PROMPT-Viz: Ontology Version Comparison Visualizations with Treemaps

by

David Stephen John Perrin
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

Supervisor: Dr. Margaret-Anne Storey

Abstract

Current trends indicate that the prevalence of ontologies will continue to increase within many domains. They are already commonly used to define controlled medical terminologies and form the backbone of the Semantic Web initiative. Very few tools that support versioning of ontologies are currently available, and those that provide difference detection and visualization are particularly lacking. We have implemented a tool called PROMPT-Viz that provides advanced visualizations using treemaps to help understand the location, impact, type and extent of changes that have occurred between versions on an ontology. PROMPT-Viz runs as a plug-in for the popular Protégé knowledge engineering environment and as such should be applicable to a large number of ontology developers.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisor Dr. Margaret-Anne Storey for her endless energy and support in completing my research.

I am also grateful to all the advice, help and ideas that the members, both past and present, of the CHISEL research group have provided. A little bit of each of them is part of this work. Finally, I would like to thank Neil Ernst, Ian Bull and Elizabeth Hargreaves for helping to transform my rough draft into a coherent thesis.

Chapter 1: Introduction

The field of knowledge engineering is becoming an increasingly important area of computer science. Initiatives such as the Semantic Web [1] "... in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [2], will rely on ontologies to share data. Ontologies provide a shared conceptualization of a domain by defining the concepts in the domain and describing how those concepts are related to each other. However, most domains of discourse are not static, but evolve as the understanding of the domain grows. In order for ontologies to evolve successfully, there is a need for effective tool support. Representation standards for ontologies such as the W3C's Web Ontology Language (OWL) [3] and development tools like Protégé [4] are becoming prevalent, but the tools to support version control and difference comprehension are still lacking for ontology development.

Of the ontology development tools currently available, the open source Protégé project developed by the medical informatics group at Stanford University is one of the most mature and best-adopted. The key feature that has contributed to Protégé's success is its open source plug-in architecture that allows it to be easily extended to better suit the needs of particular users. The PROMPT [5] plug-in for Protégé supports four versioning tasks for ontology development: merging, mapping, factoring and difference detection. During software development, the use of version control systems provide programmers with the ability to determine where changes have occurred to the code base and create visualizations of the differences between file revisions to help reduce the cognitive load on the programmers. The difference detection portion of PROMPT called PROMPTDiff

[5, 6] performs the analogous function for ontologies by determining the differences between ontology versions.

The PROMPTDiff tool provides two different views of the differences it finds between two versions of an ontology. First, it presents a table listing all of the concepts that exist in both versions of the ontology and describes the change (if any) that has occurred between versions. Selecting the corresponding row in the table reveals the specific details about any change. The second method that PROMPTDiff uses to show differences between versions is an expandable tree that merges all of the concepts from both versions of the ontology, arranging them according to their location in the is-a hierarchy of the latest version. While the change data that PROMPTDiff extracts is sound, the default visualizations provided by PROMPTDiff are not effective for large ontologies. For example, to answer a question such as "Where have the most changes occurred in the ontology?" is difficult for an ontology with 50,000 concepts using the default views in PROMPTDiff. We have also identified that the answers to the following questions are not easy to infer from the views provided by PROMPTDIFF:

- **L**ocation: Where have the changes been made to the ontology?
- **I**mpact: Do the changes directly or indirectly affect parts of the ontology the user is concerned about?
- **T**ype: What kinds (additions, deletions, moves, direct changes, etc) of changes have been made to the ontology?
- **E**xtent: How much of the ontology has changed?

We refer to these questions as the **LITE** questions to ease discussion in the remainder of this thesis. Like so many software version control systems [7], we hypothesized that incorporating additional visualizations could enhance understanding of the change data and help us answer these questions.

To test this hypothesis, we designed a tool called PROMPTViz that augments PROMPTDiff with information visualization techniques to provide enhanced cognitive support for understanding the differences between versions of ontologies. We combine the treemap layout technique with a zoomable user interface to allow questions such as "Where have the most changes occurred in the ontology?" to be answered easily. The treemap layout is a computationally fast layout technique that makes effective use of screen space and the zoomable user interface allows us to conform to Shneiderman's information seeking mantra "Overview first then details on demand" [8]. The treemap layout has been used effectively to show changes in the stock market (see http://www.smartmoney.com), another large information space. Since PROMPT-Viz is also a plug-in for Protégé, it can be applied to any ontology that is compatible with Protégé. This makes PROMPTViz applicable to many people in the knowledge engineering community.

1.1 Thesis Outline

In order to develop Prompt-Viz, there were several broad areas of study that were investigated. These areas comprise the major topics in this thesis and include:

- Information Visualization

- Ontologies, Ontological Systems and Knowledge Engineering
- Versioning of Ontologies
- Difference Visualizations of hierarchical structures

Throughout the thesis, we refer to the work we have done with the U.S. National Cancer Institute (NCI) as a case study. The NCI thesaurus is an ontology comprised of data about the cancer field including topics such as diseases, genes, proteins, biological processes, clinical trials and much more. Currently, the thesaurus contains about 35,000 concepts. The NCI has a team of modelers that generate a new baseline of the thesaurus on a bi-weekly basis. According to Frank Hartel, Director of the enterprise vocabulary group at the NCI, the modelers each have a strong mental model of how the ontology is organized and how the modeling decisions they have made effect the thesaurus. However, due to the large size and complexity of the NCI thesaurus, it is difficult for them to verify their mental models. We conjecture that one way to provide the necessary cognitive support for the knowledge engineer is with overview visualizations of the differences between the current and old baseline of the thesaurus. This real world case study was in fact the motivation for doing this work and is referred to frequently throughout the thesis.

The rest of the thesis is organized as follows. In Chapter 2, the motivation for developing a visualization tool to help humans understand the differences between ontology versions is given. The concept of an ontology is introduced and the need for versioning tools to support their development is highlighted. Parallels between ontology engineering and

software engineering are drawn and finally a presentation of the ontology development process used for the NCI thesaurus [9] is presented.

Chapter 3 further elaborates on the topic of Ontological Systems and Knowledge engineering. The two major representation paradigms are briefly explored along with the various representation languages and emerging standards in the field. We emphasize the paradigms that are supported by the Protégé environment. To conclude the chapter, an in-depth look at the Protégé system and the PROMPT plug-in is provided.

In Chapter 4, a discussion of Information Visualization techniques is provided. This discussion includes the layout techniques that can be used to draw trees and more general graphs with specific attention being paid to the treemap layout technique developed by Shneiderman and Johnson in 1991 [10]. Navigation and interaction styles are also reviewed here with the focus on zoomable user interfaces and coupled views. The chapter concludes by considering some difference visualizations that have been successfully used in other domains.

A detailed description of the tool, PROMPT-Viz, is provided in Chapter 5. This includes a discussion of the requirements, features and implementation details. In Chapter 6, we present the results of a user study that was conducted to evaluate if PROMPT-Viz improved the ability to answer overview tasks about how an ontology had changed between versions as compared to the default table and tree views provided by PROMPTDiff.

Finally, in Chapter 7 the thesis is concluded with an outline of my contributions and a discussion of some possible future work.

Chapter 2: Ontologies and their development

Ontologies provide a formal specification of a domain of discourse and are becoming increasingly prevalent in the high tech world. This chapter begins with a brief discussion of what an ontology is and how they are defined. Given the rapid adoption of ontologies, the definitions are followed by a discussion of ontology development, focusing on the issue of versioning. A brief comparison with the field of software engineering is then made as version control is more mature within software engineering. The chapter concludes with a presentation of the development process used for the NCI thesaurus.

2.1 What is an ontology?

There are many different definitions of an ontology and also some question of where an ontology ends and a knowledge base begins [11]; however, for our purposes, Gruber's short definition is suitable. "An **ontology** is an explicit specification of a conceptualization" [12]. The use of ontologies to construct knowledge base systems is growing rapidly. As already mentioned, they are widely used in the medical community and will provide the backbone of the Semantic Web. On the surface ontologies may appear to be like database schemas; however, ontologies are not a way of organizing a specific data set for efficient retrieval, but rather a reusable structure for data within a domain that is designed to capture all the inherent relationships and meta-data among the knowledge that will be stored in there. Ontologies are intended for both humans and computers to manipulate. In short, ontologies provide a common vocabulary for communication of knowledge within domains [13].

There are two primary methods that have been used to construct ontologies. Description Logic based systems and Frame based systems. The following is a description of the top level items of a frame-based knowledge model as described by Noy [14]:

- **Classes** are collections of objects that have similar properties. Classes are arranged into a subclass-superclass hierarchy using either single or multiple inheritance. Each class has slots (described next) attached to it. Slots can be inherited by the subclasses.
- **Slots** are named binary relations between a class and either another class or a primitive object (such as a string or a number). Slots attached to a class may be further constrained by facets.
- **Facets** are named ternary relations between a class, a slot and either another class or a primitive object. Facets may impose additional constraints on a slot attached to a class, such as the cardinality or value type of a slot.
- **Instances** are individual members of classes.

The primary difference between description logics (DL) and frame based knowledge systems is that as a subset of first order predicate logic [15], DL includes the ability to automatically classify new concept descriptions with respect to previously defined concepts and to check the consistency of declared statements [16]. This frees knowledge engineers from having to explicitly enter all the information about a new concept because the system will automatically add any implied information (based on previously defined concepts). Thus, a description logic based system has both explicitly and implicitly

defined information as opposed to a frame based system where all information must be explicitly defined.

2.2 Ontology development process

Yiling Lu (a former student in the CHISEL group) makes the association between ontology development and software development in his thesis [17]. He asserts that as ontologies become more complex their development becomes increasingly collaborative, requiring a group of domain experts and engineers to construct them. This parallels the historical development of software systems where, as systems grew in size and complexity, more people were required to complete the project and *ad hoc* development procedures were not suitable. Formal models were required to define the software development process and workflow management tools were required to help engineers adhere to the model. Nowadays, a suite of tools is often used to support software development projects. Two key tools within such a suite are some sort of version control software to track the evolution of the system and a difference tool to compare versions of files. Likewise, ontology development is beginning to enter the stage where projects require a formal development process [17] and the support of tools to help engineers to adhere to a defined process. The set of tools is similar to those used in software engineering with ontology development sharing the need for versioning and difference detection tools.

2.3 Ontology and Software Versioning

Given the relative immaturity of versioning and change representation tools for ontology engineering and the parallels of ontology engineering with software engineering, we will briefly look at the general techniques that are used in the software engineering field and then compare them with some of the current efforts for ontology engineering. Conceptually, the general methods that have been used to compare versions of software systems are quite applicable to ontologies; however, the fact that ontologies are usually stored as a single monolithic entity (such as a single file) as opposed to many small source files adds an additional challenge to ontology versioning and comparison.

2.3.1 Software Versioning

In many software system version control systems like CVS, a single file represents the finest granularity of change that is tracked by the system [18]. Either a file has been changed or it has not. For the most part this does not represent a large barrier to the use of version control systems as software systems are composed of many files, usually organized into some sort of package or directory hierarchy. Thus the information describing the extent and location of changes in the code base is readily at hand. In such systems if more specific information is required about the changes, it may be necessary to consult the log files to view comments made when changes were submitted, or to perform a difference operation between the repository version and a local copy. Differences between two versions of a file can be done on a line-by-line basis, highlighting the positions where insertions, deletions and modifications have taken place; however, if significant restructuring of the code has taken place, the comparison can be challenging.

In most development efforts, many programmers are working with in the same code base and inevitably, more than one of them needs to make changes to code that resides within the same file. This can be dealt with in a formal or informal manner but the outcome is always the same. The changes made must be reconciled with the version in the code base before a programmer checks their version back into the repository. Finer grained version control systems such as the Stellation plug-in for Eclipse [19] that track changes at the variable and method level can help lessen the difficulties that occur when more than one person needs to make modifications to the same file.

2.3.1.1 Software change representation and visualization

There are several levels of detail where changes in software systems can be visualized. The finest level of detail is represented by difference visualization between versions of a file in the system. A line-by-line comparison of the two versions, with the differences highlighted, is usual for this type of comparison and is common in software version control systems. This line-by-line comparison does a good job of allowing developers to compare the specific differences between versions of source files and has been extended to provide higher-level visualizations. Many tools have been developed over the last 10-15 years that aim to provide overview visualizations of the changes to large software system. Auger [20], SeeSoft [21], SeeSys [22], Xia [7], Beagle [23] and Palantir [24] are some noteworthy examples of such systems. Auger and SeeSoft both expand on the traditional line based comparison by compressing each line of code to a small colour coded line of pixels. Using this technique, Augur can display up to 40,000 lines of code (LOC) on the screen at once while providing information about the author that made the

change, the relative change date and type of structure the code belongs to, such as functions and comments.

SeeSys and Beagle both take a higher level metric based approach to visualizing changes and are aimed more at analysis of software evolution as opposed to offering direct support to programmers during the development process. SeeSys uses a treemap display where each node in the treemap displays statistics about each of its children as a histogram bar drawn within the node. Beagle provides several different methods of displaying the results of various metrics and queries including tables, file browser trees and node and link graphs.

Palantir and Xia both aim to provide direct support to engineers during development by providing visualizations to improve awareness about changes to the code base. Xiaomin Wu states that Xia is designed to help support developers to answer the 5W+H questions. That is: Who, What, Where, Why, When and How [7]. This is accomplished with a set of node and link and nested node diagrams that can be coloured, filtered, resized, etc according to various attributes stored by a CVS repository.

As illustrated by these examples, there are many different techniques used for visualizations of software changes and certainly some of the techniques are equally applicable to ontologies. One key feature that binds all of these software techniques together is that they heavily rely on the information collected by a version control system like CVS, something that is not yet available in ontological development systems.

2.3.2 Versioning for ontologies

The primary difference between version control for software systems and version control of ontological systems is the storage mechanism. Where software systems are partitioned into many small files, ontologies can have tens of thousands or even millions of concepts that are commonly all stored in a monolithic entity [25] such as a single large file. This storage mechanism can be a significant impediment to analyzing the differences between versions of ontologies. Additionally, unlike software development, where configuration management systems that log incremental changes to the source code are commonplace, in ontology development tools, logging changes is still uncommon [26].

The lack of logs, especially in the decentralized environment of the semantic web [27], combined with the monolithic storage mechanism currently makes version control of ontologies a challenging endeavor. According to Klein and Noy, even with change logs available, determining exactly how the changes made by several developers interact is a difficult task in itself [27]. Fortunately, it is possible to determine how two versions of an ontology differ without knowing the exact changes that caused them. This is not as simple as the standard Unix **diff** function. Line by line textual comparison of two ontologies is not adequate because the meaning of an ontology can be unchanged even when the stored file is dramatically different and vice versa. In order to determine the differences, a tool needs to perform what is termed **structural diff [6]** on the two versions of the ontology. A structural diff matches pairs of frames between two versions of an ontology and determines if and how the frames differ [6]. Michel Klein's

OntoView and Natasha Noy's PROMPT plug-in for Protégé are two currently available tools that can accomplish this task. The structural difference generated by these tools still lacks the richness that a configuration management tool could provide by detailing the actions that caused the changes, but it is adequate to create visualizations to help with the cognitively challenging task of comparing versions of large ontologies.

OntoView presents the results of its structural diff as a linear side by side listing of the RDF source for each version [28]. RDF is the acronym for Resource Description Framework and is an XML derived language that can be used to encode the specification of an ontology [29]. Different types of changes are highlighted in different colours, with the specific changes lines of the definition shown in bold. The PROMPTDiff portion of Noy's PROMPT plug-in presents the differences at a slightly higher level than OntoView by presenting a tabular listing of all the concepts in the ontology organized by change type and a tree view organized according to the is-a hierarchy of the ontology indicating changes with colours and icons. Unlike the Beagle, SeeSys and Augur, tools that exist for showing higher level views of changes in software systems, PROMPT and OntoView both lack visualizations that supports understanding how the ontologies have changed at an abstract level. PROMPT will be discussed in more detail in the next chapter as we use the changes it detects to provide better overview visualizations.

2.3.3 NCI Thesaurus development process

The ontology development process used by the National Cancer Institute to continually update and publish their Thesaurus is a motivational example for the need for better versioning tools. The NCI Thesaurus is a deep and complex biomedical vocabulary implementing rich semantic relationships between its nodes and taxonomies. According to Golbeck *et al.* [9], the Thesaurus is not a true ontology, as it contains many primitive concepts but it is strongly ontology-like across several of its taxonomies. The Thesaurus is currently developed using Apelon Inc.'s Terminology Development Environment and Workflow Manager software tools [30]. It is published on a monthly basis in several formats including in the Ontology Web Language (OWL) [9] which provides compatibility with Protégé through a new OWL plug-in whose development is supported by the NCI [31]. The workflow process they use to maintain their editing and publication cycle is shown in Fig. 2.1. In point form, paraphrased from Golbeck *et al.*[9], the process consists of the following tasks as numbered in Fig. 2.1:

1. Working from a separate database containing the current version of the thesaurus, the lead modeler creates worklists for each of the modelers to complete during the cycle.
2. The worklists are exported through the Apelon workflow manager for each of the modelers.
3. The modelers makes the changes detailed in their worklists on local copies of the thesaurus.
4. A change set is sent back to the workflow manager.

5. The changes are analyzed for potential conflicts and new assignments are made and the cycle repeats until no conflicts are found.
6. On a weekly basis, the changes are consolidated and each modeler's local database is updated with a new baseline of the thesaurus.
7. All the changes from the cycle are imported and classified with description logic rules into a trial database and a final check of the new version of the thesaurus is made.
8. The new version is published.

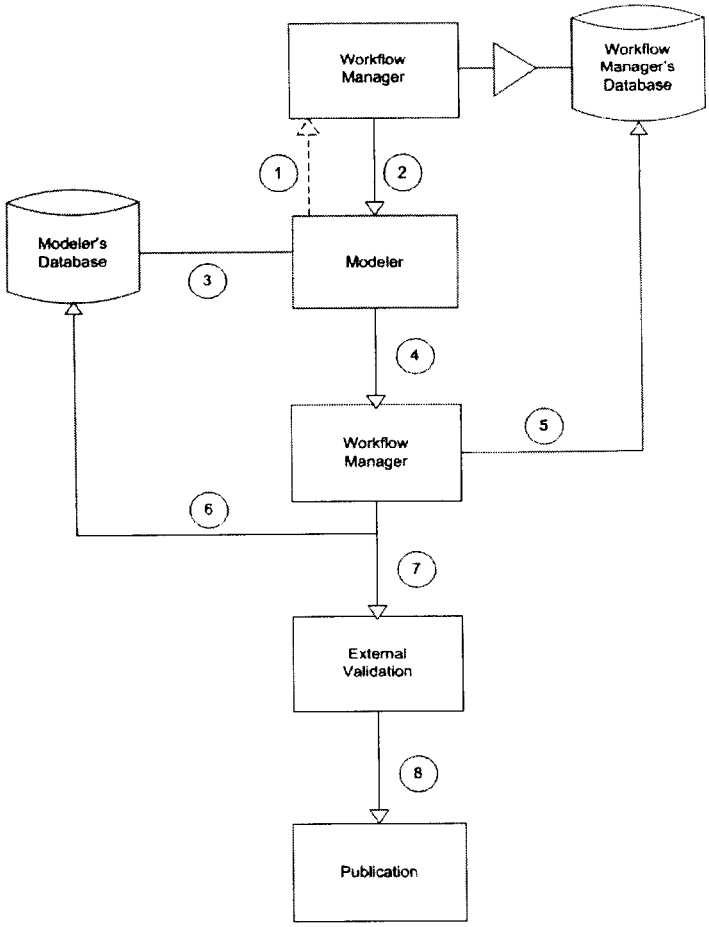


Figure 2.1 Workflow diagram of the NCI thesaurus editing and publication cycle [9]

During the 5th and 7th items of the process the use of a difference tool would greatly enhance the ability of the modelers to determine conflicts and ensure that the actual changes that have been made to the thesaurus match the intended changes. We are striving to place PROMPT-Viz to be appropriate for use at these locations within the cycle in order to provide the tool support the modelers are currently lacking. As alluded to in point 4, the Apelon tools do provide change logs, but logs themselves only state the changes and when multiple users are changing the same ontology their usefulness is reduced [26] and the need for a difference tool like PROMPT-Viz is heightened.

2.4 Chapter Summary

By providing a formal specification of the concepts within a domain, ontologies provide a common language for communication within the domain they describe. However, ontologies are not static definitions and therefore change, as knowledge about the domain they describe is accumulated and refined. Both the modelers who are evolving the ontologies and the people who rely on ontologies for their work need tools to help them understand how their ontology is changing. Within the software development domain, tools to support change management are relatively mature. We conjecture that many of the methods used in the software field are applicable to ontology development. As such, PROMPT-Viz utilizes techniques that have been applied to the software development domain. Since PROMPT-Viz is a plug-in for the Protégé ontology development environment and an extension of Natasha Noy's PROMPT plug-in, it is necessary to discuss both Protégé and PROMPT before providing a detailed description of PROMPT-Viz. The discussion of Protégé and PROMPT is provided in the next chapter.

Chapter 3: Protégé

The Protégé system is a component based ontology development platform developed by the Medical Informatics group at Stanford University [32-34]. There have been four versions of Protégé. The latest version of Protégé incorporates several key improvements over the previous versions including [32, 33]:

- An open source component based plug-in architecture written in Java that makes Protégé easy to customize for a particular domain or set of needs and easy to deploy on many different operating systems and hardware combinations.
- The adoption of the Open Knowledge-Base Connectivity (OKBC) knowledge model to increase compatibility with other ontology development systems.

3.1 Protégé Knowledge Model

The knowledge model of Protégé is built around the frame based Open Knowledge-Base Connectivity protocol (OKBC) [35]. The OKBC provides compatibility with other knowledge representation systems by providing a common application-programming interface (API) upon which to build them. The frame based knowledge model and access operations and behaviors specified by the OKBC are as general as possible to allow differences among knowledge representation systems.

There are four components of an ontology in Protégé: *classes*, *slots*, *facets* and *axioms* all of which are represented by frames. Classes represent concepts, slots describe class attributes, facets describe slots and axioms provide additional constraints. In Protégé a

knowledge base is considered to be the sum of an ontology and instances of classes with specific slot values [34].

Classes within a Protégé ontology are arranged in a taxonomic hierarchy. For example, this means that if **Graduate Student** is a subclass of **Student**, then every instance of a **Graduate Student** is also an instance of a **Student**. Additionally, classes are allowed to have more than one parent class for example, a **Graduate Student** is both a **Student** and an **Employee**. Protégé also contains the notion of a *metaclass*, whose instances are classes. This powerful mechanism will be discussed later in this section.

Slots in Protégé are used to attach attributes and values to *classes* and *instances*. *Slots* are defined independently of *classes* and are then attached to one or more different *classes*. There is also the notion of *template slots* and *own slots* with the difference being that *template slots* are inherited by subclasses and propagated to *instances*, but *own slots* are not. Figure 3.1 from [34] does a good job of illustrating these points.

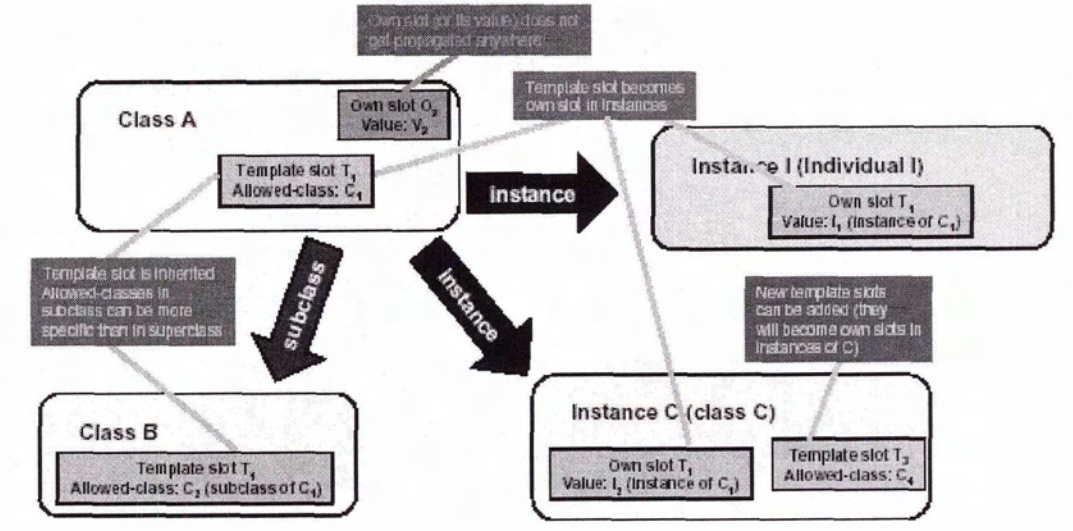


Figure 3.1 The frame based knowledge model of Protégé [34]

Facets provide a mechanism to apply constraints to allowed *slot* values. These constraints can include the type of value the *slot* can hold, the cardinality of the *slot*, restrictions to the allowed range etc.

Protégé uses the flexibility of the OKBC to implement its *metaclass* system. A *metaclass* is a template from which other *classes* are instantiated. In fact, at the core of the default knowledge model of Protégé is a *metaclass* structure that defines the default attributes of *classes*, *slots* and *instances*, so every user-defined frame in a Protégé knowledge base is an instance of a *metaclass*. This is generally hidden from Protégé users to keep things simpler, but it enables the possibility of changing the knowledge model used by a Protégé knowledge base or ontology. This is accomplished by defining a new *metaclass* structure that represents the knowledge model you would like to use. Knublauch *et al.* describe the implementation of the Protégé OWL plug-in using Protégé's *metaclass* system in *The*

Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications [36]. This plug-in allows Protégé to be compatible with the NCI thesaurus.

The Protégé designers have tried to keep as much of the generality and flexibility of the OKBC as possible while reducing ambiguity and conforming to the Protégé user interface. The flexibility of the OKBC has allowed the implementation of the Protégé *metaclass* architecture that makes it possible to define other knowledge models (such as OWL) within Protégé.

3.2 Protégé Architecture

The architecture of Protégé is designed to make it easy to customize and extend in both task and domain specific ways [33]. This is accomplished with a three layer, plug-in style architecture based on the Java language [32, 33]. The three layers as shown in Fig. 3.2, are the user interface layer, the control layer and the storage layer. Each of the layers communicates with the layers below using a well-specified API.

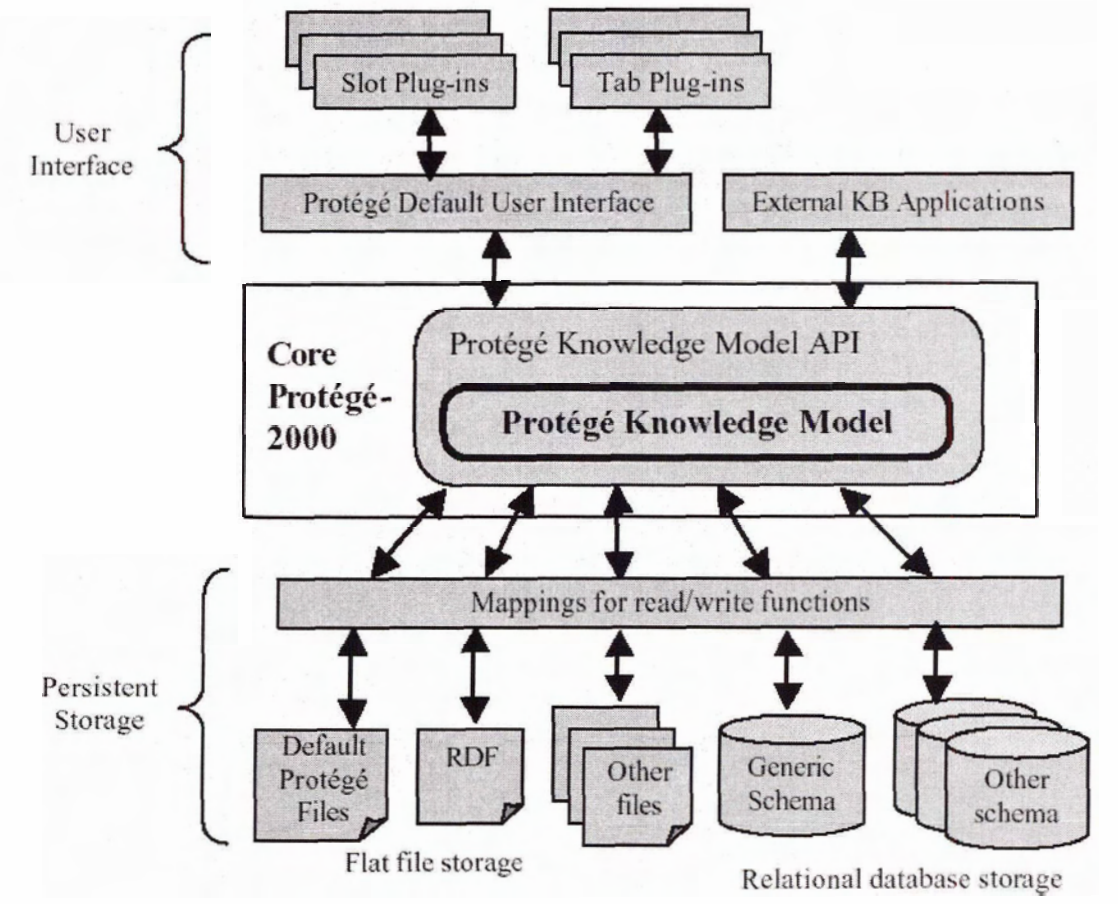


Figure 3.2 The three layers of Protégé's component architecture include a user interface layer, a core Protégé layer and a storage layer. The UI and storage layers can be replaced or modified to allow extensive customization [32]

The UI layer is the top layer of Protégé's architecture and a frequent candidate for customization as it was for Prompt-Viz. There are three different levels of customization possible at the UI level of Protégé. In order of coarse to fine granularity they are:

1. Replacing the entire user interface is the most dramatic customization possible for Protégé and is chosen when there are tight restrictions on how users are allowed to access the underlying ontology or knowledge base. An example of

this type of customization is ShrimpBib [37], by my colleague Polly Allen in the CHISEL Group. ShrimpBib is a bibliographic reference sharing tool with a web interface that uses Protégé to organize our group's bibliographic references into a knowledge base using criteria such as who has read the paper, the ratings the paper has received, the research area of the paper etc.

2. Tab Plug-ins offer an intermediate level of customization for the inclusion of domain or task specific interface elements or visualizations, but still retain the default Protégé UI. This is the method of extension that PROMPT uses to connect to Protégé and will be discussed further next. Other notable examples of Tab Plug-ins for Protégé include Jambalaya [38] and OntoViz [39].
3. Slot Widget Plug-ins provide customization at the finest level of granularity and allow Protégé to be customized to handle new data types. Some examples that are downloadable from the Protégé website include: a gif slot widget to allow gifs to be displayed in slots, media slot widgets that provide the ability to display different audio and video formats in slots, several variations for date slot widgets and many others.

Tab plug-ins for Protégé are used if a developer is satisfied with much of the default UI implementation or wishes to retain significant elements of the UI. Much of the default Protégé UI is constructed using tab plug-ins so they integrate into the existing UI in a nearly seamless manner. Additionally, since tab plug-ins function nearly independently from Protégé [32], there are few restrictions to the type of tools that can be developed as tab plug-ins. Be it a custom knowledge acquisition interface, a visualization environment

or a tool to help with aligning and versioning of ontologies like PROMPT, all are within the scope of a tab plug-in. Finally, by using a tab plug-in, the programming overhead can be significantly reduced through the reuse and/or slight modification of the standard widgets in the Protégé API.

The plug-in style, open source, component based architecture facilitates users customizing the environment to suit their particular sets of needs. PROMPT is one such plug-in for Protégé and is described next.

3.3 Prompt

The PROMPT plug-in for Protégé is a framework of four multiple-ontology management tools [26]. The four components share a common UI and infrastructure that enhances the interrelated tasks of finding overlapping concepts between ontologies, merging ontologies and discovering the differences between ontology versions.

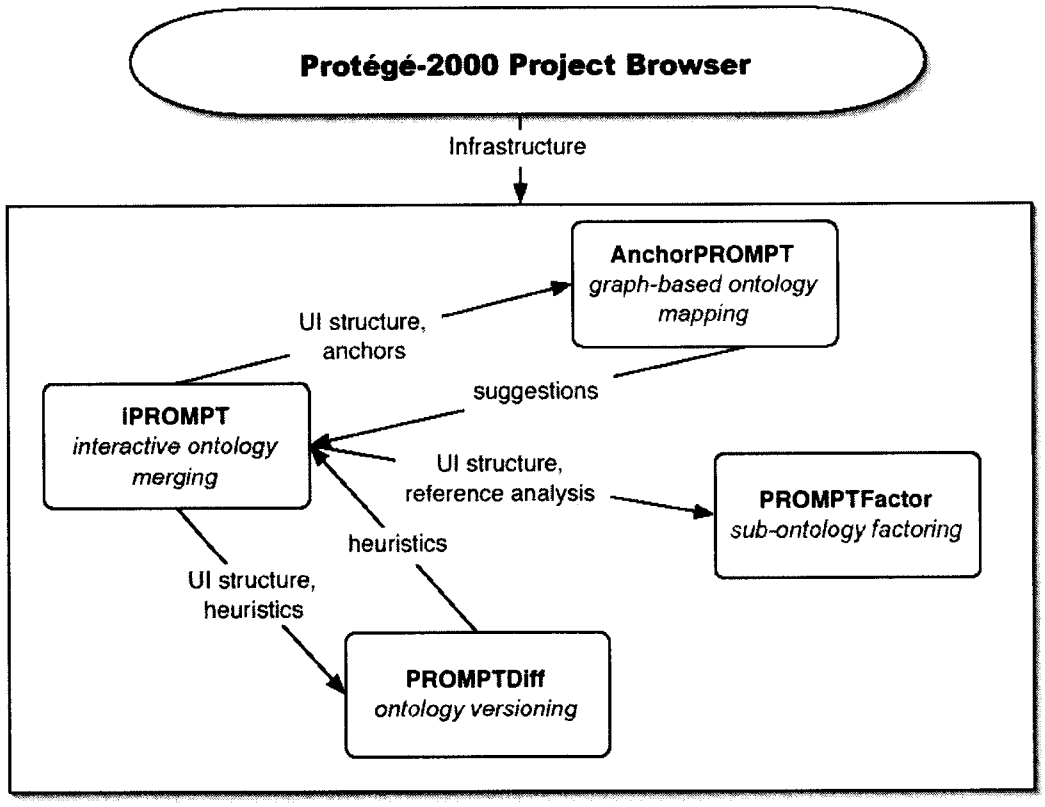


Figure 3.3 The dependencies of the three tools within the PROMPT framework [26]

iPROMPT was the first tool within PROMPT to be realized and thus, AnchorPROMPT [40], PROMPTDiff and PROMPTFactor [5] all utilize the core UI structure developed with it. The iPROMPT component is an interactive ontology merging tool that aids in merging different ontologies that represent the same or overlapping domain. The semi-automatic approach of iPROMPT guides the user as they perform merges by providing suggestions of which frames to merge, determining inconsistencies that may arise from user actions and providing suggestions to remedy any possible inconsistencies.

AnchorPROMPT finds semantically similar concepts in pairs of input ontologies using a graph structure based approach, where classes are nodes and slots are links, using a set of

initial anchors between the ontologies. The user can specify the set of anchors (pairs of related terms) or they can be generated automatically by lexical matching.

The PROMPTFactor tool can extract a portion of an ontology as a new sub-ontology ensuring broken links do not lead to any inconsistencies in the sub-ontology. The user specifies the concepts that he/she wants in the new sub-ontology, and the PROMPTFactor extracts those concepts and all concepts reached by traversing subclass-of and slot references. In addition, it traverses the superclass-of relation to retrieve subclasses, but only for the selected terms (otherwise the entire ontology would be selected).

The final component, PROMPTDiff, is a tool that identifies the differences between two versions of the same ontology by performing a structural diff. This is our particular interest and as such, how PROMPTDiff operates warrants deeper discussion than the other three tools that are part of the PROMPT framework.

3.2.1 PROMPTDiff

The PROMPTDiff tool fills the important role of comparing two versions of the same ontology. The name is derived from the standard Unix **diff** process that is used by most software version control tools to identify the differences between two versions of a text document like a source file. Comparing two text files can be accomplished at the simplest level simply by identifying the lines in the two files that are not the same; however, as mentioned in Chapter 2, the information stored within two versions of an

ontology can be conceptually identical yet their textual storage very different. The PROMPTDiff algorithm, therefore, compares not the textual representations of the versions, but the structure of the versions. This is accomplished in two parts. First an extensible set of heuristic matchers is run and their results are combined by a fixed point algorithm to create the structural diff.

Each of the heuristic matchers looks at a particular property of the yet unmatched frames. They only add to the set of matches and never retract any of the matches found by previous matchers, in this way, they always converge to a complete solution. As each of the matchers is relatively simple, the strength of the solution lies in the combination of the matchers. It is possible that incorrect matches could be found, but Noy and Musen state [26] that they have never seen this happen in practice and that a human expert is always present to catch a mistake if it does happen. Some examples from the set of heuristic matchers are:

- **Matching frames that have the same type and same name**. Since ontologies usually do not change too much between versions this can be an effective way of finding many matches.
- **Single unmatched sibling**. If two classes C1 and C2 are matched and each has one unmatched child subC1 and subC2 then subC1 and subC2 match.
- **Multiple unmatched siblings**. Similar to single unmatched siblings except the set of slots in the subclasses is used to differentiate them.

PROMPTDiff categorizes the changes it finds with its heuristic matchers according to five different operations:

1. Adds – The frame exists only in the new version.
2. Deletes – The frame exists only in the old version.
3. Merges – Two frames from the old version have been combined in the new version.
4. Splits – A single frame from the old version has been split into two frames in the new version.
5. Maps – The frame exists in both version and the previous four operations do not apply.

Additionally, there are three different levels of Map operations:

1. Unchanged – The frames are identical.
2. Changed – The frames have slots or facet values that are not images of each other.
3. Isomorphic – The frames slots and facet values are images of each other, but not identical images. For example the frame referenced by one of the slots may have changed between versions.

3.4 Chapter Summary

The flexibility afforded by Protégé's *metaclass* knowledge model combined with the extensibility of its plug-in architecture has contributed to Protégé's continued success within the knowledge engineering community. These attributes make the development of vital tools like PROMPT relatively easy and allow Protégé to be extended to read and

write important new ontology representation languages like OWL. The PROMPTDiff portion of the PROMPT plug-in is one of the few tools that can detect the differences between versions of ontologies and the flexibility of the Protégé API has facilitated our extension of it to create PROMPT-Viz. The only topic remaining to be covered before we outline the implementation of PROMPT-Viz is a discussion of relevant visualization techniques and is the focus of the next chapter.

Chapter 4: Information Visualization

The field of Information Visualization is concerned with reducing the cognitive load for a user as they interact with an information space. The idea of presenting information visually is not limited to the field of computer science. Humans have been using visual metaphors to present information long before the invention of computers [41]. What computers have done, is to allow the generation of immense volumes of information and have it stored at the fingertips of a single user. They have also provided the ability to search, sort, filter, and aggregate this information with relative ease. These abilities lead naturally to providing many dynamic visual views of the information.

4.1 Visual Perception and Cognition

Before discussing some of the specific information visualization techniques relevant to our work, it is appropriate to provide a brief discussion of human visual perception and cognition. The strengths and weaknesses of human visual processing systems must always be considered during the creation of information visualization systems. Seemingly small details, such as mapping an inherently quantitative attribute like length to a nominal variable can have a negative impact on the effectiveness of a visualization [42].

In his seminal work [41], Bertin laid out the foundations for the graphical representation of data. He separated the process into three key components:

1. Analysis of the Information: Determination of the number of components, the number of categories for each component and the level of organization of the categories (quantitative, ordinal and nominal).
2. Properties of the Graphic System: 8 variables are available for the encoding of information. Two dimensions of a plane and the six retinal variables (color, shape, size, saturation, texture and orientation).
3. Rules of the Graphic system: Correct mapping of data to variables and schemas of construction.

Mackinlay formalized Bertin's rules as part of his work to automate the design of graphical presentations of relational information. Drawing upon both the empirical work of Cleveland and McGill, and psychophysical results and various analyses of different perceptual tasks that were available at the time, Mackinlay created the ranking of perceptual tasks show in Fig. 4.1 [43].

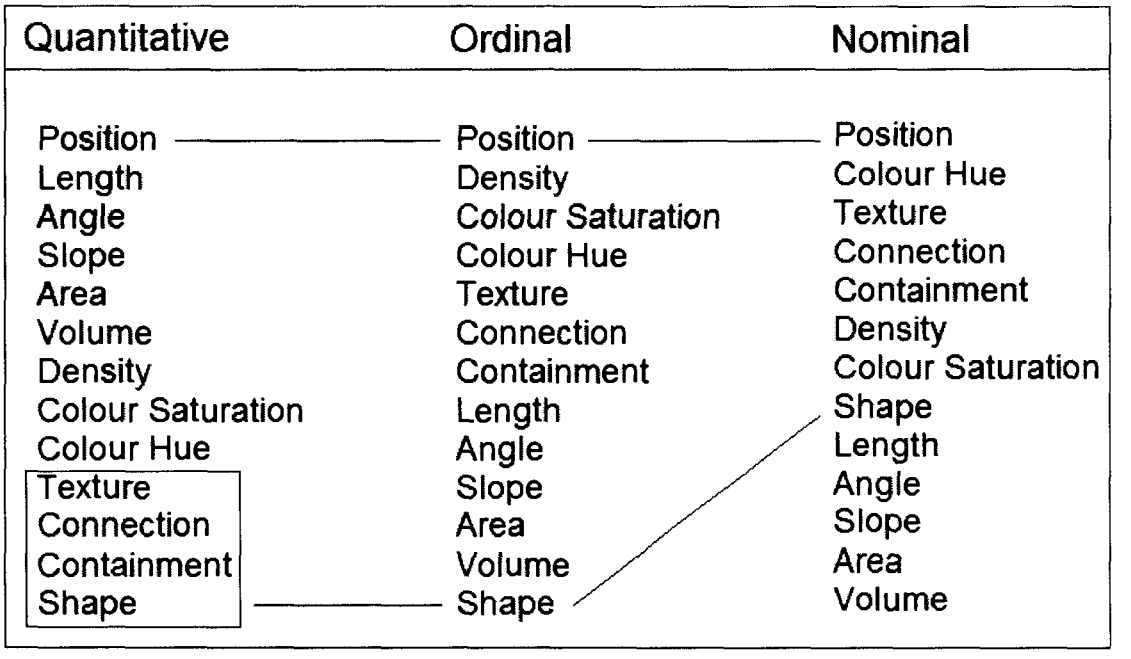


Figure 4.1 Mackinlay's ranking of perceptual tasks. Attributes are listed in order of their ability to portray each of quantitative, ordinal and nominal variables. As can be seen, **position** is always the best attribute to encode information with and **shape** is the worst for quantitative and ordinal variables. The four attributes contained in the inner box in the quantitative column are of equal rank.

4.2 Trees (Hierarchies)

One of the common structures in information is a hierarchical structure. Ontologies have a hierarchical structure based on the is-a relationship between concepts so visual layouts based on trees are a natural way of representing them. Due to the prevalence of hierarchal structures within data, there has been a lot of research into finding effective ways to display trees. The two primary techniques that are employed for drawing trees are connection and containment. Both methods have strengths and weaknesses. For our work with large ontologies like the NCI thesaurus, there are two primary concerns when

investigating potential display techniques. The time complexity of the layout algorithm needs to be low in order to maintain good interaction for the users and ideally the layout needs to make efficient use of available screen space so large numbers of concepts can be displayed simultaneously.

4.2.1 Connection

The node and link technique used to join children to their parents is the most familiar tree drawing technique to the general populace [42]. In such visualizations, nodes represent concepts and the is-a relationships are represented by links connecting the nodes. One of the primary problems with node and link diagrams is their poor utilization of screen space. Even a tree with a low branching factor may have a width that grows exponentially as the depth increases. With a typical hierarchical tree layout, like the one shown in Fig. 4.2, a tree with 10 levels and a branching factor of only two would have only one pixel per node on a common 1024x768 display. For a connection layout technique, there are really just two possible alternatives that can be used to overcome this problem. The first is to layout nodes and links in a more space efficient manner and the second is to incorporate dynamic interaction, including filtering, abstracting and focus plus context techniques. These techniques can be combined, but no matter what option is selected there are always tradeoffs. A discussion of interaction techniques is provided in Section 4.4.

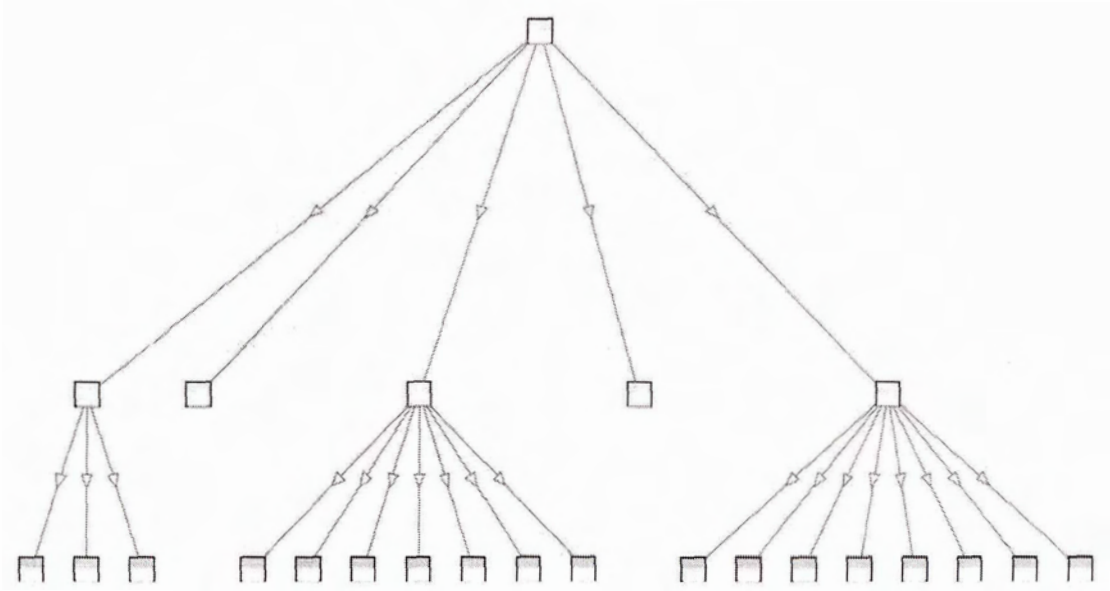


Figure 4.2 A traditional vertical tree layout

Radial layouts are one of the attempts to overcome the problem of hierarchical layout degenerating into a single line. Radial tree layouts recursively place the children of a subtree into circular wedges starting with the root of the tree at the center of the layout [44, 45]. A radial layout offers a partial solution to the exponential width of a tree with respect to its depth, but not a complete one since the circumference of the largest circle/oval that can be placed on a screen is only about three times larger than the width of the screen. Also, like a hierarchical layout, they tend to waste a lot of valuable screen real estate as can be seen by the abundant white space in Fig. 4.3 [45].

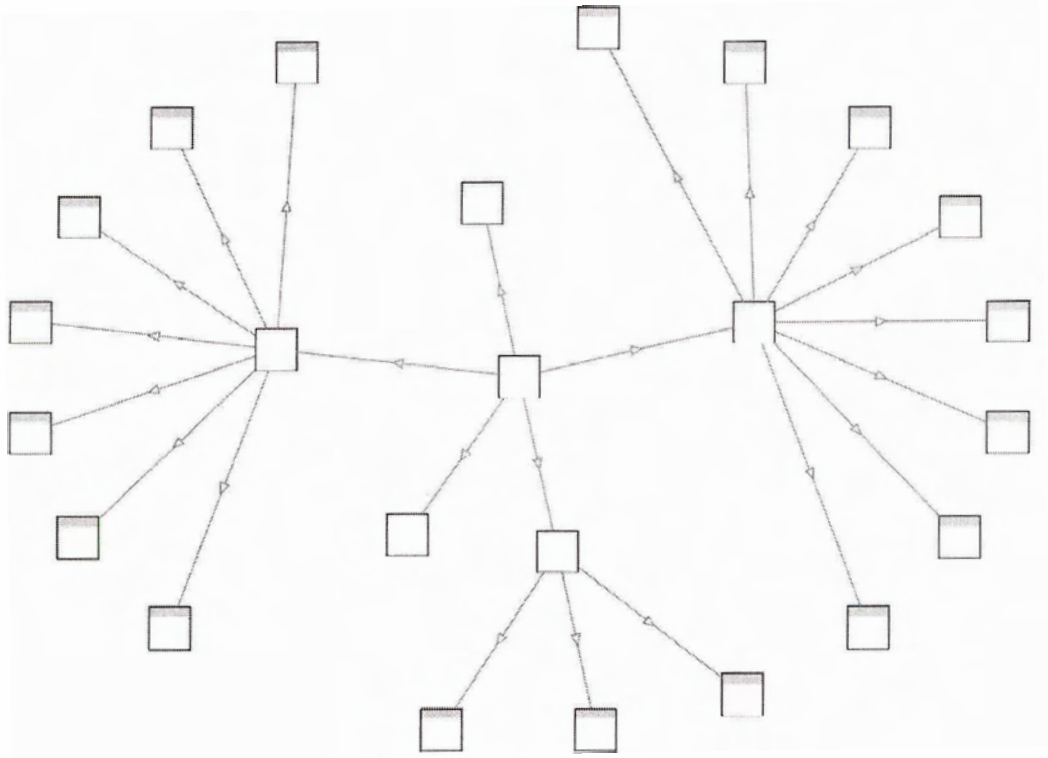


Figure 4.3 A radial tree layout

The hyperbolic layout the last node and link method I will discuss. Hyperbolic layouts mathematically solve the problem of limited screen real estate at the expense of distortion by drawing a hierarchical layout on a hyperbolic plane and then mapping it to Euclidian space. In a hyperbolic plane, parallel lines diverge, leading to the excellent property that the circumference of a circle in a hyperbolic plane grows exponentially with increasing radius, thus there is exponentially more space with increasing distance from the root of the tree [46]. The problem of wasted space is still present in a hyperbolic layout and it additionally suffers from nodes and links becoming exponentially small (in Euclidian space) as they approach the edge of the visualization.

Hierarchical, radial and hyperbolic layouts of trees all share the advantage that their algorithms are simple and fast to perform. This lends all three to dynamic interaction by the user, and has made them popular choices for many visualization tools.

4.2.2 Containment

The idea of using containment to draw a tree is not as well known to most people as the node and link structures of the connection method [42]. A tree drawn using the containment technique considers the root node to occupy all or nearly all of the available screen area. This area is then partitioned into segments for each child continuing until only leaf nodes remain. Figure 4.6 shows the same tree drawn with both node and link and containment techniques. The primary advantage of containment tree drawing methods is that they use available screen space much more effectively than node and link diagrams [42]. For large ontologies such as the NCI thesaurus, containment methods are one of the few practical techniques for displaying the entire ontology at once.

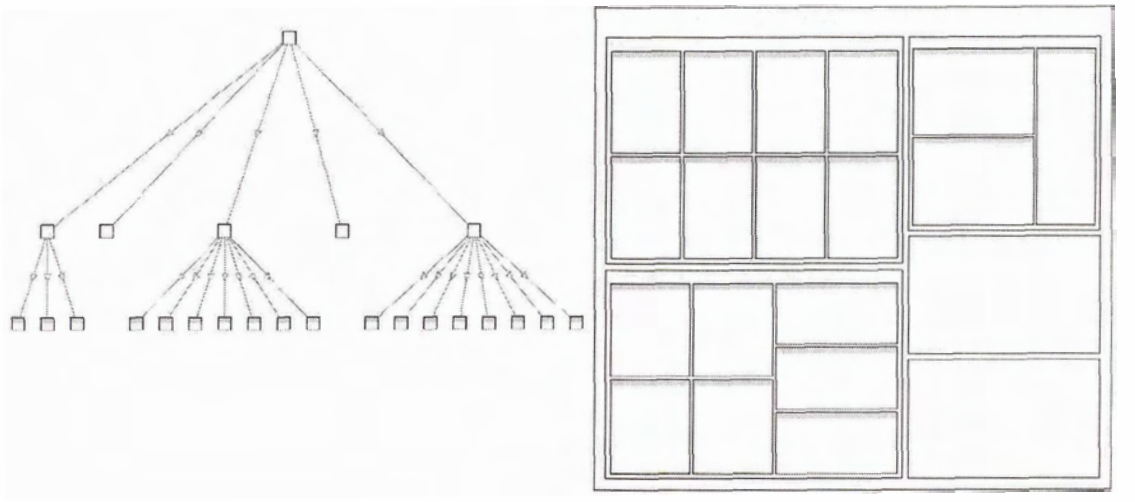


Figure 4.6 A vertical tree layout on the left and the equivalent nested treemap on the right.

Shneiderman and Johnson created the treemap approach in 1991 in response to the lack of tools that could adequately visualize the contents of a hard disk drive. This example is one of the first examples of the containment method of tree drawing in the information visualization community [10]. A Treemap can be drawn in either a flat or nested style. The flat treemap style shows only the leaf nodes of the tree whereas the nested view attempts to preserve a visualization of the hierarchical structure of the tree. Since their introduction in 1991, treemaps have been widely used in visualizations where the features of nodes are of greater importance than the hierarchical structure of the tree. It should be quite apparent that the greatest weakness of the treemap technique is their poor ability to portray hierarchical structure [47]. Despite this limitation, treemaps are a powerful technique for displaying large trees. The algorithms required for their layout are efficient (Shneiderman's original is O(n) [10] and variants that require sorting or look ahead are only slightly worse [48]), lending treemaps to highly interactive visualizations and the large space available on nodes can be used to display additional information about the node and its descendents. Some important extensions to the treemap algorithm include:

- Squarified treemap by Bruls *et al.* that reduces the aspect ratio of nodes at the expense of order [49].
- Cushion treemap by van Wijk and van de Wetering that uses a 3D cushion affect to emphasis the hierarchical structure of the tree [50].
- Quantum treemap by Bedersen *et al.* [48] that makes all leaf nodes the same size and shape and lays them out in a grid within their parent. This technique differs slightly from other treemap algorithms because the fixed size of leaf nodes prevents complete space filling in many cases; however, it can be very useful if leaves in the information space represent photos, as is the case with the PhotoMesa [51] program.

Each of the different treemap layouts has different benefits and limitations that must be considered before using them. According to Bederson *et al.* [48] three important factors that should be considered are:

- **Change**: This is a measure of how much and how quickly the position of nodes in the treemap change with changes to the underlying data. The amount the layout of a treemap changes can be important if it is desirable to have users learn the layout of the treemap. For example, when presenting stock market data in a treemap it is advantageous to have a layout that is stable over time so that investors learn where the stocks they are interested in are in the treemap. Algorithms such as Shneiderman's original Slice and Dice and Strip [48] generates a layout using the order of the data and it is inherently more resistant to changes in data than algorithms like Squarified which reorders the data to achieve

lower aspect ratios. Fortunately, if the layout attribute such as market capitalization or number of descendents is known to be relatively stable, a layout like Squarified can still be a good choice.

- **Aspect Ratio**: The goal of many of the refinements to the original treemap algorithm has been to decrease the average aspect ratio of the nodes in the treemap. In general, lowering the aspect ratio of nodes in a treemap makes the nodes easier to select and easier to compare their relative sizes [49].
- **Readability**: Bederson *et al.* define the readability of a treemap according to how easy it is to find a particular item in the layout [48]. One of the main factors that can affect readability is the order of the nodes drawn in the treemap. There are three possibilities for how the nodes will appear in the treemap. The first possibility is that nodes are drawn in an ordered fashion. The original treemap algorithm by Shneiderman and Johnson is an example of such an algorithm. The second choice is to draw nodes roughly in order, but not at the total expense of aspect ratio. The last possibility is to totally ignore order and draw them with the best aspect ratio possible.

4.3 Graphs (Networks)

Unfortunately, not all information is organized according to hierarchical structures that can be conveniently displayed by containment and trees. A lot of the information spaces such as ontologies are more complex and may have multiple inheritance and cycles. Formally, a graph is a set of vertices (nodes) and a set of edges (links) where edges are defined by a binary relationship between two vertices. The space efficient containment

drawing methods available for trees cannot be directly applied to these information sets and connection methods can quickly start to resemble nothing more than a blob of lines and nodes with no apparent order. There are several approaches that are used to create useful visualizations of graphs, including finding spanning trees, clustering, filtering, and abstraction. As with tree visualizations, the use of dynamic interaction can significantly enhance the effectiveness of graph visualization techniques.

Force directed layouts are a clustering approach to creating a readable layout from a network. Typical force directed layouts incrementally calculate a minimal energy configuration for a network based on attractive forces supplied by arcs and repulsive forces between nodes; however most of these layouts suffer from $O(n^2)$ time complexity [52-54] making them unsuitable for dynamic interaction especially when used for large data sets such as the NCI thesaurus.

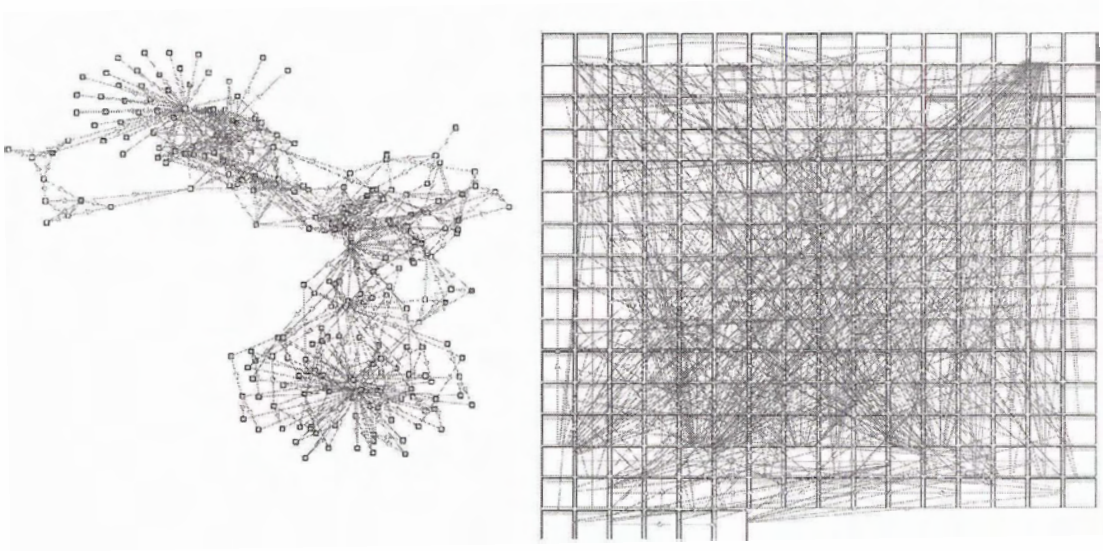


Figure4.7: A Force directed graph layout versus the same graph with nodes arranged in a grid

4.4 Interaction

Even when it is relatively easy to compose a layout for a data set, a static display may not be that informative to the viewers. The viewers may need to interact with the display to understand it further. The processing capability of a computer allows rapid interaction with visualizations and provides significant advantages over non-interactive layouts by shifting the workload from the user to the computer [42]. Navigating and understanding large information spaces is difficult without interactive navigation and comprehension aids [55] and research is ongoing to create suitable visualization environments for the ever-increasing size of information spaces [56]. What follows is a discussion of some interaction techniques that help users navigate and comprehend an information space even when it is large and complex.

4.4.1 Zoomable user interfaces

Although the idea of zoomable user interfaces (ZUIs) has been around for several decades [57], Ken Perlin and David Fox were the first to implement the paradigm of zoomable user interfaces in their programmable Pad system [58]. Pad is an infinite 2-dimensional information plane that contains graphics and portals occupying well-defined geographic regions on the Pad surface [58]. Graphics were the visible objects on the Pad and portals were cameras to view the contents of the Pad. Perlin and Fox also introduced the concept of semantic zooming; the idea that objects on the Pad would have different graphical representations at different magnifications. This original work on zoomable user interfaces has been extended by Ben Bederson with Pad++ [57, 59], Jazz [60], and Piccolo [61].

It would seem amiss to discuss ZUIs without providing some sort of definition; however, Hornbæk *et al.* [62] say that there is not an accepted one. Bederson compiled a list of eight criteria that a ZUI should meet in [60]. The most important aspects of a ZUI to us are: smooth animated pan and zoom view navigations, semantic zooming, the ability to handle large numbers of objects and a drawing surface whose size is restricted only by the floating point precision of the system the ZUI is running on.

Just as most developers would not consider developing a GUI without using an API such as SWING or MFC, building a program based on the ZUI paradigm is greatly facilitated by the use of an appropriate library and API. One such example is the Piccolo toolkit for creating Java applications with ZUIs. It was developed by the HCIL group at the University of Maryland College Park. Piccolo is based on a compact monolithic architecture [61] that allows quick and easy development of new visualization tools based on the ZUI paradigm. Piccolo works in much the same way as Perlin's original Pad system. Piccolo provides the PCanvas class that provides the core functionality of a drawing surface and the PNode class that virtually all canvas items descend from (hence the monolithic design). PNodes can be both visible canvas objects, like nodes and arcs, or cameras to provide a portal to view the canvas. Piccolo provides event handling, double buffering for smooth animations, intelligent repainting algorithms and the rest of the features you would expect of a current ZUI API.

Finally, it is worth noting some tools and programs that have been created using ZUIs. Bederson's Photomesa [51]is an image browsing and management tool using the Quantum treemap layout. The CHISEL group at the University of Victoria has created several tools such as SHriMP [63] and its Protégé incarnation, Jambalaya [38], that utilizes many different graph layout techniques all based on the Jazz ZUI API. One other ZUI based tool that takes a unique approach by providing a semi-transparent overview in the background of the current view is called Zomit [64]. The creators of ZOMIT have used it to create a tool to browse genomic data based on the location of the coding sequence within the genome.

4.4.2 View Coordination

The technique of linking involves providing multiple views of the same data that are tightly coupled so that interaction with any one of the views is immediately reflected in the other views. As with all interaction techniques, speed is important. It is known from basic usability theory that in order for an action to be perceived as immediate, it must happen within $1/10^{th}$ of a second of the control input [42]. Thus, when multiple views are linked, one must ensure they are all updated within $1/10^{th}$ of a second in order to be perceived as a single event by the user.

The linking effect has its origins with Becker and Cleveland's work on Scatterplot brushing [65] in 1987. Since that time, linking has become a standard technique [66] and there are many examples of its usage in the research literature. Tweedie *et al.* used linked scatterplots in their Interactive Visual Artifacts [67] to visualize complex

multidimensional data generated by mathematical models. North, Shneiderman and Plaisant linked multiple 2-dimensional cross sections of the Visible Human project's 3-dimensional image data [68] and Seo and Shneiderman used linking to help visually explore multidimensional microarray data [69]. In 1997, North and Shneiderman offered a taxonomy of possible linking techniques between two views for coordinating different views of the same data and views of different but related data [70]. They proposed three methods of coordinating views:

1. Selecting items ↔ selecting items. An example of this method of coordination is an item selection in one view results in the synonymous item being selected in other views.
2. Selecting items ↔ navigating views. An example of this method of coordination is when an item is selected in one view a navigation view (like an overview window) moves so the selected item is visible in the navigation view.
3. Navigating views ↔ navigating views. This method of coordination refers to navigation events in one view, such as a zoom or pan, being reflected by a navigation event in another view. This keeps the views synchronized.

North and Shneiderman also noted that the benefits of linking multiple views in visualization environments has been previously shown to increase user performance and believe it also aids in the discovery of unforeseen relationships [70].

4.4.4 Overview + Detail

The traditional usage of an overview is to provide a small overview window that is typically at least 1/16 [62] of the size of the detailed view. The overview shows the area that is currently the focus of the detailed view and also allows quick navigation around the detailed view. Plaisant *et al.* state that the difference in scale between the overview and detail windows can be as great as 20 to 1, but concede that the choice of scale between the two windows is a delicate issue [71]. It is also agreed by most researchers [62, 71, 72] that tight coupling of the overview and detail windows is imperative and expected [72] by users.

There are many research tools and commercial software that use the overview + detail paradigm such as SeeSoft [21], Adobe Photoshop and many computer games. This is not surprising given that both theoretical guidelines and empirical evidence support the addition of overviews in many situations. Studies by North and Shneiderman [72], Hornbæk *et al.* [62], and Hornbæk and Frøkjær [73] have shown that overview + detail visualization environments can significantly enhance user satisfaction and in the cases of exploring census data [72] and document reading [73] provide excellent gains to user performance as well.

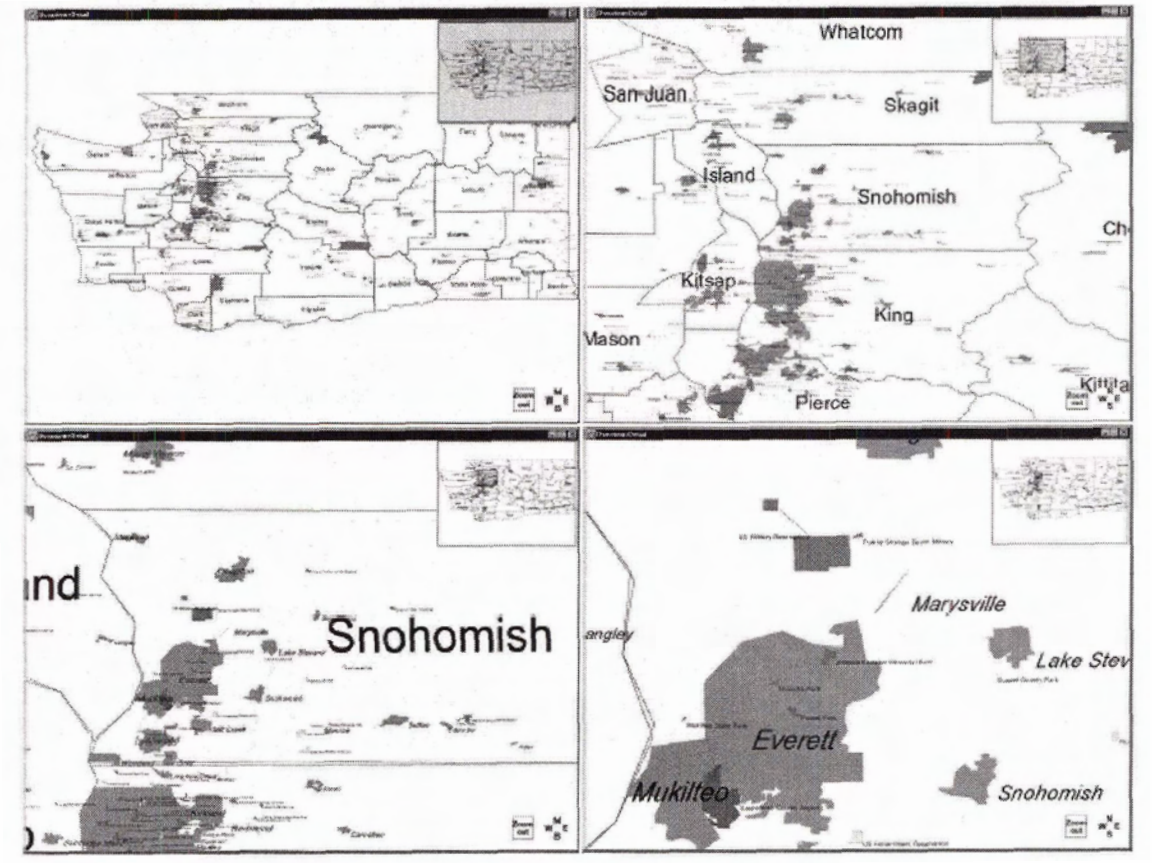


Figure 4.8: Traditional overview + detail [62]

4.5 Difference Visualizations

Before beginning the development of our tool, an effort was made to find examples of previous systems that created difference visualizations of ontologies. The only two published attempts to date appear to be PROMPT and OntoView. As discussed in Chapter 2, neither of these tools provides adequate overview visualizations. Given the dearth of examples from the ontology engineering domain, examples from other fields of study where visualizations of the differences between hierarchical structures have been researched were sought instead.

There are many parallels between software engineering and knowledge engineering and this lead to the obvious path of looking at visualizations that have been done to compare software systems. Inspiration has been drawn from Baker and Eick's space-filling software visualization system SeeSys [22] and from Xiaomin Wu's work on visualizing CVS repository change logs called Xia [7]. Another area of study that has tackled difference visualizations is that of taxonomy comparison. Given the hierarchical nature of most ontologies, the work for visualizing differences in taxonomies is a close match. Martin Graham's taxonomy comparison visualizations are one such work that was found to be influential to the task of visualizing the differences between ontology versions [66].

4.5.1 SeeSys

The SeeSys system by Baker and Eick, is a space filling visualization for code bases that are organized in an hierarchical fashion consisting of subsystems, directories and files. The technique creates a visualization based on statistics about the code base such as:

- Lines of code in each file, directory, subsystem.
- Location and scope of changes to the source files.
- Number of programmers making changes.
- Location and number of bugs.
- Other code statistics maintained by a version control system.

By displaying these statistics on a treemap layout of the code base, SeeSys can help project managers answer questions about the code such as:

Which subsystem is the largest?

- Where are the latest development activities occurring?
- Are any subsystems error prone?
- What areas of the project are most stable/most volatile?

The strong similarity between the hierarchical organization of subsystems, directories and files in large software projects and the is-a hierarchy of ontologies makes the visualization techniques of SeeSys applicable to ontology versioning. An example of SeeSys's treemap layout can be seen in Fig. 4.9.

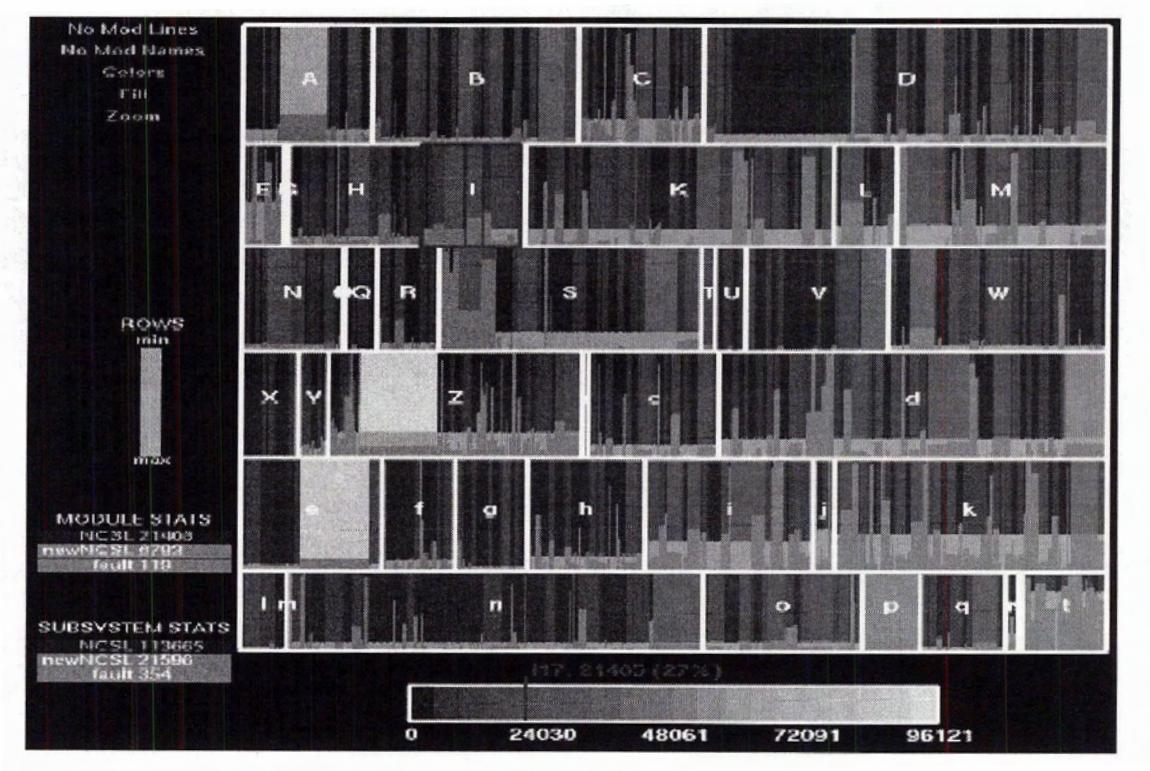


Figure 4.9 SeeSys software system visualization using a treemap layout

4.5.2 Xia

Xiaomin Wu's Xia tool is a plug-in for the eclipse platform that uses multiple layout techniques within a ZUI to visualize statistics about code changes from a CVS repository. The visualizations are a customization of the SHriMP visualization framework developed by our colleagues in the CHISEL lab at the University of Victoria and as such, they are based on the Jazz toolkit from UMD. One of the key components of Wu's Xia system is the attribute panel. The attribute panel allows users to customize various aspects of the visualization including:

- Tooltip information
- Which attribute to base node colouring on
- Filtering based on any attribute

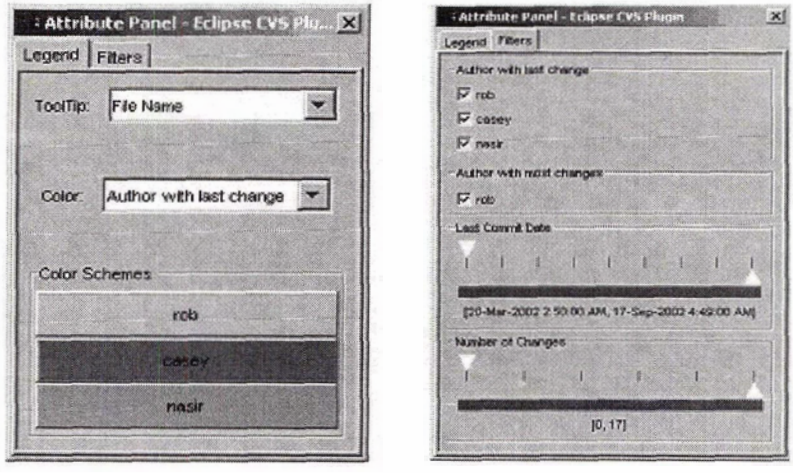


Figure 4.10: Xia attribute panel allows the user to customize what is shown in the visualization [7]. Used with permission from the author.

Wu identified what she coined the 5W + H questions (Who, What, Where, Why, When and How) that are key for developers attempting to understand the revision history of a

software project. The detailed dynamic queries that are made possible by the attribute panel allow users to answer her set of questions. While the idea of dynamic query visualization is not new, as Shneiderman published on this topic in 1994 [74], Wu's application to software versioning is an important contribution.

4.5.3 Graham – Thesis work on taxonomy comparison

The visualizations developed by Graham [66] were designed to help researchers developing taxonomies understand how concepts moved within the classification. His prototype based on linking multiple icicle plots was found to be beneficial to his users. Icicle plots are a tree drawing technique that could be described as a traditional vertical tree layout drawn without arcs. They are a compact tree representation that arranges branches of a tree into discrete columns. The root node is represented by a bar the width of the display area. This width is partitioned into columns for each of the root's children which are again represented by a bar the width of their column. This process continues until the leaf nodes are reached. Barlow and Neville found that icicle plots equaled traditional vertical node and link tree layouts for ease of hierarchy interpretation while using screen space more efficiently than vertical tree layouts [47]. As can be seen in Fig. 4.11, an icicle plot was created for each version of the taxonomy of interest and a brushing technique was used to highlight synonymous concepts across the multiple versions.

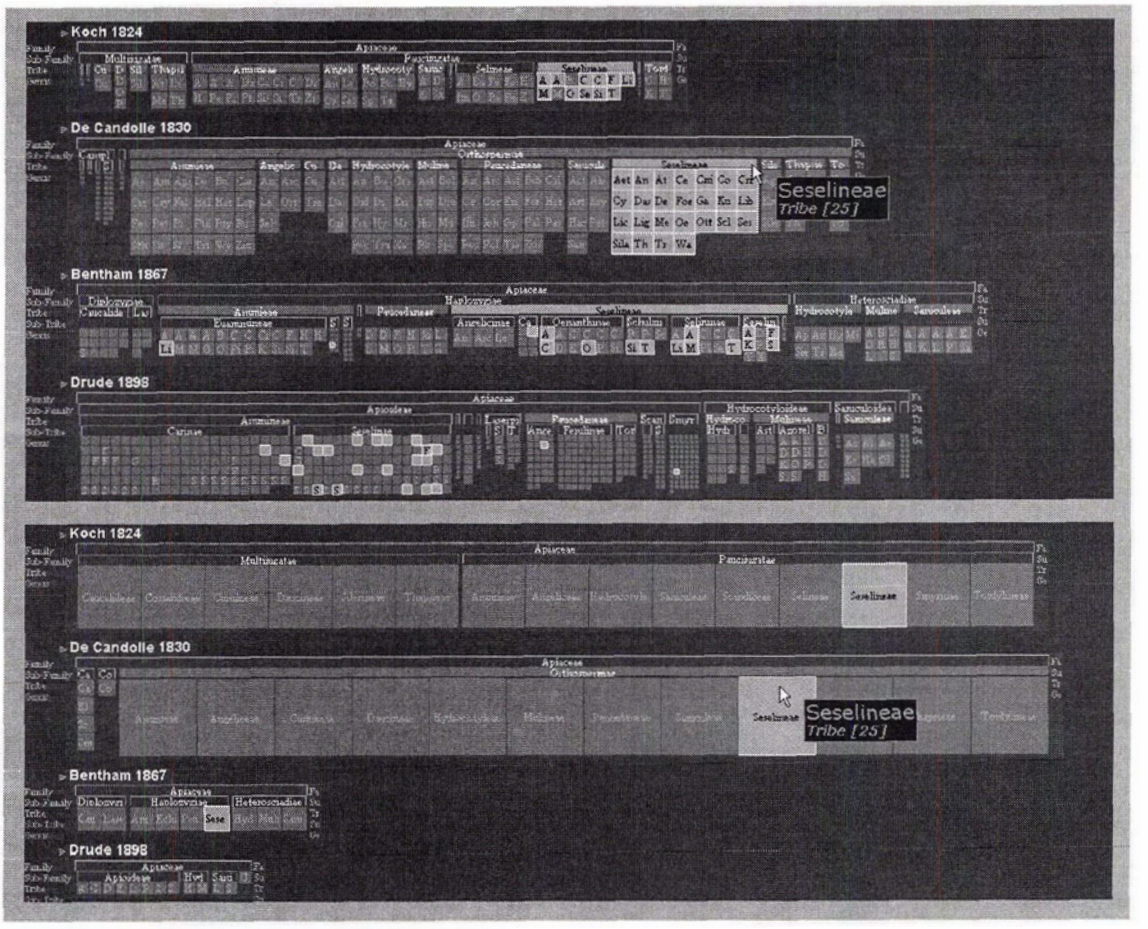


Figure 4.11: Graham's visualization for overlapping classification hierarchies using brushing to highlight synonymous concepts across several different hierarchies [66]. Used with permission from the author.

Graham found the display technique did not work for taxonomies with more than 1500 concepts due to individual elements becoming too small for users to accurately select using the mouse; however, up to that point the brushing quickly allowed his users to ascertain the reclassification of concepts. Additionally, Graham found that if not all the taxonomy versions were displayed on the screen at once, his users would forget about the versions that were not currently visible. In a ZUI, this problem would need to be addressed, as much of the display area is often not shown.

4.5.4 Difference visualizations Summary

There are many tools that have been written to visualize the changes. We have described three here that were an influence to our work and several others more briefly in Chapter 2. Other visualization systems such as Augur and SeeSoft that use compact line wise representations to visualize source code changes were two of the systems mentioned in Chapter 2. They are also important techniques but they had no direct influence on our design beyond their use of colour coding to accentuate details such as author, structure type and date of last change.

4.6 Chapter Summary

It is always a challenge to create a good visualization, but understanding the strengths and weaknesses of various techniques, the limits of human perception, the limitations of your data set and the needs of your users are steps in the right direction. In this chapter we summarized many of the key techniques from the area of information visualization that could be applicable to ontology version visualizations. Particular emphasis was placed on the techniques that were used in PROMPT-Viz and in the three difference visualization tools that were discussed in this chapter. With all the required background now presented, the details of the implementation and design of PROMPT-Viz can be presented.

Chapter 5: PROMPT-Viz

To recap the motivation for the development of PROMPT-Viz, the increasing adoption of ontologies as a means of organizing and sharing knowledge is resulting in large collaborative ontology development efforts like the NCI thesaurus and the increased likelihood of several different versions of the same ontology being available. As soon as there are multiple modelers working simultaneously on the same ontology, the need for tools that can determine the differences between ontology versions is increased. The same need is true for ontology users who need to know exactly how the changes to the latest ontology will affect their work when they make the transition. At least two tools have been developed to help knowledge engineers discover the differences between versions of the same ontology [6, 28] but, neither includes extensive visualizations like those that have been applied for versioning tools in other domains such as software engineering. More extensive visualizations may better support the cognitive demands of understanding the differences these two tools can detect.

PROMPT-Viz builds on the PROMPTDiff part of the PROMPT plug-in for Protégé. PROMPT-Viz leverages the difference table generated by PROMPTDiff to create a treemap display showing the differences between the two versions of the ontology. This visualization aims to provide the ability for users to determine the **L**ocation, **I**mpact, **T**ype and **E**xtent of the changes that have occurred to the ontology (i.e. we call these the **LITE** set of questions for the purposes of brevity in the text). The original table and tree views of PROMPTDiff that are effective for providing the details about changes to each concept in the ontology are left intact. The additional treemap view added by PROMPT-

Viz complements these two original views by allowing users to determine which concept changes they should explore. PROMPT-Viz also contains an overview window to help users keep track of where they are in the ontology, a panel containing a detailed list of the changes to a currently selected concept and a traditional horizontal tree view of the ontology. Figure 5.1 shows how PROMPT-Viz extends the existing architecture of the PROMPT plug-in for Protégé.

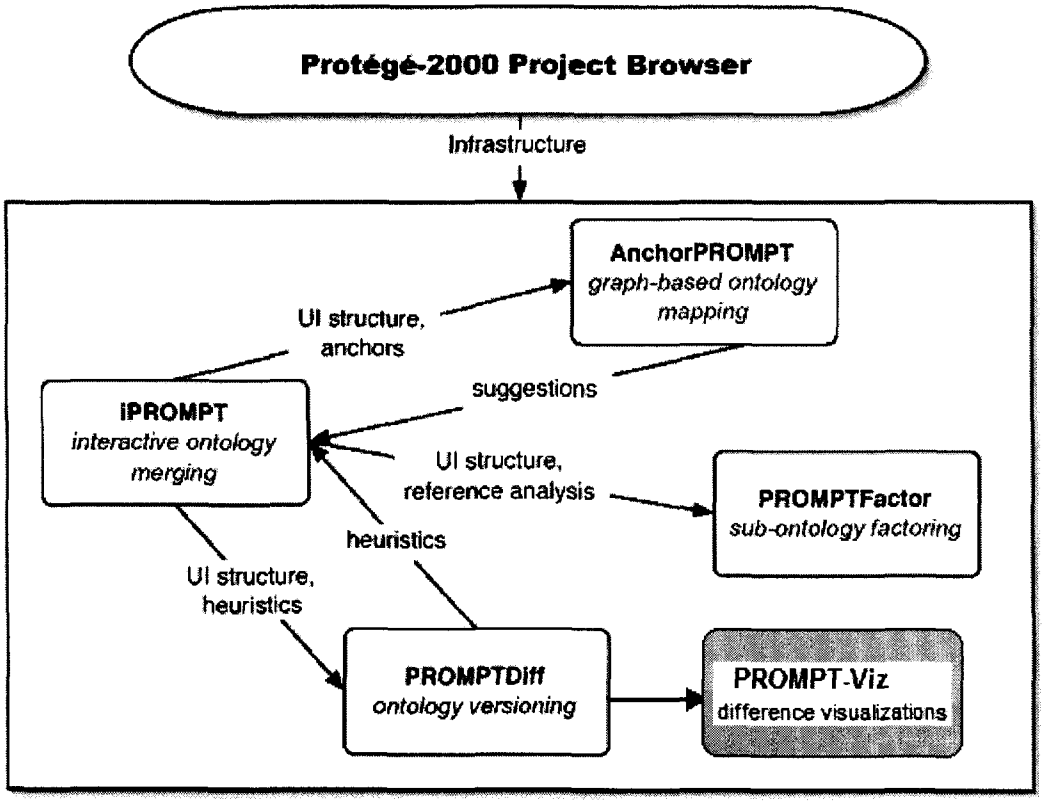


Figure 5.1 Architecture of the PROMPT plug-in showing PROMPT-Viz extending the PROMPTDiff component.

5.1 Requirements

The goal of PROMPT-Viz is to provide cognitive support to users of PROMPT who need to understand the **L**ocation, **I**mpact, **T**ype and **E**xtent of differences between versions of large ontologies. My colleague Neil Ernst visited the enterprise vocabulary group with the purpose of determining their visualization requirements for the NCI thesaurus. He had meetings with the managers and performed contextual inquiries with 2 or 3 of the modelers to understand their visualization requirements. The specific requirements we inferred were based on the needs of the modelers and managers at the enterprise vocabulary group of the NCI as discovered during Neil Ernst's visit and subsequent conference calls, and from our discussions with two expert ontology users from the University of Washington and several members of the Protégé team at Stanford University. It was based these needs that we determined that PROMPT-Viz should help users answer what we termed the **LITE** set of questions.

While realizing the aforementioned requirements, PROMPT-Viz has to fit within the following constraints:

- PROMPT-Viz must provide meaningful visualizations for differences between ontologies containing tens of thousands of concepts.
- PROMPT-Viz must remain interactive for such ontologies, i.e. it must be efficient.
- PROMPT-Viz must run as an extension of the existing PROMPT plug-in for Protégé.

5.2 Features

The PROMPT-Viz display is divided into frames as shown in Fig. 5.2. The four linked frames are:

1. An expandable horizontal tree layout of the ontology showing the differences. This widget is part of the original tree view of PROMPTDiff.
2. A treemap layout of the ontology embedded in a ZUI.
3. A path window that shows the location of the currently selected concept within the is-a hierarchy of the ontology.
4. A detailed list of the changes (if any) that have occurred to the currently selected concept. This widget is part of the original table view of PROMPTDiff.

The technique of linking has successfully been used in the past to tie multiple views of the same data together [56, 75]. Linking is used to synchronize the familiar Protégé tree view, the path to the current selection within the is-a hierarchy and a detailed listing of the changes to the selected concept with the treemap display. This allows the strong hierarchical representation of the tree view and path to complement the additional relationship and change data that is represented in the treemap and still have immediate access to the details of any change. An individual discussion of each of the components is presented next.

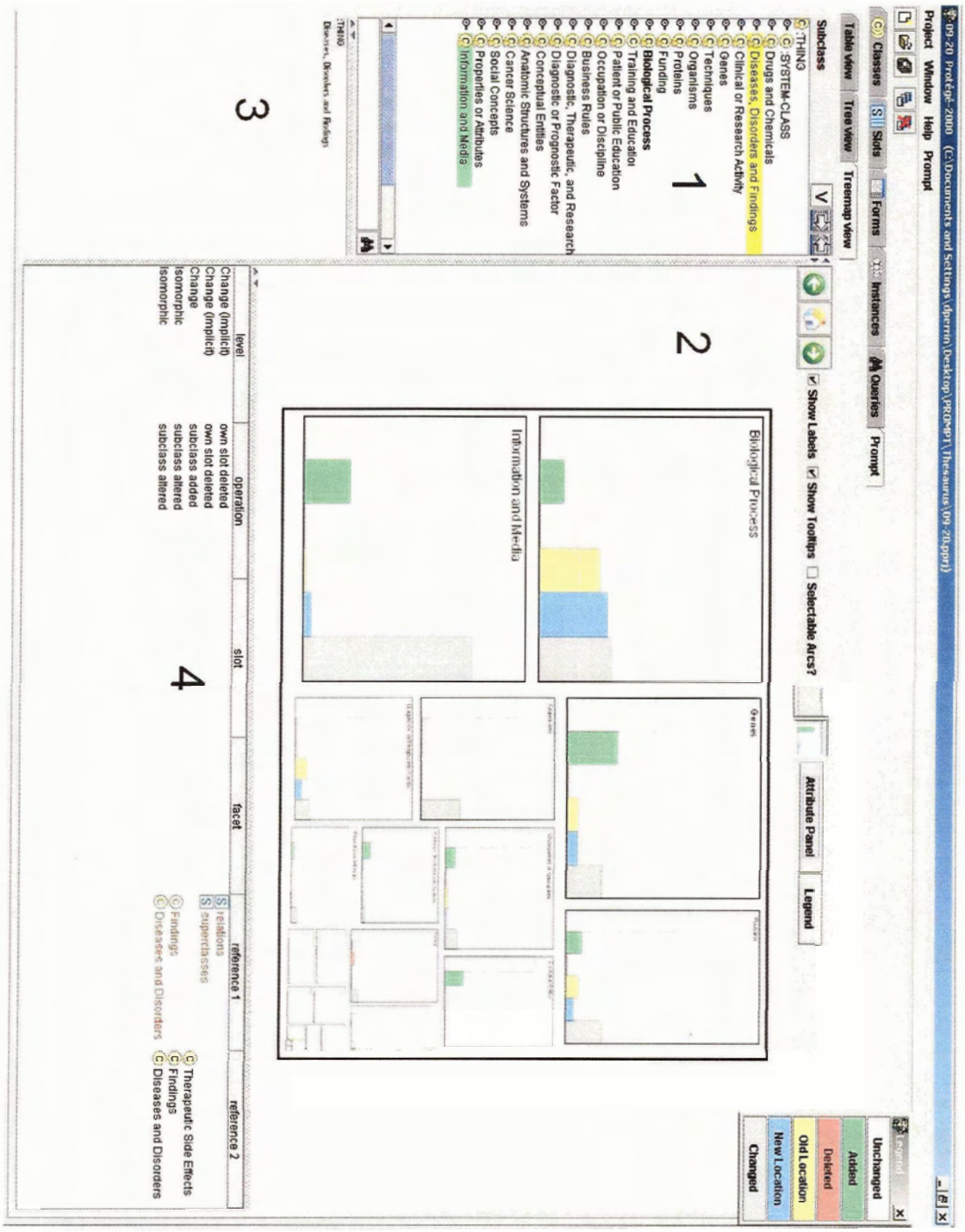


Figure 5.2 PROMPTViz in histogram colouring mode, with the treemap sized so classes with the most number of changed descendents are drawn largest. The bars in the histograms, ordered from left to right, represent the percentage of descendents classified as unchanged, added, deleted, moved-from, moved-to and directly changed respectively.

5.2.1 Horizontal Tree component

The expandable horizontal tree layout (Area 1 of Fig. 5.2) is a standard Protégé frame with a customized renderer to allow changes to be represented. This component was part of the original PROMPTDiff tool and was integrated into our tab because it contains a search tool so that specific concepts can be quickly located. It provides a strong hierarchical representation of the ontology and a familiar interface for Protégé users.

5.2.2 Treemap component

The main visualization component of PROMPT-Viz is the treemap layout embedded in a ZUI seen in area 2 of Fig. 5.2. The ZUI is a powerful visualization technique that allows us to provide an overview of the changes to an entire ontology or the details of changes to a specific area of interest. Given the large size of the NCI thesaurus and our previous positive experiences using ZUIs, a ZUI was chosen as the foundation on which to build PROMPT-Viz. As Pook mentions [64], ZUIs are no longer a new idea and it is generally well accepted that the first view should be at an appropriate scale to show the entire information space. Correspondingly, the first view in PROMPT-Viz is an overview of the entire project. This gives users a familiar base to start their navigation.

Although drawing a tree using a containment style like treemaps does not provide the best representation of the hierarchical structure [47], the space efficiency is unrivalled by node and arc representations [42]. The space efficiency of the treemap display and the low time complexity of the algorithm allow the display of changes to tens of thousands of concepts simultaneously. In PROMPT-Viz, the squarified treemap algorithm is

employed because the low average aspect ratio of the nodes are easier for users to select, easier to compare the relative size of, more aesthetically pleasing and better for displaying additional change data.

By using the treemap space-filling tree drawing technique, a large area is available for displaying additional statistics about nodes and their descendents. In PROMPT-Viz, this area is painted in two different styles to convey this information. Firstly, the nodes can be filled with a solid color to represent how the class it represents differs between the two versions of the ontology under investigation. There are six different colors a node (class) can be filled with, each signifying a different type of change. These colours were chosen based on the colour scheme of PROMPTDiff and are outlined in table 5.1.

Color	**Representation**
White	The Class has no direct changes between versions.
Gray	The Class has direct changes compared to the previous version. This difference may be as simple as a name change or as complex a multiple changes to the inheritance and slots of the class.
Red	The Class has been deleted from the new version of the ontology.
Green	The Class has been added since the previous version of the ontology.
Yellow	The Class no longer exists at this location of the is-a hierarchy, but it still exists in at least one other location.
Blue	The Class did not exist in this location in the previous version of the ontology, but it did exist in some location.

Table 5.1 Fill color representation in PROMPT-Viz

The second filling technique is called the histogram colouring mode and provides information about the percentage of descendents that are categorized according to each type of change (see the categories in Table 5.1). The histogram colouring was inspired by Baker and Eick's SeeSys tool for exploring changes to software systems [22], but differs from their approach in that each histogram bar in PROMPT-Viz represents a subset of descendents whereas each bar in SeeSys represented statistics about a single child. Nodes are filled with six colored (white, grey, red, green, yellow and blue) bars, each representing the percentage of descendents that were classified according to the type of change by PROMPTDiff. Each bar's height is proportional to the percentage of descendents of the node that are of the change type the bar represents. In the case of leaf classes, the fill type reverts to a solid color.

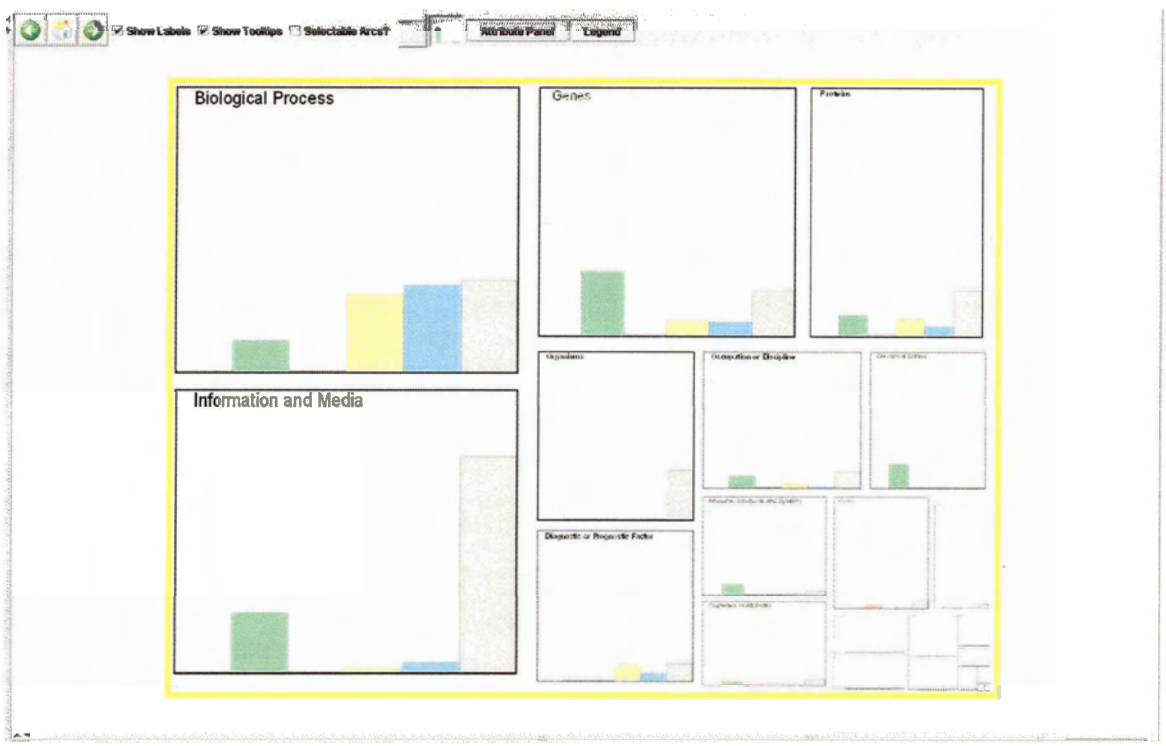


Figure 5.3 The histogram colouring mode with the treemap sized by percentage of descendents that have any type of change

One common scenario that occurs during ontology development is the decision that a class should be reparented or should have one of its parents removed. In such a case, the entire subtree under the class in question will no longer exist. As previously described, such classes are coloured yellow. Additionally, all the remaining nodes in that subtree are made non-selectable and semi-transparent so the yellow color of the moved class will be visible even when the subtree is expanded. A dotted arc connects the moved sub-tree other instances within the ontology. These features are shown in Fig. 5.4.

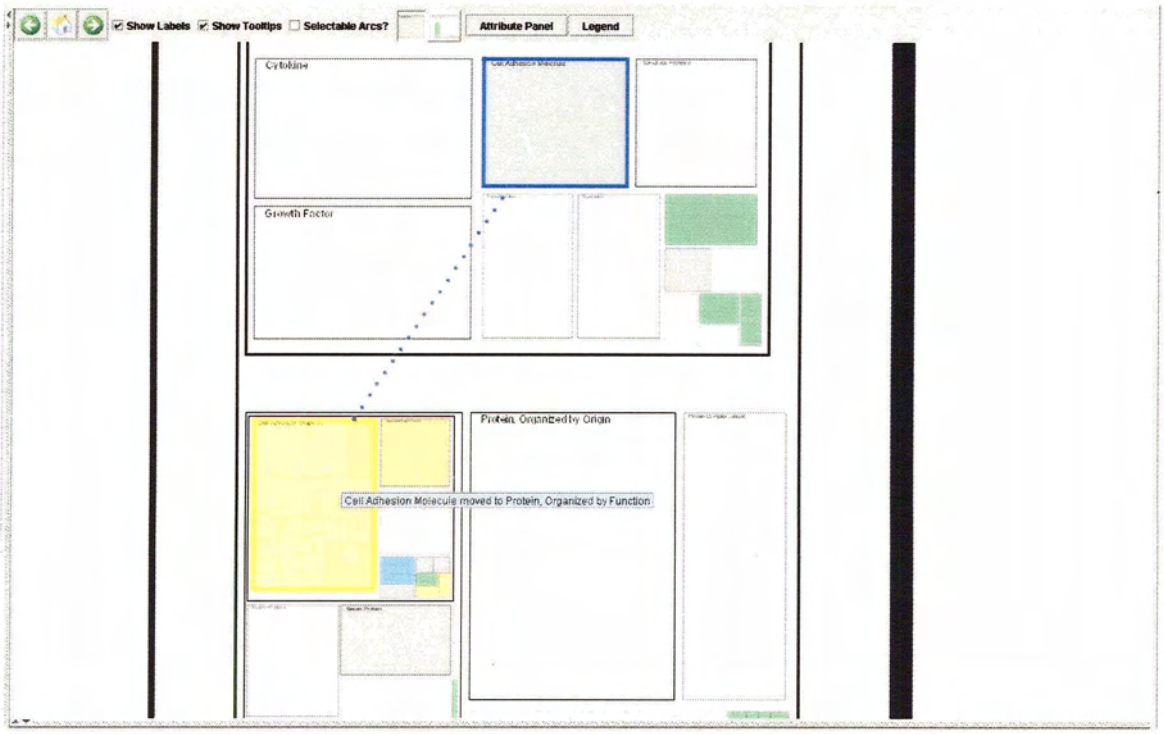


Figure 5.4 Reclassification of cell adhesion molecule shown by dotted arc. Note the subclasses of the old position of cell adhesion molecule are greyed (actually yellowed) out because they are no longer in this position.

Treemaps also have the desirable property that the nodes can be sized based on any statistic or attribute of interest. This allows many resizing options for the treemap display, providing users with the ability to create a view that emphasizes number of descendents, number of any combination of change type or percentage of any combination of change type. Resizing the treemap display can help users quickly answer questions concerning type and location of changes such as "Where have the most new concepts been added to the ontology?"

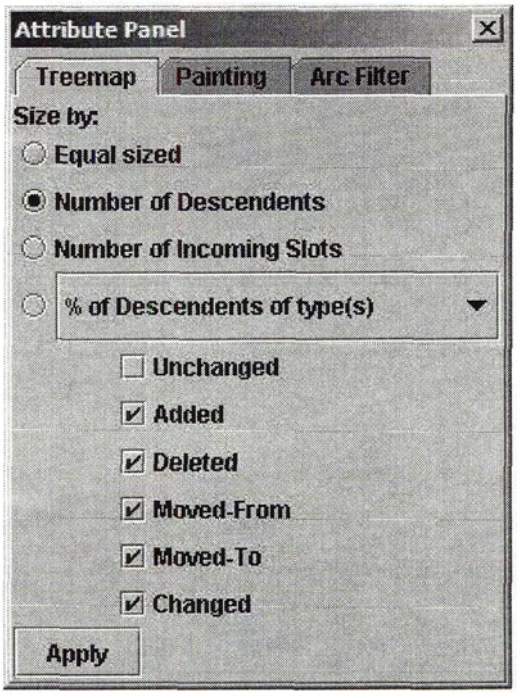


Figure 5.5 The attribute panel showing the numerous ways the treemap algorithm can be customized

Treemaps were designed for displaying simple hierarchical structures; however ontologies do not have this constraint. As was alluded to earlier, many of the differences between versions of ontologies are due to changes in the multiple inheritance of classes. Dotted arcs are used to link the multiple locations of classes together, this provides visual

cues to any potential multiple inheritances. The arcs and outlines of the location are drawn in red for old locations and blue for new and current locations.

Arcs are also used to show slots to and from other classes. The arcs are colored according to how the destination class has changed. Clicking on a node will reveal a web of connections to and from other nodes. By looking at the colours and the pattern of where the arcs go to and arrive from, it is possible to discover how changes to other portions of the ontology are affecting the selected concept. This view may help expose unforeseen ramifications of modeling choices made in other areas of the ontology. Due to their potential to clutter the display, arcs are only drawn for the currently selected concept and can be turned off if they are unwanted.

Simple tooltips and labels that display the name of the class a node represents and a brief description of the change are also part of treemap display and like arcs, can be turned off.

5.2.3 Path through the hierarchy

Located in area 3 of Fig. 5.2, the path component has a twofold purpose. First, it provides an overview of the position of the selected concept within the is-a hierarchy of the ontology. This is something that can be difficult to determine from the tree or treemap views especially if the selected concept is many levels down the hierarchy and the ontology has thousands of concepts. Second, it acts as a navigation aid for the treemap component. The treemap display can be zoomed to show the entire bounds of any level in the path by a right mouse click on the equivalent level in the path.

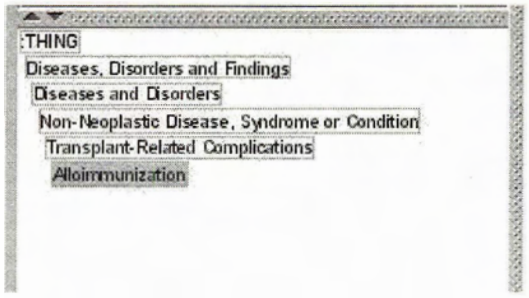


Figure 5.6 The path to Alloimmunization, the currently selected concept

5.2.4 Detailed list of changes

The final component of the PROMPT-Viz display is a detailed list of the changes to the selected concept. This pane is also contained within PROMPTDiff's "table view" tab, but the importance of providing this information without switching tabs warranted its inclusion within the same tab as the treemap. The details that are provided in this pane for each change that has occurred to the concept include the classification of the change, the change operation and the reference frame in the old and new versions of the ontology. Like all four components in Fig. 5.2, the user can adjust the screen space used by this list.

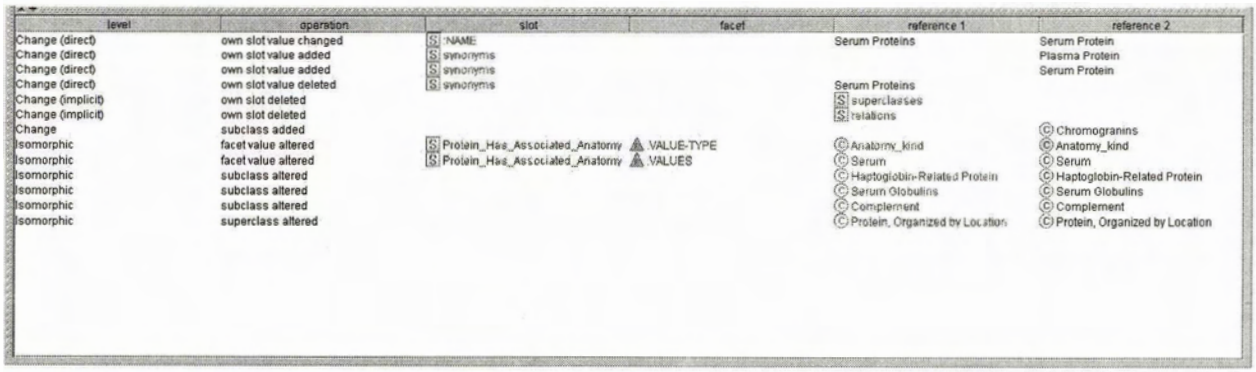


Figure 5.7 The detailed change panel

5.3 Implementation

The implementation of PROMPT-Viz extends Protégé's API [76] and the PROMPT plug-in for Protégé. It uses the Piccolo [61] ZUI toolkit from the University of Maryland and the publicly available treemap algorithms from the University of Maryland [77]. The purpose of this section is to provide further technical details on the implementation.

5.3.1 Protégé plug-in development & PROMPT extension

As mentioned in Chapter 3, Protégé is an ontology editor, a knowledge-base editor and an extensible open-source project. Because Protégé was developed from the ground up with a plug-in architecture, it is relatively easy to develop new plug-ins for. PROMPT-Viz is written as an extension of PROMPT and the best way to integrate was to add our visualizations as an additional tab within the existing PROMPT UI. PROMPT generates a merged data structure of the differences it finds between the ontologies it is comparing and our treemap is simple a visual representation of this structure. There were several static methods available to determine the change status of a frame or sub-tree within the merged knowledge base and they have been sufficient for our prototype; however, they may ultimately be insufficient if we wish to provide more detailed change information visually, as the change status is not truly as simple as the single state returned by the aforementioned methods. To provide more detail it would be necessary to directly access the diff table entries for each frame. Accessing this detail information could allow the calculation of a change metric that could be visualized by different shades of gray for drawing classes with direct changes.

5.3.2 Piccolo ZUI from UMD

The compact monolithic architecture of the Piccolo toolkit from the HCIL group at the University of Maryland that Bedersen discusses in his paper "Toolkit design for Interactive Structured Graphics" [61], made it straight forward to create an interactive treemap representation of the merged knowledge bases that is provided by PROMPT. The monolithic architecture of Piccolo is build around the PNode class, which was extended to create both the nodes that hierarchically define the treemap and the arcs that are used to show moved concepts and relationships between nodes. One necessary feature that the Piccolo toolkit does not supply is tooltips. Our solution to this problem was solved by the creation of a custom tooltip manager and the use of the tooltip that the drawing canvas inherited from SWING's JComponent.

5.3.3 Treemap algorithms from UMD

Part of the key functionality that was built into the treemap nodes was the ability to layout their children using various treemap algorithms. The package of treemap algorithms that we employed were written by Ben Bedersen and Martin Wattenberg from the HCIL lab at the University of Maryland and used under the Mozilla public license. This code required only minor modifications to successfully use both the Squarified and Ordered treemap algorithms that were part of the distribution. The only slight problem that resulted was a difference in the implementation of the two algorithms. The Ordered algorithm when called for the root of the tree would recursively layout all of its descendents, but the Squarified algorithm only applied the layout to its children. This

resulted in two separate methods being required within the treemap nodes to do layout depending on which algorithm was being used.

5.4 Chapter Summary

Our treemap display combined with the horizontal tree, hierarchy path and detailed change frame was developed to provide more cognitive support for users to quickly understand how an ontology has changed between versions. The histogram colouring mode and the ability to resize the treemap layout based on any combination of change attributes provides the means to quickly answer location, type and extent questions. Impact questions are addressed by our implementation of arcs. We performed an evaluation of PROMPT-Viz and compared it to PROMPTDiff to determine if our visualizations did provide the conjectured cognitive support. The results of this user study along with the results of less formal focus groups and individual discussions are presented next.

Chapter 6: Evaluation

In this section we describe the evaluation of PROMPT-Viz with respect to the **LITE** set of questions. Our main method of evaluation was a user study, but because it was not possible to conduct the study with expert ontology users, we complemented the user study with focus groups and consultations with experts in the field.

During the course of the development of PROMPT-VIZ, two focus group sessions were conducted with members of the CHISEL Group at the University of Victoria; several demonstrations of the work in progress were made to knowledge engineering and information visualization experts; and finally a user study was conducted once the prototype was completed. The first focus group was held near the beginning of the development cycle to solicit feedback and discussion about the proposed visualizations. The second was held near the completion of the prototype to again get suggestions and feedback to incorporate into the final iteration of the Prompt-Viz prototype. The demonstrations and discussions with experts from the fields of information visualization and knowledge engineering were held during the development cycle. These discussions helped us focus and iterate on the requirements for PROMPT-Viz and develop the set of tasks for our user study.

During the fall of 2003, we conducted the user study with eight non-expert users to evaluate the utility and usability of the visualizations that PROMPT-Viz adds to PROMPTDiff in terms of addressing the **LITE** set of questions. The users unanimously agreed that gauging the **E**xtent, **T**ype and **L**ocation of changes was easier when using the

visualizations provided by PROMPT-Viz. They also provide valuable feedback on the usability of PROMPT-Viz's features and suggestions for future work.

6.1 Goal of the User Study

The PROMPTDiff portion of the PROMPT plug-in for Protégé can determine the differences between two versions of the same ontology and provides good mechanisms to show how individual concepts have changed, but could benefit from visualizations that help users determine which concepts they may want to know more details about. PROMPT-Viz adds a treemap based visualization that supports the cognitive demands of understanding the **L**ocation, **I**mpact, **T**ype and **E**xtent (**LITE**) of changes that have occurred to the ontology. The goal of our user study was to determine if our visualizations helped users better understand the **LITE** changes to the ontology.

6.2 Methods

6.2.1 Participants

The user study was performed by the author and involved eight computer science graduate students from the University of Victoria. Participation was solicited via an email to the computer science graduate student announcement list and in person. Ethical permission to perform this study was obtained from the University of Victoria Human Research Ethics Committee. No financial incentives were offered to the participants; however, they were told they would be provided with snacks and beverages during the experiment. All of the participants were non-expert Protégé-2000 users and had little

experience with the concepts of an ontology, the treemap layout technique, and zoomable user interfaces. Non-expert users were used for the study because it was not possible to assemble a group of experts due to logistical constraints and the heavy time demands that are often faced by such people.

6.2.2 Apparatus

The study was conducted on a single PC using Protégé-2000 version 1.9 and the most current build of PROMPT-Viz. The ontologies being compared were two versions of the NCI thesaurus from 2002. Each of the versions of the thesaurus contained roughly 20,000 concepts and many of the differences were clustered in certain subsections of the hierarchy. The computer was equipped with a 3-button mouse (several of the functions of the ZUI are mapped to the middle button) and an LCD monitor (Video cameras do not record usable images from CRT style monitors). The sessions were recorded using a separate video camera that could capture both the screen and the users.

6.2.3 Design

We hypothesized that the visualizations of PROMPT-Viz would improve user's ability to answer the **LITE** changes between ontology versions. To test this hypothesis, we had the participants solve a set of tasks that included several **LITE** type questions using PROMPTDiff and PROMPT-Viz. We chose the LITE type questions because from our requirements gathering for PROMPT-Viz we had determined that they are the types of questions ontology users want visualizations to help them answer. The independent

variable in this experiment was the order the participants used PROMPTDiff and PROMPT-Viz in. The dependent variables were the answers to the tasks and the participants' subjective opinions of how much easier or difficult the tasks were to answer while using PROMPT-Viz in comparison to PROMPTDiff.

The set of tasks and the pair of ontologies being differenced were the same for the two tools and the eight participants to allow us to gauge if familiarity with the data set and tasks would alter the results as most experts are intimately familiar with the ontology they work with and with the set of questions they need answered.

6.2.4 Procedure

For each participant, the study proceeded in 3 phases:

1. **Setup**: Consent was acquired, a pre-study questionnaire filled out, training was provided.
2. **Tool usage**: Six tasks were performed with each tool. The order of tool usage was randomized.
3. **Feedback**: Post-study questionnaire was administered once both tools had been used.

The details of each phase are as follows:

Setup:

At the beginning of the experiment, each user was read the consent script found in Appendix A that described how the experiment would proceed and remind them that they

were under no obligation to participate and could leave at any time. After they were read the consent script, the participants were asked to read and sign a consent form before they participated in the study. Before performing the set of tasks with each tool, each user was asked to fill out a pre-study questionnaire (see Appendix B) regarding their familiarity with Protégé-2000, treemap layouts and ZUIs. Once they had completed the questionnaire, they were all made familiar with what an ontology is and how Protégé-2000 represents an ontology. The notions of a treemap layout and ZUIs were both explained. This explanation was repeated when the users reached the portion of the experiment where they used PROMPT-Viz.

Tool Usage:

Each user was then shown the first of the two versions of PROMPT (either PROMPTDiff or PROMPT-VIZ). The six tasks the subjects were asked to complete were chosen based on conversations with managers from the NCI and other expert ontology users. All of these experts expressed the desire to see overview visualizations so that they could quickly determine the **LITE** questions that have occurred between versions of an ontology. Consequently, the tasks we selected for the experiment were of this nature.

Tasks:

The tasks were of increasing specificity with regards to how the ontology had changed between the two versions. The tasks were:

1. How much has the ontology changed between versions?
2. How would you summarize the changes that have occurred with respect to their type?
3. Where have the changes occurred in the ontology?

4. What area of the ontology has had the most new concepts added to it?
5. Have any new "analgesic agents" been added to the ontology? If so what are they?
6. Describe the reclassification of "Cell adhesion molecule".

As mentioned above, half of the users used PROMPTDiff first and half used PROMPT-Viz first. A description of the features for the tool was given and then the users were allowed as much time as they required to practice with the tool's functionality on a sample ontology. Once they were comfortable with the first tool's functionality, they were presented with the experimental ontology and the set of six tasks. Upon completion of the tasks, the additional functionality of the second tool was explained to them and they were again able to explore the sample ontology and were then assigned the same six tasks.

Feedback:

Once the set of tasks had been completed with both tools, the users were asked to fill out a post-study questionnaire that is copied in Appendix D. The questionnaire asked the users to rate their satisfaction of the following four points using a five point Likert scale:

1. Ease of use
2. Ease of learning
3. Intuitive to use
4. Usefulness of controls and visualizations

Next they were asked to describe which features of PROMPT-Viz they found useful or confusing and which features were missing or superfluous. Finally, they were asked to

rate whether each task they completed was more difficult to complete or easier to complete using PROMPT-Viz as compared to PROMPTDiff on a five point Likert scale.

6.3 Results

There were many interesting and positive results that emerged from the user study. They are presented according to the order of the post-study questionnaire (Appendix D).

6.3.1 User Satisfaction

The first question in the questionnaire asked the users to rate their satisfaction with PROMPT-Viz in terms of ease of use, ease of learning, intuitiveness to use, and usefulness of controls and visualizations. The average satisfaction with PROMPT-Viz for each of these four aspects is presented in Fig. 6.1. Of the rating given by the eight participants for each aspect, ease of use received one rating of neutral or less, ease of learning received two ratings of neutral or less, intuitive to use received two ratings of neutral or less and usefulness of the controls and visualizations receive no ratings below satisfied.

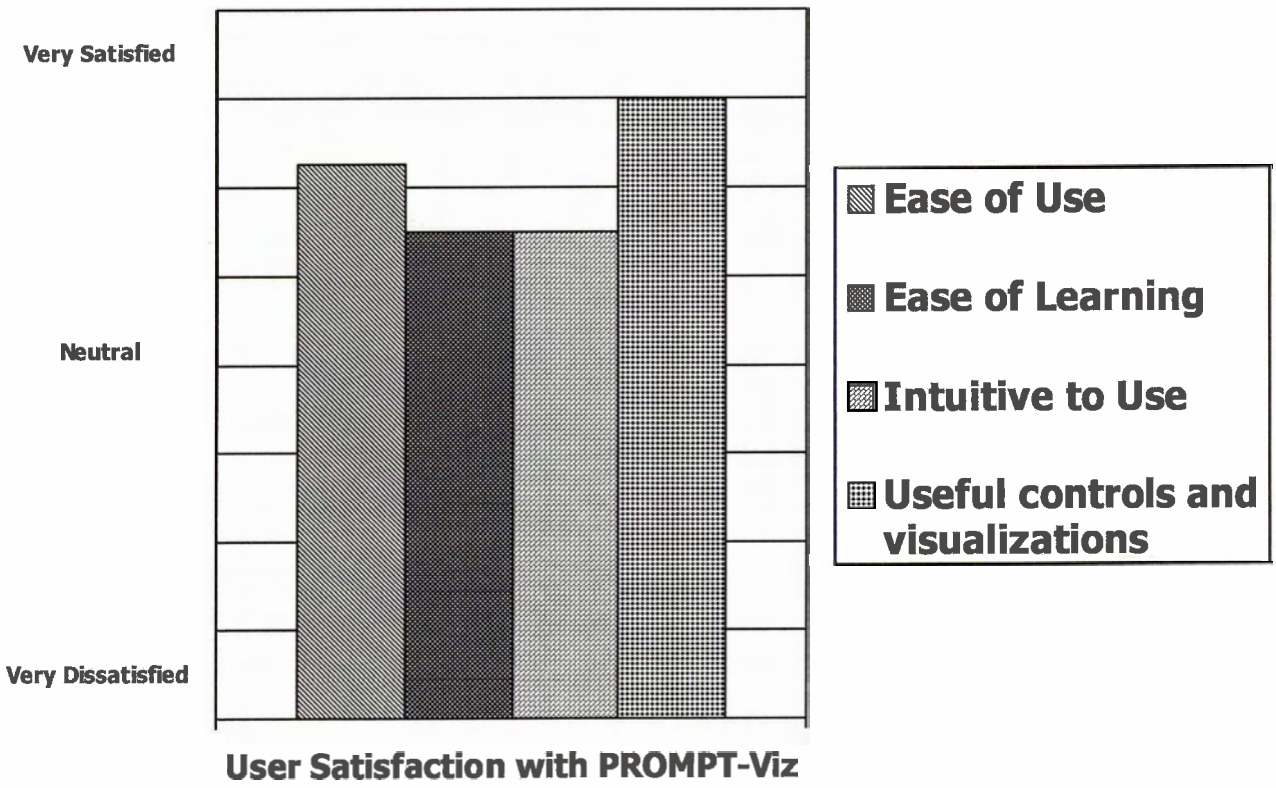


Figure 6.1: Average user satisfaction with PROMPT-Viz

6.3.2 Useful, Difficult, Missing and Extraneous Features

In the post study questionnaire the users were asked what features of PROMPT-Viz they found useful, difficult, or missing. The users were encouraged to look at PROMPT-Viz again to aid in them in formulating their responses, but no list of features was supplied. Their responses are provided in point form with the number of users providing the response in brackets at the end of each description.

Useful Features:

- Histogram colouring mode (5 users)

- Sizing treemap nodes based on various combinations of change attributes (5 users)
- Zooming and Navigation controls (2 users)
- Overview treemap sized by number of descendents so relative size of branches could be determined (1 user)
- Ability to visualize the position of a concept in the ontology (1 user)

Difficult Features:

- Navigation in treemap (4 users)
- Arcs between nodes in the treemap(2 users)
- Children difficult to distinguish from siblings (1 user)
- Parent versus children coloring (1 user)
- Coupling between tree and treemap navigation was not tight enough (1 user)
- Entries in the trail canvas were too small (1 user)

Missing Features:

- Labels for number of classes represented by the size of a node in the treemap and labels for number of classes represented by each bar in the histograms (5 users)
- Tooltips on individual bars in the histograms (2 users)
- Filter unwanted class types or individual classes from the treemap (1 user)
- Complexity metrics & change metrics (1 user)
- Graphs and reports & Annotations (what has changed and why it has changed) (1 user)
- Stop button when resizing treemap (1 user)

Superfluous Features:

- There were no features of PROMPT-Viz any users felt should be removed.

6.3.2 Task completion: PROMPT-Viz vs. PROMPT

The last question on the post-study questionnaire asked the users to rate whether each of the six tasks were easier or harder to solve when using PROMPT-Viz as compared to PROMPT. The average responses are shown in Fig. 6.2. Overall, the users found that the first four tasks that concerned type, location and extent were easier to answer with the aid of the visualizations incorporated into PROMPT-Viz. In particular, the two tasks that concerned the location of changes were ranked by all the participants as being much easier to solve with the aid of the visualizations. When asked if PROMPT-Viz made determining if any new analgesic agents were added to the ontology, one user responded that it was slightly more difficult, three users responded that there was no difference compared to PROMPT, three users said the task was slightly easier with PROMPT-Viz and one user found the task much easier with PROMPT-Viz. The final task was to describe the reclassification of the "Cell Adhesion Molecule" concept between the two versions. One user found the task much easier with PROMPT-Viz, one user found it slightly easier, two user found no difference, three users found it slightly more difficult and one user found the task much more difficult.

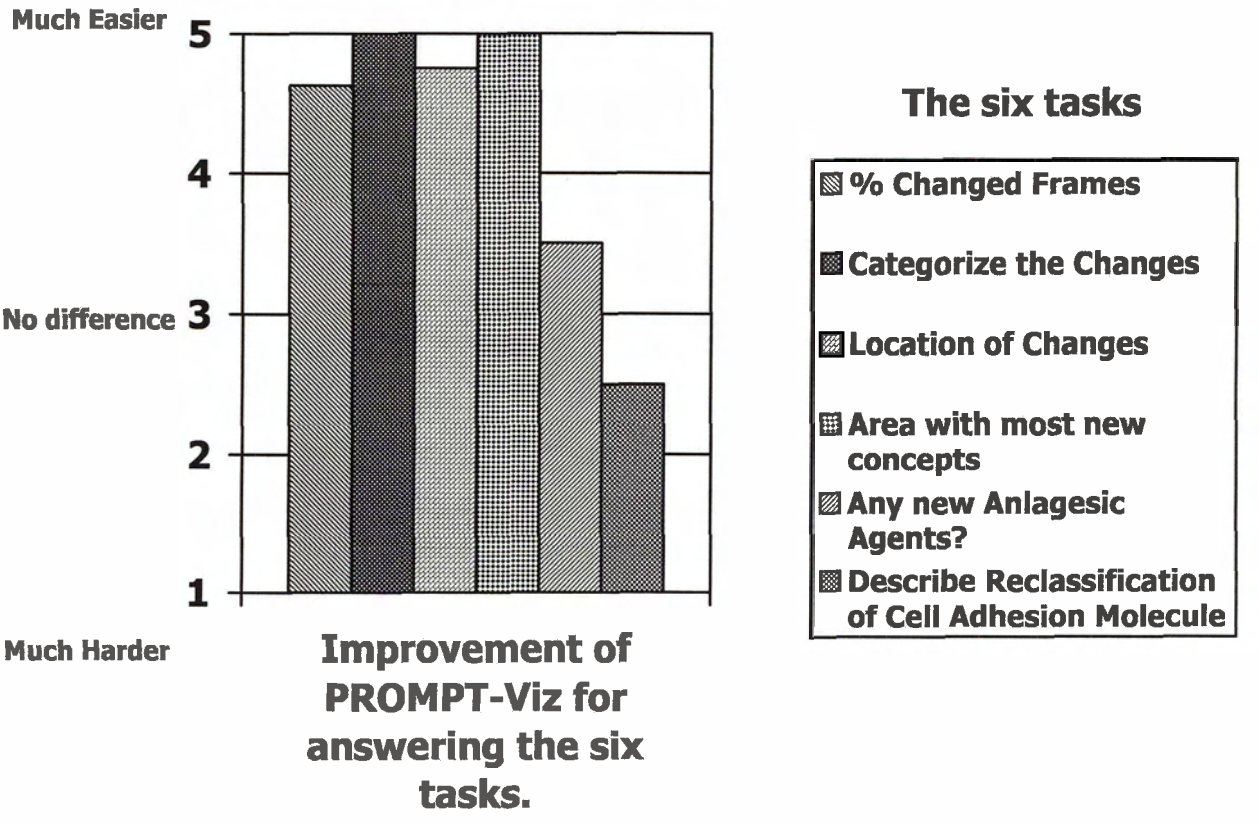


Figure 6.2: User ratings on whether PROMPT-VIZ made solving the six tasks easier or harder

6.4 Discussion

Overall, the results of this study support our hypothesis that the visualizations of PROMPT-Viz improve the ability of users to answer questions about the location, type and extent of changes that have occurred between versions of an ontology. There was unanimous agreement of all the participants that answering the four questions concerning location, type and extent of changes was easier with PROMPT-Viz. Since the participants were all non-expert users, it was not practical to ask questions that accessed the ability of PROMPT-Viz to help users understand the impact of changes; however, based on our observations during the user study, we speculate that PROMPT-Viz may

fall short in this regard. The study also exposed problems with the arcs used in PROMPT-Viz to depict changes to the location of concepts within the is-a hierarchy and the users provided valuable feedback on the overall usability of PROMPT-Viz's visualizations and interface. Finally, discussions with the participants combined with the questionnaire results and observations have inspired new ideas for improvements to PROMPT-Viz. A more detail discussion of each of the questions from the post-study questionnaire proceeds below.

6.4.1 User Satisfaction

The only two user satisfaction categories where the participants gave dissatisfied ratings were for the ease of learning and intuitiveness to use categories. This was unsurprising given that previous studies have also concluded that many people do not initially find the way a treemap displays a hierarchy to be intuitive. Reading treemaps is very much a learned skill. The fact that all of the users felt that the visualizations provided in PROMPT-Viz were useful was especially encouraging considering this common difficulty of novice treemap users. A learning effect can be observed in the user satisfaction ratings of PROMPT-Viz between using PROMPTDiff first and then PROMPT-Viz. The group that used PROMPTDiff first gave a higher overall satisfaction score for PROMPT-Viz; this can be attributed to their greater understanding of the tasks, data set and domain. Because of this learning effect, they could instead concentrate on using the visualizations of PROMPT-Viz to solve the tasks. Given the cognitive challenges placed on the users during the study, this is to be expected, but it is still an

important result because typical expert users are very familiar with the tasks, data set and domain.

6.4.2 Useful, Difficult and Missing Features

Most of the responses given for these questions were to be expected, so the discussion of them will be brief. For example, most of the users listed the histogram colouring mode and ability to resize the treemap to be useful features. Both of these features use the common visual mapping of size to quantitative data and it would have been surprising if the participants had encountered any difficulty with these two features. Similarly, several users expressed difficulty in navigation within the treemap and this was not that surprising given that most of the users were unfamiliar with ZUIs. Using ZUIs effectively is also a learned skill and once the effort has been made to learn the technique, it can be very powerful. The single user who was most familiar with the technique found it to be one of the more useful features.

The most important difficultly that users had was that of interpreting the use of arcs in PROMPT-Viz. Only two users listed this as a difficulty, but it was observed that nearly every one of them found the arcs to be confusing. Part of the confusion with the arcs may have been because no specific questions required the users to make use of all the different types of arcs; however, it is also probable that mixing node and arc and nested node metaphors can result in confusion. Perhaps a better solution to help users assess the impact of a change could be another colouring mode like the histogram mode, but showing relationships rather than descendents. If the user wants to see arcs for a

particular relationship, it could be done on a slot by slot basis. This may be better than the current solution of just showing arcs for every slot attached to the class. Finally, on the topic of arcs, PROMPT-Viz also depicts multiple superclasses by drawing dashed arcs to all the locations within the hierarchy where the selected class exists, including locations that were only valid in the older version of the ontology. The current treemap view contains all the concepts from both the new and old versions of the ontology, but after witnessing the confusion this can lead to, drawing the old and new versions separately and using linking to show the location of synonymous concepts like Graham did in his taxonomy comparison visualizations may be a better alternative solution.

Finally, many users noted that each bar in the histograms should have had individual labels and tooltips. These features had already been identified as being necessary, but there was insufficient time to implement them prior to the user study.

6.4.3 Task completion: PROMPT-Viz vs. PROMPT

It was reassuring to have all the users confirm that PROMPT-Viz made answering the location, type and extent questions easier than could be done with PROMPTDiff. Since the inability of PROMPTDiff to provide a good overview of the changes it finds between two versions of an ontology was one of the primary deficiencies PROMPT-Viz attempts to solve.

As expert users of the system, we would have expected that the users would have also found that task 5 was easier to solve with PROMPT-Viz as it concerned location and type

parameters. Task 5 asked the users to determine if there were any new analgesic agents added to the ontology, and if so, what were they. PROMPT-Viz provides several mechanisms to help discover specific types of changes; however, it appeared that because the branch of the ontology representing Analgesic agents was quite small, many of the users were content in answering the question using the more familiar treeview of PROMPTDiff (recall that the treeview in PROMPTDiff is a traditional hierarchical layout like that of the folders frame used by Microsoft for file system browsing in their Windows operating systems). Perhaps if the branch of the ontology that they were required to search for new concepts had been larger, then more of participants would have been inclined explore the visualizations that PROMPT-Viz offers in order to solve the task.

The sixth task was easily solved using the list of detailed changes about the class that were presented by both tools or by recognizing a class equivalency arc in PROMPT-Viz; however, many of the users found the arcs in PROMPT-Viz confusing and said this task was harder to solve using PROMPT-Viz. For the class in question there were many other types of arcs that needed to be filtered from the screen before the class equivalence arc was apparent. Only one of the users had learned the interface and visual metaphors well enough to find this task easier with PROMPT-Viz.

6.4.4 Discussion and other observations

In the experiment, we decided to use the same ontology and tasks for both tools. We were aware that this would introduce a learning effect, but in the knowledge engineering

community, users are typically very familiar with the ontology they use and the questions that they need to answer. By allowing the users to become more familiar with these we conjectured they might provide results closer to those of expert ontology users. By having half the users start with PROMPT and half start with PROMPT-VIZ we could observe both the different approaches taken to use the two tools and the users interpretation our visualizations with different levels of domain knowledge. Overall, we found that the increased knowledge did lead to the users being more satisfied with the ability of PROMPT-Viz to answer location, type and extent questions and less satisfied with its ability to answer the other questions. The lower ratings to the non-LITE questions may support our conjecture as real domain experts could likely answer these questions quickly and efficiently with the existing features of PROMPT.

Although the number of participants in the study is too small to make a statistical argument that PROMPT-Viz is better than PROMPT for answering questions about the general differences between two versions of a large ontology, the unanimous agreement of the users suggest this to be true. Studies by Jakob Nielsen and others have shown that the majority of usability problems with software systems can be exposed by as few as five users. This makes it likely that the 8 users discovered most of the usability problems with PROMPT-Viz, the most significant of which were the lack of tooltips and labels in the histogram mode and the general confusion surrounding the interpretation of arcs. We have subsequently added labels to the histogram mode and are working on the other features.

This experiment was an exploratory study and in the future a more formal study should be conducted to compare PROMPT-Viz with PROMPTDiff.

6.5 Chapter Summary

Overall, the feedback that we received so far about PROMPT-Viz is very encouraging. Expert knowledge engineers believe that the visualizations provided by PROMPT-Viz will help them with their work. This includes both modelers who need verification of changes they have made to an ontology and the end users of large ontologies who need to know where changes have occurred between versions. The group of 8 non-experts who performed a set of change understanding tasks using both PROMPT and PROMPT-Viz found that PROMPT-Viz greatly enhanced their ability to answer questions about overall change characteristics without hindering their ability to discover specific changes. This group also provided valuable suggestions about usability concerns and the need to redesign some of the features such as arcs. An evaluation of PROMPT-Viz by expert users is ongoing at the NCI.

Chapter 7: Contributions and Conclusions

PROMPT-Viz is the first attempt, that we are aware of, to use advanced visualization techniques for presenting changes between ontology versions. We have applied the treemap layout algorithm to visualizing the differences between ontology versions. This technique has been used in the domain of software engineering for visualizing change data captured by a version control tool. As ontologies become more prevalent, we expect the need for tools like PROMPT-Viz to become increasingly important. Since PROMPT-Viz is a plug-in for the popular ontology editing and knowledge acquisition environment Protégé and an extension of the existing and well known plug-in PROMPT, it already has a large base of potential users and is well positioned to help them deal with the cognitive challenges of understanding changes between ontology versions. The enterprise vocabulary group at the NCI has been supportive of our work and hopefully PROMPT-Viz will bring them one step closer to adopting Protégé as their ontology editing environment.

The participants in our user study found that our technique of embedding histograms that represented how all the descendents of concept had changed on the nodes in the treemap layout was particularly effective for determining overall changes between ontology versions. They also found the ability to resize the treemap layout to accentuate particular types of changes was very useful.

7.1 Future work

Our users did experience difficulties understanding the use of arcs to display the movement of a frame from one location in the ontology to another. Given how easily the users understood and accepted the histogram painting scheme, we think that a similar painting scheme could be used to show the actual changes that have occurred to a frame. This would also alleviate the problem of multiple arcs being drawn for a single slot. A block could be drawn for each slot and selection by the users would result in arcs being drawn. This would be a third painting mode for the program and it would allow users to quickly inspect the specific nature of the changes to a frame.

A side-by-side comparison of the two versions of the ontology similar to Treejuxtaposer [75] may be a better solution than drawing a merged version as we have done. The fact that both old and new locations of concepts appear in the treemap was confusing to the users in our study suggesting that showing parallel views may be a good avenue for future work. Certainly we need to compare the advantages and disadvantages of both techniques.

All of the users in our study requested tooltips on individual bars of the histogram. They expected this feature as tooltips are used so frequently in other aspects of the user interface. All users were also interested in getting specific statistics about the number and type of changes that had occurred in a particular branch of the tree. A future enhancement should provide numerical information on the histogram and also allow a more detailed labeling setting to provide this information.

Large numbers of incoming and outgoing arcs can be difficult to comprehend, even if you know what they represent. One future enhancement could be to reduce the cognitive load by using a pop-up window that will list all the incoming and outgoing slots in for the class and colour code them based on the changes that have occurred. The user would then be able to have any slot arcs drawn at their discretion by selecting the slots of interest from the list.

Although PROMPT-Viz does run sufficiently fast for the ontologies we have been comparing, the current implementation does not look like it will scale beyond ontologies of about 50,000 concepts. Similar techniques could perhaps scale to larger ontologies if we were to port our visualization to OpenGL and make use of high-end consumer graphics cards.

Finally, our work on visualizing changes between ontology versions has provided some inspiration for visualizing the differences between declared and inferred classifications in OWL ontologies.

7.2 Contributions

The contributions from this work include:

1. The LITE set of questions that are important for an ontology difference visualization to try and help answer.

2. A review of the literature of visualization techniques and how it could be applied to ontology versioning.
3. The implementation of one of the first attempts at applying advanced visualization techniques to the problem of ontology versioning.
4. A user study to evaluate the ability of our implementation to help users answer the LITE set of questions about the differences between ontology versions.
5. Suggestions for future approaches to visualizing the differences between ontology versions.

We believe that work on visualizing the differences between ontology versions will continue to be an important area of research. The prevalence and scale of ontologies will surely increase dramatically as the Semantic Web is realized and the developers will need to understand how their ontologies are changing. It is our hope that PROMPT-Viz will useful to the knowledge engineering community and that future researchers will be inspired by our approach.

References:

[1] W3C Semantic Web: http://www.w3.org/2001/sw/

[2] T. Berners-Lee, J. Heldler, and O. Lassila, "The Semantic Web," in *Scientific American*, vol. 284, 2001, pp. 34-43.

[3] W3C Web Ontology (WebOnt) Working Group (OWL): http://www.w3.org/2001/sw/WebOnt/

[4] The Protégé Project: http://protege.stanford.edu

[5] N. F. Noy, and M. A. Musen, "The PROMPT Suite: Interactive Tools For Ontology Merging and Mapping," *International Journal of Human-Computer Studies*, vol. 59, pp. 983-1024, 2003.

[6] N. F. Noy, and M. A. Musen, "PromptDiff: A Fixed-Point Algorithm for Comparing Ontology Versions," presented at *Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, Edmonton, AB, 2002.

[7] X. Wu, "Visualization of Version Control Information," Computer Science, University of Victoria, Victoria 2003.

[8] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 3rd ed: Addison Wesley Longman, Inc., 1998.

[9] J. Golbeck, G. Fargoso, F. Hartel, J. Hendler, B. Parsia, and J. Oberthaler, "The National Cancer Institute's Thesaurus and Ontology," *Journal of Web Semantics*, vol. 1, 2003.

[10] B. Shneiderman, "Tree visualization with Tree-maps: A 2-d space-filling approach," *ACM Transactions on Graphics (TOG)*, vol. 11, pp. 92-99, 1991.

[11] N. F. Noy, and D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," Stanford University, Stanford, CA 2001.

[12] T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing.," *International Journal of Human-Computer Studies*, vol. 43, pp. 907-928, 1995.

[13] A. Farquhar, R. Fikes, W. Pratt, and J. Rice, "Collaborative Ontology Construction for Information Integration," Stanford University, Stanford, CA 1995.

[14] N. F. Noy, and M. A. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment," presented at *Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX, 2000.

[15] B. N. Grosof, I. Horrocks, R.l Volz, and S. Decker, "Description logic programs: combining logic programs with description logic," presented at *Twelfth International Conference on World Wide Web*, Budapest, Hungary, 2003.

[16] D. L. McGuinness, and J. R. Wright, "An industrial strength description logic-based configuration platform," *IEEE Intelligent Systems*, vol. 13, pp. 69-77, 1998.

[17] Y. Lu, "Roadmap for Tool Support for Collaborative Ontology Engineering," Computer Science, University of Victoria, Victoria 2003.

[18] R. Conradi, and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Computing Surveys*, vol. 30, pp. 232-282, 1998.

[19] M. C. Chu-Carroll, J. Wright, and D. Shields, "Supporting Aggregation in Fine Grained Software Configuration Management," presented at *Tenth ACM*

SIGSOFT symposium on Foundation of software engineering, Charlestion, South Carolina, USA, 2002.

[20] J. Froehlich, and P. Dourish, "Unifying Artifacts and Activities ina Visual Tool for Distributed Software Devlopment Teams," presented at *International Conference on Software Engineering ICSE 2004*, Edinburgh, UK, 2004.

[21] S. G. Eick, J. L. Steffen, and E. E. Sumner, "SeeSoft - A Tool For Visualizing Line Oriented Statistics Software," *IEEE Transactions on Software Engineering*, vol. 18, pp. 957-968, 1992.

[22] M. J. Baker, and S. G. Eick, "Space-filling Software Visualization," *Journal of Visual Languages and Computing*, vol. 6, pp. 119-133, 1995.

[23] Q. Tu, and M. W. Godfrey, "An integrated approach for studying software architectural evolution," presented at *2002 Intl. Workshop on Programming Comprehension(IWPC'02)*, Paris, France, 2002.

[24] A. Sarma, and A. van der Hoek, "Visualizing Parallel Workspace Activities," presented at *IASTED International Conference on Software Engineering and Applications (SEA 2003)*, Marina Del Rey, California, 2003.

[25] H. Stuckenschmidt, and M. Klein, "Integrity and change in modular ontologies," presented at *18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.

[26] N. F. Noy, and M. A. Musen, "Ontology Versioning as an Element of an Ontology-Evolution Framework," *IEEE Intelligent Systems*, 2003.

[27] M. Klein, and N. F. Noy, "A component-based framework for ontology evolution," presented at *Workshop on Ontologies and Distributed System, IJCAI '03*, Acapulco, Mexico, 2003.

[28] M. Klein, A. Kiryakov, D. Ognyonov, and D. Fensel, "Ontology Versioning and Change Detection on the Web," presented at *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, Sigüenza, Spain, 2002.

[29] Resource Description Framework (RDF): http://www.w3.org/RDF/

[30] Apelon: http://www.apelon.com/

[31] NCI Center for Bioinformatics Enterprise Vocabulary Services: http://ncicb.nci.nih.gov/core/EVS

[32] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy, and S. W. Tu, "The Evolution of Protégé: An Environment for Knowledge-Based Systems Development." Palo Alto: Stanford University, 2002.

[33] W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen, "Knowledge Modeling at the Millennium (The Design and Evolution of Protege-2000)," 1999.

[34] N. F. Noy, R. W. Fergerson, and M. A. Musen, "The knowledge model of Protege-2000: Combining interoperability and flexibility," presented at *2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000)*, Juan-les-Pins, France, 2000.

[35] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, and J. P. Rice, "OKBC: A Programmatic Foundation for Knowledge Base Interoperability," presented at *Fifteenth National Conference on Artifical Intelligence (AAAI-98)*, Madison Wisconsin, 1998.

[36] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen, "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications," *Submitted for publication*, 2004.

[37] M. M. Allen, "Empirical Evaluation of a Visualization Tool for Knowledge Engineering," Computer Science, University of Victoria, Victoria 2003.

[38] M.-A. D. Storey, M. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. Noy, "Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protege," presented at *K-CAP-2001*, Victoria, B.C. Canada, 2001.

[39] OntoViz: http://protege.stanford.edu/plugins/ontoviz/ontoviz.html

[40] N. F. Noy, and M. A. Musen, "Anchor-PROMPT: Using Non-Local Context for Semantic Matching," presented at *International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, WA, 2000.

[41] J. Bertin, *Semiology of Graphics: Diagrams, Networks, Maps*. Madison, WI.: University of Wisconsin Press, 1983.

[42] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. San Francisco: Morgan Kaufmann Publishers, Inc., 1999.

[43] J. D. Mackinlay, "Automating the Design of Graphical Presentations of Relational Information," *ACM Transactions on Graphics (TOG)*, vol. 5, pp. 110-141, 1986.

[44] I. Herman, G. Melancon, and M. S. Marshall, "Graph Visualization and Navigation in Information Visualization: a Survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, pp. 24-43, 2000.

[45] Q. V. Nguyen, and M. L. Huang, "A Space-Optimized Tree Visualization," presented at *IEEE symposium on Information Visualization 2002 (InfoVis'02)*, 2002.

[46] J. Lamping, R. Rao, and P. Pirolli, "A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies," presented at *CHI 95: Human Factors in Computing Systems*, Denver, CO, 1995.

[47] T. Barlow, and P. Neville, "A Comparison of 2-D Visualizations of Hierarchies," presented at *IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, San Diego, CA, 2001.

[48] B. B. Bederson, B. Shneiderman, and M. Wattenberg, "Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies," *ACM Transactions on Graphics (TOG)*, vol. 21, pp. 833-854, 2002.

[49] M. Bruls, K. Huizing, and J. J. van Wijk, "Squarified Treemaps," presented at *Joint Eurographics - IEEE TCVG Symposium on Visualization*, Vienna, 2000.

[50] J. J. van Wijk, and H. van de Wetering, "Cushion Treemaps: Visualization of Hierarchical Information," presented at *IEEE Symposium on Information Visualization (InfoVis'99)*, San Fancisco, CA, 1999.

[51] B. B. Bederson, "PhotoMesa: a zoomable image browser using quantum treemaps and bubblemaps," presented at *ACM Symposium on User Interface Software and Technology*, Orlando, FL, 2001.

[52] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: algorithms for the visualization of graphs*. Upper Saddle River, New Jersey: Prentice-Hall, 1999.

[53] M. C. Hao, U. Dayal, D. Cotting, T. Holenstein, and M. Gross, "Accelerated Force Computation for Physics-Based Information Visualization," presented at *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)*, Grenoble - France, 2003.

[54] A. J. Quigley, "Large Scale Relational Information Visualization, Clustering, and Abstraction," The Department of Computer Science and Software Engineering, The University of Newcastle, Newcastle, PhD 2001.

[55] G. W. Furnas, and B. B. Bederson, "Space-Scale Diagrams: Understanding Multiscale Interfaces," presented at *Conference on Human Factors in Computing Systems (CHI '95)*, Denver Colorado, 1995.

[56] S. G. Eick, and G. J. Wills, "Navigating Large Networks with Hierarchies," presented at *IEEE Visualization 1993*, 1993.

[57] B. B. Bederson, and J. Meyer, "Implementing a Zooming User Interface: Experience Building Pad++," *Software: Practice and Experience*, vol. 28, pp. 1101-1135, 1998.

[58] K. Perlin, and D. Fox, "Pad: An Alternative Approach to the Computer Interface," presented at *SIGGRAPH '93, the 20th International Conference on Computer Graphics and Interactive Techniques*, 1993.

[59] B. B. Bederson, and J. D. Hollan, "Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics," presented at *UIST'94, ACM Symposium on User Interface Software and Technology*, Marina del Rey, CA, 1994.

[60] B. B. Bederson, J. Meyer, and L. Good, "Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java," presented at *UIST 2000, ACM Symposium on User Interface Software and Technology, CHI Letters 2(2)*, 2000.

[61] Toolkit Design for Interactive Structured Graphics: http://www.cs.umd.edu/local-cgi-bin/hcil/rr.pl?number=2003-01

[62] K. Hornbæk, B. B. Bederson, and C. Plaisant, "Navigation Patterns and Usability of Zoomable User Interfaces With and Without an Overview," *ACM Transactions on Computer-Human Interaction*, vol. 9, pp. 362-389, 2002.

[63] J. Wu, and M-A. D. Storey, "A multi-perspective software visualization environment," presented at *CASCON'2000*, Mississauga, Ontario, Canada, 2000.

[64] S. Pook, E. Lecolinet, G. Vaysseix, and E. Barillot, "Context and interaction in zoomable user interfaces," presented at *AVI 2000, the working conference on Advanced visual interfaces*, Palermo, Italy, 2000.

[65] R. A. Becker, and W. S. Cleveland, "Brushing Scatterplots," *Technometrics*, vol. 29, pp. 127-142, 1987.

[66] M. J. Graham, "Visualising Multiple Overlapping Classification Hierarchies," Napier University 2001.

[67] L. Tweedie, R. Spence, H. Dawkes, and H. Su, "Externalising Abstract Mathematical Models," presented at *CHI 96, ACM Conference on Human Factors in Computing Systems*, Vancouver, BC. Canada, 1996.

[68] C. North, B. Shneiderman, and C. Plaisant, "User Controlled Overviews of an Image Library: A Case Study of the Visible Human," presented at *ACM Digital Libraries '96 Conf*, Bethesda, MD, 1996.

[69] J. Seo, and B. Shneiderman, "Interactive Exploration of Multidimensional Microarray Data: Scatterplot Ordering, Gene Ontology Browser, and Profile Search," University of Maryland, College Park, MD 2003.

[70] C. North, and B. Shneiderman, "A Taxonomy of Multiple Window Coordinations," University of Maryland, College Park, MD 1997.

[71] C. Plaisant, D. Carr, and B. Shneiderman, "Image Browsers: Taxonomy, Guidelines, and Informal Specifications," *IEEE Software*, vol. 12, pp. 21-32, 1995.

[72] C. North, and B. Shneiderman, "Snap-Together Visualization: A User Interface for Coordinating Visualizations via Relational Schemata," presented at *Advanced Visual Interfaces 2000*, Palermo, Italy, 2000.

[73] K. Hornbæk, and E. Frøkjær, "Reading of electronic documents: the usability of linear, fisheye, and overview+detail interfaces," presented at *Human Factors and Computing Systems*, Seattle, Washington, United States, 2001.

[74] B. Shneiderman, "Dynamic Queries for Visual Information Seeking," *IEEE Software*, vol. 11, pp. 70-77, 1994.

[75] T. Munzner, F. Guimbretiere, S. Tasiran, L. Zhang, and Y. Zhou, "TreeJuxtaposer: Scalable Tree Comparison using Focus+Context with Guaranteed Visibility," presented at *SIGGRAPH 2003*, San Diego, 2003.

[76] Protégé Programming Development Kit: http://protege.stanford.edu/doc/pdk/index.html

[77] Treemaps for space-constrained visualization of hierarchies: http://www.cs.umd.edu/hcil/treemap-history/index.shtml

Appendix A: User Consent Script

CONSENT SCRIPT FOR USER OBSERVATIONS

Please remember that participation in this process is voluntary and that you have the right to withdraw at anytime without consequences. If you choose to withdraw part way through the study, data gathered your participation will not be used in the analysis. We would also like to clarify that the investigators do not hold any position of authority or power over you.

Please take the time to read over the following consent form before signing it.

I will now explain how the experiment will proceed. If you have any questions during this explanation please let me know and I will answer them as we go. At the beginning of this experiment we will ask you to fill out a pre-observation questionnaire to gauge your familiarity with the tools, concepts, and domain knowledge involved. We will then provide some training for the Protégé-2000 tool and the PROMPT tab, and will give you as much time as you would like to familiarize yourself with these tools. We will then load the experimental file. During this experiment, we will be filming your actions, filming the screen and recording your voice. We will ask you to complete various tasks using the tools, and ask you to speak aloud your thoughts as you are trying to complete these tasks. You can ask questions to the investigators, and if you feel you do not understand the task or how to complete it you do not have to continue with that task. Please remember that if you have trouble completing a task, it does not demonstrate a personal weakness, but more likely a weakness of the tool. We will then repeat the training, familiarization and tasks with a second variation of the PROMPT-tab. Finally, we will ask you to fill out a post-study questionnaire where you can provide feedback about the tools.

Appendix B: Pre-Study Questionnaire

User Number

Pre-study Questionnaire

Please answer the following questions to the best of your ability. If you are unable or unwilling to answer a question please feel free not to answer.

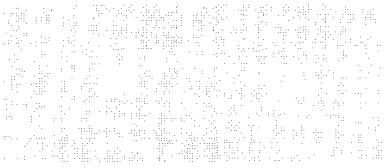
1. How would you categorize yourself as a Protégé-2000 user?

☐ I've never used Protege ☐ beginner ☐ advanced ☐ expert

2. Are you familiar with the treemap layout technique? (yes/no)

3. Have you ever used a zooming user interface before? (yes/no) If so, what was the tool?

Appendix C: User Study tasks

Tool:  *User Number*

Questions to answer while using the tool

Please use PROMPT or PROMPT-Viz to answer the following questions to the best of your ability. If you are unable to answer a question please continue to the next question.

1. PROMPT classifies changes in many ways, at the broadest level they are:
   Changes (direct): Gray or bold text. Examples include name change, added or deleted slot.
   Moves: Blue or Yellow. Examples are added or deleted superclasses.
   Additions: Green.
   Deletions: Red.

Considering the above categories, how much has the ontology changed between the two versions? I'm looking for the percentage of classes that have been modified in some way.

2. With respect to the categories from question 1, how would you categorize the changes that have occurred between the two versions?

3. How would you describe the locations of the changes considering only the categories from question 1. ie. Please ignore isomorphic changes.

4. What area of the ontology has had the most new concepts added to it?

5. Have any new "Analgesic Agents" been added to the ontology? If so, what are they?

6. The Class "Cell Adhesion Molecule" has been reclassified. What was the change in classification?

Appendix D: Post Study Questionnaire

User Number

Post study Questionnaire.

1. Please rate your satisfaction for the following aspects of PROMPT-Viz (The treemap view of PROMPT).

	Very Dissatisfied	Dissatisfied	Neutral	Satisfied	Very Satisfied
Overall ease of use	☐	☐	☐	☐	☐
Ease of learning	☐	☐	☐	☐	☐
Intuitive to use	☐	☐	☐	☐	☐
Informative (The visualizations and controls were useful)	☐	☐	☐	☐	☐

2. What features of PROMPT-Viz did you find most useful?

3. What features of PROMPT-Viz did you find confusing? Do you have any suggestions on how they could be improved?

4. Were there any features that you thought were missing?

5. Are there any features you think should be removed? Why?

6. Recall the six questions you were asked to answer while using PROMPT and again with PROMPT-Viz. For each question rate whether the Visualizations of PROMPT-Viz helped or hindered you in discovering the answer in comparison to PROMPT.

Question	Much more difficult	Slightly more difficult	No difference	Slightly easier	Much easier
1	☐	☐	☐	☐	☐
2	☐	☐	☐	☐	☐
3	☐	☐	☐	☐	☐
4	☐	☐	☐	☐	☐
5	☐	☐	☐	☐	☐
6	☐	☐	☐	☐	☐