Towards Cognitive Support in Knowledge Engineering: An Adoption-Centred Customization Framework for Visual Interfaces

by

Neil A. Ernst
B.Sc., University of Victoria, 2001

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

Supervisor: Dr. Margaret-Anne Storey

ABSTRACT

Constructing large knowledge models is a cognitively challenging process. In order to assist people working with these models and tools, this thesis proposes considering the tools used in light of the cognitive support they provide. Cognitive support describes those elements of a tool which aid human reasoning and understanding.

This thesis examines the use of advanced visual interfaces to support modelers, and compare some existing solutions to identify commonalities. For many problems, however, I found that such commonalities do not exist, and consequently, tools fail to be adopted because they do not address user needs. To address this, I propose and implement a customizable visualization framework (CVF) which allows domain experts to tailor a tool to their needs. Preliminary validation of this result revealed that while this approach has some promise for future cognitive support tools in this area, more work is needed analyzing tasks and requirements for working with large knowledge models.

Examiners:

Dr. Margaret-Anne Storey (Department of Computer Science)

Dr. Hausi Müller, Department Member (Department of Computer Science)

Dr. Yvonne Coady, Department Member (Department of Computer Science)

Dr. Francis Lau, External Examiner (School of Health Information Science)

Table of Contents

List of Tables

List of Figures

Acknowledgement

To my supervisor, Dr. Peggy Storey, for wisdom and patience and encouragement.

To my committee members for their helpful suggestions and interest.

Thanks to Ian Bull and Elizabeth Hargreaves for their suggestions and comments which strengthened this work. Thanks also to my fellow lab members, who have been instrumental in guiding this research, particularly Mechthild Maczewski, for seeing the big picture. Thanks to Rob Lintern for his help and programming expertise.

Funding for this research was provided by an NSERC scholarship and a grant from the U.S. National Cancer Institute Centre for Bioinformatics. I wish to thank each of these funding bodies for their support.

I wish also to acknowledge the support, financial, moral, and otherwise, from the Protégé team at Stanford, including Dr. Mark Musen, Dr. Natalya Noy, and Dr. Ray Fergerson. The Protégé resource is supported, in part, by grant P41 LM007885 from the National Library of Medicine.

Portions of this thesis (chapter 3) appear in [28].

Dedication

To Kambria, for everything.

Chapter 1 – Introduction

Three of the many benefits which computer science (particularly Internet-based computer technology) can provide to society are new ways of addressing the themes of *discovery*, *understanding*, and *communication*.

Discovery, in the sense of exploring and uncovering new truths and explanations for any number of questions;

Understanding, in the sense of leveraging existing knowledge and using this new knowledge to do new and different things, and

Communication, in the sense of dealing with other human beings and non-human intelligent ‘agents’;

Two research streams which address these broad themes are knowledge engineering and cognitive science. This thesis examines these streams with respect to supporting the understanding of complex knowledge models, and applies software customization to improve the cognitive support for this mentally demanding task. The combination of cognitive science and knowledge engineering is a powerful mechanism for empowering people to find new things to discover and attempt to understand, and to find new ways to help people communicate and collaborate.

1.1 Knowledge engineering

Knowledge engineering is an area of research that is increasingly relevant today. Typically associated with the heady days of Artificial Intelligence (AI) research (culminating in the mid-1980s), and similarly part of the AI ‘winter’ that ensued, knowledge engineering nonetheless remains highly relevant to many ordinary users. The term ‘knowledge engineering’ refers to the development of intelligent, knowledge-aware applications, both in traditional AI arenas like expert systems, but also in areas such as the creation of end-user wizards—like the much-maligned paper-clip from Microsoft Office. Throughout this thesis, the term ‘intelligence’ will be used in the sense it is in the following quotation:

> . . . “intelligent” refers to the ability of a system to find implicit consequences of its explicitly represented knowledge. Such systems are therefore characterized as knowledge-based systems. ([61], p. 5)

In other words, the focus is on a program which can do more than strict computation, and to do so requires some form of representation of knowledge to solve a problem. Knowledge engineering is:

- concerned with building knowledge-centered, intelligent software
- interested in leveraging existing data (increasingly networked) to build more powerful tools
- follows a systematic process.

On the other hand, it is **not**

- trying to build a replacement for human reasoning or common-sense
- going to solve all problems
- characterized by *ad-hoc* tool development

Following a systematic process—a methodology—in tool development is an important step in developing robust and well-understood tools. The software development field has been involved in an evolutionary process, as software development becomes more and more a large-scale, industrial effort, largely deserving of the ‘engineering’ moniker (see [71] for a detailed essay on this phenomenon).

Knowledge engineering is further back on this development trajectory, but progressing. The preceding years have been characterized by *ad-hoc* development, and lately there has been a move towards more of a systematic approach to the development of knowledge-based systems. As stated in [82], “[t]his requires the analysis of the building and maintenance process itself and the development of appropriate methods, languages, and tools specialized for developing [knowledge-based systems] ([82], p. 1).” In a survey I conducted, detailed in 3.1.2.1, I found more than half of knowledge engineering projects involve five or fewer individuals. In terms of the increase in reliability and size of applications, knowledge engineering is largely dominated by the hobbyist and the researcher. However, demand for these applications is causing many to grow in scope and scale. Such growth in turn demands adoption of more systematic processes of development; a trajectory which closely mirrors that of software development.

In addition, the successes of the World Wide Web have shown many people the power of distributed information—such as searches using Google—and led some researchers and developers to leverage knowledge engineering techniques to provide even more meaningful information - such as those the semantic web initiative [9] describes. This vision foresees many different systems used to describe data and domains, enable interoperability, and secure transaction success, and as such will have a large impact on the knowledge engineering domain.

One of the byproducts of this success will be a requirement for improved metaphors for understanding the claims about the world that these different models make. Numerous different taxonomies, vocabularies, and data dictionaries will place a heavy cognitive burden on knowledge engineers and domain experts as these individuals attempt to comprehend them; researchers have shown that formal, logic-based language is difficult for humans to comprehend [73].

Artificial intelligence has, by and large, failed to impress, and is typically seen as having little practical value. Despite that perception, however, many new products incorporate AI technology, rather than being developed as the stand-alone 'Expert Systems' of the past. This distinction is akin to the difference between robot and cyborg—the one a pure machine, the other a melding of the best of biology and the best of technology; it seems much more likely that we will see a robot-like human before a human-like robot.

1.2 The growth of intelligent systems

One potential 'killer app' for knowledge-based systems is the Semantic Web initiative [9]. The semantic web describes an effort, sponsored by the World Wide Web Consortium (W3C), which aims to develop machine-understandable data on the Internet; the plan is to take the current information on the Web or Internet, add additional markup (metadata), and allow computers to perform operations on that data (this idea is covered in Chapter 2). The Semantic Web itself is not an application, but rather, a platform for applications. The reason this concept has great potential to serve as a vehicle for the first truly widespread knowledge-based tools is two-fold: first, it leverages the existing wealth of data on the Web, which grows nearly exponentially; secondly, it uses an open, global standards process to agglomerate the large amount of existing knowledge about machine-enabled reasoning, coming from the early developers of these tools (such as MYCIN [15]), with existing web knowledge and standards.

The emergence of personalized web journals, known as web logs or 'blogs', illustrates this development. Blogs themselves are not particularly revolutionary, and many of them are quite banal; however, when combined with the distributed nature of the Internet, and a handful of standardized protocols, blogs become information sources of unparalleled promise. One set of standards, the Rich Site Summary or RSS standards, provide blogs with a means to syndicate content with standard metadata. This typically consists of items like publication date, title, author, category, and a preview. RSS clients can interpret this data (much like an email reader) and allow the end-user to determine whether a particular entry is worth reading in its entirety.

Blogs often form an essential part of social networks, since most blogs have a highly personal, informal aspect to them. Social networks have other models, most popular of which is the Friend of a Friend vocabulary (FOAF), which defines some standard social networking terminology, such as biographical information and relationships (e.g., Person A knows Person B). The semantic aspect of this terminology allows machine agents to perform complex queries on the networks formed as each individual creates a FOAF-formatted description of herself. This information could then be used to determine which blogs may be of interest: for example, all bloggers who live in the UK and are interested in football (as long as everyone agrees what 'football' is—interesting disambiguation problems still remain). This example demonstrates the enhanced semantics that the connected nature of the Internet provides. Such networks have the potential to scale exponentially (for example, you have two friends, and they each have two friends, and so on). This places a heavy cognitive burden on the human making use of this network.

1.3 Cognitive support enhances knowledge engineering tools

At the beginning of this chapter, I mentioned that there are two streams of research that deal with the issues of discovery, understanding, and communication of knowledge - knowledge engineering and cognitive science. Knowledge engineering techniques alone will not provide all the means to address these issues. Leveraging the power of knowledge engineering in a way which seeks to address human and task needs is complex, and current knowledge engineering tools do not provide much analysis into how or why they approach this problem. Cognitive science provides some techniques to do this.

The actual method used to accomplish a particular task (e.g., which knowledge representation scheme is used, what underlying problem-solving is used) is not the focus of this thesis *per se*. What is the focus is the way in which humans can access that power. One of the things preventing people from accomplishing this is the lack of understanding about the importance of *cognitive support* [88], the elements of a tool which aid human reasoning and understanding. As Walenstein states, “The first rule of tool design is to make it useful; making it usable is necessarily second, even though it is a close second . . . [A tool’s] usefulness is ultimately dependent upon [its] utility relating to cognition: i.e., to thinking, reasoning, and creating. Assistance to such cognitive work can be called cognitive support ([88], p. 5).”

The relationship between usability, utility, and cognitive support is a complex one. Norman [63] defines three models of how a system works (where system is any external device a human can interact with). The designer has a mental model of how it should work, the user has a mental model of how the system is working, and the system itself has a model, which Norman terms the system image, of what is actually happening (reality). This is shown in Fig. 1.1.

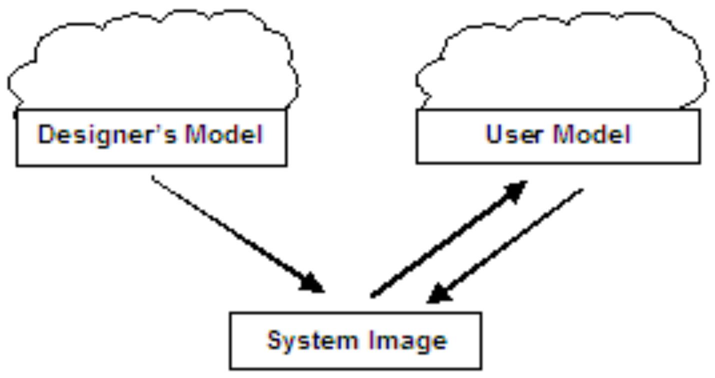


Figure 1.1. *Norman's model of how users understand tools such as software [63]*

Usability and utility exist as abstract concepts in the system image, created at design-time. They define how easy it is to do something with a tool, and what can be done with a tool, respectively. Cognitive support measures how well the tool supports a given user's cognitive processes, and is the product of the interplay between the system image and the user's needs and desires. Thus, usability and utility affect cognitive support based on the user's perception of the system image. Designing a tool to provide cognitive support requires understanding the specific needs of users of the tool, as well as what functionality to provide (that is, addressing both the domain and the user requirements).

Cognitive support research is in its infancy in software engineering [88], and more so in knowledge engineering. Tool designers in knowledge engineering certainly consider issues such as utility and usability (and implicitly cognitive support). However, what is lacking is a formal exposition of why and how such design considerations were made. Chapter 3 presents a preliminary analysis of what cognitive support is required in knowledge engineering tasks, specifically for users who perform modeling tasks. It is one of the goals of this work to identify cognitive support requirements in knowledge engineering processes, in order to make knowledge engineering projects more productive.

Lack of complete understanding of cognitive support issues for knowledge engineering tools partly explains the lack of adoption. This is particularly relevant in knowledge engineering since, as was shown in preceding paragraphs, the cognitive burdens on users will only increase. For example, in one prototypical example the size of the domain model, let alone the domain itself, is much larger than what any one human user can make sense of. This model, known as the National Cancer Institute Thesaurus (NCI Thesaurus), "facilitates the standardization of vocabulary across the Institute and the larger cancer biomedical domain" (see `http://ncicb.nci.nih.gov/core/EVS`), containing "detailed semantic relationships among genes, diseases, drugs and chemicals, anatomy, organisms, and proteins [39]". Providing cognitive support at appropriate places will be of great assistance to modelers.

Few existing tools have dealt with cognitive support issues in a systematic manner. The majority approach the problem by identifying areas they *conjecture* may need support, often specific to a particular domain, then building that support into a tool, and finally attempting to identify whether the tool met those requirements. Chapter 2 examines these solutions in more detail. This thesis attempts to divorce the specifics of particular solutions from the larger challenges of cognitive support tools for knowledge engineering.

All tools can be said to provide various degrees of cognitive support, most often in the form of simple representations of the knowledge being modelled. For example, most knowledge modelling tools capture lists of the concepts and relationships of interest in an indented tree list, similar to popular file management programs. The tools I examine go beyond this to provide pictorial representations of the knowledge, commonly using some form of directed graph. Representing information and knowledge in this form is particularly important because it allows one to leverage techniques from information visualization research.

Information visualization techniques are used in many domains to help provide insight or to communicate information [16]. Information visualization leverages innate human abilities to perform spatial reasoning and make sense of relatively complex data using some form of graphical representation language. In the domain of knowledge engineering, such a language is often based on graph theory and has two components: one, the use of nodes to represent concepts in a domain; the other, the use of edges to represent relationships between concepts. The language for visualizing information in this domain therefore consists of manipulations of graphs in some form or another. Information visualization is one technique for constructing advanced visual interfaces to provide additional utility in tools.

One problem many tools have, and information visualization solutions in particular, is evaluating their implementations in real-world situations. Often cutting-edge solutions are developed in research situations and fail to recognize the significance of the 'last-mile' problem - the stage of development which involves marketing, distribution, and final adjustments for usability. Tools which tend to languish on the web equivalent of store shelves are said to suffer from lack of *adoption*. Adoption, also known as technology transfer or diffusion of innovations [69], is a complex notion and the subject of much study. For example, merely showing that a tool is used more in a particular environment does not indicate that specific changes were responsible for that adoption; other factors, such as social pressures, may be responsible. Incorporating the adoption perspective into tools is an area of active research (see, for example, the Adoption-Centric Reverse Engineering website [58]).

1.4 Software customization: a possible solution?

One potential way to resolve the challenges of creating a tool which provides cognitive support for knowledge engineering is to focus less on the domain specific requirements for the tool, and look instead to certain capable users in that domain to make the tool fit the requirements themselves. One way of doing this is to incorporate customization features into the tool. Customization, described in more detail in §2.4, allows users to alter either the data, the presentation, or the functionality of a tool in order to reflect their needs. The domain customizer is the only person in the technology transfer process who has acceptable knowledge of both the cognitive support a tool offers and the domain knowledge of what the tool should support. This thesis describes how I implemented customization support in a tool for knowledge engineering and examines how this change might impact adoption of this tool and its cognitive support.

1.5 Outline

This thesis is laid out as follows. This chapter provided a brief overview of the challenges involved in knowledge engineering, and suggested some ways of thinking about the problems which underpins the remainder of the work. Chapter 2 provides background on the relevant technologies and related work, defining some key concepts and definitions used in subsequent chapters. Chapter 3 identifies some techniques I used to identify problems with the cognitive support in current knowledge engineering tools. It also examines some non-functional design goals cognitive support tools need to consider and concludes with an approach to addressing the issue of adoption using one of these goals, that of customization. In Chapter 4 I describe extending a tool our lab has produced to incorporate customization. I explain these changes in detail and motivate their use. I conclude this chapter with a description of how I validated the changes I've made to the tool using domain experts as evaluators. Chapter 5 concludes the thesis by describing how these customization changes may affect the adoption of a cognitive aid for knowledge engineering visualization.

Chapter 2 – Background

This chapter extends the introduction of concepts mentioned in the previous chapter, and identifies existing tools and research which contend with those issues. It begins with a broader focus, discussing knowledge engineering and its tools, how information visualization is applied to knowledge engineering, and then discuss what adoption and customization are, and how they are relevant to the topics at hand. The chapter concludes by putting these topics in context of the work discussed in my thesis.

2.1 Knowledge engineering

I defined knowledge engineering in the first chapter as “the development of intelligent, knowledge-aware applications” and defined intelligence (in §1.1) as the process of deriving the implicit from the explicit. Knowledge engineering typically consists of a knowledge engineer following an established methodology [70], involving knowledge acquisition (or elicitation), creating a formal representation of some form, and then testing the representation to ensure accuracy (in concordance with the requirements gathered). Of these three steps, the one I focus on in this thesis is the knowledge representation phase. Issues such as extracting knowledge and verifying models are beyond the scope of this work; I focus on the modeling and representation steps because this is where most conceptual problems occur. Designing models of the world is very difficult to do, and formalizing such a model for use in software applications more so. One must keep in mind the importance of modeling a domain properly—to direct thinking—and the inherent bias involved in any such modeling task—they reflect a particular world-view. The knowledge representation phase therefore requires a great deal of cognitive effort from the modeler.

2.1.1 Knowledge representation and ontologies

There are several ways to store knowledge for later use. Natural language is one such way, and most closely approximates human usage. Typically, though, we wish to use computers to operate on the knowledge, and natural language is a poor choice for doing this, due to its lack of formality and its implicit syntax and semantics. A formal language which contains mappings from a syntax to computer-recognizable symbols, as in first-order logic or programming languages, is best for this purpose. A formal language is defined as

> An alphabet and grammar. The alphabet is a set of uninterpreted symbols. The grammar is a set of rules that determine which strings of symbols from the alphabet will be acceptable (grammatically correct or well-formed) in that language. The grammar may also be conceived as a set of functions taking strings of symbols as input and returning either "yes" or "no" as output. The rules of the grammar are also called formation rules [83].

Knowledge bases store representations of knowledge in a formal language. Traditionally they took the form of a set of statements or atoms about the world, together with a collection of rules describing how to operate on those axioms to produce new atoms [66]. This approach to knowledge representation is well-suited to earlier, logic-based AI tools such as MYCIN [15], a knowledge base for the domain of blood-borne illnesses developed in the late 1970s and early 1980s. It was quite successful at determining medical diagnoses for this limited domain, but suffered from an inability to adapt to new information—as this involved re-entering knowledge from a domain expert and re-configuring the knowledge model.

Later knowledge representation schemes evolved to store 'default' knowledge—that a chair typically has four legs— in constructs known as 'frames' [57]. These frames were instantiated when a situation arose that invoked that knowledge—for example, entering a dining room with a table and chairs. Other representations have been created, many from the large amount of work done on human cognition: neural networks, for example, try to mimic the distributed nature of human neural processing abilities in software. For each representation scheme a model had to be created of what knowledge was to be captured, and the knowledge itself had to be acquired, and these have proven to be the bottlenecks. Formal knowledge models are termed ontologies, and capture the concepts and relationships in a domain.

2.1.1.1 Ontologies

The word ontology is derived, it should be noted, from the philosophical usage, where it refers to the study of being and existence. In AI, the meaning has been subverted, and Gruber's definition is widely agreed-upon: "ontologies are formal specifications of a conceptualization ([42], p.2)." The term conceptualization refers to an abstraction of a real-world issue of interest; in the case of MYCIN this issue was blood infections. Ontologies do not prescribe the technology used to define them, and indeed take many forms; Uschold [87] provides an extensive discussion of differences between ontologies.

Ontologies are used in knowledge engineering to do domain modelling, and are excellent at capturing static knowledge [82]. Once an ontology is created, the claims it makes about the domain—for example, that breast neoplasm is found in the breast—are its ontological commitment, and implicitly agreed to by the users of that ontology. Ontologies are used to facilitate different types of communication, and range from the highly informal to the rigourously formal. Ontology development is increasingly following standardized methodologies (such as CommonKADS [70]). *Ad-hoc* methods have their uses, however, particularly for smaller, prototype models. One such method is described in Noy and McGuinness [64]. This document describes a five step methodology for constructing an ontology.

In this work, I focus largely on modelers working with formal, frame-based ontologies, and specifically, ontologies created with a particular tool, the Protégé ontology editor from Stanford University (see §2.2.1). Studying the ontology modeling process is of interest because it is a highly cognitive process, demanding detailed understanding of both modeling techniques and domain knowledge from modelers. As tools grow broader, the knowledge models they use also increase in complexity. The example of the NCI Thesaurus given in the previous chapter (§1.3) illustrates this: given its mission of standardizing vocabularies in cancer biomedicine, and the ever-increasing knowledge generated by research in the

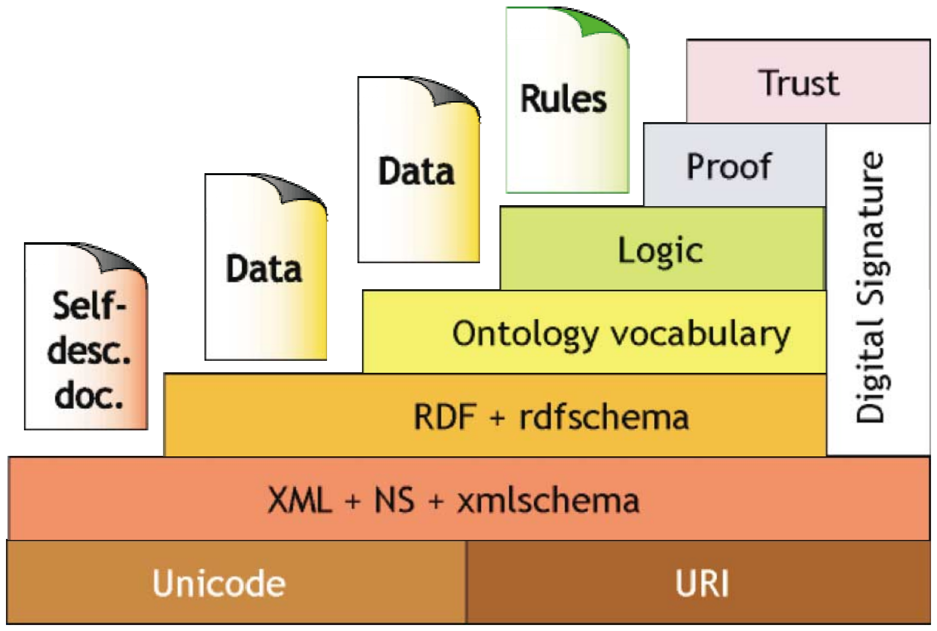


Figure 2.1. *The components of the 2nd Generation web*

field, it is safe to say the NCI Thesaurus will increase fairly quickly in size, perhaps by 10-12 concepts a day, as the NCI has several modelers working on various aspects of the ontology each day. A description of these tasks is provided in §3.1.2.2.

2.1.2 The Semantic Web initiative

The Semantic Web initiative of the World Wide Web Consortium (W3C) ([8], [65]) proposes strategies to enable the "abstract representation of data on the World Wide Web" [56] such that additional, machine-comprehensible metadata might be created. Global standards have been developed for this initiative, such as the Resource Description Framework (RDF) [53], a formal language for describing subject-property-object relationships, the Web Ontology Language (OWL) [77], a knowledge representation formalism, and XML, a data serialization format. Together with renewed interest in intelligent systems, these promise to increase the semantic information available. Combined with the power of distributed application development via the Internet, any number of tasks, such as making inferences on web site metadata, to intelligent e-commerce shopping agents [9], become more feasible and capable. Figure 2.1 illustrates how these components may fit together, leveraging existing Internet and World Wide Web technology such as Uniform Resource Identifiers and Unicode.

2.1.3 Building the semantic web

Machine-readable knowledge, when fully standardized, promises to make the idea of using a machine to make decisions at once clearer and more concise for the large body of pre-existing software developers working on the Web. Although it was only 5-10 years ago that most large companies had never heard of the Web, it now seems obvious that nearly all applications will be web-enabled in some form, for example, as web services (see Curbera *et al.* [21] for an introduction). A further 5-10 years from now new Web users will find it hard to believe that people ever had to search for airline tickets, as their computers will be able to present them with the result automatically. The Semantic Web promise will add further complexity to the knowledge modelling task. To re-use the example from the National Cancer Institute, a semantic web-enabled thesaurus, which is fairly close to reality (there is already an OWL ontology describing it [39]), brings new challenges. For example, modelers and editors (modelers tasked with strategic development) need to consider other ontologies, such as anatomy models. They also need to ensure their model is more precise, stable, and accurate than before. Knowledge engineering in the Semantic Web vision will only increase demand for adequate cognitive support. Some questions that might need answering: What commitment do I make in using this external ontology? What is the provenance of the knowledge represented by the ontology?

2.2 Graphical knowledge engineering

As mentioned in the introduction, there has been some research into cognitive support for knowledge engineering, largely in the area of user interfaces for expert systems. This work was motivated in large part by the realization that few users could easily understand what a tool was doing or how to make it work. Some early knowledge representations were directly graphical, such as Sowa's work on conceptual graphs [78]. This representation format, while certainly more readable than first-order logic based representations such as KIF [38], focused more on being logically rigorous than providing cognitive support for end-users. A similar idea, concept maps [34], were more focused on user support, but lacked rigorous logical representations for knowledge acquisition and inferencing. CYC, an effort to represent common-sense knowledge, had several user interfaces built for it, of which one, by Travers, used a room metaphor to browse different areas of the ontology [86]. However, use of this metaphor has difficulty with different relationships. Another knowledge representation tool, CODE4 [76], focused in more detail on the user experience, and also combined that focus with a logically rigorous representational semantics. A key detail that CODE4 focused on was providing multiple methods to view the knowledge, emphasizing the separation of presentation from model. For example, the system provided graph layouts of the knowledge base, but also provided a tabular interface.

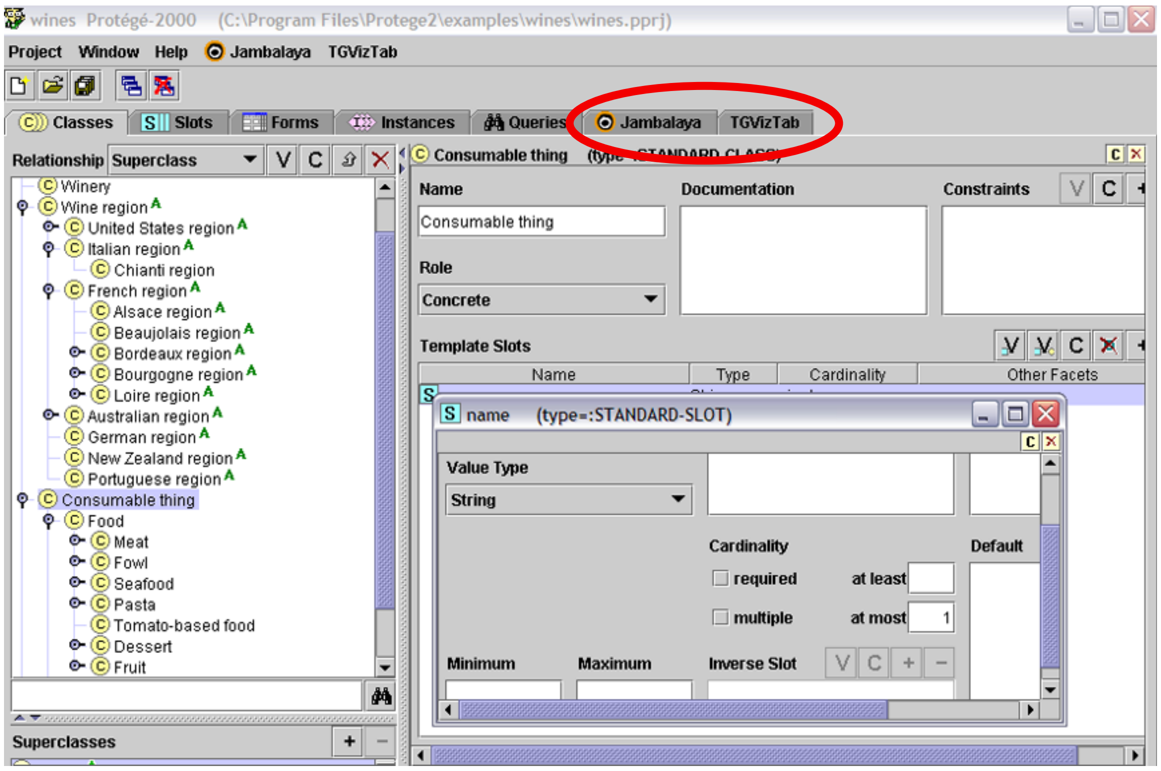
Other early work that is applicable to this subject includes the research done in visual programming, particularly in Expert Systems. Visual programming is important because the tasks associated with it (program understanding, control flow, model checking) are highly consistent with ontology engineering tasks, as we shall see in more detail. A good example of such a system is KEATS, a knowledge engineering tool with strong emphasis on visual programming and program visualization [24]. KEATS supported the notion of sketching early versions of a knowledge base before the actual design commenced. This differs slightly from the focus in this thesis, which is more concerned with how modelers understand or verify a model after it has been (largely) completed. The GKB-Editor tool [47] has a graphical interface for visualizing and editing a frame-based knowledge base. It has several views, such as a hierarchical view of the concept hierarchy, a relationships viewer, and a spreadsheet viewer. However, the views are static once defined, and do not allow much customization and interaction on the part of the user.

Another set of tools dealt with visualization techniques in information retrieval and management. An early work, SemNet [30], had several complex metaphors for visualizing personal information, including fisheye views, zooming, and hyperlinking; however, the hardware available at the time (1988) greatly limited its adoption, as did the relatively small amount of electronic data. Other work built on the graph visualization theme, discovering new techniques for browsing networked data. A lot of work has been done on visualizing hypertext networks (closely related to concept maps). For example, VISAR [20] was an early graphical tool to aid in the construction of hypertext systems, again using CYC.

2.2.1 Protégé

The tool my lab uses in our research is Protégé, an ontology editor from Stanford Medical Informatics with a Java-based graphical user interface [37]. Originating as a system for modelling medical guidelines and protocols, this tool provides an interface to create and model ontologies, as well as to acquire knowledge based on that ontology. Protégé has traditionally used a frame-based language to construct ontologies, an OKBC-compliant frame language according to the specification available at `http://www.ai.sri.com/~okbc/spec.html`. Frame languages, first mentioned in §2.1.1, use frames to model objects in the world, and slots in each frame to represent relationships or properties of an object. Furthermore, slots can have constraints, or facets, restricting allowed values.

A popular example used to illustrate this process is an ontology of wines and associated meal courses [64]. An ontology modeler creates an ontology which defines what wine is, what food is, and how they relate to each other, among other things. For instance, the modeler may state that the frame *Bordeaux* has a slot *produced-by* with a value of *Chateau Lafite*. Another slot may be *year-bottled*, describing when a particular wine was bottled, with a facet restricting this to *greater than 1990*. When this model satisfies the requirements for the system (gathered at a preliminary stage), instances are collected/acquired from domain experts (vintners, oenophiles, etc.) to expand the knowledge model to include data that fits the model. Additional projects can be referenced using Protégé's project inclusion mechanisms. Including a project is a form of importing it. The imported project's concepts and relationships are made available for use, but cannot be modified. This allows modelers



Name: Consumable thing
Role: Concrete
Value Type: String
at most: 1

Figure 2.2. *The Protégé user interface with the Jambalaya and TGViz tabs visible (circled)*

to use concepts in these external ontologies without altering them directly.

The combination of data and knowledge model (ontology) can now be termed a knowledge base, and software tools are used in conjunction with the knowledge base to create knowledge-aware applications. As the application is used, the original ontology may be refined to improve accuracy.

2.2.2 More recent tools

Leveraging Commercial Off The Shelf Software (COTS) to create visual knowledge engineering tools offers well-supported development environments with a reduced learning curve for new users [4]. The most sophisticated of these is SemTalk (`www.semtalk.com`), now known as Construct, developed by Network Inference (Fig. 2.3). Construct uses Microsoft Corporation's Visio diagramming product and creates a separate symbol library for the tool with associated semantics. For example, connecting certain shapes to-

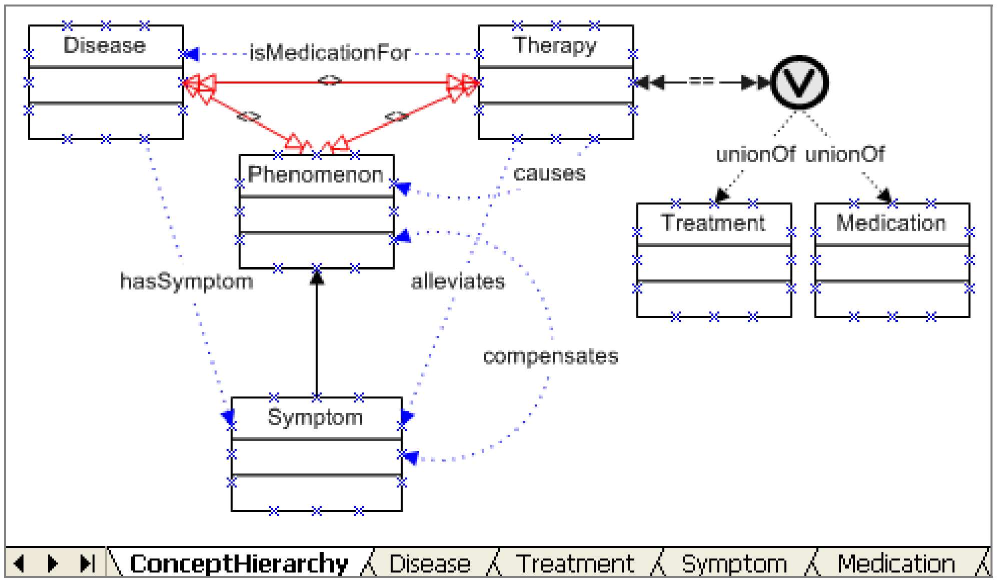


Figure 2.3. *Screenshot of the Construct ontology editor (networkinference.com)*

gether creates an associated sentence in the knowledge base.

EzOwl (`http://iweb.etri.re.kr/ezowl/`) is a plug-in for the Protégé editor which allows modelers to visually compose OWL ontologies using graphical operators. For example, in Fig. 2.4, the modeler is defining a class by taking the intersection of two other classes (an OWL-specific semantics that frame-based representations do not support).

2.2.2.1 IsaViz

Isaviz [67] is a tool designed by the World Wide Web Consortium (W3C) to visualize knowledge representations constructed using the Resource Description Framework (RDF). It uses the GraphViz library from AT&T. Although the user can configure how the views appear, they are not very interactive nor easily customized. The parsing and generating of graphs can be quite slow. It also has facilities for styling the graph using a stylesheet concept, exporting to SVG, and simple editing functions. This stylesheet concept has a lot of potential for handling customizations.

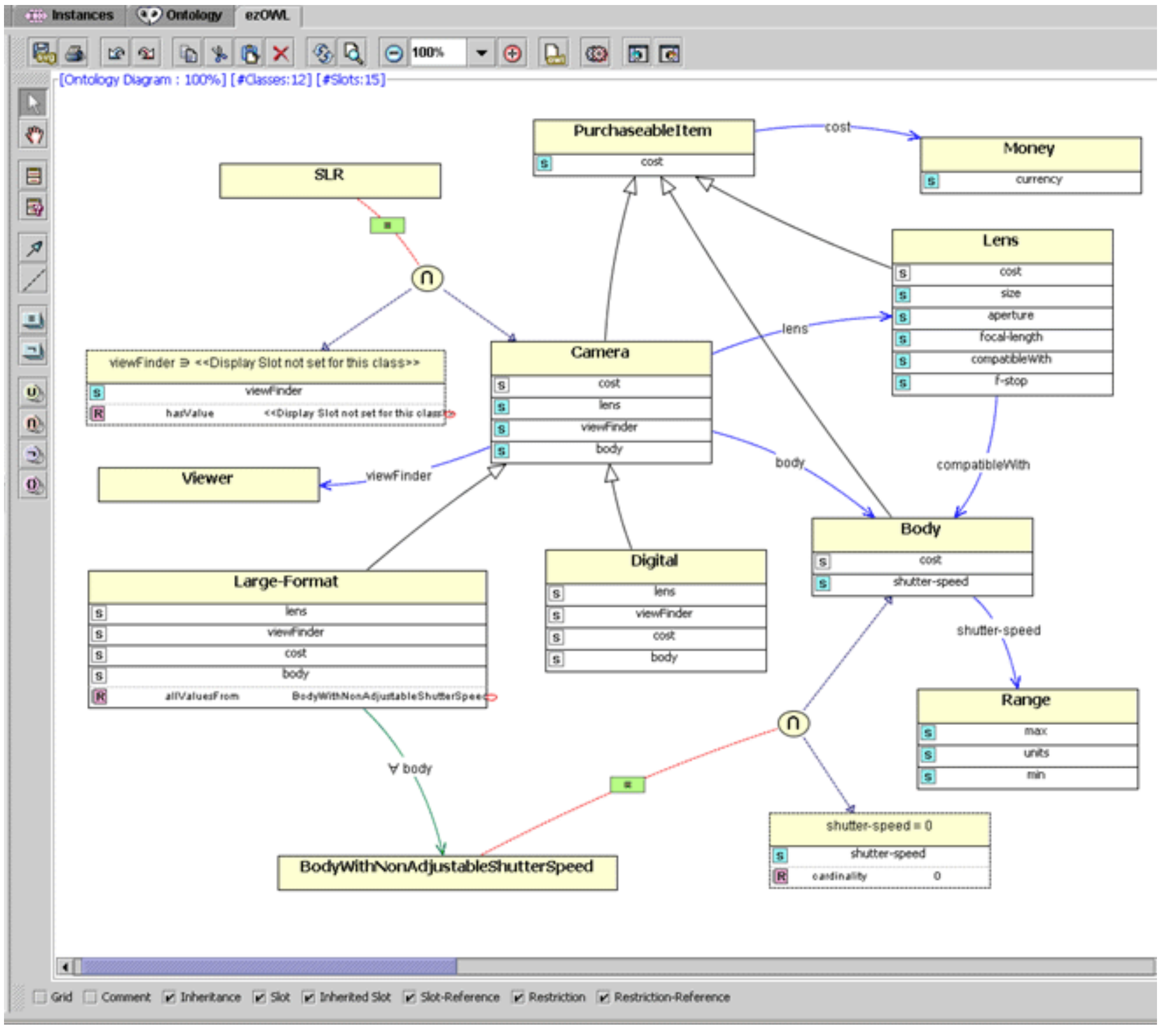


Figure 2.4. *Protégé's EZ-OWL visual ontology editor*

2.2.2.2 Ontorama

Ontorama is a visualization tool for RDF-based graphs, detailed in Eklund *et al.* [25]. It presents RDF graphs in a hyperbolic graph layout using radial layouts to align the nodes on a spherical surface. A significant challenge for Ontorama and other hyperbolic browsers is that not all ontologies are trees (in the mathematical sense) according to the inheritance hierarchy (is-a). For example, some domain models are constructed using partonomy (part-of) as the key structural relationship. This means these tools must somehow handle the case where these relationships break in order to display all the nodes—i.e., be able to visualize forests as well as trees.

2.2.2.3 Ontobroker/Kaon

The Ontobroker tool [22] uses a similar hyperbolic view technique to aid in the navigation of ontologies. It has recently been superceded to some extent by KAON [52], a similar tool with more of a focus on the Semantic Web. The strengths of these tools lies in the degree of integration between the tool and the visualization engine, which makes the representations in the graphs more salient.

2.2.3 Advanced visual interfaces in knowledge engineering

While graphical knowledge engineering tools tackle some problems, the solutions provide little or no justification for how the approach might be defined as successful (for example, number of users, novelty of approach), let alone evaluating success. To design better visual interfaces in knowledge engineering requires stepping back from the problem and examining the knowledge engineering tasks that need better cognitive support. One type of visual interface represents knowledge structures as graphs, and uses well-researched techniques from graph visualization to introduce new ways of manipulating the model.

Graphical *aide-memoires* can greatly assist human cognition. For example, when adding several large numbers together, nearly everyone needs the assistance of pencil and paper to store the intermediate values. Based on studies such as Bauer and Johnson-Laird's [6], it would seem diagrammatic representations of complex models can be of similar assistance. However, the use of such a cognitive aid has overhead. For example, using Venn diagrams to represent complex logical sentences such as ”Jenny is a student or Jenny is a teacher, but not both, and Paul is a student” can help identify what exactly is being stated, but requires a certain degree of ‘graphical literacy’. In other words, while most people are quite able to interpret meaning from sentences, not everyone can do this from a diagram. This inability is attributable to lack of experience. One of the major challenges for graphical cognitive aids, then, is the ability to leverage innate abilities for spatial reasoning without demanding too much in the form of graphical interpretation.

I provide detail on tools which create advanced visual interfaces for knowledge engineering in the following chapter, which introduces such tools in the context of knowledge engineering tasks which were identified through several research methods, and thus have more relevance there.

2.3 Adoption and innovation diffusion

Throughout this thesis reference is made to the challenge of technology transfer, or technology adoption. These terms refer to the transfer of an innovation from developers to potential adopters. The seminal work in the study of technology transfer is by Everett Rogers, “Diffusion of Innovations” [69], first published in 1962. In this book, he examines some reasons why things are or are not adopted, from new cereals to new medical techniques. In software engineering, (and by extension, knowledge engineering [27]), the ‘adoption problem’ refers to the fact that many tools, particularly research tools, fail to be deployed in industry [58]. Adoption is a complex process which defies simple explanations: for instance, the fact a tool shows great usefulness and usability do not guarantee it will see use at the target company. I elaborate on this point when justifying my choice of evaluation techniques in Chapter 4.5.

Traditionally, technology diffusion has been seen as an imperialistic process, where the innovator comes up with something he thinks is clearly better than what is currently used.

In this perspective, not adopting an innovation is ‘bad’, and using it is ‘good’. Researchers have come to realize this process is not so simple, and that there are many, many factors affecting adoption. Rogers illustrates this with his description of women and public health in Latin America. While boiling water was technically healthier, the women refused to change their ways due to social pressures and traditional beliefs.

Stan Rifkin writes [68] of how this new perspective affords a greater understanding of technology and its use, particularly in software. Developers of new products—in this thesis, advanced visual interfaces for knowledge engineering—must understand how adoption processes work in order to effectively design a tool. For example, Rifkin mentions ([68], p. 24), that one way of looking at tool development is by characterizing the functionality a tool offers as ‘competency-enhancing’ or ‘competency-destroying’. The latter category describes tools which require one to learn new skills, which fits the tools described in the following chapter. Such tools are likely to be resisted and feared, at least initially.

Rogers lists five factors which affect diffusion ([69], p. 14-15):

1. *Relative advantage* is the degree to which an innovation is perceived to be better than the idea or product or process it supersedes.
2. *Compatibility* is the degree to which an innovation is perceived to be consistent with the existing culture and needs.
3. *Complexity* is the degree to which the innovation is difficult, or at least more difficult that its competitors.
4. *Trialability* and *divisibility* are measures of the degree to which an innovation may be taken apart and only a part tried. A thick, monolithic innovation has a lower trialability than one that has separable components, each of which adds some value.
5. *Observability* is the degree to which the results of the implementation will be visible.

To relate this to the discussion of cognitive support, usability, and utility in §1.3, I define utility as the set of functions a tool offers. Usefulness is a measure of how well the tool helps the human user accomplish something, that is, how much cognitive support it provides. Usefulness is equivalent to Rogers's concept of relative advantage. Usability is a measure of how easy or difficult the the tool's utility is to access. Thus, it is a measure of complexity and compatibility.

Adoption analysis is as essential an aspect of software development as requirements gathering (although it might be characterized as one facet of comprehensive requirements gathering). For this research, adoption analysis provides a means to assess, for designing new tools, what the reaction will be. For understanding existing tools, adoption analysis illustrates why those tools were successful or unsuccessful. I use both approaches in the next chapter in analyzing current tools that provide cognitive support for knowledge engineering.

2.4 Customization and domain models

One promising method for addressing some of the concerns adoption research uncovers is software customization. In designing tools, in this case advanced visual interfaces to provide cognitive support in knowledge engineering, the ability to adapt to domain-specific requirements is important. The chief advantage for knowledge engineering lies in a customizable tool's ability to adapt to specific requirements, something which characterizes knowledge engineering projects. For instance, tool requirements at the NCI may require viewing large numbers of concepts in very specific ways. An ontology integration project may be more interested in overviews of a number of smaller models. Functional customizations, presentation customizations, and data customizations are the three main forms customization can take.[1] In data customization, the format and content of data and metadata can be customized. Applying an XSL (Extensible Stylesheet Language) transform to an XML file to produce a different format is an example of this. Presentation customizations change either the organization of information in a display (information architecture) or the graphical design. Changing the appearance of a web page using Cascading Style Sheets (CSS) is an example of graphical design. Finally, one can customize the functionality a

[1] This and the following discussion are taken largely from [54], but any errors are mine.

tool offers using control/behaviour customization. This can include removing features, constraining what the user can do, or enhancing features to extend functionality for the user's needs. Chapter 4 describes a customization implementation operating partially on the information architecture of a tool, as well as constraining and selecting which features to provide.

2.4.1 Who is involved in the customization process?

Michaud [54] lists three idealized roles for players involved in customizing software. The person creating the software, the Designer, defines the major architectural framework for customizations, such as feature set and interaction styles. The Customizer is a person who has extensive domain knowledge and refines the tool using that knowledge. Finally, the End User is one for whom the customizations are performed and, for various reasons, has no interest in customization. Throughout this thesis I will refer to Michaud's user roles using initial capitals, as in Customizer or End User.

His roles parallel those described in Finnigan et *al.* [32], which describes the creation of a domain independent 'software bookshelf' for program understanding. In that work, the equivalent roles are titled builder, librarian, and patron. In both cases, the middle role (Customizer or librarian) is central, and is a person with good tool knowledge as well as more domain-relevant knowledge. It is the domain knowledge which produces the benefits for the End User or patron.

2.4.2 Customization approaches

Many tools offer some form of customization support. Microsoft Word, for example, gives users the ability to remove and add features, and also allows programmers to operate on the application using the Visual Basic for Applications (VBA) scripting language. Two areas of active research are worth noting, model-driven architectures and scripting.

2.4.2.1 Model-driven architectures

Model-driven architecture uses a formal and explicit model (often an ontology) to create an architecture for a software tool. A prime example of this is the Object Management Group's MDA initiative (see `http://www.omg.org/mda/`). Changing the model produces a corresponding change in the application. In this way, customization is reduced to modeling changes. More discussion of this is given in §5.6.

KNAVE — A knowledge-based system created by researchers at the Stanford Medical Informatics group [17], KNAVE (Knowledge-based Navigation of Abstractions for Visualization and Explanation) explores the time-sensitive nature of medical records, presenting an interface for clinicians to explore and understand recommendations the application makes regarding protocol-based clinical care. The system is a timeline style visualization, using structured semantic information plotted against a temporal axis. This additional level of knowledge is a weakness, because it "relies heavily on detailed knowledge provided by a domain expert . . . [therefore] we see no way to achieve this level of fluency without the help of a domain expert." The KNAVE project obtains its successes by tightly integrating the domain and the visualization. Separating these two models is important to achieve more domain independent tools.

Ontoweaver — Ontoweaver is an ontology-based hypermedia application [48]. Hypermedia applications are applications which use rich media and complex relationships to create dynamic applications, typically over a network. Ontoweaver makes use of a set of ontologies to determine what representation to present to users of a web site, as well as what customizations can be made. For example, a website designer could modify the user ontology based on an assessment of to whom the site is targeted, then modify the presentation ontology and domain ontologies based on his understanding of the required information. Ontoweaver would update the website automatically based on these modifications.

2.4.2.2 Script-based environments

Scriptable environments, a version of end-user programming [60], allow for customization by the End User by extending the features of the tool using a scripting language. The RIGI software reverse engineering tool, for example, allows users to extend the tool to support their own cognitive requirements, which the tool Designers could not fully understand, as mentioned in Tilley *et al.* [85]. Their paper refers to the non-configurable approach as 'builder-oriented', and describes the implementation of a system which allowed the End User to tailor the environment to suit his needs.

Interfaces themselves can be customized. Modern scripting languages, being interpreted and not compiled, can represent an interface in editable source format, and then dynamically update the interface. Jelly (`http://jakarta.apache.org/commons/jelly/jelly-swing.html`) is one example of this. Jelly stores presentation information in XML format, using that to describe a Java application's user interface. The traditional approach to UI design in Swing involves complex positioning and instantiation of graphic widgets, which are then compiled. Jelly allows designers to determine the interface dynamically. Another tool, described in [36], uses XML Schemas transformed with XSL to create XForms, an emerging standard for web-based forms.

The work in this thesis combines these two perspectives on customization. As shown in Chapter 4, a model is created of a tool, allowing Customizers to modify the user ontology as needed. Customizers can also create customized scripting actions which can be run by end users to perform the needed functions.

2.5 Chapter summary

I have illustrated selected challenges created by the lack of appropriate cognitive support tools for discovering, understanding, and communicating knowledge models. While visual knowledge engineering tools exist, for the most part they fail to analyze what functionality is useful, and why. This chapter discussed challenges tools face in getting adopted, and suggested that the adoption question is one which must be considered early on at the requirements analysis phase. The notion of software customization was introduced as a potential means to address these adoption challenges. Customization allows domain experts, not the tool designers, to make decisions as to what functionality is necessary or valuable for their specific problem.

In the next chapter, I examine some tools which partially address the questions of determining what cognitive support might be required, and using a combination of research methods, attempt to present a preliminary assessment of cognitive tasks requiring support in the knowledge modeling domain.

Chapter 3 – Cognitive support for knowledge engineering

In the previous chapters of this thesis, I explained how knowledge engineering tasks are complex, and becoming more so. I then proposed that visual cognitive support tools could aid user comprehension of large, complex models of reality. Now, I examine what features such a tool should have, using an existing effort from my research group as a form of case study. This chapter then concludes with a more detailed exploration of problems such tools face in adoption by users, and suggests a solution using customization.

3.1 Determining where cognitive support can help

In order to discover more details about where cognitive support might help, I used a number of research techniques, including background research, surveys, and qualitative evaluations.[1] I make use of the work of others in my research group, CHISEL, giving credit where appropriate.

3.1.1 Impetus for the research

Over the past few years, our research group at the University of Victoria has developed a tool similar to those mentioned in the background review. Jambalaya—available at `http://www.shrimpviews.org`—was initially developed as the result of a collaboration between the Protégé team and Jambalaya developers [80]. Jambalaya was based on a tool for visualizing software systems, and we recognized some obvious synergies with knowledge engineering. This initial development and release was characterized by a lack of any formal requirements process. While it was apparent to both teams that this tool was potentially very useful, insufficient effort was made to identify what it should look like; thus, the existing features were simply inserted wholesale into Protégé. We had identified requirements for the tool based on our work in software comprehension [79], but had not

[1]Portions of this chapter were published in [26].

done any work to identify the requirements in this different domain. Approximately six months following the initial release of the plug-in, we began to question whether the tool was meeting the requirements of the users, as we had not received much feedback, and had never truly considered this in the beginning. We did a limited initial analysis of the Jambalaya tool: we conducted some studies to analyze how useful the various interface elements were, and a heuristic evaluation was also done to identify some user interface challenges, which helped to make the tool more usable. Neither approach, however, provided a clear understanding of how the tool might support knowledge engineering tasks.

3.1.2 Requirements gathering

An examination of other visualization tools for knowledge engineering, and the requirements they were built to satisfy, revealed the lack of an established theory about user tasks and the cognitive support they require. This examination is discussed in more detail in Allen's thesis [3]. Without this theoretical guidance, quantitative approaches—such as formal user testing—would fail to reveal new requirements for such tools. Furthermore, Allen identifies many difficulties encountered when performing user testing in the knowledge engineering domain, including gaining access to expert users, generalizing results over different domains, and quantifying the knowledge acquired and used by such tools. These issues led us to focus on more qualitative approaches which included a user survey, two contextual inquiries, and investigation of related work; using these different techniques provided a series of useful perspectives on the problem.

3.1.2.1 User survey

One of these approaches was to survey the general user population of the Protégé tool and determine some of their preliminary needs for a visualization tool. This resulted in a user survey which I disseminated to the Protégé user list and other knowledge engineering lists [29]. The survey sought to determine the background of people interested in the area of visualization for knowledge engineering, and received 44 responses. After some preliminary questions designed to establish background, the survey asked what tasks respondents carried out on their ontologies, and where they saw a graphical interface helping. I found that there are a wide variety of users of knowledge engineering tools, as well as many domains to which ontology engineering is being applied. Furthermore, the results indicated that visualization *is* a desired feature. The lesson for those working with tools which manipulate and create ontologies is that this diversity must be supported. I believe that the wide-ranging degree of domains uncovered is a sign of the future, and that tools that operate at a meta level to assist users to understand the modeling decisions, such as Jambalaya, will be increasingly important in maintaining clear communication and understanding.

3.1.2.2 Contextual inquiries

I used the survey as a pointer to areas where more detailed investigation might be useful, and this drove the second aspect of the requirements gathering: contextual inquiries at two separate venues. Contextual inquiry is a form of ethnographic research where the investigator both observes and questions current work practices, alongside a user [11]. I conducted one at the U.S. National Cancer Institute, discussed previously in §1.3. While they currently are using a tool suite named Apelon (`www.apelon.com`), and not Protégé, they had expressed interest in using Protégé, and in our implementation of Jambalaya. I conducted two site visits to the NCI team to determine how their ontology engineering workflow proceeded, and what some requirements for that workflow might be. Using contextual inquiry techniques, I sat alongside users to observe their daily activities. I also conducted in depth discussions with the technical gurus and managers of the project, to gauge their needs. The inquiry demonstrated a need for a number of mechanisms. One user demonstrated a reporting tool, custom built, which showed an indented text-based layout of the concept hierarchy and the associated relationships, designed to show “what this branch [of the ontology] looks like”. Modelers also relied extensively on text searching for navigation, using this mechanism to jump around the concepts. They also wanted to define new concepts using existing concept definitions (i.e., make a copy and edit that), but this was not supported. Another user used a third-party tool to model the concept textually, and then added the concept using that external information. Finally, all users expressed an interest in more collaborative, real-time modeling work.

My colleague Polly Allen conducted a site visit to the University of Washington Foundational Model of Anatomy Project (see http://sig.biostr.washington.edu/projects/fm/AboutFM.html). Here, she employed similar techniques to gauge the needs of the users of that domain. While watching the users perform domain modeling and verification, using Protégé-2000, she asked specific questions about the process, and concluded with a demonstration of Jambalaya in its current form to gauge their reaction. From this demonstration several important points of feedback were gathered on how the tool could be changed to better suit their needs. For example, users demonstrated that knowledge engineers would like to visualize not only the taxonomy hierarchy but the metamodel hierarchy as well. Other users identified more support for editing being desirable, such as the ability to 'drag and drop' concepts of interest onto a list on the side of the screen. It was observed here as well that modelers spent a lot of time performing text searching operations. Finally, the modelers noted that errors in modeling were found about once every two weeks, indicating that existing verification mechanisms were insufficient. Allen detailed this visit in [3].

These two visits, in conjunction with the survey, as well as our previous work in software engineering [81], gave me some specific data to form preliminary visualization requirements for the knowledge engineering domain, in conjunction with a detailed literature review, discussed next.

3.1.3 Background review

I conducted an extensive background review looking for discussions of user challenges in knowledge engineering. Considering when and where the tool or method was first detailed was important because, owing to a rapidly changing computer technology landscape, tools that were not feasible 10 years ago have now become commonplace. For example, SemNet

[30] had interesting ideas for visualizations, such as fish-eye views, but the technology of the time could not perform quickly enough. The implementations are now fairly commonplace. The implication is that previously discarded tools may now be worthwhile. Four studies I discovered during our literature review were highly relevant and I discuss them here. Other studies concerned user interface details, but often did not suggest specific areas of concern or problems users encountered. The studies used are listed in ascending order of the time period they cover.

During user studies of a Knowledge Acquisition system conducted by Tallis and Gil [84] the experimenters observed users performing the following high-level tasks:

- Understanding the given knowledge acquisition task
- Deciding how to proceed with the knowledge acquisition task
- Browsing the knowledge base to understand it
- Browsing the knowledge base to find something
- Editing (create or modify) a knowledge base element
- Checking that a modification had the expected effects on the knowledge base
- Looking around for possible errors
- Understanding and deciding how to fix an error
- Recovering from an error by undoing previous steps (to delete or restore a knowledge base element)
- Reasoning about the system

This study serves to detail some of the tasks that users go through while engaged in the traditional knowledge engineering process. It also hints at some of the problems that an otherwise valid knowledge engineering project may encounter if it fails to address the specific cognitive needs of the users. For example, a system which failed to provide simple, usable methods for adding knowledge or looking for errors would be quickly rejected by users without a large investment in the system (typically these would be the actual designers of the system).

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 were that the users do not know where to start and where to go next, the users do not know if they are adding the right things, and the users often get lost as it takes several steps to add new knowledge. This study is interesting because it clearly shows that the standard knowledge engineering methodology, consisting of the steps of modeling, acquiring, and verifying knowledge, fails to accommodate the specific needs of users, even modelers in the domain. It is no use to have a crisp and detailed methodology if users cannot easily make use of it in any one of its stages.

Gary Kwok-Chu Ng conducted user studies as part of his Ph.D. research [62]. Based on an evaluation of user requirements in ontology modeling tasks, he designed a tool, InfoLens, to browse description logic ontologies, using a combination of 'lenses' which revealed different information about the domain as they were interactively moved about the model representation. He found that one issue was scalability for practical sized systems. For cognitive support specific tasks, he identified the need for a tool to support information integration (between different representations), to support the often cyclic task-switching between navigation (around the model) and visualization (of a specific aspect of the model). Initial user surveys were quite positive but some aspects of the implementation hindered the evaluation. Despite these problems, Ng reports his tool was "effective in finding patterns useful for modellers, with a few limitations in both scalability, flexibility and completeness ([62], p. 212)". I analyze why these limitations occurred in §3.4. Ng also identifies the same high-level tasks that I outline below in §3.2, although I use the term 'modeling' rather than his term 'authoring'.

Finally, Clark *et al.* [19] conducted studies of SHAKEN, a graphical tool for knowledge acquisition. Although they only tested it on four users, and those users were not modelers or knowledge engineers, I still present the results for the insight it offers into the benefit of increased cognitive support. The users were able to enter a few hundred concepts into a large medical knowledge base within a week, and also verify the model using competency questions. From the discussion:

> [the results suggest] the basic machinery works, providing a basic vehicle for axiom-building without the users having to encode axioms directly (or even encounter terms like "concept," "relation," "instance," "quantification," etc.) . . . ([19], p. 8)

Some of the issues Clark *et al.* identify as areas needing improvement include multi-faceted representations, active critiques from the system, and more expressivity in the interface (such as temporal relations and conditions).

3.2 Knowledge engineering tasks requiring cognitive support

This section provides some of the motivation for functional requirements of knowledge engineering tools. Functional requirements refer to specific knowledge engineering tasks—such as knowledge acquisition, modeling, verification, and usage—that cognitive support tools should address. I prefer to term these 'tasks requiring cognitive support', rather than functional requirements, as the term requirements traditionally refers to domain and tool specific needs, whereas what I term 'tasks' address high-level commonalities seen over a range of knowledge engineering projects. There are also other, non-functional requirements (variously known as effectiveness criteria, quality attributes, and constraints [59]), which I discuss in §3.4.

My list of tasks where cognitive support is expected to be of most assistance is based on four streams of research: 1) the detailed literature review of user interfaces to knowledge engineering tools and papers on the few user studies that are available; 2) the qualitative analyses from site visits to two large and well-known knowledge engineering efforts; 3) our lab's background in software program comprehension, as many researchers have made the case that software engineering and knowledge engineering have many parallels (such as [24], [23], [31], [46]); and finally, 4) my survey of ontology users identifying visualization requirements and domain specific issues [29].

What follows is a non-exhaustive list of tasks in the knowledge engineering process identified as requiring cognitive support; non-exhaustive, because they are based on my experiences and research, in collaboration with other members of my lab. They could also change depending on a variety of factors, including the domain of interest and user characteristics. The objective is to present a fairly detailed list of problem areas that have *emerged* through the work my research colleagues and I have done over the past few years. Furthermore, this list is tailored, as mentioned above, to modelers working on the ontology. While some of these may seem obvious, I wish to make them explicit. To indicate from which research method these were developed, I use the following code: **[s]** for survey, **[r]** for review, **[pc]** for program comprehension work, and **[ci]** for the contextual inquiries. Table 3.1, following this section, summarizes the findings presented below, and gives specific details on the origin of the requirements mentioned. I highlight three encompassing terms to structure the taxonomy: *navigation*, *modeling*, and *verification*. These terms are intended to organize the tasks into different sub-areas in the knowledge engineering process, and are not exclusive. For example, navigation tasks may be performed during a verification exercise.

⋆ **Navigation**–support the navigation of ontologies for understanding, discovery, and search

1. *Provide overviews and support top-down exploration of the ontology* (**[s] [r] [pc] [ci]**) – top-down exploration is most useful when people are unfamiliar with the ontology. For example, in a meal ontology, users may start with the higher-level concepts of ‘wine’ and ‘food’, and drill-down into more detailed areas of the ontology. In large ontologies, in particular, this task could be highly useful, because complex ontologies rapidly overwhelm the abilities of users to make sense of where they are, and where they need to be. By providing an overview, a cognitive aid can address some of these challenges. Often, top-down exploration is goal-directed and based on a hypothesis. There is evidence,

from the software engineering literature, that navigation strategies often alternate between top-down and bottom-up [81], and tools should support them. In the case of modelers, this may occur when working with ontology merging, but I have found no specific evidence to that effect.

2. *Support slot-based browsing* (**[r] [ci]**) – Navigation should support following hierarchical and network structures. In OWL, for example, slots may be composed of other properties and classes using set-theoretic semantics such as Union and Intersection. For example, one may want to see subclasses of Wine, and then navigate to the regions those wines are produced in. The tool would provide mechanisms to discover relationships that are not clear from the standard view of the model. Furthermore, a tool should show the user values for all slots.

3. *Allow users to view query results* (**[r] [pc] [ci]**) – in some cases, it is sufficient to highlight the concept, but in other cases it may be important to show the user what context the result is in, and perhaps why certain results were ranked lower – to allow users to determine what they 'meant' by a search.

4. *Provide a mechanism for saving, annotating, and sharing views* (**[s] [r] [ci]**) – many times we noticed users wanting to share their recent work to show others or save intermediate results. For example, it was noticed that many users wished for a way to save intermediate modeling in the event that they were forced to abandon it. At the NCI, since the elements in the domain of discourse (cancer genetics) change so frequently, it was not uncommon for the modeling staff to restructure entire subtrees. The modeler with chief responsibility for a particular sub-ontology expressed a desire to share his intermediate efforts with the other experts.

⋆ **Modeling** – support modelers in their tasks of constructing the knowledge base or ontology.

5. *Provide graphical editing techniques* (**[r] [pc]**) – allow users to directly edit

concepts in an ontology. The tool should maintain the user's context, allowing them to see how the concept being edited relates to neighbours.

6. *Editing navigation* (**[ci]**) – support for "jumping" between two concepts as they are being joined by a relationship during knowledge modeling. During observations of the FMA team, modelers were continually visiting several detailed parts of a large ontology which were difficult to navigate between. For example, the `Lung` and `Chest Cavity` concepts were in very different areas of the `is-a` hierarchy, but needed to be joined directly by the `part-of` relationship. Creating and verifying that relationship involved using text searching to switch between the two concepts.

7. *Support ontology reuse* (**[s] [r] [pc] [ci]**) – allow users to quickly and easily identify external ontologies and knowledge bases that may be useful, and directly include these in the model; furthermore, the namespace model that many representations are using, based on the World Wide Web Consortium specification, should be clear and explicit.

⋆ **Verification** – support for the visual checking of a knowledge base's structure, such as the model-model and model-domain consistency.

8. *Identify incoming relationships* (**[pc] [ci]**) – the tool should provide support for the identification of a concept's incoming relationships, or the lack thereof, during model verification. It is difficult to identify such concepts that depend on other concepts without doing text searches and navigating to each of the results individually. This can be quite cumbersome if the interface does not have specific support for such a task. For example, a modeler may wish to know what concepts link to the one being examined, to determine what effect changes will have. In frame-based systems, it is not difficult to detect outgoing relationships as they are explicitly defined as part of the concept definition.

9. *Incremental navigation* (**[r] [ci]**) – provide support for incremental navigation of a knowledge base. A tool providing this would be able to show the modeler

all nodes which are *N* slots out from the central concept. Here, the concept is clearly of interest, and the user wishes to start investigating from there.

10. *Browse multiple and complex relationships at once* (**[ci]**) – the tool should provide: complex relationship support, including support for understanding a concept's place in multiple relationships simultaneously, to help verify newly-modeled concepts; and support abstraction of reified relationships. For instance, the FMA modelers reify the `part-of` relationship based on the 'type' of part a concept is, such as anatomical or physical.

Category	**Task #**	**Program Compr.**	**Contextual Inq.**	**Readings**	**Survey**
Navigation	1	x	x (NCI, FMA)	x ([19], [84], [12], [62])	x
	2	-	x (NCI, FMA)	x ([84],[12], [62])	-
	3	x	x (FMA)	x ([12])	-
	4	-	-	x ([19], [62])	x
Modeling	5	x	-	-	x
	6	-	x (FMA)	-	-
	7	x	x (NCI, FMA)	x ([19], [12])	x
Verification	8	x	x(NCI, FMA)	-	-
	9	-	x (FMA)	x ([62])	-
	10	-	x (FMA)	-	-

Table 3.1. *Research methods used to derive tasks requiring cognitive support.*

3.2.1 Summary

Table 3.1 succinctly describes the list of tasks I have identified, as well as their provenance. Task numbers refer to the description given above. An **x** indicates the task was determined from that research method, and parentheses indicate the origin (literature reference or project). While not necessarily complete, this list of tasks serves as a useful model for conducting analysis of tools supporting the knowledge engineering process. The following section applies this analysis to a specific set of tools, those providing advanced visual interfaces for Protégé.

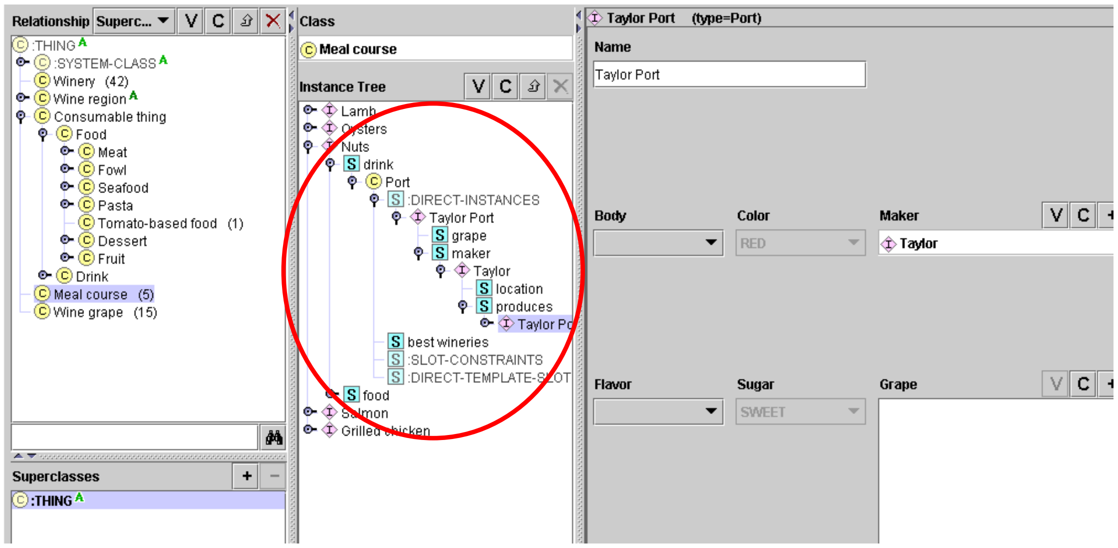
3.3 Approaches to cognitive support

Many tools already support some of these areas I have identified. Protégé itself provides extensive cognitive support for navigation, modeling, and verification tasks, otherwise it would not be as successful as it is. As mentioned in §1.3 and in more detail in §2.2.3, I have chosen to focus on tools which provide cognitive support using information visualization to construct an advanced visual interface (as well as using core Protégé as a baseline reference). The recently released Ez-Owl tab, described briefly in §2.2.2, does allow editing, but I did not examine this in detail because it is focused solely on OWL ontologies, which have different semantics from the frame-based ontologies with which these five tools work.

General knowledge engineering tools were mentioned in §2.2.2. What follows is a detailed analysis of the existing visualizations provided in Protégé at the time of writing, and a summary of the essential features of each tool. Next, each tool is evaluated using the tasks identified in §3.2 and Table 3.1; Table 3.2 summarizes these findings. To show examples of each tool, a well-known ontology is used. This ontology describes the domain of wines, first used in a paper describing the CLASSIC knowledge representation system [14], and now also in other examples, such as the Web Ontology Language guide (see `http://www.w3.org/TR/owl-guide/`).

3.3.1 Protégé core

Protégé itself has a fairly standard indented tree view interface, as shown in Fig. 2.2. Users can navigate the class tree by opening and closing class concepts, or by using a search panel located at the bottom of the screen. This default view supports several of the tasks I identified, including top-down browsing (task #1) and search visualization (task #3), to varying degrees. The chief limitation of the default Protégé interface makes it very difficult to grasp more complex relationships, such as multiple inheritance or complex slot compositions (as used in reification). Particularly interesting is the lack of support in core



Name: Taylor Port

Body:

Color: RED

Maker: Taylor

Flavor:

Sugar: SWEET

Grape:

Figure 3.1. *Instance Tree tab for Protégé, supporting slot-based browsing*

Protégé for the verification tasks. No methods are provided, for example, to understand complex relationships, or to see dependent concepts for a particular artifact of interest.

3.3.2 Instance Tree widget

This tool (shown in Fig. 3.1) extends the default Protégé interface. It allows users to browse through the classes using the slot values (related classes) defined in a particular concept, providing a means to perform slot-based browsing (task #2), as well as navigating between nodes during editing (task #6). Some support is also provided for navigating to arbitrary levels in the hierarchy. As a plug-in for Protégé, the widget leverages the other capabilities of Protégé to handle more complex tasks.

3.3.3 Ontoviz

Ontoviz [75] relies on the GraphViz graph viewer to visualize ontologies in Protégé (see Fig. 3.2). Exploring the ontology is very difficult in Ontoviz, as the user interactions are restricted to panning and simple zooming (navigation tasks for top-down browsing (task #1)). Furthermore, OntoViz does not support more than one layout, which can only be based on the inheritance relationship. Presentation customization (a concept described in

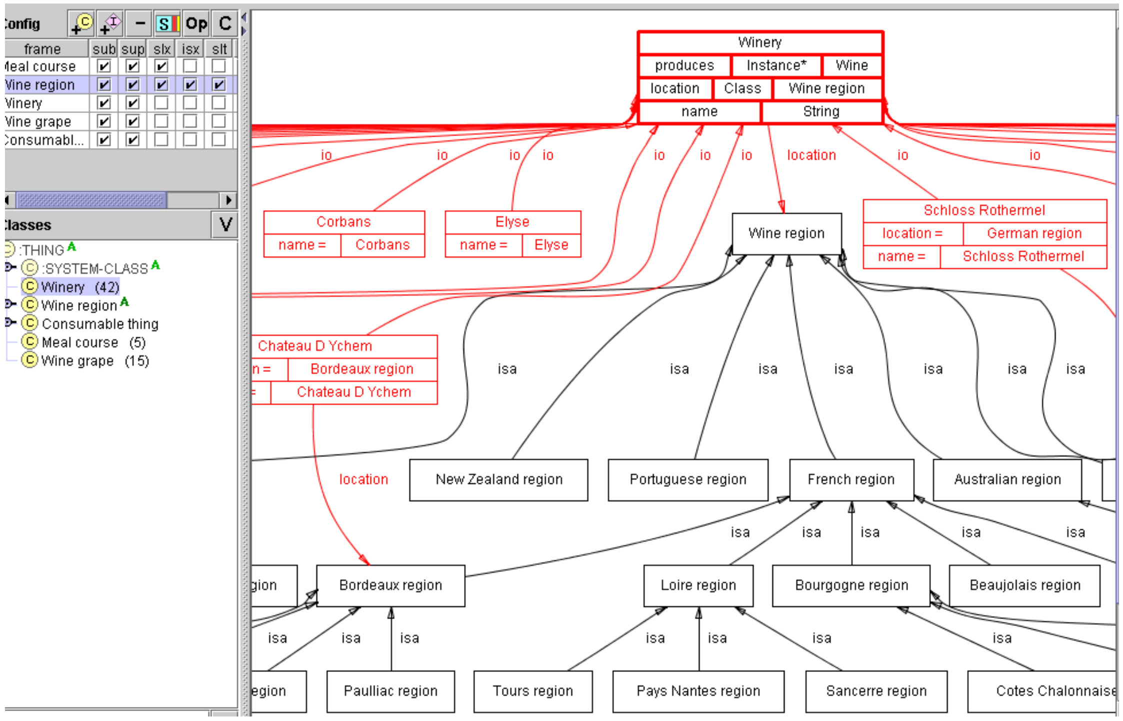


Figure 3.2. *Ontoviz plug-in for Protégé, showing a portion of the wines ontology*

§2.4) is limited and the visualization does not scale beyond a few hundred entities. Ontoviz, as shown by the task support it provides, lacks the support typical of more interactive tools; for example, one cannot browse multiple relationships.

3.3.4 TGVizTab

First released in 2002 by Harith Alani [2], TGVizTab (Fig. 3.3) makes use of a spring-embedding algorithm [33] to implement a customizable layout for concepts and relationships. The true strength of TGVizTab lies in its salience. The graph shown is typically readily apparent to users. Some of the weaknesses of TGVizTab concern the difficulty of seeing all relationships (e.g., non-structural relationships), screen clutter, and the difficulty in synchronizing this view with the Protégé view. TGVizTab does support showing search results, and is excellent for incremental browsing of the ontology. The user can easily turn on or off certain nodes and arcs for an exploration of interest. It is not great at top-down

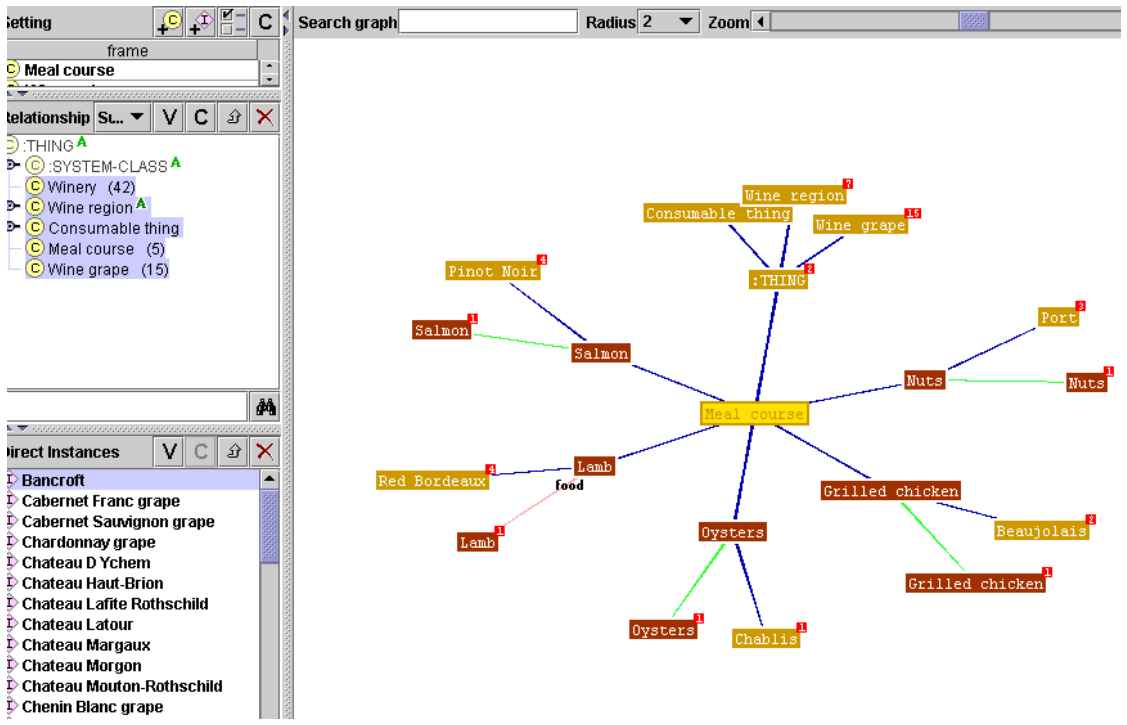


Figure 3.3. *TGVizTab plug-in for Protégé, using a hyperbolic layout on the wines ontology*

exploration, because the graph is fairly unordered. One can save views, but only in the Touchgraph (XML) format.

3.3.5 Jambalaya

Jambalaya [80], developed by my research group and shown in Fig. 3.4, is a suite of tools and views for viewing ontologies with graph metaphors; there are several different mechanisms for viewing data in Jambalaya. As mentioned, Jambalaya uses a graphical language based on the mathematical theory of graphs to map from the Protégé meta-model (based on frame-based knowledge representations) to a visual representation. Jambalaya provides several different mechanisms supporting navigation tasks, although browsing search results is not always simple. Like the other tools, Jambalaya has little or no support for editing tasks. In the verification tasks, Jambalaya does fairly well; one can abstract information and search for arbitrary levels of interest, and identifying incoming relationships is easy as

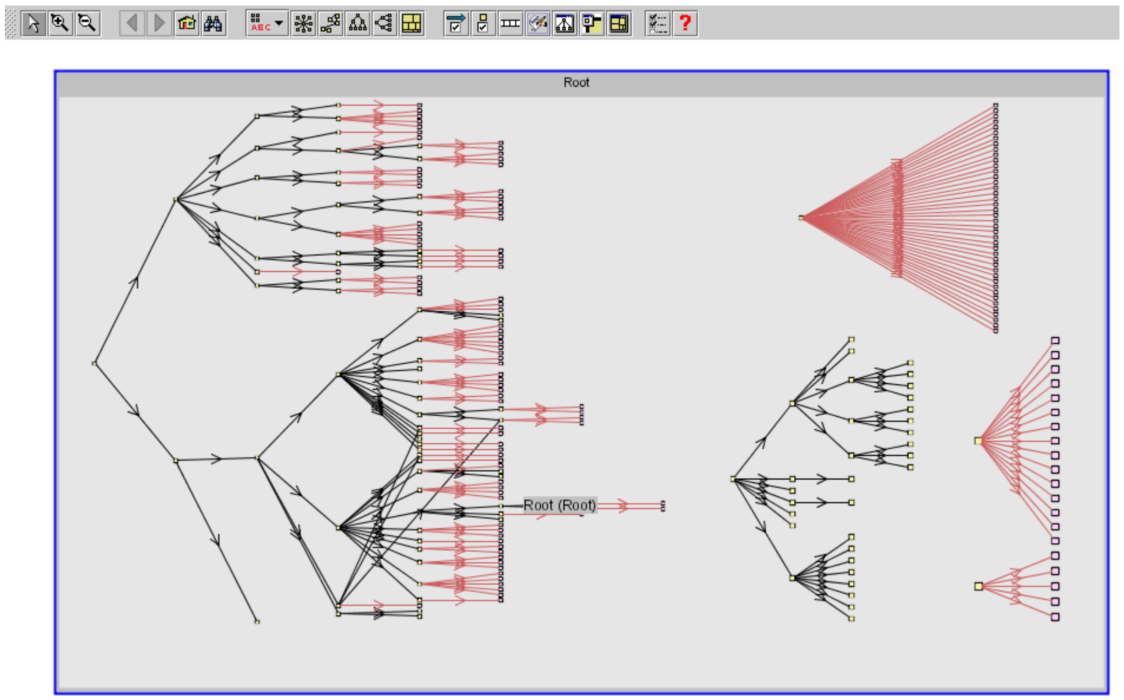


Figure 3.4. *Jambalaya plug-in for Protégé, showing the concepts and relations in the wines ontology*

well. While there is no support for incremental navigation, there is rudimentary support for complex relationships: one can see which hierarchies a node belongs to, but there is no support for reification.

3.3.6 Summary

Based on the areas where cognitive support is required (as listed in §3.2), I conclude certain things about the five tools examined. The tools do not support editing tasks, particularly graphical modeling; this probably reflects the focus on ontology understanding these tools have. While all tools provide different but complementary approaches to navigating and verifying domain models, the lack of certain cognitive support hinders the usefulness for real-world users. Table 3.2 outlines the cognitive support offered by the five tools.

Category	Area of Cognitive Support	Protégé	Inst. Tree	Ontoviz	TGViz	Jamb.
Navigation	1. Overviews	x	p	x	p	x
	2. Slot-based browsing	p	p	-	x	x
	3. Show queries	p	-	-	x	p
	4. Save views	-	-	x	p	x
Editing	5. Graphical editing	x	-	-	-	-
	6. Editing navigation	-	x	-	-	-
	7. Ontology reuse	p	-	-	-	-
Verification	8. Incoming relations	-	-	x	x	x
	9. Incremental navigation	-	x	-	x	-
	10. Multiple relations	-	-	-	-	p

Table 3.2. *List of Protégé and its extensions evaluated against knowledge engineering tasks. An* ***x*** *indicates the support was provided in that tool, a* ***p*** *that there was partial support, and a dash that there was no support.*

3.4 Evaluating cognitive support using design goals

I described some tasks in knowledge engineering which require cognitive support and outlined them in Table 3.1. I then examined the cognitive support that was provided by a popular knowledge engineering tool, Protégé, and some of its related extensions (as shown in Table 3.2). This section presents some thoughts on how designers can evaluate the potential cognitive support in a particular tool (and for a particular domain).

3.4.1 Trade-offs in the design process

Non-functional requirements ([59], [13]), also known as quality attributes, constraints, and effectiveness criteria, are high-level objectives such as scalability, usability, and customizability, which can be thought of as goals which guide the design of features in a system, particularly in terms of what support is provided for the functional requirements or tasks [18].

These goals are often conflicting, in much the same way as design goals for knowledge representation languages are. For example, just as language designers must balance the goals of expressiveness and tractability, designers of cognitive support tools for knowledge engineering must balance scalability and usability. As such, design of these tools can be thought of as a series of trade-offs and architectural decisions which attempt to balance these goals. Choosing which goals to focus development efforts on involves a series of choices about which goals are more important to the tool, as well as any domain-specific needs. The level of focus determines to what extent each goal is met, or at least, to what extent the developers care about it (as it may not be met). Goals exist on a continuum, with a range of concerns possible.

Some examples of such trade-offs are listed in [1], which details certain trade-offs that visualizations of knowledge-based systems have to contend with. These trade-offs cover both functional task requirements and non-functional design goals of the tool. For instance, a tool should provide *human readable* and comprehensible information and views, while still allowing for semantically rich, *machine-processable* data. Both can be considered functional tasks the tool needs to address. Showing sufficient *local detail* while still giving the user adequate *global context* requires that designers understand what focus their tool will have for these functional tasks—customizability is an important consideration for this trade-off. In a similar vein, a tool should give an *overview* to abstract the underlying *complexity*, but sometimes that complexity is necessary and essential. Sometimes what is important is the *structure* of the model; at other times, the *content* is more important. As mentioned in §3.4.6, users typically want systems which can handle large problems—*scalability*—but can still maintain adequate response times (*usability*), again a trade-off in design goals. A system should also maintain its *flexibility* to support the user's unique mental approach to his task, but it should strive to maintain *consistency* as well - for example, always showing key elements at the same locations. This represents a trade-off in non-functional design constraints and must be decided based on the domain the tool will address. Finally, another non-functional trade-off is that in some cases, *known metaphors* are easily understood and most usable, but at times only *innovative approaches* can address the task.

These design trade-offs illustrate the complexity of designing a tool which provides cognitive support for knowledge engineering tasks. Not only should those tasks be identified and supported, but developers must keep in mind the non-functional goals of the tool, such as how well it scales, whether it can be adapted, and how usable its interface is. Clearly, this set of factors produces a large number of solutions which can be created. Based on the findings of this thesis, I believe that in practice a well-designed tool, with a carefully determined set of functionality, will be able to accommodate many disparate domains and users.

3.4.2 Five important design goals

To elaborate on what these goals consist of, I've chosen five of the most important goals discussed above. The next section explains them and how they impact the provision of cognitive support in knowledge engineering tools. These goals are:

- Usability
- Learnability
- Expressivity
- Scalability
- Customizability and extensibility

I refer only to how Jambalaya meets these goals in this section, although they could be applied to all the tools I have looked at. There is no definitive list of non-functional requirements, as this is antithetical to what they represent—domain specific goals that are often only identified as the system is implemented (suggesting an iterative, spiral model ([13]) of development might be best).

3.4.3 Usability

An important criteria is that of usability, which supports users staying 'in the flow' [7], that is, staying on-task and not being unnecessarily distracted with interface issues. Usability affects tools by constraining the complexity and variety of metaphors and techniques a tool can use to provide the cognitive support; there is often a direct relationship between scalability and usability. Usability is best assessed by conducting user evaluations. Jambalaya has made some strides towards improved usability, including some preliminary, unpublished user studies, a heuristic evaluation of the interface, and informal evaluations. Neglecting usability as a design goal often impacts tool use—preventing users from accessing the task-specific functionality, and thereby, the cognitive support we aim to provide. Eisenstadt *et al.*, in [24], also mention some usability principles for knowledge engineering tools, including ensuring there is mapping between what is shown and what is present in the model, and that all views should be coupled so that changes in one are shown in the others. They also mention that tools should be able to abstract and switch views, and objects in a view should be manipulable.

3.4.4 Learnability

This design goal focuses on making the key tools and functionality in the application readily apparent. Whereas usability is focused on making the tool itself easy to use, learnability requires that the functions that will most help a user be salient and visible (also mentioned in [24]). For example, in the NCI domain, a useful visualization in Jambalaya is to perform a spring layout on the `has-biological-process` relation, which shows which genes or proteins play roles in biological processes. Jambalaya does not make this powerful visualization apparent, since its design was focused more on the expressive power of the tool. Learnability is also related to the customizability of a tool, since, as demonstrated, different domains and different users will have various sets of functionality which would be most useful.

3.4.5 Expressivity

Expressivity has typically been used in the sense of formal languages, such as first-order logic, to denote what the language can represent. In tools with user interface components, such as the ones discussed here, expressivity refers to what aspects of a model can be represented in the tool (refer to [19]) for another example). Eisenstadt *et al.* call this completeness—a tool should ideally show all implications of a model (and only those implications). Expressivity often has an inverse relationship with scalability. For example, showing all statements that are implied by a description logic language may not even be computable, so tools working with, for example, OWL ontologies, need to be aware of these constraints. In Protégé, there is also a notion of system classes, which are meta-objects used to construct a domain of discourse. Initially Jambalaya did not represent these objects, yet they can have major implications for modelers: the FMA project, for example, uses metaclasses extensively (see §3.1.2.2). Jambalaya now addresses the non-functional goal of expressivity by allowing users to determine whether they wish to see these classes. This improves the potential cognitive support of the tool by offering more features to match tasks the user may have. However, it is still not clear to us how to support the user in knowing which concepts to render in a visualization.

3.4.6 Scalability and responsiveness

A very important criteria in many tools in the knowledge engineering domain is scalability, the ability of a tool to handle large problems. A user will not see adequate cognitive support if a tool cannot deal with the problem of interest, which may be quite large; the NCI ontology consists of approximately 40,000 concepts. We are working on improving the responsiveness of Jambalaya so that using it with reasonable sized ontologies (currently around one hundred thousand artifacts) does not affect one's work patterns. We are also examining how different visualization metaphors and toolkits can be used to rapidly visualize changes between versions of ontologies (to better support navigation and verification tasks). Jambalaya provides a range of views that the user can select from, rather than one or two metaphors. The reason we focus less on innovative views on the data is that we believe that the techniques are not the major hurdle in people's use of the tool; often, the issue is related to one of the design goals I have discussed, such as scalability or usability. Scale can have a negative impact on how a metaphor is used, and much of the design challenge lies not in determining what useful metaphors might be, but rather, in how to adapt metaphors for different problems (for example, by using abstraction techniques).

3.4.7 Customizability and extensibility

Users differ, even within themselves. Thus, providing cognitive support to users requires addressing this variability. The *customizability* criteria measures how well a tool allows an end-user (in this domain, a modeler) to tailor a tool's functional support to his or her needs. It is very difficult for designers to determine what cognitive support for which tasks is necessary in all cases. Jambalaya partly addresses the customizability goal with rudimentary scripting support to provide access to the application domain. This customization may take the form of hiding additional complexity that is not needed, by removing unwanted tools or views, or by writing custom scripts. In the next chapter, I describe a prototype implementation of customization support in Jambalaya. A user-modifiable ontology describes the Jambalaya tool domain (the set of visualizations, tools, and elements in Jambalaya) and allows the application to be customized by either end-users (modelers at the NCI, for example) or their technically proficient colleagues (who could be termed Customizers [55]).

Our research group is also interested in making Jambalaya itself more flexible at the source code level by adding some form of extension points, similar to the extension features Protégé currently offers. This could allow, for instance, other teams to develop plug-ins for Jambalaya that offer new views or layouts. We would like to take some ideas from the CODE4 tool mentioned in the background section, such as fully separating what is being viewed from how it is viewed, and incorporating non-graph visualizations. While CODE4 did not support an extensibility goal, incorporating its notions of model and view separation will provide a powerful way of addressing extensibility in Jambalaya.

3.5 Summary

I have explored various tools and how they addressed cognitive support. I then proposed some design goals for these tools such as scalability, expressivity, and learnability, mentioning that implementing these goals involves trade-offs. When adoption is used as a metric for successful design, the design goal that can make the biggest impact in knowledge engineering tools is that of customizability.

Focusing on customization has one main advantage. Despite some commonalities, such as the need for slot-based browsing, editing support, and incremental navigation (refer to §3.1), individual domains require subtle yet important changes to standard functionality. Visual interfaces are difficult and time-consuming to construct. In order to have a realistic chance of being adopted, this design must be done on a case by case basis. For example, the visualizations needed by the NCI team will be different than those needed by the Foundational Model of Anatomy team. The FMA modelers need a tool which can show complex relationships between meta-model and model, a tool which shows modeling errors of commission or omission. While requiring similar task support, the NCI modelers are more interested in the location of new concepts in the hierarchies. This distinction involves differences in the degree to which a tool supports specific tasks.

The notion of domain-dependent tasks may seem to contradict my earlier suggestions that there are certain tasks common to knowledge engineering which all tools should support. However, this is not so. While there are tasks common to all projects, such as top-down navigation or slot-based browsing, it is in the details of *how* these tasks are supported, and, as mentioned in the preceding chapter, to what degree design goals are examined, that determine what a tool will look like, and to what degree it will be adopted.

Considering customization as a high-priority design goal allows tools to address these different degrees of interest, without having to make these decisions at design time. For example, incorporating mechanisms to support feature selection and enhancement allows the tool to be tailored to individual domains. A prototype implementation of this is discussed in the following chapter.

Chapter 4 – Implementing and evaluating customization support in Jambalaya

Customization support removes or at least reduces the need for the tool Designer to make guesses and generalizations about the requirements for a particular domain. An example of a Designer would include the developers of Jambalaya or Ontoviz. Instead, customization allows a local expert to make these decisions. This chapter details the steps involved in making changes to Jambalaya in order to implement such customization support, and then validates this particular approach using experience reports from domain experts.

4.1 Modeling Jambalaya

While Jambalaya fulfilled many of the areas mentioned in the survey of cognitive requirements in knowledge engineering, the rich set of functionality it provides remains largely inaccessible to the majority of users because such features are often hidden behind a complex user interface. Furthermore, as developers, the Jambalaya team are not experts in all the domains the tool is used in; these domains have different needs and tasks, as shown by the site visits and surveys. In order to outline how these different needs might be addressed, a prototype of a customizable form of Jambalaya was designed and implemented. Just what is meant by customizable in this case will be defined in this chapter. It is important to note that this approach does not add more functionality to Jambalaya, because lack of features is not an issue for this tool. This section details how I implemented customization features in Jambalaya. Other papers address modifications to Jambalaya to address other design goals such as scalability [49] and extensibility [10].

4.1.1 Customization in Jambalaya

Customization has many definitions. It is possible to consider customization from a variety of scales, from the smallest (being able to change the language of a dictionary the system provides) to the largest (providing options to completely reconfigure the entire application). In this paper, I defined customization as involving non-source level changes made for or by a user, and extensibility as being source-code modifications typically made by a developer. Customization in this sense refers to changes to either the data, the functionality, or the presentation of a tool, as described in §2.4.

For example, while using the Jambalaya tool, the developers noticed they frequently changed from its default view (nested nodes using the *is-a* relation) to a tree layout showing the *is-a* relation. This action suggests a simple and obvious presentation improvement that would help the usability of the tool (and hence, improve its cognitive support). This shows that offering customization opportunities seems to improve domain-specific tool usability and, in so doing, improves cognitive support for the knowledge engineer. The reason this thesis is not a simple usability study is because, as described in other chapters, I do not propose that I can even identify such problems in many domains. The preceding usability problem was discovered only after long use of the tool in our research lab, and there is no such detailed study in other domains. The appeal of a customization approach is that Designers can provide to domain users (Customizers, in the terminology of [54]) the power to make such changes themselves, just as the Designers do as they come to understand it.

Using the Jambalaya source code, I modified the application to support simple customizations, consisting of presentation customizations in the form of specifying initial views on the graph, and functional customizations in the form of feature selection and addition. To model the possible customizations in Jambalaya, an application ontology was created to describe the possible customizations. An application ontology uses a formal knowledge representation to model the concepts in the Jambalaya tool domain, and is detailed below. To construct the ontology, I make use of the ontology construction steps outlined in Noy and McGuinness [64] and discussed in §2.1.1.1.

4.1.2 Step 1. Outline the domain and scope of the ontology

The customization support in Jambalaya takes the form of an ontology, written in Protégé's representation format, that describes the entire set of graph manipulations and tools that Jambalaya provides, and allows Customizers to modify the ontology directly. These modifications take the form of new instances added to the ontology which merge the Jambalaya application model and the domain-specific ontology. These changes are then reflected in the Jambalaya tool on start-up. For example, one thing users may want to do is to set a new default view, which is possible with this tool. To decide what to include in the framework, I iterated over the ontology with other developers working on the tool to determine the entire set of features and tools Jambalaya provides. This is a valid approach because only the developers at our research lab truly understand the set of features Jambalaya has. I term this Jambalaya ontology the Customizable Visualization Framework (CVF) ontology. The term CVF properly refers to the combination of the ontology/domain model with Jambalaya code enhancements to produce a customization solution for Jambalaya.

Noy and McGuinness also suggest composing 'competency questions' about the domain, in order to verify how true the model is to reality. I have not used this approach because I believe the utility of the tool will best demonstrate how well the model was constructed. That is, if the model failed to capture salient features of Jambalaya, that should present itself as a failure to accomplish some task using the CVF.

Another critique of this approach is that it could as easily be accomplished using a simpler set of configuration options, perhaps consisting of a simple XML file and a Document Type Definition (DTD) or XML Schema. For the current functionality of the CVF, this is certainly a valid comment. However, there were two motivations for deciding to use an ontology. On a practical level it is much easier to use an ontology because that is what Protégé supports, and therefore no extra processing was required other than that provided by Protégé. More importantly, though, I wished to allow for the possibility that this ontology could be extended and modified as Jambalaya changed. An ontology is the best way of capturing the complexities of this tool.

4.1.3 Step 2. Consider other ontologies

Norman [63] defines three models of how a system works (where system is any external device a human can interact with). The designer has a mental model of how it should work, the user has a mental model of how the system is working, and the system itself has a model, which he terms the system image, of what is actually happening (reality). This is shown in Fig. 1.1. Most software has some form of model, whether it is formally specified or not (see §1.3). For example, the Unified Modeling Language is a formalism that represents software artifacts in a standard manner. Various systems, such the Model-Driven Architecture initiative (see `http://www.omg.org/mda/`) and its implementations such as the Eclipse Modeling Framework (see `http://dev.eclipse.org/viewcvs/indextools.cgi/˜checkout˜/emf-home/docs/overview.html`), aim to use formal models to design software systems. These describe design-time approaches, whereas this ontology is capturing knowledge about an existing system, one not constructed with a formal model we can rely on.

Furthermore, the models made to design a system often fail to capture detailed information about user interfaces, leaving these for implementation decisions. However, this is vital to the information the CVF ontology must capture. Some other research has been done into modelling an application's 'beliefs about the world' using ontologies, for example, [45]. There is no one way of capturing an application's ontological commitment, though, since, as shown in Fig. 1.1, the designer's model can never completely match that of the system image. As such, I feel the somewhat *ad-hoc* modeling performed on Jambalaya is best validated by expert analysis, in this case, using the other developers. Since I provide the ontology describing the tool as well, the schema can be altered at any moment to reflect new classes or relations between classes.

4.1.4 Step 3. Enumerate important terms in the ontology

Modelers use this step to make an exhaustive listing of the concepts in the domain. Having used Jambalaya for several years, I was intimately familiar with its feature set. I went

- :view_elements
    - node
    - arc
        - grouped arc
        - grouped node
- :layouts
    - force-directed
        - spring
    - nested
        - alphabetically
        - number of children?
    - space-filling
        - treemap

Table 4.1. *Sample enumeration of elements in Jambalaya*

through all the potential tools and elements in Jambalaya, as well as the interaction elements the tool has (such as buttons and menu items) and proposed a list, similar to that shown in Table 4.1.

4.1.5 Step 4. Define the classes and the class hierarchy

Having enumerated the possible terms in the domain, I now structured them into a class and instance hierarchy using my domain knowledge. This process is by no means simple, and there are modeling decisions to be made at nearly every step. I began with a top-down approach to the problem, taking the central aspects of Jambalaya and then decomposing those. My top-level classes are the tool itself, modeled as “Shrimp_Application - Shrimp_Project - Jambalaya_Project”, so named to reflect the nature of Jambalaya as one product in a line of similar tools (see [10] for more details); and “User”, identifying the people who use the tool. For the application itself, there are the five high-level organizing classes: “Actions”, “Scripts”, “Layouts”, “View_Elements” and “Interface_Elements”. There is also a class termed ‘Future’, which is a useful place to store elements of Jambalaya which may be developed in the future.

I now explore each of these in more detail, but as the purpose of this thesis is the customizations, and not Jambalaya itself, I refer the interested reader to our work on Jambalaya, published in [28] and [80]. Detailed explanations are available online at `http://chiselog.chisel.cs.uvic.ca/manual`.

4.1.5.1 Actions

Actions in Jambalaya refer to things the user can do with the interface to make something happen. For example, clicking a button or selecting a menu item both cause an action to occur. Actions are distinct from the interface element that caused them. There are two action subtypes, filters (Arc and Node) and navigation actions (Back, Forward, Home), as well as direct instances. The actions I've modeled in Jambalaya include the Attribute Panel, Filmstrip, Options Panel, Help, Bookmarks, Hierarchical View, Thumbnail View, TreeMap View, Snapshot, Script Manager, and Search. It should be noted that many of these 'actions' are represented by buttons and menu items in the default Jambalaya interface. Although actions should always be associated with these UI elements, I found it too cumbersome for a Customizer to have to add both a button and the corresponding action. Therefore, when manipulating a user instance as part of the customization process, actions are added and removed, not buttons and menu items. Jambalaya implicitly associates actions with menu items and buttons, so removing an action will force the related button or menu item to be removed as well.

4.1.5.2 Layouts

Layouts are operations on the graph-structured data Jambalaya uses. For example, using a relation to traverse the graph, we can direct the tool to place nodes down the screen in a parent-child pattern, producing a tree (possibly a series of trees, or forest). Typically, and this is shown in the standard Protégé interface, the relationship used is the inheritance hierarchy, which we term the ”– is-a –” relationship. There are four categories of layout used in Jambalaya: force-directed layouts, using a modified Spring layout; tree layouts, including vertical, horizontal, and radial trees; space-filling layouts, currently using the

TreeMap algorithm [74], and a nested layout, where nodes are placed within other nodes to indicate a relationship; and grid layouts, where nodes are ordered in a grid according to name, number of children, number of relationships (outgoing), user-specified attribute, or node type. Not otherwise classified is the UML layout, designed to mimic the style and conventions of UML.

4.1.5.3 Scripts

Jambalaya also supports end-user programming with a scripting engine. This engine takes a text listing of Jambalaya specific interaction commands, such as ”select node x”, and the actions and layouts described above, to perform a series of commands without further user input. The current scripting language used is Javascript, a popular language used most commonly in World Wide Web applications. Scripting allows users, particularly expert users, to represent a common set of navigation and interaction tasks with one action, and then to share this with others. An area of future work is to explore making this process more automatic, much like a macro in common word processors. This concept is broken into System and User scripts, the former representing scripts the developers feel have some commonality across domains, the latter for scripts most useful for this particular application.

4.1.5.4 View Elements

The View Elements concept describes the components on the screen in a Jambalaya view, broken into Nodes, Labels, and Arcs. Labels are associated with nodes, in particular, but could conceivably be used with arcs as well. They are given first-order status to indicate their importance to non-functional design goals such as scalability. Associated with Labels are Label Options, which allow users to specify how the labels are shown. Arcs and Nodes have similar children concepts. Both have a Grouped_(Arc/Node) concept, to reflect the ability in Jambalaya to concatenate several elements into one abstract representative. Both also have a System_(Arc/Node) concept, reflecting the meta-modeling facility of Protégé.

Finally, the arc concept has a sub-concept Reified_Arc to allow users to represent which arcs in their ontology may use reification. Reification allows complex relationships to be modeled as concepts themselves.

4.1.5.5 Interface Elements

Interface elements are components of the UI that users interact with. These typically reflect programming language divisions (in this case, Java's Swing libraries). I have modeled it with three categories: menus, buttons, and icons. Menu bars have individual menu items, and button bars likewise have individual buttons. Menus and buttons have icons, and associated actions (described above). The inclusion of this set of concepts is not strictly necessary to implement customizations, as it would be sufficient for the user to specify the actions and layout he or she wants; the mechanics of this are detailed in §4.3. These concepts were included to attempt to present as complete a model as possible, and reflect an underlying struggle in this model: whether to construct a complete domain ontology of Jambalaya, or focus on an ontology which serves (current) customization issues. For example, it is conceivable that the CVF will be used in future to drive Jambalaya's development in an MDA-style tool; in that case, a complete ontology is necessary.

4.1.5.6 Not included or future work

A portion of the ontology is dedicated to future implementations, as well as elements of the Jambalaya UI that did not have a bearing on the customizations as currently implemented. For example, our potential additional actions, such as a concept explorer (to support slot-based browsing) is included here, as is the ability to change the hierarchy, label options, and navigation mode from a bottom toolbar. The concept of navigating using a fisheye, zoom, or focus mode was not included in this first version of the ontology for simplicity. They would be added in a more complete revision. As well, navigation is a user-specific task which is difficult even for a Customizer, who presumably knows the User well, to ascertain.

4.1.5.7 User concepts

Also included in the ontology is a notion of user types. This serves to distinguish between users who are End Users, and those who require more functionality, like the Customizer or Designer. Since it is up to the Customizer to define the different instances of users, this set of classes currently serve as place-holders. A set of sample instances for each class provide guidance to potential Customizers, giving an example to follow. Specifying a user is shown in §4.3.

4.1.6 Step 5. Define class properties

There are several slots, or relations, that are part of the CVF, and serve to relate the various components of Jambalaya to the user specific customizations. I describe these relationships and give examples of how they are used.

- allowed_layouts - relates a user instance to multiple layout instances, indicating which layouts the user should be allowed to use.
- allowed_tools - the tools or actions a particular user can access.
- display_slots - the relationships in the domain ontology to show on first displaying Jambalaya for this user.
- has_collection - a means to model a collection of elements (an integral part of the RDF specification), denotes a set of elements to use for this ontology, though not currently. For example, what set of nodes make up a Grouped Node.
- has_element - denotes the elements a particular layout should operate on. Not currently used, this option would specify that a certain layout should *only* use the elements it is defined for.
- has_icon - maps a button to an icon instance.
- hierarchy - Used to refer to the nesting relation when the Nested Layout had a special status.

- loaded - a boolean relationship for a user instance, used to indicate whether this particular instance should be loaded
- location - a substitute relation for the RDF concept of URI, this slot indicates where something can be found, for example, where a script is defined.
- initial_layout - defines which layout to begin with for a particular user.
- slot_to_nest_on - defines which relation to use as the structural relation at first
- use_script - maps a custom user action to a particular script to run when that action is run.
- uses_buttons - defines which custom buttons to load for this user.

4.1.7 The CVF ontology: summary

Figure 4.1 shows an overview of the CVF ontology, with the user component in the top right corner. Instances have been excluded for clarity.

4.2 Implementation

Once I had decided on a suitable model of Jambalaya, this model was created using Protégé. Once this was complete, the ontology needed to be integrated into Jambalaya in order to realize the customizations the user would create.

4.2.1 Creating the ontology

Using Protégé, I first attempted to create the CVF project in OWL format, previously described in §2.1.2. The motivation for using OWL was that it provides URI support, namespaces, and the other advantages of Semantic Web technology, as well as being the direction most ontologies seem headed. Using the CVF as a project a developer could include it using Protégé to leverage OWL's inclusion mechanisms, to offer extra definitions (Jambalaya-specific extensions to his own model). Using OWL would also allow the framework to be

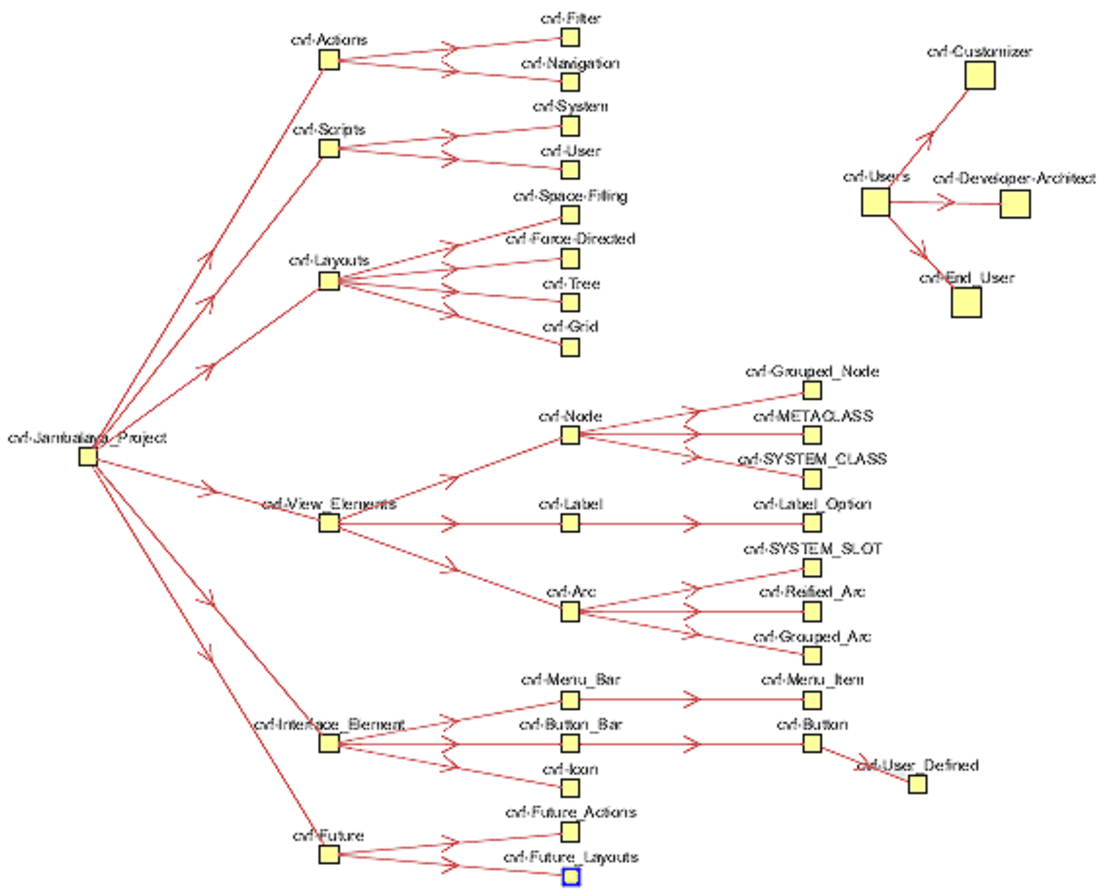


Figure 4.1. *Jambalaya view showing the CVF ontology as a horizontal tree*

imported for other tools which Jambalaya might support in the future (i.e., it isn't tool-specific). I ran into difficulties with OWL support in Protégé as the development of the tool was ongoing while I was developing the ontology. I moved to Protégé's default standard text file with the intention of switching back to using OWL when the plug-in matures, which is anticipated by February 2004. Once this was determined, it was straightforward to implement the class structure defined above.

4.2.2 Integration with Jambalaya

Having constructed an ontology which mapped the properties and concepts of Jambalaya, the CVF now needed some way to determine what the user had specified, and customize Jambalaya accordingly. One option would have been to add a rule engine to Jambalaya, such as Algernon (see `http://algernon-j.sourceforge.net/doc/algernon-protege.html`). This extension provides forward and backward chaining and querying processes on the ontology. Algernon is implemented for frame-based knowledge representations, so implementation in OWL would require the use of an OWL inference engine such as FaCT [44], as well as the support of a query language (none are as yet defined strictly for OWL, but there are several for RDF, the syntactic underpinning of OWL).

Rather than tie the CVF to one specific rule engine or query language, I decided to customize the Jambalaya source code to handle cases where an CVF ontology is loaded. For example, when Jambalaya is added to the Protégé UI, a search is made for the included CVF ontology. If this is found, the extensions to the source code begin assessing which options the modeler specified, and loads the tool accordingly. There are three main categories of the tool the modeler can customize: 1) the starting view and tool that Jambalaya uses, such as which layout to perform, which nodes to show, and which relationship to use; 2) which actions the user should have access to, such as Help menu items, or different types of layout; 3) custom actions, created using Jambalaya's scripting engine to run a set of commands to produce a view.

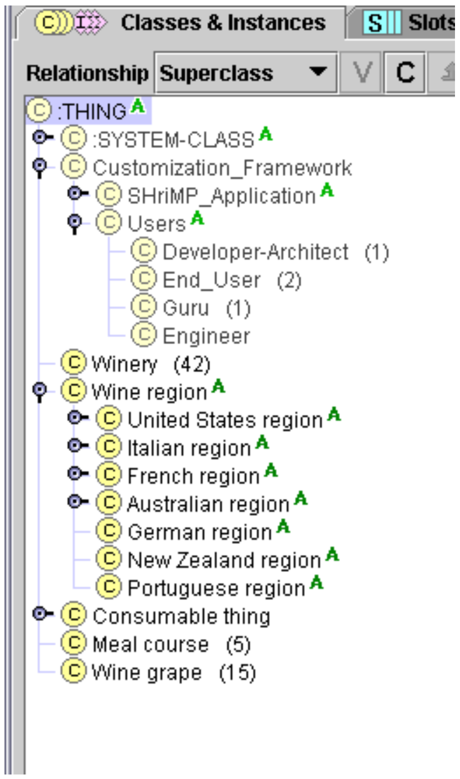
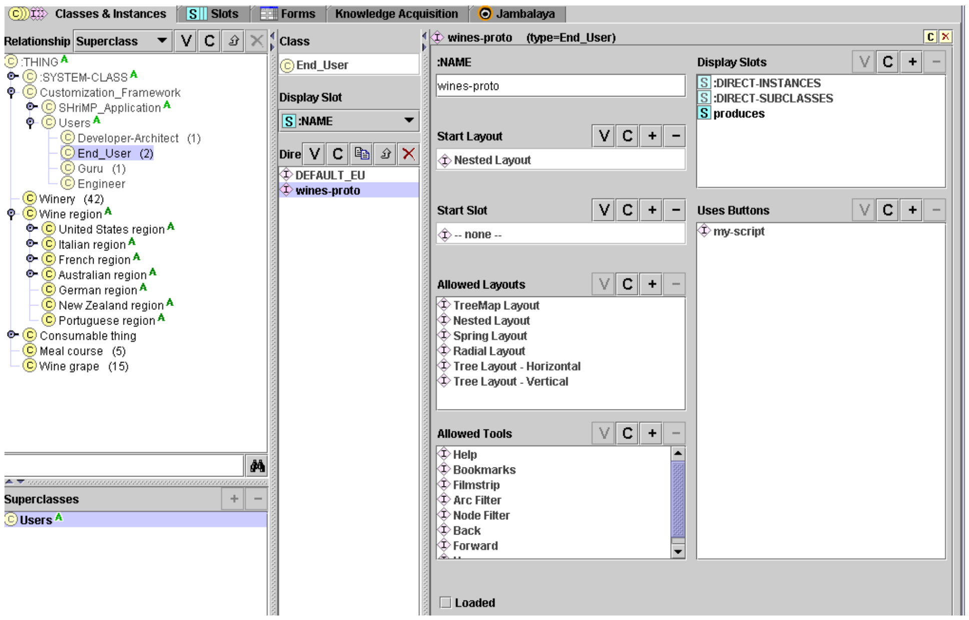


Figure 4.2. *Including the CVF ontology in Protégé. The faded letters for the CVF classes indicate they cannot be modified*

4.3 Interacting with the CVF

How does one use the CVF customization extensions for Jambalaya? The first step is to include the CVF ontology in an ontology project in Protégé, using Protégé's mechanism for inclusion. This process loads the concepts and relationships in the CVF into the domain ontology, and saves the domain ontology anew. The class panel of Protégé now looks like Fig. 4.2. The user, who we take to be the Customizer, according to Michaud's customization framework [54], can now interact with the CVF without fear of altering its core model in any way. Should Jambalaya add a new feature, all that is required is for the user to replace the old CVF ontology files in her directory with the new files. The changes will be automatically reflected in her Protégé project. This stratagem completely eliminates inter-ontology coupling, allowing simple updates with no concerns over dependencies.



Relationship: Superclass

- :THING
  - :SYSTEM-CLASS
  - Customization_Framework
    - SHriMP_Application
    - Users
      - Developer-Architect (1)
      - End_User (2)
      - Guru (1)
      - Engineer
  - Winery (42)
  - Wine region
    - United States region
    - Italian region
    - French region
    - Australian region
    - German region
    - New Zealand region
    - Portuguese region
  - Consumable thing
  - Meal course (5)
  - Wine grape (15)

Superclasses: Users

Class: End_User

Display Slot: :NAME

Direct instances: DEFAULT_EU, wines-proto

wines-proto (type=End_User)

:NAME: wines-proto

Start Layout: Nested Layout

Start Slot: -- none --

Allowed Layouts: TreeMap Layout, Nested Layout, Spring Layout, Radial Layout, Tree Layout - Horizontal, Tree Layout - Vertical

Allowed Tools: Help, Bookmarks, Filmstrip, Arc Filter, Node Filter, Back, Forward

Display Slots: :DIRECT-INSTANCES, :DIRECT-SUBCLASSES, produces

Uses Buttons: my-script

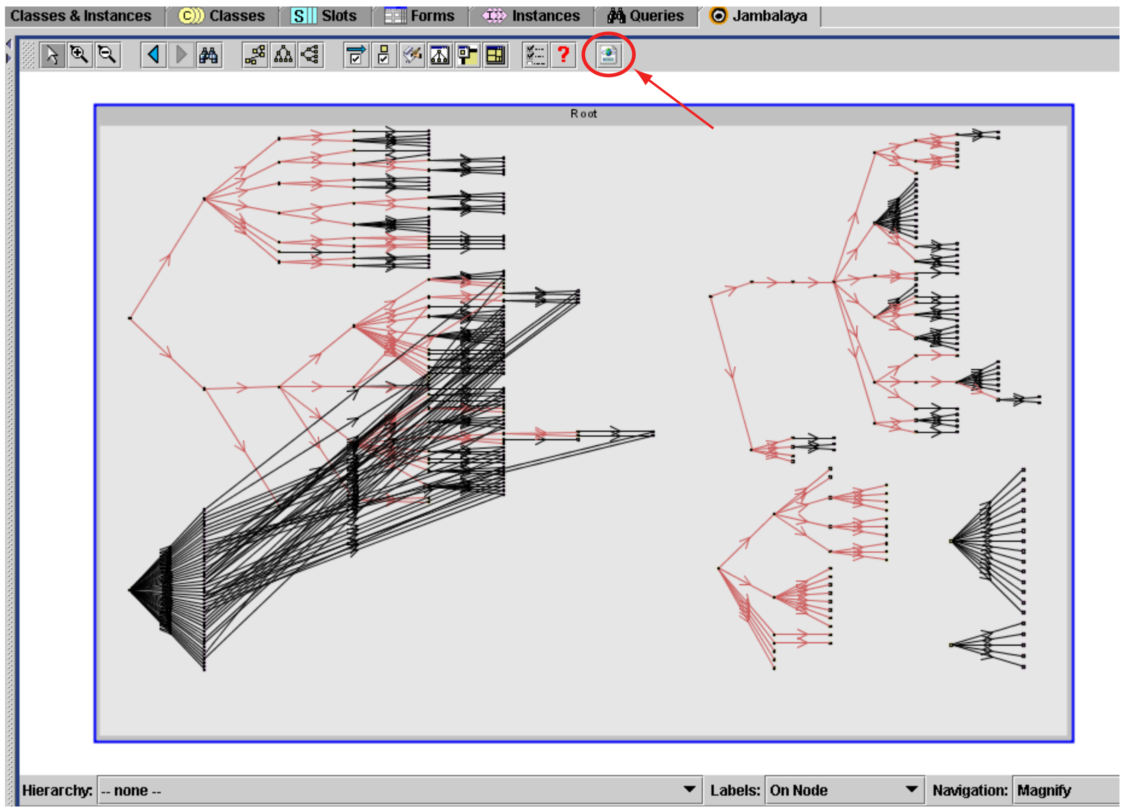
Loaded

Figure 4.3. *The user instance creation form in the CVF. New instances can be created and options set using this form.*

To use the CVF to customize the ontology, the Customizer navigates to the user panels as seen in Fig. 4.3, and creates a new user from one of the pre-defined abstract user classes. In the example shown, the Customizer has decided to create a new End-User instance.

Using the form, the Customizer now combines his knowledge of the Jambalaya tool with his knowledge of the domain (in this case, wines), to create a meaningful initial layout, the actions needed, and any custom scripts an end-user in his group may need. This last sentence describes what I feel is the power of the CVF framework. In this scenario, the person most able to make accurate and useful decisions is empowered to do so, and the developers of the tool are left to do what they are best able to: creating a suite of useful actions, layouts, and tools for manipulating arbitrary graph-based data.

An important point about the CVF is that it makes some assumptions about the user's abilities or knowledge with Jambalaya: if there is something in the CVF that is incompatible with Jambalaya, it is up to the user to determine this. For example, it is not possible to perform a tree-layout on one node in Jambalaya, yet it is possible, depending on the ontology, to make this happen using the CVF customizations. This is the flip-side to allowing a



Hierarchy: -- none --
Labels: On Node
Navigation: Magnify

Figure 4.4. *Jambalaya view showing customizations, such as new button in top-right corner.*

Customizer to change the Jambalaya tool: there is the potential to break Jambalaya, if the user is not fully aware of how Jambalaya works. However, such errors are not fatal, and can be seen as part of the learning process. More detail on this issue is discussed in the next chapter. One resolution may be to include some explanation of why a view was not meaningful, for a limited definition of meaningful, such as only one node showing, no tree layout done, too many arcs on the screen, too many nodes on screen, and so on.

4.4 Results of the customization

Having detailed some specifics for a particular user, the ontology (representing the combined version) is now saved and the Jambalaya tab reloaded. Changes are now apparent when the Jambalaya tab is initialized, and the customizations the user specified are cor-

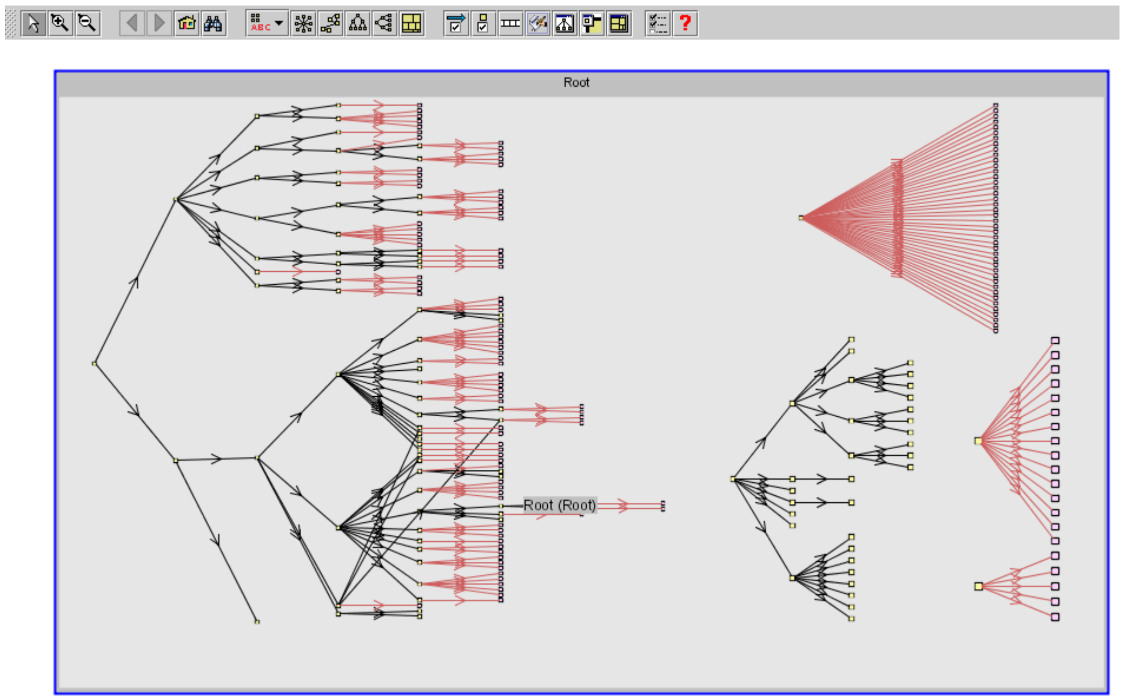


Figure 4.5. *Jambalaya view without customizations. Note number of buttons, and different initial layout.*

rectly implemented in Jambalaya. In Fig. 4.4, for example, we can see the custom button the Customizer specified in the instance form, as well as fewer buttons available than the default interface, representing the removal of those actions. I conjecture that such a customization makes the Jambalaya plug-in more intuitive and simpler to understand, resulting in increased adoption. To assess this claim, I evaluate the CVF using a qualitative approach, discussed in the next section.

4.5 Validating the Prototype and Approach

The conjecture that underpins this thesis is that providing domain experts a means to customize a tool to their unique needs (that is, the needs of the users, and the domain constraints), will increase their perception of its cognitive support. Using the CVF to select or enhance features, or to tailor the user interface, places the burden for determining effective visualizations on the shoulders of the Customizer, and not the Designers.

Whether this leads to increased tool adoption is more difficult to assess. Tool adoption is a very complex problem, with many different models and theories proposed to describe both the technology transfer process and the nature of adoption. As such, this thesis does not aim to show that the CVF model increases tool adoption, because this is a complex process. This complexity is described in the diffusion literature (see [69] or [68] for introductions). For example, in the NCI project, the decision to use or not use Jambalaya (and for that matter Protégé itself) certainly has something to do with their respective merits. However, many other factors will play their part, including social aspects of the organization, competing tools, budget constraints, and of course individual preferences. Rogers [69] outlines some of the factors involved in adoption decisions by organizations. These include

1. knowledge about innovation and the need for change
2. persuasion and evaluation by decision makers
3. decision about acceptance
4. communication of decision to adopters
5. action or implementation of decision (and potential rejection)

Currently the status of the CVF with the NCI is at the second phase, which shows how complex and lengthy the process can be. The other problem with examining the success or failure of the CVF conjecture in the NCI context is that it would be quite difficult to extract the personal qualities of the individuals involved so as to produce an unbiased case study. For instance, part of the reason for adopting the tool may have to do with evangelizing on the part of the motivated investigator (the author). Therefore, the validation of this conjecture cannot be done in the NCI adoption context; that is, the relative merits of this conjecture cannot be assessed in this particular aspect. However, while the adoption/non-adoption of Jambalaya using the CVF by the NCI does not indicate whether the particular approach is worthwhile, there are certain aspects that we can assess independent of this adoption criteria.

4.6 Selection of validation technique

What, then, would be an effective (i.e., tractable) and meaningful (allows conclusions to be drawn about the approach and implementation) validation of the CVF conjecture? Mary Shaw, in [72], describes a taxonomy of research techniques and the means for validating them. Her taxonomy gives five chief methods for validating research (she writes for the software engineering domain, but her analysis is applicable in this domain as well). These types are analysis, evaluation, experience, example, and persuasion (as well as 'blatant assertion', which will not be used!). Analysis typically suggests some empiricism using controlled experiment or formal proof, not valid here because of the qualitative, exploratory nature of the work. Evaluation suggests that one has examined the results and determined the hypothesis accounts for what is occurring. This work is concerned with suggesting useful hypotheses and does not propose to defend a particular one. Experience best fits what this thesis is trying to accomplish, being a narrative or comparison of real-world problems to demonstrate effectiveness. The work also proposes an example of how to accomplish certain goals. Finally, the persuasion method of validation consists of proposing a hypothetical solution and endeavoring to convince others of its suitability, but such a validation is not adequate for a project of this size.

As shown, each validation type is best suited to a particular research objective. For this thesis, the project takes the form of a generalization of using domain and tool models in combination to customize visualization techniques, as well as an exploration of whether this approach has any merit. Generalization refers to the work of this thesis in summarizing the problems in this space and raising them to a higher level of awareness. Using this

Type of result	Examples
Procedure or technique	New or better way to do some task, such as design, implementation, maintenance, measurement, evaluation, selection from alternatives; includes techniques for implementation, representation, management, and analysis; a technique should be operational—not advice or guidelines, but a procedure
Qualitative or descriptive model	Structure or taxonomy for a problem area; architectural style, framework, or design pattern; non-formal domain analysis, well-grounded checklists, well-argued informal generalizations, guidance for integrating other results, well-organized interesting observations
Empirical model	Empirical predictive model based on observed data
Analytic model	Structural model that permits formal analysis or automatic manipulation
Tool or notation	Implemented tool that embodies a technique; formal language to support a technique or model (should have a calculus, semantics, or other basis for computing or doing inference)
Specific solution, prototype, answer, or judgment	Solution to application problem that shows application of SE principles – may be design, prototype, or full implementation; careful analysis of a system or its development, result of a specific analysis, evaluation, or comparison
Report	Interesting observations, rules of thumb, but not sufficiently general or systematic to rise to the level of a descriptive model.

Figure 4.6. *Types of software engineering research results ([72], p. 4)*

characterization, the thesis then outlined a feasibility study (implementation) to demonstrate whether this technique was even useful or possible. I would classify this work as a procedure or technique to improve adoption of complex visual interface (as per Fig. 4.6). For the qualitative model (the model of how people customize software provided by Jeff Michaud [54] and applied to this problem) we can analyze the evidence for effectiveness using a narrative, qualitative report by the users. I also provide a description of how this implementation worked for a real-world task.

4.7 Validation technique: implementation report

In order to assess some of the technical aspects of using the CVF, I tested it using a small sample ontology, the wines example from the preceding chapter. The CVF was used to customize views of the wines ontology for different users. For end users, I removed features I deemed confusing for the wines demo, such as Script Manager, Hierarchical Views, and so on. I initiated a default layout which would show an overview of the ontology in horizontal tree layout, and added some scripts via buttons that would allow users to navigate to specific nodes, change the nesting relation, and return to a tree layout. This user instance was included as a default example with the copy of the CVF I sent to people evaluating my approach. I did this to provide people a starting point to identify what was occurring in the customizations, and what some useful choices might be (as some users find it easier to follow a path and deviate, rather than beginning anew).

4.8 Validation technique: experience report

The experimental set-up for the validation involved using domain experts in the CVF target audience (larger knowledge-engineering projects involving 2 or more users). I selected five such experts, as well as one pilot user, and requested that they attempt to use the Jambalaya tool, with which all were previously familiar to varying degrees, having downloaded it in the past or seen demonstrations of it. The five experts also all have backgrounds in knowledge engineering with Protégé, with at least two years of experience, and in most cases more. Selecting users who were already familiar with Jambalaya has its faults, however. This approach risks bias from the users towards a tool they may appreciate or dislike. The questionnaire endeavoured to emphasize the fact that the approach taken with the CVF is merely exploratory and proposed, and highlighted the fact that the approach was aimed at addressing some of the known issues with Jambalaya. Thus, users were expected to set aside their previous interpretations of the tool and evaluate the CVF as a proof-of-concept approach. This was the only means available to deal with this tool bias, and was not entirely successful.

4.8.1 Initial contact and questionnaire

Prospective evaluators were emailed a message as shown in Appendix A, after first testing the approach on a colleague as an initial iteration. Due to busy schedules, two users were unable to complete the analysis at the time of this writing. I received three questionnaires in return (in addition to the pilot study). The other evaluators have promised results and these will be included when available. The questionnaire asked respondents to evaluate the CVF by installing it, using it, and then answering the following:

1. Does the customization mechanism meet your expected difficulty level for something of this nature?
2. Does the customization appear to make Jambalaya easier to use? You may want to compare the un-customized version with the new version.
3. What other customizations do you consider important to reduce barriers to using Jambalaya in your domain?
4. What are the biggest barriers, in your mind, to adopting Jambalaya (or other visualization tools) for your domain?
5. Are there other ways to accomplish the goal of increasing the usefulness of generic visualization approaches this methodology may not have brought up?

4.8.2 Pilot User

The pilot user was deeply familiar with the Jambalaya tool, being a developer. She also had adequate experience with Protégé and ontology creation. As maintainer of a small-sized research application leveraging ontologies, she also was familiar with working with different end-users of her ontology. I asked her to complete the evaluation of the CVF by following the instructions in the email, keeping in mind how it could be applied to end-users of her application. Her responses were generally positive. She found the difficulty level “very simple”, but wanted the ability to customize more than the initial layout with the tool. This Customizer has a slightly different mental model of what an End-User may want than my design anticipated, which suggests altering the CVF ontology. For instance, she wanted to be able to present users with several static views (starting points), and then “create buttons to toggle between them”. This user also mentioned that potential adoption barriers in Jambalaya were its “heavyweight nature”; she wondered if a smaller web applet would be a good introduction to the tool. Finally, this user mentioned that the ability to “start somewhere” might be a better approach for users navigating a hierarchy.

4.8.3 User 1

This user initially had difficulty installing the application, due to a pre-existing error in the Jambalaya installation. As part of dealing with this, he sent the ontology he was using. I had the opportunity to examine what he had done and suggest improvements. The reason for my suggestions was that the options he had specified would produce a graph view that showed relatively little. He replied that this was because he had not used Jambalaya that much, more 'dabbled' with it, and commented on the 'steep learning curve'. His suggestion for useful cognitive support in Protégé referred to the Instance-Tree widget described in §3.3.2. Implementing similar functionality in Jambalaya, he felt, would be helpful. This user did not perform the tasks, nor respond directly to the questionnaire.

4.8.4 User 2

This user found the installation trouble-free. Part of the reason for the difference with the first user may be that this user did not have an existing Protégé installation. The user was able to perform all the tasks save adding custom buttons to his instance. He did not provide an explanation for this omission. In response to the questionnaire, this user made the following comments:

1. As to whether the difficulty level was appropriate, the user found it too difficult. He felt more emphasis should be made that the customizations are for Jambalaya users, and suggested an extremely simple instance with no layouts as one option. He recommended taking a set of "beginner", "medium", and "advanced" user instances, and showing users how to copy these to create new values, using Protégé's "copy instance" feature. He also noted that the interface should not permit people to create new layouts, as these are defined by the Designers, not the Customizers.
2. This evaluator felt that the customization options were good, as "one of the problems of [Jambalaya] is its bewildering set of options. Your work can get rid of this." He felt it would be essential to closely integrate the changes in user instances with

immediate feedback in the Jambalaya panel. Currently, such changes appear only after reloading the Jambalaya tab.

3. As to what other customizations might help, this user felt that the customizations should be subtler, and driven by Jambalaya, rather than changing Protégé instances. He suggested a "create new user type" option that operated as a wizard, allowing someone to interactively specify which settings he or she wanted to use.
4. The biggest barrier for this user was Jambalaya's complexity—"Jambalaya is hard to understand and control". He also mentioned software bugs as causing annoyance.
5. Finally, this user believed that the concept of generic visualizations are "tough to sell". To overcome this, he believes one of two things need to be done: "You either need to appeal to pre-conceived visual notions of users or you need to do a good job of educating and helping your users."

4.8.5 User 3

This user had some initial installation problems, but did not report anything beyond that. As to the questionnaire, he had the following comments:

1. He determined that the customization process was of acceptable difficulty, "but I expect it to be difficult for non-ontology-experts". The chief problem was that what each option did (e.g., various layouts) was not explained anywhere, so he was "selecting blindly" while determining the effect on Jambalaya's display.
2. As to whether this customization approach made Jambalaya easier to use, he wasn't convinced, largely because the user needs to learn all the options to select the correct profile. He also suggested that if a user had to understand Jambalaya anyway, it might be easier to make the changes in Jambalaya itself. He saw the benefit in different user profiles, but thought the key was to select the *correct* profile, and prevent the user from making wrong choices. I took his use of "wrong" here to mean unhelpful or useless choices for that user. Finally, he felt making Customizers include separate

ontologies in their projects that are not domain related might “have an undesirable effect on the service results”.

3. He expressed a desire to enable users to also customize other aspects such as zooming speed and label fonts in the customization settings, and not just in Jambalaya’s options panel.
4. The biggest barrier in getting Jambalaya or other visualization solutions adopted he listed as “scale, scale, and scale”.
5. Finally, this evaluator did not have an answer to the last question, although he felt it was an important one.

4.8.6 Discussion and analysis

Several themes can be identified in these experience reports.

First, the evaluators felt that Jambalaya’s complexity would prevent customizations being properly used. Since the pilot user had extensive Jambalaya knowledge, manipulation of the CVF ontology, particularly those aspects requiring significant knowledge of what Jambalaya is capable of, was easy for her. The other users, by comparison, had much less knowledge of Jambalaya. They experienced problems due to lack of knowledge of Jambalaya’s features, such as useful views for a domain. This result indicates that a key assumption of the CVF approach, that the Customizer be experienced with both Jambalaya and his domain, is an extremely challenging one to meet. My conclusion is that Jambalaya as it is distributed remains too complicated and therefore underused, even by technically adept, motivated users, such as the ones in this evaluation.

Secondly, the evaluators either did not understand or did not appreciate the role of a Customizer. This is clear from the comments that mentioned how customization options might be better done in the Jambalaya application itself, rather than as an included ontology. This issue may be due to poor explanation in the questionnaire, or it may be a dismissal of the concept itself.

Finally, another source of problems was the instability and lack of features of a prototype. I term this an ‘implementation bias’ because when presented with a medium or high-fidelity prototype (I classify the current version of the CVF as ’high-fidelity’) users inevitably assume it is stable, thoroughly tested, and complete. This is not the case here, as there are several omissions and bugs still lingering in the implementation. Despite the repeated message, though, people assumed this was yet another feature of Jambalaya, and not a re-examination of the general approach. Although it didn’t happen directly, it is also conceivable that prototypes could receive poor evaluations if they crash, despite the new approach they present.

4.9 Summary

The evaluation techniques chosen were an experience report of my own attempts at using the tool and qualitative reports from expert users. These two methods were preferred over more empirical approaches, such as user studies, because the issue of cognitive support is a highly personal one , and adoption success is difficult to interpret. Furthermore, developing a highly mature tool was beyond the scope of this work. My own experiences allowed me to assess how this approach could actually work, and revealed potential problems. I asked other people their experiences using the CVF, and got them to detail the potential usefulness of the approach. This allowed me to assess how likely it is that the CVF will have a positive impact on adoption, at least in the sense that the CVF made it simpler for these experts to customize the tool to their circumstances. The conclusions from the experts who responded seems to highlight the fact that the Jambalaya tool remains highly complex. This complexity revealed itself in the low numbers of respondents who considered themselves skilled with Jambalaya. In turn, this suggests that the assumption behind this approach, namely, that it most benefits those who have knowledge of both the domain and the tool, will be a difficult one to meet. The following chapter summarizes this in more detail, and presents some potential directions to take this work.

Chapter 5 – Conclusions

Jambalaya sees on average 3-4 downloads per day. This is a respectable amount for a research tool, and suggests a great demand for the tool's represented features. It remains difficult to conclude much else from these statistics, however. As developers, we hear little or nothing from these people, so it is hard to tell whether they are now satisfied, regular users, or were merely experimenting, and have now given up. What is known, however, is the reaction of individuals and organizations we talk to regularly. A good representation of these people was captured in §4.8.2. Their experiences suggest that the complexity of Jambalaya greatly limits its usefulness to users.

Like many research tools, Jambalaya is feature rich and advanced. In this problem space, though, the requirements of end users are much more complex than any one approach can address. For example, User 1 demanded features that Jambalaya has never supported. Attempts to retrofit customization frameworks on it, such as the CVF, must first address these underlying cognitive issues.

5.1 The use of customization

This thesis examined the modeling phase of knowledge engineering methodologies, and analyzed other research in the field to present a preliminary list of tasks that might need cognitive support. Several design goals were presented that tools trying to support these tasks needed to consider. Of these, the customization goal is usually overlooked (often in favour of scalability), and yet seems to present one potential mechanism to encourage knowledge engineers to adopt cognitive support tools. With that in mind, the Customizable Visualization Framework for Jambalaya was created. The CVF created a model of Jambalaya which allowed knowledgeable users, termed Customizers in the parlance of [54], to create domain-specific adaptations of the tool.

That this was possible was shown by the implementation report. That this was feasible was examined in qualitative experience reports by domain experts. These reports were inconclusive; some users felt the approach made sense, but generally, the overwhelming number of features in the underlying Jambalaya tool made users conflate the customization approach with the Jambalaya implementation. As such, it cannot be concluded that the customization approach works for this specific tool. However, these uncertain results lead me nevertheless to conclude that customization is a valuable design goal for these types of tools. This conclusion is based on other research into customization, such as Wendy Mackay's work (see [50] and [51]), as well as the presence of select customization support in major software packages such as Microsoft Office and Eclipse, and the general enthusiasm which the respondents displayed towards the idea. An example of this enthusiasm was the idea, put forth by the NCI, that a customized version of Jambalaya posted on their web page would be an excellent tool for users to download. The customizations would tailor that version of Jambalaya specifically for browsing the NCI ontology.

Plenty of customization work incorporating formal conceptual models (unlike the customization in Microsoft Office, for example) exists in the hypermedia research space. Ontoweaver, designed by Lei, Motta, and Domingue [48] is a similar approach for the domain of hypermedia. Ontoweaver uses a combination of domain, user, and presentation ontologies, together with the JESS inference engine, to present customizable websites to different users. This approach is very similar to mine. For example, the CVF ontology also maps user, domain, and presentation knowledge, albeit in one, and not three ontologies. I implemented the application logic in Java code, and not explicit rules, which makes my approach more difficult to modify, but for this limited prototype, this method is sufficient.

The Ontoweaver work makes no mention of evaluation, save for implementation reports similar to that I provided in Chapter 4. This makes it difficult to assess how well these techniques work. One possible criticism is the overhead additional models introduce. As stated previously, modeling is difficult to do, and requiring Designers to conceive of three models, as in Ontoweaver's case, seems unwieldy. This is perhaps why the implicit conceptual model of Microsoft Word works best. The difference between the CVF and other model-based interfaces is that they do not support the notion of customization by the domain experts, just the Designers. For example, Teallach [5] designs interfaces on databases, so it can be customized by designers for specific applications (like any other GUI builder), but not by the actual users of the tool.

Mackay, in [50], makes no mention of user models or domain models. In this sense, customization describes a user-initiated process. Gantt and Nardi [35] make mention of 'Gurus', users in a particular setting with the ability to customize their tools and extend the benefits to colleagues. They make no mention of formalizing these customization efforts beyond Designers providing examples of customizations. However, their work used a User and Customizer perspective, not a Designer perspective. Michaud and Storey [55] discuss the Designer perspective, but forego formal models.

While the results of the CVF prototype were mixed, the customization aspect is a useful approach. The NCI team, for example, have suggested posting a customized version of the Jambalaya tool on their website, alongside the Thesaurus. They appreciate the idea of customizing views on the data for non-domain experts. Customization is perhaps most useful as a focus for user-centered design efforts. It can help Designers assess what aspects of their tool should or should not be customizable, based on an assessment of the requirements for the user base. For example, for the community Jambalaya was addressing, needs and requirements were quite varied, although commonalities existed. This should suggest to developers that customization is a useful design goal for this space. However, customization needs to be carefully thought out, lest too large a cognitive burden is placed on the Customizers, as was the case with the CVF.

5.2 Enabling customization vs. improving usability

A critique of this work one might expect is, “All you did was improve the tool’s usability”. Usability improvements will almost certainly result in more favorable reviews of the tool, because user complaints have been directly addressed by the designers. It stands to reason that solving user complaints will result in an increased appreciation for the tool. However, the problem with this approach, particularly in this case, is that the usability improvements

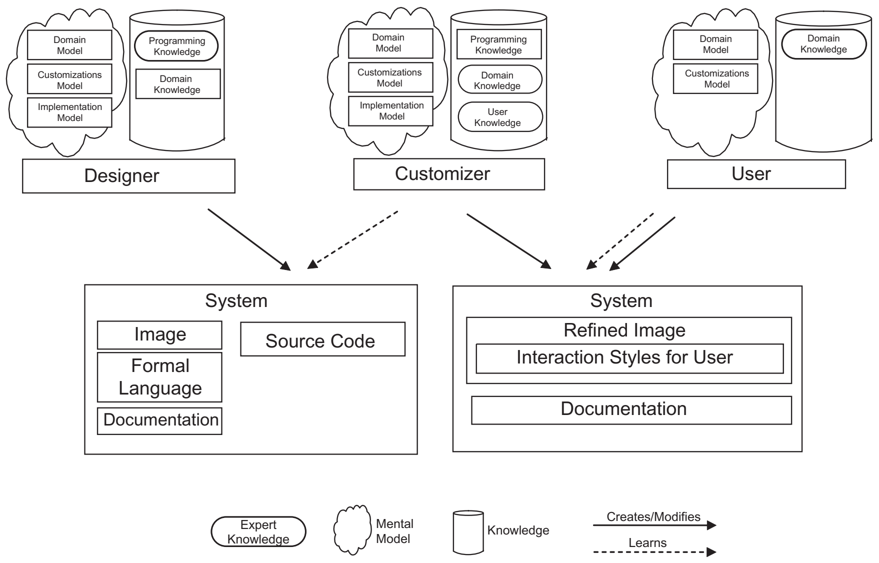


Figure 5.1. *Customization model, showing the relationships between Customizer and Designer, and Customizer and User*

require access by the designers to the users. In the domain of knowledge engineering, however, each set of users may have a fairly unique list of requirements, as demonstrated in Chapter 3. Hence, usability improvements by the designers are unlikely to resolve the issues at a wider scale. Implementing customization features in the tool, however, shifts some of this responsibility for usability to the Customizer. The CVF operates at the intersection between Designer and Customizer, as shown in Fig. 5.1, and this is reflected in the evaluation, as none was performed on the customizations created for Users. While usability studies remain important, the success of the CVF cannot be analyzed solely by the number of users who adopt Jambalaya with customizations; because the CVF is for the Customizer, the success or failure of this person to customize for her domain will largely determine adoption. This second aspect of the tool requires more detailed analysis.

5.3 Why knowledge engineering should care about adoption

Shipman and Marshall, in [73], write extensively about the problems with users and explicit representations, a problem seen in the evaluation of the CVF, and also in the wider knowledge engineering community. They make the case that requiring users to formally represent their knowledge and work (for example, capturing metadata) is likely to be met with a high degree of resistance, a reference to the Knowledge Acquisition bottleneck. They conclude that such efforts are unlikely to be successful. In one study, although the explicit model was successful in reducing long-term costs (by making assumptions explicit, for instance), it nonetheless was not adopted outside the domain of use, and success in that domain was attributed to “social pressure, extensive training, or continuing human facilitation ([73], p. 5)”.

Their paper illustrates the dangers of developing tools without considering the social issues they create. Adoption should therefore play a larger role in the development of all knowledge engineering tools, if only as a metric for assessing tool development. Such a focus has a meta-implication for this thesis. Cognitive support is required for reducing the burden formalizations place on users, and therefore the efforts this thesis analyzed can also be seen in the light of reducing some of the difficulties users face when using general knowledge engineering tools. However, these support tools need also to focus on adoption, particularly by the Designers of the knowledge engineering tools. The CVF implementation, and subsequent evaluation, have shown that the problems of high cognitive load exist in this space as well. While the CVF approach was unique in trying to focus on appealing to a Customizer, or local champion of the technology, because it retained all the functionality of a pre-existing tool, it overwhelmed these champions. Naturally, some tools will not concern themselves with adoption. They exist as experimental efforts, and whether they see widespread use may not be relevant to the developers. For most other tools, however, adoption of the tool provides useful feedback on the approach, tested in real-world settings.

5.4 Cognitive support needs consideration

This thesis has dealt extensively with the interplay of usability, usefulness, and cognitive support. Too few tools elaborate on what they are trying to accomplish, and how to establish that this was in fact done. This might take the form of a list of requirements addressed and a means to assess these were met. Considering cognitive support in tool design means appreciating the roles different aspects of the design process play. For example, tool designers must consider what design goals their tool is trying to address, and why. Is the tool to be highly scalable? Emphasize usability? Or, as in this work, enable customizability? Thinking of these different goals will help the designers clearly understand what is being achieved.

Also important is a good understanding of the requirements the tool is trying to address. What functionality will be included? Why? What knowledge engineering tasks will these functions help with? What user community is being targeted? Is this tool focused on one particular set of users? If so, this should be mentioned in the documentation. Is a generic tool being designed (like Jambalaya)? Perhaps customizability and usability should be big focuses.

Finally, comprehensive design requires consideration of what cognitive support the tool might provide. Cognitive support is an amalgam of the previous considerations, and results from the interaction between a specific user, his or her problems, and the functionality a tool provides. The objective here is to have a good understanding of what makes a particular tool useful. Is it the user? The features of the tool? The problem space it addresses? Understanding these questions will ensure that the designer of the tool, as well as users, have some comprehension of how the tool is to work, and what it can offer users. The human interest in discovering, understanding, and communicating new knowledge will require knowledge engineering tools which can support a variety of cognitive tasks.

5.5 Contributions

This thesis has made several unique contributions in the area of developing cognitive support tools for the knowledge engineering community.

- Sounded a call for more attention to end users by tool developers. This should be done by gaining a deeper appreciation of the cognitive support a tool can provide. This implies understanding both the utility a tool should provide, as well as the usability criteria a tool needs to address.
- Emphasized the importance of addressing the issue of tool adoption by end users. Although complex, using adoption as a metric will provide developers an opportunity to assess how well they addressed the domain's requirements and characteristics. Making the usefulness of a tool explicit will help in this area.
- Empirically derived a taxonomy of knowledge engineering tasks requiring cognitive support. This taxonomy addresses the general lack of such explicitly stated tasks and allows tool designers to evaluate their efforts based on this list.
- Provided a detailed evaluation of four tools providing advanced visual interfaces for Protégé, based on the task taxonomy, something previously poorly understood.
- Suggested the need to consider non-functional design goals when constructing these tools. I proposed five such goals and illustrated them and their trade-offs in this problem space.
- Finally, implemented the CVF, a combination of a domain, user, and presentation ontology, with Java-encoded application rules. With this tool, one can customize the presentation of information in Jambalaya, as well as determine feature selection and enhancement. This tool and the underlying approach were validated using outside expert users.

5.6 Future research directions

The CVF project was created as a prototype for testing some hypotheses about the use of customization in advanced visual interfaces for knowledge engineering. Given the conclusions I make above, there is no need to improve the CVF itself beyond what currently exists. Putting more effort into adding features to Jambalaya is not useful, either, unless such an effort is placed in the context of this work. There are two opportunities that I see for future work on Jambalaya.

One is to target a more specific application of the tool, such as the National Cancer Institute's Thesaurus. More effort would be expended on identifying domain-specific requirements, and combining those with the general knowledge engineering tasks I identified earlier. This combination would then allow the Designers to alter Jambalaya to specifically address those tasks.

Second, and perhaps most interesting, is to position Jambalaya as a collection of components which can be reconfigured based on specific requirements. In the customization model used in [54] and shown in Fig. 5.1, there are three major players in the process: a Designer, Customizer, and End User. This model could be extended to add a fourth role, tentatively titled Cognitivist, after Green and Petre [40]. They position the Cognitivist as the person who determines (in a broad-brush, task-specific analysis) what cognitive requirements are for a particular tool and problem, possibly using Green's Cognitive Dimensions framework [41]. The Cognitivist would operate between the Designer and the Customizer to identify the general domain requirements, and thereby "[illustrate] some of the cognitive consequences of making a particular bundle of design choices that position the artifact in the space ([40], p. 133)". This allows the Designer to assemble the required components accordingly. For example, the tasks requiring support might suggest building a tool that provided slot-based browsing and concept editing, but not top-down exploration. This would shift the burden of understanding Jambalaya in its complex entirety to the Cognitivist from the Customizer. The Customizer's role would then consist of selecting useful default views and designing helpful scripts for the End User. The cognitive burden on the Customizer would be reduced because the Cognitivist had already performed some task analysis for that domain.

There are other potential approaches to the problem of building advanced visual interfaces for cognitive support in knowledge modeling. One approach which might eliminate some complexity is to offload the Customizer's cognitive processing onto the tool. Such a tool would use intelligent learning techniques and recommend, based on previous use, important view paths to new users. A view path consists of a series of steps through a model; this might consist of navigating from an overview, along a series of relationships, to a node of interest. The tool would assign weights to the different view paths and recommend other, similar paths to experienced users, and also make a list of popular paths to new users.

Another technique, currently used by the Isaviz tool (§2.2.2.1), is to externalize the presentation of the graph into a customizable stylesheet. The stylesheet would consist of a series of transformations of the graph, such as changing node shape, removing unnecessary arcs, or changing layout parameters. The danger with this approach is that it asks Customizers to learn yet another formalism for viewing their data. The CVF ontology used an explicit model to capture customizations. While the stylesheet approach is not as explicit, it still requires users to understand how the model is constructed. Learning this model is time-consuming. It might be more useful to examine ways of making customizations for this space more implicit (similar to the Microsoft Word approach, for example).

Implicit customizations would allow items in the tool itself to be removed directly, for example via right-clicking and selecting "remove item". Similarly, nodes and arcs in the layout could be customized by selecting them and choosing whether to show them or not, and with what styles. This set of preferences could then be saved in an external representation so that they could be re-applied to other domains. Currently, Jambalaya allows one to customize the arc types and node types to a small extent (colour and limited shapes), but only for a particular ontology. It would be interesting to examine making these changes more extensive and reproducible.

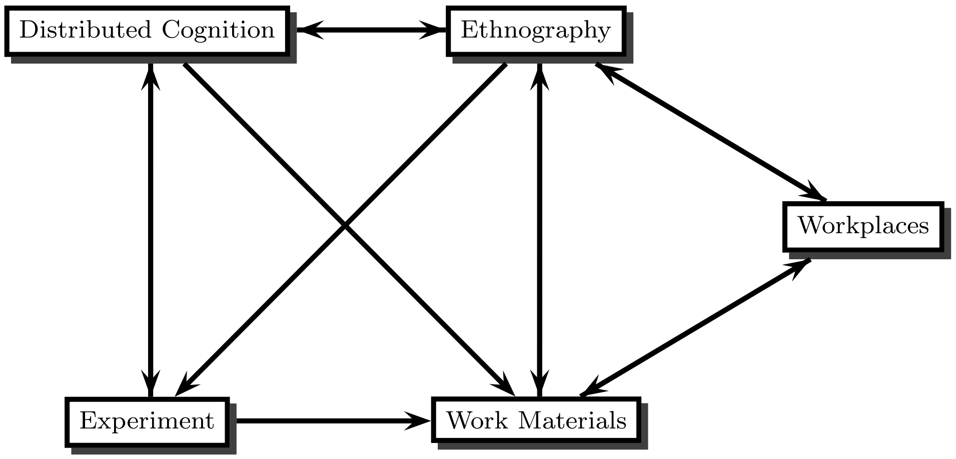


Figure 5.2. *Distributed cognition research loop [43]*

5.6.1 Critical assessment of the research

To conclude the thesis, a more critical look is taken at what the research has established, and where it positions future efforts in this problem area. In Hollan *et al.* [43] a useful framework for assessing this type of research is given in the form of a 'research loop', as shown in Fig. 5.2. This research loop is used to illustrate the benefits of cognition research; in this particular figure, using ideas from the *distributed cognition* paradigm. Distributed cognition (DC) examines cognitive processes holistically, considering not just the human and tool interaction, but related aspects of that interaction, such as work setting, other actors, and different tools.

The research loop highlights the process used in this thesis. Using DC, theories arise about problems and issues of interest to the researcher. In this case, these theories take the form of proposed tasks requiring support, design goals, and the notions of cognitive support and adoption. In order to expand knowledge about such theories, techniques from cognitive ethnography are used, including contextual inquiry, participant observation, or observational study. These techniques are then used to design experiments, studies designed to enhance understanding of the problem. In this work, such experiments involved heuristic walkthroughs, the fieldwork at the NCI and Anatomy groups, and the experience reports from expert users. While Hollan *et al.* are concerned almost exclusively with qualitative, ethnographic research methods, quantitative approaches can be equally effective. Good research requires understanding the benefits and problems with each approach, and applying those most suited to the task at hand. These experiments then produce work materials, such as the CVF+Jambalaya product, as well as impacting the development of other tools, such as Protégé. These changes in turn affect the workplace and work practices of the organization.

The cyclical nature of the framework implies that the work carried out in this thesis has no definitive endpoint—there is no ultimate product that will solve all problems. Rather, having undergone the process, researchers are now poised to assimilate the results in order to embark on more theory generation, more experimentation, and more tool design. The real product of this research is a deeper understanding of some of the problems, and approaches to those problems, in this domain.

Bibliography

[1] J. Ahlers and H. Weimer, "Challenges in interactive visualization for knowledge management," in *Sixth International Conference on Information Visualisation (IV02)*. London, UK: IEEE, 2002, pp. 367–371.

[2] H. Alani, "TGVizTab: An ontology visualisation extension for protege," in *Knowledge Capture 03 - Workshop on Visualizing Information in Knowledge Engineering*. Sanibel Island, FL: ACM, 2003, pp. 2–7.

[3] M. M. Allen, "Empirical evaluation of a visualization tool for knowledge engineering," M. Sc., Computer Science Department, University of Victoria, 2003.

[4] C. Alves and A. Finkelstein, "Matching multiple COTS: Can we achieve a happy marriage?" in *Workshop on Adoption-Centric Software Engineering at ICSE03*. Portland, OR: SEI, 2003, pp. 62–66.

[5] P. J. Barclay and J. Kennedy, "Teallach's presentation model," in *working conference on Advanced visual interfaces*, Palermo, Italy, 2000, pp. 151 – 154.

[6] M. I. Bauer and P. Johnson-Laird, "How diagrams can improve reasoning," *Psychological Science*, vol. 4, no. 6, pp. 372–378, 1993.

[7] B. Bederson, "Interfaces for staying in the flow," University of Maryland Human Computer Interaction Lab, Technical Report HCIL-2003-37, October 2003.

[8] T. Berners-Lee, M. Fischetti, and M. Dertouzos, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. San Francisco: Harper, 1999.

[9] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, p. 279, May 17 2001.

[10] C. Best, "Designing a component-based framework for a domain independent visualization tool," M.Sc., Computer Science Department, University of Victoria, 2002.

[11] H. Beyer and K. Holtzblatt, *Contextual Design: Defining Customer-Centred Systems*. San Francisco: Morgan Kaufmann, 1998.

[12] J. Blythe, J. Kim, S. Ramachandran, and Y. Gil, "An integrated environment for knowledge acquisition," in *Int. Conf. on Intelligent User Interfaces*, San Francisco, CA, 2001, pp. 13–20.

[13] B. Boehm, "A spiral model of software development and enhancement," *IEEE Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[14] R. Brachman, D. L. McGuinness, P. Patel-Schneider, L. Resnick, and A. Borgida, "Living with CLASSIC: When and how to use KL-one-like language," in *Principles of Semantic Networks*, J. F. Sowa, Ed. Morgan Kaufmann, 1991, pp. 401–456.

[15] B. Buchanan and E. Shortliffe, *Rule-Based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley, 1984.

[16] S. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. London: Academic Press, 1999.

[17] C. Cheng, Y. Shahar, A. Puerta, and D. Stites, "Navigation and visualization of abstractions of time-oriented clinical data," Stanford Medical Informatics, Tech. Rep. SMI-97-0688, 1997.

[18] L. Chung, B. Nixon, and E. Yu, "Using non-functional requirements to systematically select among alternatives in architectural design," in *1st International Workshop on Architectures for Software Systems*, Seattle, 1995, pp. 31–43.

[19] P. Clark, J. Thompson, K. Barker, B. Porter, V. Chaudhri, A. Rodriguez, J. Thomere, S. Mishra, Y. Gil, P. Hayes, and T. Reichherzer, "Knowledge entry as the graphical assembly of components," in *1st International Conference on Knowledge Capture (K-Cap '01)*. Victoria, BC: ACM Press, 2001, pp. 22–29.

[20] P. Clitherow, D. Riecken, and M. Muller, "VISAR: a system for inference and navigation of hypertext," in *Conference on Hypertext and Hypermedia*. Pittsburgh, Pennsylvania, United States: ACM, 1989, pp. 293 – 304.

[21] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, pp. 86–93, March/April 2002.

[22] S. Decker, M. Erdmann, D. Fensel, and R. Studer, "Ontobroker: Ontology based access to distributed and semi-structured information," in *DS-8*, ser. Semantic Issues in Multimedia Systems, R. Meersman, Ed. Kluwer, 1999, pp. 351–369.

[23] V. Devedzic, "Understanding ontological engineering," *Communications of the ACM*, vol. 45, no. 4, pp. 136–144, 2002.

[24] M. Eisenstadt, J. Domingue, T. Rajan, and E. Motta, "Visual knowledge engineering," *IEEE Trans. on Software Engineering*, vol. 16, no. 10, pp. 1164–1177, 1990.

[25] P. Eklund, N. Roberts, and S. P. Green, "Ontorama: Browsing an RDF ontology using a hyperbolic-like browser," in *First International Symposium on CyberWorlds (CW2002)*. Tokyo: IEEE, 2002, pp. 405–411.

[26] N. Ernst, M.-A. Storey, and P. Allen, "Cognitive support for ontology modeling," *submitted to Int. Journal of Human-Computer Studies*, October 2003.

[27] N. A. Ernst, "Adoption-centric knowledge engineering," in *Workshop on Adoption-Centric Software Engineering at ICSE 2003*, H. Mueller, Ed., Portland, OR, 2003.

[28] N. A. Ernst, M.-A. D. Storey, P. Allen, and M. A. Musen, "Addressing cognitive issues in knowledge engineering with Jambalaya," in *Workshop on Visualization in Knowledge Engineering at KCAP 03*, Sanibel Island, FL, 2003, pp. 26–30.

[29] N. A. Ernst and M.-A. Storey, "A preliminary analysis of visualization requirements in knowledge engineering tools," University of Victoria," CHISEL Technical Report, August 19 2003.

[30] K. Fairchild, S. Poltrock, and G. Furnas, "Semnet: Three-dimensional graphic representations of large knowledge bases," in *Cognitive Science and its Applications for Human-Computer Interaction*, R. Guidon, Ed. Lawrence Erlbaum Associates, 1988, pp. 201–233.

[31] R. d. A. Falbo, G. Guizzardi, and K. C. Duarte, "An ontological approach to domain engineering," in *International Conference on Software Engineering and Knowledge Engineering, SEKE02*, Ischia, Italy, 2002.

[32] P. Finnigan, R. C. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. A. Müller, J. Mylopoulos, S. G. Perelgut, M. Stanley, and K. Wong, "The software bookshelf," *IBM Systems Journal*, vol. 36, no. 4, pp. 564–593, 1997.

[33] T. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software - Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, November 1991.

[34] B. Gaines and M. Shaw, "Concept maps as hypermedia components," *Int. Journal of Human-Computer Studies: Special Issue on Knowledge-Based Hypermedia*, vol. 43, no. 3, pp. 323–361, 1995.

[35] M. Gantt and B. Nardi, "Gardeners and gurus: patterns of cooperation among CAD users," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, Monterey, California, 1992, pp. 107 – 117.

[36] P. Garvey and B. French, "Generating user interfaces from composite schemas," in *XML 2003*, Philadelphia, Pennsylvania, 2003.

[37] J. H. Gennari, M. A. Musen, R. Fergerson, W. E. Grosso, M. Crubzy, H. Eriksson, N. F. Noy, and S. W. Tu, "The evolution of Protégé: An environment for knowledge-based systems development," *International Journal of Human-Computer Studies*, vol. 58, no. 1, pp. 89–123, 2003.

[38] M. Ginsberg, "Knowledge interchange format: The KIF of death," *AI Magazine*, vol. 12, no. 3, 1991.

[39] J. Golbeck, G. Fragoso, F. Hartel, J. Hendler, B. Parsia, and J. Oberthaler, "The National Cancer Institute's thesaurus and ontology," *Journal of Web Semantics*, vol. 1, no. 1, 2003.

[40] T. R. G. Green and M. Petre, “Usability analysis of visual programming environments: a ‘cognitive dimensions’ framework,” *J. Visual Languages and Computing*, vol. 7, no. 2, pp. 131–174, June 1996.

[41] T. R. G. Green, “Cognitive dimensions of notations,” in *Fifth conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and computers V*. Nottingham, UK: Cambridge University Press, 1990, pp. 443 – 460, iSBN:0-521-38430-3.

[42] T. Gruber, “Towards principles for the design of ontologies used for knowledge sharing,” *International Journal of Human-Computer Studies*, vol. 43, no. 5-6, pp. 907–928, November 1995.

[43] J. Hollan, E. Hutchins, and D. Kirsh, “Distributed cognition: Toward a new foundation for human-computer interaction research,” *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 2, p. 174196, June 2000.

[44] I. Horrocks, “FaCT and iFaCT,” in *Proceedings of the International Workshop on Description Logics (DL’99)*, P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, Eds., 1999, pp. 133–135.

[45] I. Hsi, C. Potts, and M. Moore, “Ontological excavation: Unearthing the core concepts of the application,” in *10th Working Conference on Reverse Engineering*. Victoria BC: IEEE, 2003, pp. 345–352.

[46] Y. Kalfoglou, “Exploring ontologies,” in *Handbook of Software Engineering and Knowledge Engineering*, C. Chen, Ed. World Scientic Publishing Company, 2000, vol. 1.

[47] P. D. Karp, V. K. Chaudhri, and S. M. Paley, “A collaborative environment for authoring large knowledge bases,” *Journal of Intelligent Information Systems*, vol. 13, pp. 155–194, 1999.

[48] Y. Lei, E. Motta, and J. Domingue, “Design of customized web applications with Ontoweaver,” in *International Conference On Knowledge Capture*, Sanibel Island, FL, USA, 2003, pp. 54 – 61.

[49] R. Lintern, J. Michaud, M.-A. D. Storey, and X. Wu, “Plugging-in visualization: Experiences integrating a visualization tool with Eclipse,” in *SoftVis 2003*, 2003, pp. 47–56.

[50] W. E. Mackay, “Triggers and barriers to customizing software,” in *Computer Human Interaction (CHI 91)*. ACM, 1991, pp. 153–160.

[51] ——, “Responding to cognitive overhead: co-adaptation between users and technology,” *Intellectica*, vol. 30, no. 1, pp. 177–193, 2000.

[52] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz, "Ontologies for enterprise knowledge management," *IEEE Intelligent Systems*, vol. 18, no. 2, pp. 26–33, March/April 2003.

[53] F. Manola and E. Miller. (2002) RDF primer. World Wide Web Consortium. [Online]. Available: http://www.w3.org/TR/rdf-primer

[54] J. Michaud, "A software customization framework," M.Sc., Computer Science Department, University of Victoria, 2003.

[55] J. Michaud and M.-A. D. Storey, "The role of knowledge in software customization," in *15th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE03)*, San Francisco Bay, CA, 2003.

[56] E. Miller, R. Swick, D. Brickley, B. McBride, J. Hendler, and G. Schreiber. (2003) W3C semantic web. World Wide Web Consortium. [Online]. Available: http://www.w3.org/2001/sw/

[57] M. Minsky, "A framework for representing knowledge," in *The Psychology of Computer Vision*, P. Winston, Ed. New York: McGraw-Hill, 1975, pp. 211–277.

[58] H. Müller. (2003) Adoption-centric software engineering. Department of Computer Science, University of Victoria. [Online]. Available: http://www.acse.cs.uvic.ca/

[59] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using non-functional requirements: A process-oriented approach," *Software Engineering*, vol. 18, no. 6, pp. 483–497, 1992.

[60] B. Nardi, *A Small Matter Of Programming: Perspectives on End-User Computing*. Cambridge, MA: MIT Press, 1993.

[61] D. Nardi and R. J. Brachman, "An introduction to description logics," in *The Description Logic Handbook: Theory, implementation and applications*, 2nd ed., F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. P. Schneider, Eds. Cambridge UK: Cambridge University Press, 2003, pp. 1–574.

[62] G. K.-C. Ng, "Interactive visualization techniques for ontology development," Ph.D., Computer Science Department, University of Manchester, 2000.

[63] D. A. Norman, *The Design of Everyday Things*. New York: Currency and Doubleday, 1988.

[64] N. F. Noy and D. L. McGuinness, "Ontology development 101: A guide to creating your first ontology," Stanford Medical Informatics, Technical Report SMI-2001-0880, 2001.

[65] N. F. Noy, M. Sintek, S. Decker, M. Crubzy, R. W. Fergerson, and M. A. Musen,

"Creating semantic web contents with Protégé-2000," *IEEE Intelligent Systems*, pp. 60–72, Mar/Apr 2001.

[66] A. Pease, R. A. Liuzzi, and D. Gunning, "Knowledge bases," in *Encyclopedia of Software Engineering*, 2nd ed., J. Marciniak, Ed. New York: Wiley & Sons, 2001.

[67] E. Pietriga, "IsaViz: a visual environment for browsing and authoring RDF models," in *WWW 2002, the 11th World Wide Web Conference*. Honolulu, Hawaii, USA: World Wide Web Consortium, 2002.

[68] S. Rifkin, "Why new software processes are not adopted," *Advances in Computers*, vol. 59, 2003.

[69] E. Rogers, *Diffusion of innovations*, 4th ed. New York: The Free Press, 1995.

[70] G. Schreiber, B. Wielinga, R. d. Hoog, H. Akkermans, and W. v. d. Velde, "CommonKADS: A comprehensive methodology for KBS development," *IEEE Expert (IEEE Intelligent Systems)*, vol. December, pp. 28–38, 1994.

[71] M. Shaw, "Towards an engineering discipline of software," *IEEE Software*, pp. 15–24, November 1990.

[72] ——, "Writing good software engineering research papers," in *International Conference on Software Engineering*, Portland OR, 2003, pp. 1–11.

[73] F. M. Shipman and C. C. Marshall, "Formality considered harmful: Experiences, emerging themes, and directions on the use of formal representations in interactive systems," *Computer Supported Cooperative Work*, vol. 8, no. 4, pp. 333–352, 1999.

[74] B. Shneiderman, "Tree visualization with Tree-Maps: 2-D space-filling approach," *ACM Transactions on Graphics*, vol. 11, no. 1, pp. 92 – 99, 1992.

[75] M. Sintek. (2003) Ontoviz tab: Visualizing Protégé ontologies. [Online]. Available: http://protege.stanford.edu/plugins/ontoviz/ontoviz.html

[76] D. Skuce and T. C. Lethbridge, "CODE4: A unified system for managing conceptual knowledge," *International Journal of Human Computer Studies*, vol. 42, pp. 413–451, 1995.

[77] M. K. Smith, D. McGuiness, R. Volz, and C. Welty. (2002) Web ontology language (OWL) guide version 1.0. World Wide Web Consortium. [Online]. Available: http://www.w3.org/TR/owl-guide

[78] J. F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA: Brooks/Cole, 2000.

[79] M.-A. Storey, F. Fracchia, and H. A. Müller, "Cognitive design elements to support the construction of a mental model during software exploration," *Journal of Software Systems: special issue on Program Comprehension*, vol. 44, pp. 171–185, 1999.

[80] M.-A. D. Storey, M. A. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. F. Noy, “Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in protege,” in *Workshop on Interactive Tools for Knowledge Capture, K-CAP-2001*, Victoria, B.C. Canada, 2001.

[81] M.-A. D. Storey, K. Wong, F. Fracchia, and H. Müller, “On integrating visualization techniques for effective software exploration,” in *InfoVis ’97*, Phoenix, AZ, 1997, pp. 38–45.

[82] R. Studer, V. R. Benjamins, and D. Fensel, “Knowledge engineering: Principles and methods,” *Data Knowledge Engineering*, vol. 25, no. 1-2, pp. 161–197, 1998.

[83] P. Suber. (1999) Glossary of first-order logic. [Online]. Available: http://www.inf.unibz.it/~franconi/teaching/1999/3411/logic-glossary/

[84] M. Tallis, J. Kim, and Y. Gil, “User studies of knowledge acquisition tools: Methodology and lessons learned,” in *Knowledge Acquisition Workshop*, 1999.

[85] S. R. Tilley, K. Wong, M.-A. D. Storey, and H. Müller, “Programmable reverse engineering,” *Software Engineering and Knowledge Engineering*, vol. 4, no. 4, pp. 501–520, December 1994.

[86] M. Travers, “A visual representation for knowledge structures,” in *2nd ACM Conf. on Hypertext*. Pittsburgh PA: ACM, 1989, pp. 147–158.

[87] M. Uschold and R. Jasper, “A framework for understanding and classifying ontology applications,” in *IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, V. Benjamins, B. Chandrasekaran, A. Gomez-Perez, N. Guarino, and M. Uschold, Eds., Stockholm, Sweden, 1999.

[88] A. Walenstein, “Cognitive support in software engineering tools: A distributed cognition framework,” Ph.D., Computer Science Department, Simon Fraser University, 2002.

Appendix A – Body of email sent requesting evaluation

As you are aware, our lab has developed a visualization plug-in for Protégé known as Jambalaya. While the tool was initially released some two years ago, we continue to seek ways to improve the usability and utility of our tool.

Following interviews, background research, and user interviews, we conjecture that one of these improvements will be to develop a way for users to customize the tool using an included Protégé ontology. This ontology, which we call the Customizable Visualization Framework, or CVF, defines layouts, tools, and buttons to be used with your project. It makes use of scriptable interaction to tailor the user experience to your domain and your users.

In order to determine the validity of the idea of domain experts providing customizations, I would like to ask you to perform a brief evaluation on this early prototype of the CVF. This evaluation should take a half hour (or less) of your time and will require following the steps below, examining the resulting customized visualizations, and emailing results back to the investigator by December 12.

1. Ensure you have Protégé 2, build 111 or greater, and Java 1.4.2 or higher.
2. Download the latest Jambalaya installation at `http://www.cs.uvic.ca/~nernst/docs/cvf-files.zip`. Unzip the files into the (Protégé home)/plugins directory. You should overwrite any files you are asked about (e.g. older jambalaya versions). You should now see, inside the plugins directory, the relevant .jar files, the 'cvf' folder (representing the customization ontology files), and a 'scripts' directory. When including the CVF files, they will be in (Protégé home)/plugins/cvf.
3. Follow the instructions at `http://chiselog.chisel.cs.uvic.ca/cvf/quickstart` to explore ways the tool allows one to customize Jambalaya to the domain.
4. As a guide for what customizations to add for (3), try the following tasks:

   (a) create a new end-user instance
   (b) allow the user to start with a Nested Layout, using ”– :DIRECT-SUBCLASSES and :DIRECT-INSTANCES –” as the default nesting relation.
   (c) Select some other slots from the domain as ”Display Slots”.
   (d) Add two buttons for the user with some of the provided script instances attached.
   (e) Specify a set of other options to use for display, such as a set of tools and layouts.
   (f) Switch to the Jambalaya tab to see the customizations in effect.

5. You can tell if this worked by noticing the addition of the custom buttons on the far right. The CVF also writes to the console which user instance it is using.

Following your use, I would ask you to evaluate this approach to making visualization tools more useful, using the following criteria. Please reply to this email with your answers, on or before Friday, December 12th, if possible.

1. Does the customization mechanism meet your expected difficulty level for something of this nature?
2. Does the customization appear to make Jambalaya easier to use? You may want to compare the un-customized version with the new version.
3. What other customizations do you consider important to reduce barriers to using Jambalaya in your domain?
4. What are the biggest barriers, in your mind, to adopting Jambalaya (or other visualization tools) for your domain?
5. Are there other ways to accomplish the goal of increasing the usefulness of generic visualization approaches this methodology may not have brought up?

Keep in mind that the tool is a proof-of-concept model, being used to gather requirements for this approach. We realize that some of the interface features in the tool need work.

Thanks for your consideration. I will release the findings of this survey in my Master's thesis; if you would like a copy of these results, please let me know, and I will gladly share it with you. If you know of other users who would be interested in this, please forward this email to them as well.

VITA

Surname:	Ernst	***Given Names:***	Neil Alexander
Place of Birth:	Victoria, British Columbia, Canada	***Date of Birth:***	

Educational Institutions Attended

University of Victoria	1995 to 2001

Degrees Awarded

B.Sc.	University of Victoria	2001

Honors and Awards

NSERC PGS B	2001-2003
Provost's Award for Academic Excellence in Athletics	2002

Conference Publications

1. N. A. Ernst, M.-A. D. Storey, P. Allen, and M. A. Musen, Addressing cognitive issues in knowledge engineering with Jambalaya, presented at *Workshop on Visualization in Knowledge Engineering at KCAP 03*, Sanibel Island, FL, 2003.

2. N. A. Ernst, Adoption-Centric Knowledge Engineering, presented at *Workshop on Adoption-Centric Software Engineering at Int. Conf. on Software Engineering 2003*, Portland, OR, 2003.

3. M.-A. D. Storey, N. F. Noy, M. A. Musen, C. Best, R. W. Fergerson, N. Ernst, Jambalaya: an interactive environment for exploring ontologies, demonstration and poster at *International Conference on Intelligent User Interfaces*, San Francisco, USA, 2002.

4. M.-A. D. Storey, M. A. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. F. Noy, Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé, presented at *Workshop on Interactive Tools for Knowledge Capture, K-CAP-2001*, Victoria, B.C. Canada, 2001.

Work Submitted for Publication

1. N.A. Ernst, M.-A. Storey, P. Allen, Cognitive Support for Ontology Modeling, submitted to *Int. Journal of Human-Computer Studies*, October 15, 2003.