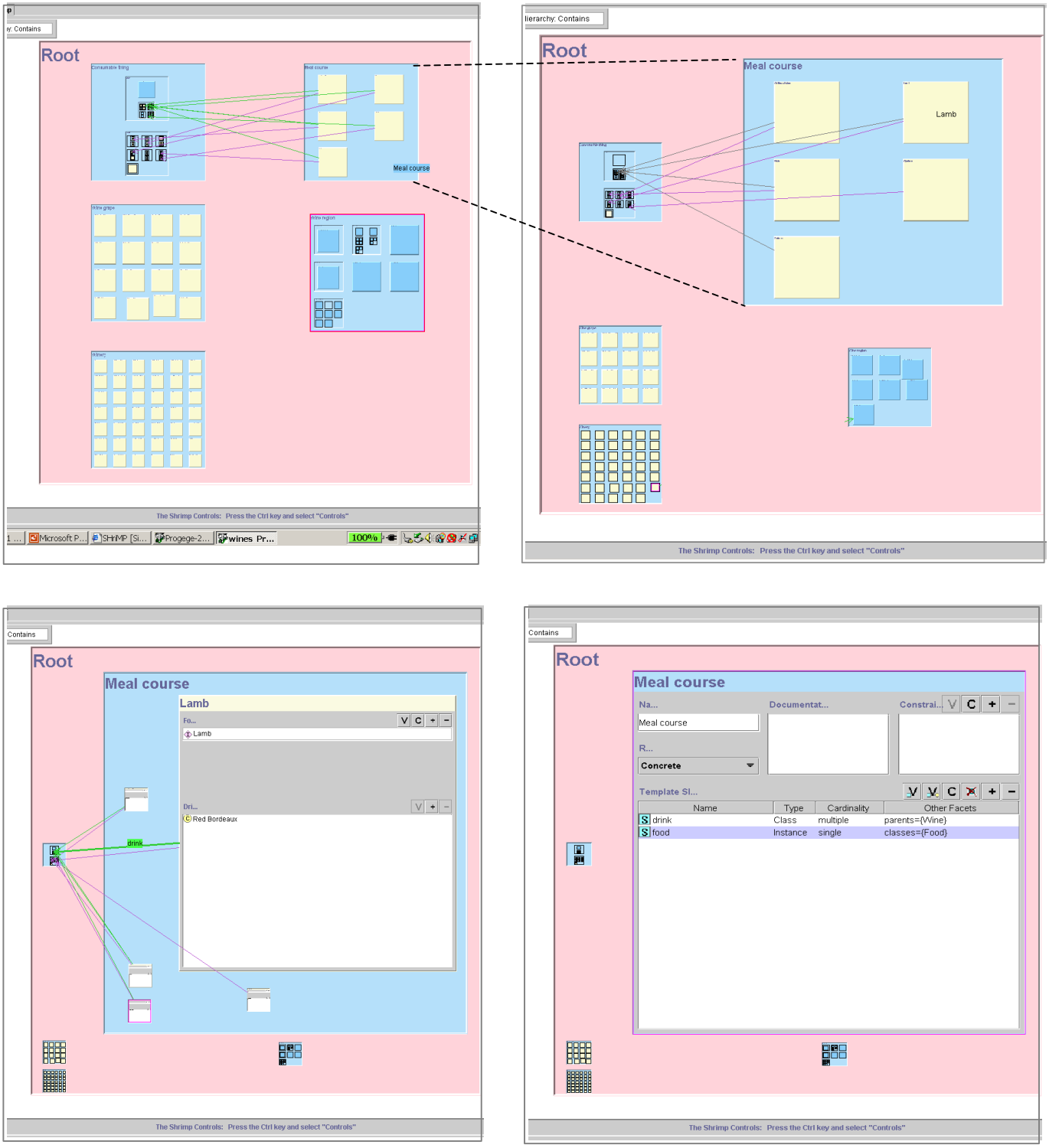

Figure 6: In the SHriMP view shown here, classes are coloured blue, instances using pale yellow. Only the slots (arcs) that are relevant to the "Meal Course class are visible. This figure demonstrates how the user can navigate among the interchangeable nested views available.

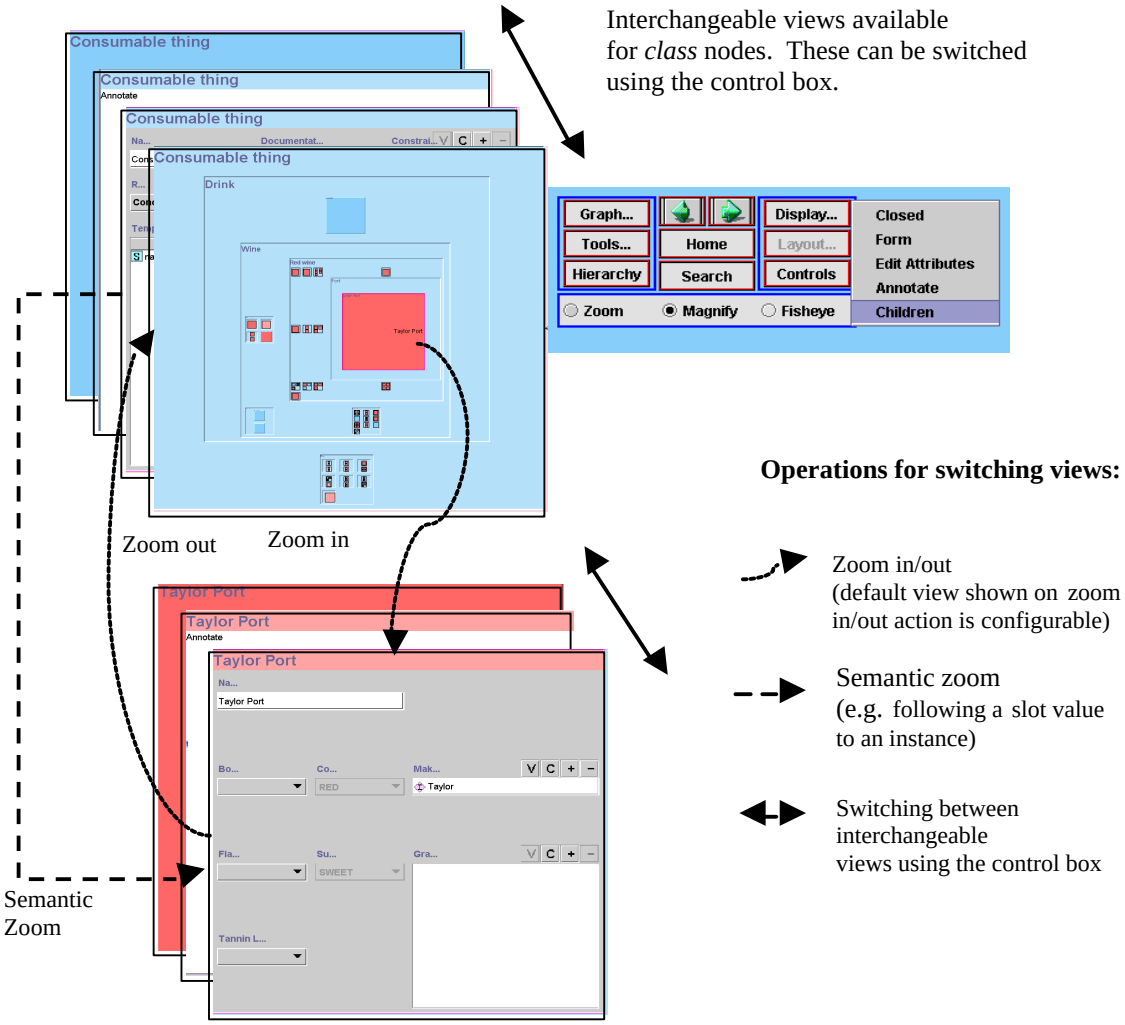

Figure 7: Nested InterchangeableViews in Jambalaya

Figure 7 summarizes how the nested interchangeable view concept is applied within Jambalaya. For class nodes, potential views include:

- Closed -- no children or further information is shown;
- Annotation View – in this view the user can document or annotate the class using a simple text editor;
- Form View – in this view the user can browse and edit the form that describes the class node;
- Children View -- there may be two possibilities
  - if there are subclasses of the class, these may be nested inside this node
  - if there are instances of the class, these may be nested inside this node (note: this option is not shown in Fig. 7 as there are no instances of any non-leaf classes in the food and wine knowledge base).

Instance nodes have similar views except that they do not have the "Children View". Switching between views is facilitated by the Control box (shown in Fig. 7). Also, when a node is focused on (i.e. enlarged through a zooming action), the default view for the node is automatically displayed inside the node. The default view for a particular type of node (i.e. class or instance node) is user-configurable.

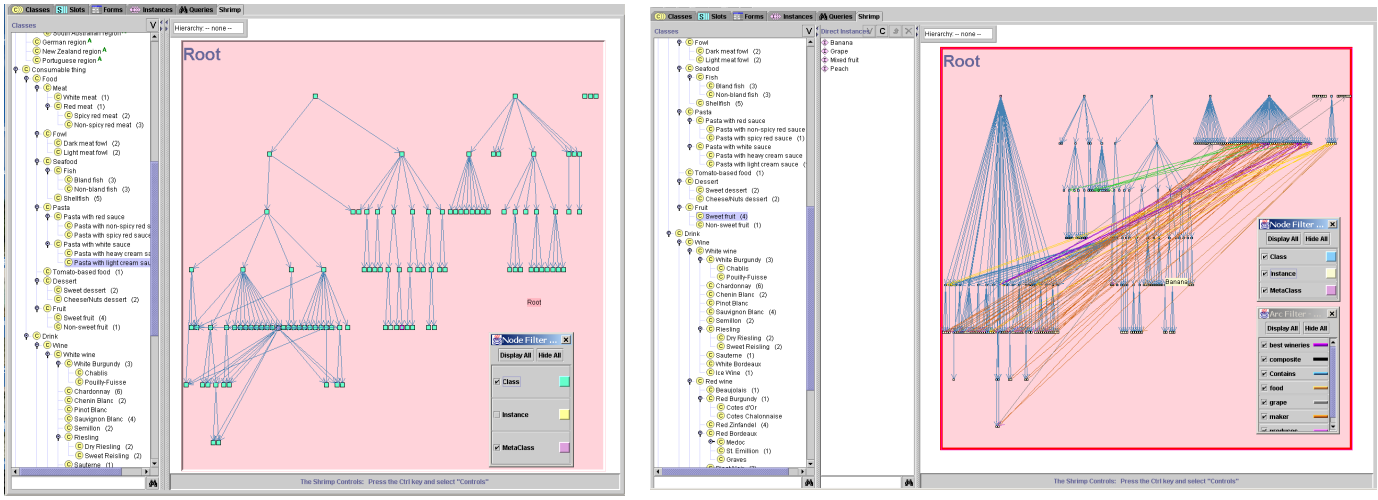

Figure 8: In the left-hand view, the user has filtered the instance nodes and issued a tree layout of the class nodes to show the class hierarchy. In the right-hand view, a tree layout of both the instance and class nodes shows the class hierarchy as well as shows which classes have been directly or indirectly instantiated. Slots relating instances are also displayed to reveal dependencies between instances.

4 Using Jambalaya during Knowledge Acquisition

In this section of the paper we briefly discuss how Jambalaya may be useful for some of the tasks required during ontology authoring and knowledge acquisition. We also consider how the provided visualizations may address some of the typical concerns faced by a user when adding new knowledge to an existing knowledge base.

A knowledge base consists of an ontology and a set of instances. According to Noy and McGuinness [4], developing a knowledge base from scratch includes the following steps:

- Define the classes in the ontology (i.e. identify the main concepts in the ontology)
- Arrange the classes in a subclass-superclass hierarchy
- Define the slots (properties) for the classes and define the allowable values for the slots, as well as define any restrictions on the slots
- Add instances and fill in the values for slots for the instances

Extending or maintaining an existing knowledge base may involve these additional steps:

- Browse and understand the existing class hierarchy in the ontology
- Browse and understand at least some of the instances, slot values and restrictions in the knowledge base
- Restructure the ontology and instances

Blythe *et al.* [12] further identified some typical concerns that users may have when adding new knowledge to an intelligent system. Some of these concerns are as follows:

- User do not know where to start and where to go next
- Users do not know if they are adding the right things
- Users often get lost as it takes several steps to add new knowledge

We consider some of these tasks and concerns from the perspective of a user maintaining the wine and food knowledge base. In Fig. 8, the leftmost column in the first view indicates that the class hierarchy is quite complex. However, it is difficult to get an overall impression of the width and depth of the class hierarchy from the traditional Protégé tree view. Furthermore, it is difficult to ascertain if multiple inheritance is used in the class hierarchy. The SHriMP view in this snapshot attempts to provide additional insight on the class hierarchy. The user has filtered (temporarily hidden) the instance nodes and slot dependencies between classes. Furthermore, instead of nesting subclasses within their superclasses, the user has decided to use directed arcs to reflect the subclass relationship (therefore nesting of nodes is optional). Finally the user issues a tree layout algorithm on the class nodes and the subclass arcs. The resulting view provides the user with a snapshot view of the width and depth of the hierarchy revealing that it is six levels deep and that there is some multiple inheritance in the class hierarchy.

The view on the right shows both the class nodes (again with no nesting) and the instance nodes, with slot relationships between instances turned on (not filtered). This snapshot gives a quick and approximate overview of the number of instances in the knowledge base as well as reflects the general nature of the slot dependencies between instances. The user can "brush" (move their mouse over) individual arcs to see which slot types dominate the dependencies between instances. As the mouse moves over an arc, a small tooltip window briefly displays the name of the slot. As the user brushes over a node, the name of the node is also shown in a tooltip. Incoming and outgoing arcs are also highlighted when the user selects a node.

By way of another example, Fig. 9 shows a diabetes clinical trial knowledge base (this knowledge base was created by Samson Tu at Stanford University). In this SHriMP view, classes are nested inside their superclasses with instances nested in their instantiating classes. All of the classes and instances in the knowledge base are visible (although some of the nodes are very small and can only be viewed when the user zooms in). Furthermore all of the slot dependencies are also shown in this view. An area of considerable activity is visible in the top right hand corner. Such an area of congestion may indicate to a user unfamiliar with a knowledge base that this may be a central concept or set of instances in the knowledge base, indicating that further investigation by the user is warranted.

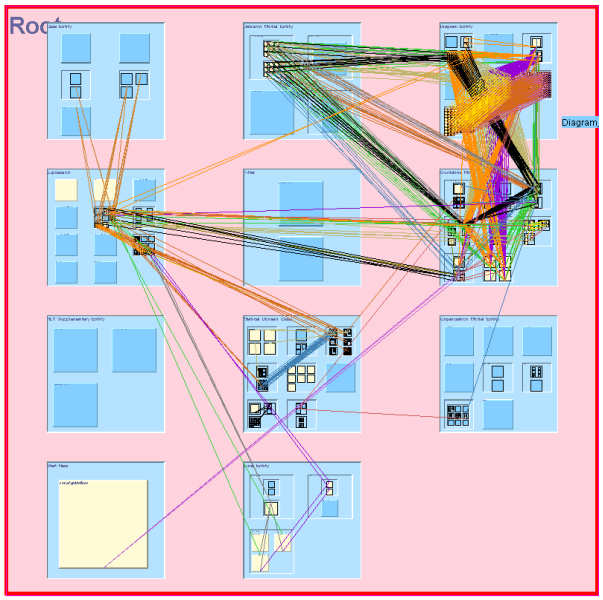

Figure 8: This figure shows a diabetes clinical trial knowledge base. Heavy dependencies between instances in this knowledge base can be easily seen in the top right hand corner

We are hoping that interacting with graphical views of the ontology and instances will help a user understand the important parts of the knowledge base and highlight parts of the knowledge base that could be a good starting point during knowledge acquisition. Furthermore the combination of the traditional views in Protégé with the graphical views provided by SHriMP should provide useful navigation assistance as a user browses a large knowledge base.

Although not implemented at this time, we also intend to add features to allow a knowledge engineer to author ontologies directly in the SHriMP view. That is, the user should be able to visually define classes and instances in the ontology by adding new nodes to the graph. Slot dependencies between classes and instances could likewise be added by drawing lines between nodes. Such a tool could perhaps mimic how humans often sketch box and line diagrams on paper to model information in an *ad hoc* fashion. Such features could potentially facilitate and enhance the brainstorming and creative processes during knowledge modeling. Furthermore a user could restructure a class hierarchy by dragging nodes from one superclass to another class node within the nested graph view of a knowledge base. However since we have not done any user testing or implemented all of these features, we can only speculate on their usefulness. The SHAKEN system discusses using graphical components to facilitate knowledge acquisition, and early results from their users indicate that this technique for KA is desirable. The next and final section of this paper further discusses the application of SHriMP to knowledge acquisition and describes our next steps in this research.

5 Discussion and Future Work

In this paper, we have described the integration of the SHriMP visualization tool with the Protégé tool. SHriMP has been used primarily for software visualization. In this section we explore if the techniques used in software visualization are applicable to more general knowledge domains. Although software engineering is a subset of knowledge engineering, there are clearly some similarities and differences between the two. Some similarities include:

- Software engineers have been shown to use different authoring and comprehension strategies when writing or maintaining software. This is also the case for knowledge engineers when authoring and maintaining ontologies. For both domains, the visualizations have to customizable and extensible by the end user.
- For both applications, the visualizations made available need to be task based, i.e. the benefit of the visualization can only be realized if they support a particular task during the evolution of the software or knowledge base. Substantial user feedback is required to test if the proposed visualizations do provide support for a set of identified tasks.

- Both domains face substantial challenges with respect to the scale and complexity of the information to be modeled. Mechanisms to abstract and filter information are required for both applications.
- Navigation and management of multiple views, bridging different levels of abstraction are likewise needed in both domains.

Despite the many similarities, there are also some differences that should be mentioned:

- Metadata and instance data are typically visually separated in software visualization tools. Often the software schema is hidden from the user and is implicit in the information presented. Therefore, SHriMP's ability to distinguish between the ontology and the instances is currently inadequate. Our future work includes extending SHriMP to address this shortcoming.
- Some of the ontologies that we have explored using SHriMP are very complex (more so than the typical schemas used by software tools). Therefore, more sophisticated visualization mechanisms will be required.

We would also like to further explore which parts of the lifecycle of a knowledge base benefit from these techniques. Are they most beneficial for knowledge engineers trying to understand an unfamiliar knowledge base, or could they be helpful for a knowledge engineer modeling a new knowledge base? Would the visualizations provided in SHriMP help detect inconsistent or ambiguous features in the knowledge base? These are just some of the questions we would like to explore in our future research.

Finally we need to get feedback from Protégé users in various domains to learn if these visualizations are indeed useful for their knowledge engineering tasks.

Acknowledgements

This research was supported in part by CSER, NSERC, IBM, Stanford University, the University of Victoria, the University of Texas at Austin and the Space and Naval Warfare Systems Center San Diego. The content of the information does not necessarily reflect the position or the policy of any of the universities nor the US or Canadian government and no official endorsement should be inferred. We would like to thank Samson Tu from Stanford for providing the diabetes clinical trial knowledge base shown in this paper. Finally, we would like to thank the anonymous reviewer for the careful review and very useful comments on our paper.

References

1. Protégé project, Stanford University, http://protege.stanford.edu.
2. Musen, M.A., Fergerson, R.W., Grosso, W.E., Noy, N.F., Crubézy, M., and Gennari, J.H. Component-based support for building knowledge-acquisition systems. *Proceedings of the Conference on Intelligent Information Processing* (IIP 2000) of the International Federation for Information Processing Sixteenth World Computer Congress (WCC 2000), Beijing, China, August, 2000, pp. 18–22.
3. Noy, F.N., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., and Musen, M.A. Creating Semantic Web contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.
4. Noy, N.F. and McGuiness D.L., Ontology Development 101: A Guide to Creating Your First Ontology by Noy, N.F. and McGuinness, D.L. *SMI technical report* SMI-2001-0880 (2001), Stanford University.
5. Buchanan, B.G., and Shortliffe, E.H. Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Reading, MA: Addison-Wesley, 1984.
6. The Fifth International Protégé Workshop, Newcastle upon Tyne, July 16-18, 2001, http://www.schin.ncl.ac.uk/protege2001/
7. Fairchild, K., Poltrok, S., and Furnas, G. Semnet: Three dimensional graphic representations of large knowledge bases. In *Cognitive Science and its Applications for Human-Computer Interaction*, R. Guindon, Ed. Lawrence Erlbaum, 1988.
8. Storey M.-A., Wong K., Fracchia F., and Müller H., On Integrating Visualization Techniques for Effective Software Exploration. In *Proc. of the InfoVis'1997*, pages 38-45, Phoenix, USA, 1997.
9. Storey M.-A., Müller H. and Wong K., Manipulating and Documenting Software Structures. In *Software Visualization*, pages 244-263. World Scientific Publishing Co., 1996.
10. SHriMP Views project, University of Victoria, http://www.csr.uvic.ca/shrimpviews/protege.htm
11. The Jazz Zoomable User Interface, University of Maryland, http://www.cs.umd.edu/hcil/jazz/
12. Blythe J., Kim J., Ramacahandran S. and Gil Y., An integrated environment for knowledge acquisition, Best Paper, *International Conference on Intelligent User Interfaces*, 2001.
13. P. Clark, J. Thompson, K. Barker, B. Porter, V. Chaudhri, A. Rodriguez, J. Thomere, S. Mishra, Y. Gil, P. Hayes, T. Reichherzer. Knowledge Entry as the Graphical Assembly of Components. To appear in *Proc 1st Int Conf on Knowledge Capture (K-Cap'01)*, 2001.