ABSTRACT

SOCIAL NETWORK ANALYSIS APPLIED TO
ONTOLOGY 3D VISUALIZATION

by En Yu

OntoSELF was recently developed to provide 3D visualization of the underlying hierarchical structure of intensional ontologies. The extensions to OntoSELF focus on enhancing perception and facilitating high-level comprehension as well as low-level detail exploiting. The extensions include various visualization cues to enhance visualization perception, processing user-defined relationships in addition to the standard hierarchical IS-A relationship, user filtering on which relationships to include in the visualization, and social network analysis (SNA) metrics for additional filtering and structuring criteria, and for finding and better understanding important concepts of interest.

To use standard social network analysis (SNA) techniques, a preprocessing algorithm is used to project an m-mode n-plex ontology structure to a 1-mode 1-plex sociomatrix. A high-level abstraction algorithm based on the notion of “communities of interest” is provided to simplify the social network view of the ontology to a higher level of abstraction.
SOCIAL NETWORK ANALYSIS APPLIED TO
ONTOLOGY 3D VISUALIZATION

A Thesis

Submitted to the
Faculty of Miami University
in partial fulfillment of
the requirements for the degree of
Master of Computer Science
Department of Computer Science & Systems Analysis
by
En Yu
Miami University
Oxford, Ohio
2008

Advisor ________________________
Dr. Valerie Cross

Reader ________________________
Dr. James Kiper

Reader ________________________
Dr. Mufit Ozden
# TABLE OF CONTENTS

1. Introduction ........................................................................................................... 1

2. Background Concepts ............................................................................................ 3
   2.1 Ontologies ......................................................................................................... 3
   2.2 Visualization of Ontologies .............................................................................. 5

3. Social Network Analysis ......................................................................................... 7
   3.1 Social Network .................................................................................................. 8
   3.2 Social Network Analysis .................................................................................. 9
      3.2.1 Centrality .................................................................................................. 9
      3.2.2 Prestige .................................................................................................... 13

4. Related Research .................................................................................................... 14
   4.1 On the Visualization of Large Size Ontologies .............................................. 15
   4.2 OntoSELF ........................................................................................................ 17
   4.3 Balancing Systematic and Flexible Exploration of Social Networks .......... 18
   4.4 Visual Analysis of Large Heterogeneous Social Networks by Semantic and Structural Abstraction .............................................................................. 20
   4.5 Network Analysis as a Basis for Partitioning Class Hierarchies ............... 21
   4.6 Semantic Analysis of Ontologies .................................................................. 24

5. Designing Extensions to OntoSELF ..................................................................... 26
   5.1 Ontology Preprocessing Approach ............................................................... 26
   5.2 Visualization Packages in OntoSELF ............................................................ 27
   5.3 Standardized vs. Customized OWL Parser .................................................. 29
   5.4 Issues of Centrality and Prestige Calculation .............................................. 31
   5.5 Abstraction Algorithms .................................................................................. 34

6. OntoSELF Enhancements for Visual Cues and SNA ......................................... 36
   6.1 Overall Processing Architecture .................................................................... 36
   6.2 User Interface .................................................................................................. 40
   6.3 Implementation of SNA Processing Features ............................................... 44
      6.3.1 Visualization Software Overview ............................................................ 45
         6.3.1.1 VTK - Visualization Tool Kit .............................................................. 45
         6.3.1.2 JUNG - Java Universal Network/Graph Framework ....................... 46
      6.3.2 Visualization Packages in OntoSELF .................................................... 48
6.3.3 OWL to GML Parser

6.3.4 Social Network Analysis Implementation

6.3.5 Clustering for Abstraction

7 Evaluation

7.1 Evaluation on Visual Cues Enhancement

7.1.1 Node Size

7.1.2 Node Color

7.1.3 Combination of Node Color and Size

7.1.4 Edge thickness

7.1.5 Edge Color

7.2 Evaluation on Filtering Criteria

7.2.1 Comparison of the SNA Metrics in Hierarchical Structure

7.2.2 SNA Metrics for Topology Understanding Tasks

7.3 Evaluation of Abstraction

7.3.1 Clustering for Semantic Abstraction

7.3.2 Clustering for Structural Abstraction

7.4 Evaluation of Task Support on Ontologies

7.4.1 Task 1: Overview

7.4.2 Task 2: Zoom

7.4.3 Task 3: Filter

7.4.4 Task 5: Relate

7.4.5 Other Tasks of Details on Demand, History, and Extract

8 Conclusions and Future Work

References

Appendices

Appendix I Visualization Software Implementation

Appendix II Implementation of OWL to GML Parser
List of Figures

Figure 2-1 The Jambalaya tree view (lower left); nested view (center right) (Storey et al., 2004) 6

Figure 2-2 OntoSphere Visualization (A) Root Focus View (B) Tree Focus View (Bosca et al., 2005) ................................................................. 7

Figure 6-1 Overall Architecture of OntoSELF represented by UML Class Diagram ............ 37

Figure 6-2 Overall processing flow of OntoSELF .................................................................. 39

Figure 6-3 The User Interface of OntoSELF-v1 .................................................................... 41

Figure 6-4 The User Interface for OntoSELF-v2 at the start of execution.......................... 41

Figure 6-5 The User Interface when the File Dialog is opened by clicking the Browse button... 42

Figure 6-6 The User Interface displayed after the user selected a umls-semantic-network.owl file and package 1 for visualization, and clicked the Open File button ................................. 43

Figure 6-7 A Typical VTK pipeline ...................................................................................... 45

Figure 6-8 Class definition of SNAnalysis class ................................................................. 53

Figure 6-9 Betweeness Centrality Algorithm (Brandes 2001) ............................................. 56

Figure 6-10 Example clustering using Newman Algorithm (Newman 2004) .................... 61

Figure 7-1 3D Visualization of OntoSELF-v1 on UMLS SN Ontology with only superclass/subclass relationship and labeling on the top 15 nodes with the highest rank prestige measures. All nodes have equal sizes. .......................................................... 65

Figure 7-2 3D Visualization of OntoSELF-v2 on UMLS SN Ontology with only superclass/subclass relationship and labeling on the top 15 nodes with the highest rank prestige measures which are also used to determine the node size. ..................................................... 66

Figure 7-3 3D Visualization of OntoSELF-v1 on NASA Sweet JPL Ontology with only superclass/subclass relationship and labels on the top 25 nodes with the highest rank prestige measures. All nodes have equal sizes. .......................................................... 67

Figure 7-4 3D Visualization of OntoSELF-v2 on NASA Sweet JPL Ontology with only superclass/subclass relationship and labels on top 25 nodes with the highest rank prestige measures which are also used to determine the node size. ..................................................... 68

Figure 7-5 3D Visualization of OntoSELF-v1 on Gene Ontology with only superclass/subclass relationship and labels on top 50 nodes with the highest rank prestige measures. All nodes have equal size ........................................................................ 69
Figure 7-6 3D Visualization of OntoSELF-v2 on Gene Ontology with only superclass/subclass relationship and labels on top 50 nodes with the highest rank prestige measures which are also used to determine the node size ........................................................................................................... 70

Figure 7-7 The area around Protein_complex after zooming in from Figure 7-6 ......................... 71

Figure 7-8 The local context of Protein_complex after selecting it as the main node in the filtering options based on its identification Figure 7-6 and using rank prestige to determine node size and labeling on the top 25 concepts.......................................................... 72

Figure 7-9 3D Visualization of OntoSELF-v1 on NASA Sweet JPL Ontology with only superclass/subclass relationship with Node Level as its weighting function ........................................ 73

Figure 7-10 3D Visualization of OntoSELF-v2 on NASA Sweet JPL Ontology with only superclass/subclass relationship. Semantic clustering at Level 0 is used to determine node color. Rank prestige is used to determine cluster selection in case of multiple inheritance ..................... 74

Figure 7-11 3D Visualization of the NASA Sweet JPL Ontology with only superclass/subclass relationship. The top 15 nodes with the biggest size based on rank prestige have been labeled but not visual cues added. ........................................................................................................ 75

Figure 7-12 3D Visualization of NASA Sweet JPL Ontology with only superclass/subclass relationship. The Semantic clustering at Level 0 for node colors is used with rank prestige to determine the cluster when multiple inheritance occurs and the size of the node. The top 15 nodes with the biggest sizes are labeled. .................................................................................. 76

Figure 7-13 U2D Visualization of NASA Sweet JPL Ontology with all user-defined relationships with rank prestige used to determine node size and structural clustering on the whole ontology used to determine node color. Top 35 biggest nodes are chosen for labeling. .... 78

Figure 7-14 HU2D Visualization on UMLS SN Ontology with hierarchical relationship in blue and user-defined relationships in red. Rank prestige determines node size and structural clustering on whole ontology determines node color. The top 15 biggest nodes are labeled........ 80

Figure 7-15 HU3D Visualization of UMLS SN Ontology with hierarchical relationship and user-defined relationships colored according to the source node’s level. Rank prestige determines node size. Structural clustering on whole ontology determines node color. Top 15 biggest nodes are labeled................................................................. 81

Figure 7-16 3D Visualization on UMLS SN Ontology with only superclass/subclass relationship. Each of the four centrality and prestige measures are used to determine the node size: Closeness Centrality (top left), Betweenness Centrality (top right), Information Centrality (bottom left), and Rank Prestige (bottom right) ........................................................................................................ 83

Figure 7-17 Find the Bushiest child node of a given node (Somasundaram 2007) ..................... 86

Figure 7-18 The initial visualization for the task of finding the bushiest child node of a given node. The number of children each node has determines the node size...................................................... 87
Figure 7-19  Zooming in on given node from figure 6.18 for the task of finding the bushiest child node of a given node. .............................................................................................................................................. 88

Figure 7-20  Find the Largest Subtree for a given a node (Somasundaram 2007) .................................. 89

Figure 7-21  Find the Largest Subtree for a given a node in H3D ......................................................... 90

Figure 7-22  H3D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Semantic Clustering at Level 1 with Closeness as clustering criteria is used. The top 35 largest nodes are labeled. .......................... 92

Figure 7-23  HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Semantic Clustering at Level 1 with Closeness as clustering criteria is used. All nodes are labeled. ......................................................... 93

Figure 7-24  Zooming in on the visualization in figure 6.21 ................................................................. 94

Figure 7-25  Zooming in on the visualization in figure 6.22. ............................................................... 95

Figure 7-26  H3D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Clustering for semantic abstraction is at Level 2 with Closeness as clustering criteria is used. The top largest 35 nodes are labeled .............................................................................................................................................. 96

Figure 7-27  HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Semantic Clustering at Level 2 with Closeness as clustering criteria is used. All nodes are labeled. ......................................................... 97

Figure 7-28  H3D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Structural Clustering on whole ontology is used. Top 35 largest nodes are labeled. .............................................................................................................................................. 99

Figure 7-29  HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Structural Clustering on whole ontology is used. All nodes are labeled. .............................................................................................................................................. 100

Figure 7-30  HU2D Visualization of UMLS SN Ontology with superclass/subclass relationship as the hierarchical relationship and has _measure as the user-defined relationship. Rank Prestige is used to determine the node size. Clustering for structural abstraction on whole ontology is used. All nodes are labeled. The edges in blue are hierarchical ones. The edges in red are user-defined ones. ................................................................. 103

Figure 7-31  HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Clustering for structural abstraction on whole ontology is used. All nodes are labeled. The filtering criterion Rank Prestige is used.............................................................................................................................................. 106
Figure 7-32  HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Clustering for structural abstraction on filtered ontology (have rank prestige measure larger than or equal to 0.2) is used. All nodes are labeled.

Figure 7-33  HU3D Top-down view of the UMLS SN ontology

Figure 7-34  HU3D 90-degree view of the UMLS SN ontology

Figure 7-35  HU3D 45-degree view of the UMLS SN ontology

Figure 7-36  Another HU3D 45-degree view of the UMLS SN ontology

Figure 7-37  HU2D visualization of UMLS SN with all relationships selected. User defined relationships are in red and hierarchical relationships are in blue.

Figure 7-38  HU3D visualization of UMLS SN with only superclass/subclass relationship as hierarchical relationship. Rank Prestige is used to determine the node size. Clustering for semantic abstraction is used. Top 20 largest nodes are labeled.

Figure 7-39  U2D Visualization of UMLS SN with all user-defined relationships selected. Rank Prestige is used to determine the node size. Clustering for semantic abstraction is used. Top 20 largest nodes are labeled.

Figure 7-40  Zooming in to view the local context of the cluster rooted at Conceptual Entity in H3D visualization. Rank Prestige is used to determine the node size. Clustering for semantic abstraction is used. Top 20 largest nodes are labeled.

Figure 7-41  Zooming in to view the local context of the concept Pathologic Function in HU3D visualization.

Figure 7-42  Zooming in to view the local context of the cluster in dark blue in U2D visualization.

Figure 7-43  Zooming in to view the local context of the upper left portion of the visualization in HU2D.

Figure 7-44  Zooming with subtree filtering based on node name “Animal” in H3D.

Figure 7-45  Zooming subtree filtering based on node name “Animal” in HU3D.

Figure 7-46  Zooming with filtering based on node name “Animal” in HU2D.

Figure 7-47  H3D visualization of UMLS SN. Concepts at level 3 or higher and having rank prestige greater than 0.2 are displayed. Clustering for semantic abstraction at level 1 with rank prestige as clustering criteria. All nodes are labeled.
Figure 7-48  HU3D visualization of UMLS SN. Concepts at level 3 or higher and having rank prestige greater than 0.2 are displayed. Clustering for semantic abstraction at level 1 with rank prestige as clustering criteria is used. All nodes are labeled. 124

Figure 7-49  U2D visualization of UMLS SN ontology. Concepts with rank prestige greater than 0.2 are displayed. Clustering for structural abstraction on the whole ontology is used. All nodes are labeled. 125

Figure 7-50  HU2D visualization of UMLS SN. Concepts at level 3 or higher and having rank prestige greater than 0.2 are displayed. Clustering for semantic abstraction at level 1 with rank prestige as clustering criteria is used. All nodes are labeled. 126

Figure 7-51  HU2D visualization of UMLS SN ontology for relating capability. 127

Figure A-1  VTK pipeline for OntoSELF. 136

Figure A-2  The class definition of SNVisualization class. 138

Figure A-3  The UML class diagram of the external class edu.uci.ics.jung.graph.Graph composed of the two customized classes, Vertex and Edge. 139

Figure A-4  Class definition of OWLtoGMLParser class. 143

Figure A-5  Snapshot of a simple 7-line OWL code from people+pets.owl. 145

Figure A-6  The Parsing Stack after reading the first line &lt;owl:Class rdf:about = "#bus+company"&gt; 145

Figure A-7  The Parsing Stack after reading the fourth line &lt;rdfs:subClassOf&gt; 145

Figure A-8  The Parsing Stack after reading the sixth line &lt;/rdfs:subClassOf&gt; 146
List of Tables

Table 1  Result comparison between structural clustering and cohesive partitioning on UMLS SN ontology. Cluster name in red means the cluster with the same size appears in both algorithms. Cluster name in blue means the cluster appears in both algorithm but with different sizes. Cluster name in black means the cluster only appears in one of the two algorithms. ........................................ 102

Table 2  Comparison between using superclass/subclass only and using both superclass/subclass and has measure relationship. Cluster name in red means the cluster with the same size appears in both algorithms. Cluster name in blue means the cluster appears in both algorithm but with different sizes. ............................................................................................................................. 105
ACKNOWLEDGEMENTS

I would like to thank Prof. Valerie Cross for her constant advice, encouragement and help. Without her enthusiasm, her inspiration and lots of good ideas, I would have been lost. I would also like to thank my committee members Dr. Mufit Ozden and Dr. James D. Kiper for their guidance and support.

I am indebted to my many professors and student colleagues for providing a positive and fun environment where to learn and grow during this and other related projects, specifically Prof. Michael T. Helmick for advising me while working on Miami Redhawks cluster in the summer scholar project, my wife Wenting Yi for her support, help and continuous encouragement.

I am also grateful to the two years I have spent at Miami University with all these excellent and nice professors and graduate classmates. For their kind assistance with giving wise advices, helping with various coding problems, writing letters and certificates, and so on, I wish to thank in addition Dr. Douglas A. Troy, Dr. Michael A. Zmuda, Mrs. Patricia Otto and Mrs. Monica Lea Baxter.

Finally, I would like to thank my parents, whose bore me, raised me, taught me, and love me all the time. To them, I dedicate this thesis.
1 Introduction

An important component of the emerging Semantic Web is the significant number of ontologies covering various knowledge domains. Ontologies as a knowledge modeling tool provide easy re-use and re-engineering of the knowledge. In order to support many comprehension tasks such as ontology selection, annotation and specification, users need methods and tools to understand the structure and content of an ontology. Since comprehension and analysis of an ontology is a subjective topic to humans, different users often have different perspectives and learning styles. Many researchers, therefore, have been focused on using visualization techniques to facilitate the ontology comprehension task. A variety of techniques are being researched to create aesthetic and easy-to-understand visualization of ontologies. A core problem of ontology visualization that most research is trying to address is that the perception of the ontology degrades as the size of the ontology increases (Tzitzikas and Hainaut, 2006). Researchers are investigating ways to complement the standard visualization techniques with other visualization techniques to improve the overall perception of large-sized ontologies.

In (Katifori et al 2007), visualization methods have been reviewed and evaluated to compare their pros and cons for ontology visualization. The methods have been grouped into the following categories which represent their visualization type: indented list, node-link and tree, zoomable, space-filling, Focus + context or distortion, and 3D Information landscapes. A comparison of these various categories was presented based on an analysis of seven major tasks related to ontologies. Their assessment results indicate that the node-link and tree category provides good visualizations for supporting each task type. Of the thirteen surveyed visualization software tools that belonged to this category, only five provide 3D visualization and of these five, only one, OntoSphere (Bosca et al 2005) directly supports 3D visualization of ontologies.

This survey of visualization methods supports the design decisions and research efforts that initially led to the development of OntoSELF (Ontology Scrutiny Exploiting Layouts and Filtering) as part of a former master’s thesis (Somasundaram, 2007). The primary objective of OntoSELF was to provide a 3D ontology visualization system that tries to address degradation of perception that occurs as the ontology size increases.
OntoSELF gives the user various weighting functions on layout selection that can use different node metrics to determine the node’s position in the 3rd dimension of the visualization and node filtering techniques using ontology structural metrics based on the primary ontology relationship, the is-a or superclass/subclass relationship. OntoSELF helps users to better understand the information contained within an ontology by exploiting these various metrics.

As pointed out in (Katifori et al. 2007), except for OntoSphere, the use of 3D visualization for the ontology domain has been limited. OntoSELF has been shown to be effective at 3D visualization of large-sized ontologies and performing various ontology understanding tasks by providing the user with flexible visualization on the hierarchical structure of an ontology. It, however, does not support the visualization of user-defined relationships and provides very limited use of visual cues to enhance a user’s perception of an ontology. In addition, it does not provide any abstraction capability to allow a user to easily identify a cluster of related concepts in an ontology.

The primary objective of this research is to provide enhancements to OntoSELF to address some of its original limitations and to increase its flexibility in assisting users in accomplishing ontology-related tasks. The following enhancements have been made: 1) visual cues such as node size, node color, edge thickness, and edge color to improve the user’s perception of the visualization, 2) a flexible method of incorporating additional relationships defined by the user and still have the capability to include the superclass/subclass relationship, 3) abstraction techniques to be used on complex ontologies to facilitate a user’s high level comprehension, 4) social network analysis metrics that can be used to analyze the structure of ontologies and applied to both hierarchical and user-defined relationships and 5) the use of 2D visualization for flexibility in visualization tasks on ontologies. The enhanced OntoSELF system with the combined power of the above listed enhancements is used on several standard ontologies, such as the Gene Ontology and the UMLS in order to evaluate the effects of these capabilities on the visualization of ontologies. The enhanced OntoSELF is also assessed with respect to the seven ontology-related tasks used in comparing the five categories of visualization methods.
This thesis begins with chapter 2 providing a brief introduction to ontologies and ontology visualization. Chapter 3 introduces social networks and some terminology and discusses various social network analysis methods for measuring importance and prestige of nodes. In chapter 4, several recent research efforts are presented, including some prominent ontology visualization tools and criteria, the use of social network analysis techniques in visualization of heterogeneous networks, and applying SNA techniques for ontology analysis. The design issues addressed in the development of the enhanced OntoSELF are overviewed in chapter 5. A detailed description of the developed software and its implementation are presented in chapter 6. The enhancements made to OntoSELF are analyzed using several well-known ontologies and a discussion of the results is presented in chapter 7. The significance of this research is highlighted and an outline of future work is proposed in chapter 8. The appendices provide implementation details of the VTK and JUNG visualization software development and the customized OWL parser for converting an ontology into a graph manipulation language (GML) file to be used in the ontology visualization process.

2 Background Concepts

The following sections provide information essential as background for understanding this thesis research. The major topics covered include an introduction to ontologies and their visualization.

2.1 Ontologies

An ontology is a specification of conceptualization in a specific domain area (Gruber, 1993). It can be viewed as a domain model which identifies the concepts in that domain and the relationships between those concepts. Because of their capability of specifying domain knowledge, ontologies have become central to the growth of the Semantic Web. To describe domain knowledge precisely and simply, a typical ontology generally contains the following four components:
1) Concepts or classes – Like classes used in object-oriented programming, concepts in an ontology are often abstract groups or sets of real-world concrete objects. For example, Person is a class of all people. Concepts in ontology may have multiple inheritance. A concept can have zero, one, or more children as usual, but it can also have zero, one, or more parents.

2) Properties (Attributes) – Properties are features or characteristics used to describe a class. Typically, a property is called an attribute when the value for the property is not an object but consists of a scalar value. For example, in Computer Graphics, the class line has a property color which can have a value red. When a property takes on a value it is typically referred to as a relationship which is described in the following item. For example, the class line has the property endpoints which can have two points as its value.

3) Relationships – Relationships are created between concepts in ontology to link the concepts. Two standard relationships existing in ontologies are the subsumption (is-a) relation and meronymy (part-of) relation. If class A is-a class B, then class A inherits all properties that class B has. If class A part-of class B, then an instance of class B is made up or consists of an instance of class B. In addition to those two relationships, user-defined relationships may be included in an ontology to provide richer semantics for the domain model.

4) Instances (Individuals) – An instance represents an occurrence of a concept. Instances are not included in the intensional ontology which describes or defines the metadata of the domain model. Instances make up the extensional ontology which describes the actual real world objects in the problem domain. An extensional ontology is populated or instantiated by instances of the concepts and relationships defined in the intensional ontology.

Various approaches have been proposed to classify ontologies. The two major approaches are the amount and type of structure of the conceptualization and the subject of the conceptualization (Van Heijst et al. 1997). The level of complexity in the description of the concepts and the relations between them determines the classification with respect to
the first approach. For example, terminological ontologies are one of the simplest since they only specify the terms used to describe the knowledge about a domain and typically the linguistic relationships such as broader than or narrower than terms. Information ontologies are more complex since they define concepts and their relationships between them along with other properties with the concepts. The subject classification includes application, domain, and generic ontologies.

2.2 Visualization of Ontologies

Ontology visualization with 2D tools is typically the norm. More recent research has begun to tackle issues facing 3D ontology visualization. With 3D visualization the added dimension permits displaying more information but often brings the problems of node overlapping and edge crossings that makes it harder for the user to manipulate the 3D graph in a 2D window by 2D mouse device.

Currently many visualization systems have been developed to support ontology visualization, including both 2D and 3D techniques. However, no universally applicable system to visualize ontologies for a wide range of purposes and tasks exists. All have their own pros and cons. For a good review of various existing ontology visualization systems see (Somassundaram, 2007). Another survey of various techniques used for ontology visualization has also just recently been published (Katifori et al 2007). Two example visualization systems, Jambalaya for 2D visualization and OntoSphere for 3D visualization are briefly summarized in the following.

Jambalaya (http://www.thechiselgroup.org), which is plug-in software for the ontology editor Protégé, is a 2D visualization system for visualizing sophisticated ontologies,. It allows the user to customize their visualization of ontologies. Like the traditional visualization drawing techniques, classes and instances are represented as nodes and have different color hues. Directed edges are used to represent relationships between nodes. In addition to that standard approach to visualization, users can choose a containment visualization approach for relationships, i.e., nodes are contained in rectangular boxes when they are subclasses of other classes. Figure 2-1 illustrates this visualization technique.
The advantage of this treemap layout technique can improve perception dramatically for relationships which are inherently containment, such as part-of. But the disadvantage is also obviously that the containment might degrade the perception if applied to some other relationships which are not related to containment. Additionally, the treemap layout often creates more line crossings.

The motivation for 3D visualization of ontologies is that the structure of, and information contained in an ontology, can be more readily perceived by the user if represented on a 3D view-port enriched by visual cues. OntoSphere ([http://ontosphere3d.sourceforge.net/](http://ontosphere3d.sourceforge.net/)) is a 3D ontology visualization systems that provides three different views to support overview and details visualization according to the different visualization tasks the user needs to undertake. In order to give users a global view of a specific ontology, it uses Root Focus view (Figure 2-2A) which only displays some top level concepts. Once a class is right clicked, a Tree Focus view (Figure 2-2B) of the subtree of that class will be displayed to help users to better understand the inheritance structure of that class. Once a class is left clicked, a Concept Focus view of the tree of its ancestors will be displayed.
Figure 2-2  OntoSphere Visualization (A) Root Focus View (B) Tree Focus View  
(Bosca et al., 2005)

The advantage of this tool is obviously its abstraction and filtering techniques to give users both global and detailed view of an ontology. However, a disadvantage is that the Tree Focus and Concept Focus views are mainly based on the hierarchical class structure and do not incorporate various other relations for analysis purposes. Also the concept focus is just that, i.e., it gives a localized view about the selected concept.

OntoSELF (Ontology Scrutiny Exploiting Layouts and Filtering) developed by Ramathan Somasundaram (2007) has a similar purpose as the previously describe tools in that it wants to facilitate users’ comprehension of the ontology structure at a high level and also provide focus and filtering within the structure. OntoSELF uses 3D visualization techniques to display several 3D layouts with different weighting functions to offer the flexibility of understanding the ontology’s topology according to different tasks or purposes of viewing it. Several filtering techniques have also been adapted to enhance the perception based on several ontological structure metrics. More discussion on OntoSELF is provided as a separate section in the related research section.

3 Social Network Analysis

Recently ontology researchers are discovering that a matured research area social network analysis (SNA) can be applied to analyze ontology structures. For example, the importance and possible applications of some SNA techniques have been proposed to
assist ontology analysis (Hoser et al., 2006). This section provides an overview of SNA and explains some of its techniques and measures used in this thesis. The major reference for this section is the book *Social Network Analysis: Methods and Applications* (Wasserman and Faust, 2007). It is one of the most widely referenced books on this topic and presents both an overview of the research field and reviews the various SNA methods.

### 3.1 Social Network

Social network is a social structure containing various nodes (usually organizations and individuals) which are connected by one or more kinds of relationships. The nodes in the social network are called actors, and the links between each pair of actors are called relationships. For example, in (Shen et al., 2006), a social network named Terrorism Network has country actors such as Greece and India, terrorism organization actors like Hezbollah, and even terrorism attacks actors such as 911 attack. The links between each pair of those nodes are the relationships between each two nodes. For example, the network links all terrorism organization and terrorism attacks by an is-responsible-for relationship. All terrorism attacks and countries are linked by an occurred-in relationship.

Social networks have two major characteristics, their mode and their relationship complexity. Typically social networks are 1-mode, 1-plex networks, i.e., they view the various nodes as one type and the relationship between these nodes as one type. For example, a social network of friendships among residents of a neighborhood would consist of one type of actor, neighborhood residents who are related by the one relation of friendship. A 2-mode network consists of two different sets or types of actors and a 2-plex network consists of two different kinds of relationships. Relations can be undirected or directed. For undirected relations, the link between two actors does not have a ‘to’ or ‘from’ consideration; the line from node i to node j is the same as the line from node j to node i. For directional relations the path from node i to node j is different from the path from node j to node i.

A *sociomatrix* is typically created for a social network. In that matrix, each actor has a row and a column. For example, actor i has row i and column i. The value in each entry means the weights or connectivity of a pair of actors. For example, if \( x_{ij} = 1 \), then there is
a directional relationship from actor i to actor j with weight 1. If the relationship is undirected, only half of the matrix is used. Regardless of whether the relationship is directed or not, there is only one entry to store the weight of a particular relationship between two actors, so the sociomatrix is created per relationship. Separate analysis can be done for each relationship type; however, as discussed later in section 4.6, the approach being used by researchers in intensional ontology analysis is to treat all the different class types as one kind and all different relationship types as the same kind of relationship. This approach requires a projection algorithm to transform a 1-mode n-plex social network into a 1-mode 1-plex social network. There is only 1 mode since all classes are considered as only 1 type, a class type.

3.2 Social Network Analysis

Social network analysis (SNA) is the study of relations among a set of actors with the goal of finding two different kinds of network patterns: ones that show subsets of nodes organized into cohesive social groups and others that uncover subsets of nodes having equivalent social positions or roles (Freeman, 2004). Since a complete social network can be viewed as a graph, social network analysis techniques are mostly based on graph analysis. In the following sections, various SNA measures based on graph metrics are presented.

3.2.1 Centrality

Based on the graph nature of social networks, the analysis of social network is highly dependent on graph theory. One of the uses of graph theory is to find out the most important actors in a social network. A prominent actor is defined as “those that are extensively involved in relationships with other actors. This involvement makes them more visible to the others.” (Wasserman and Faust, 2007). Because links may be directed with a sender and a receiver, formulas for both undirected and directed centrality measures have been developed.

The general form of centrality measurement over a specific node is simply a function calculating the centrality for that node, say $C_A(n_i)$, where A represents a specific
centrality measurement algorithm, and \( n_i \) is the ith node in a graph. However, this function may also be applied to a group of actors to measure the importance of a specific group. This application is very important when determining clusters of various network nodes. The following centrality measurement algorithms consider either the actor’s position in a global perspective or in its neighborhood view.

The actor’s degree centrality is the most basic and easiest algorithm among a variety of centrality measures. It simply calculates the number of links one actor has regardless of the direction of the link, so the formula is as follows:

\[
C_D(n_i) = d(n_i) = \sum_{j=0}^{N} x_{ij}, \quad \text{where } d(n_i) \text{ means the number of links the ith node has, } N \text{ is the total number of nodes, and } x_{ij} \text{ is the weight of the tie between node } i \text{ to node } j. \text{ If there is no weight involved, } x_{ij} \text{ can be 1. If the links have direction and the direction is to be considered, i.e., incoming and outgoing, the in-degree centrality and out-degree centrality may be calculated as:}
\]

\[
C_{D_{\text{out}}}(n_i) = d_{\text{out}}(n_i) = \sum_{j=0, j\neq i}^{N} x_{ji}
\]

\[
C_{D_{\text{in}}}(n_i) = d_{\text{in}}(n_i) = \sum_{j=0}^{N} x_{ij}
\]

The density of a network represents how connected the network is relative to the total number of possible connections between the N nodes, in an undirected graph, i.e. \( T_u = \frac{N(N-1)}{2} \). For an undirected network, the density is determined as

\[
density_U = \frac{\sum_{i=1}^{N} C_D(n_i)}{T_u}
\]

For a directed graph, at most two edges are possible between each node so \( T_d = N(N-1) \). The density for a directed network is calculated as

10
The degree measure for a node focuses only on its directly connected neighbor nodes. To get the global position of a node, this algorithm is not applicable. Additionally, it assumes the more ties a node has, the more important it is. However, it does not consider the relative importance of its connected neighbor nodes. Therefore, other centrality measures must be used to find the global position of a node within the network.

The *closeness centrality* measure views an actor’s importance is based on its closeness or distance to all the other actors in the set of actors. The idea behind this measure is that a node’s closeness is measured by how quickly it can interact with all other nodes in the network. The important actors considered as central are often productive in communication with all other actors. The closeness centrality formula for undirected graph is as follows:

\[
C_c(n_i) = \left( \sum_{j=0}^{N} d(n_i, n_j) \right)^{-1},
\]

where the subscript C is for closeness, N is the total number of nodes, and \(d(n_i, n_j)\) is the distance from actor i to actor j. By inverting the sum of distances, the one having the shortest distance gets the highest centrality measure.

Obviously, this algorithm is much more time consuming than the degree centrality algorithm since it puts each node in a global position. However, the question here is how to define the distance from one actor to another. It can be calculated either based on the number of links on a single path or the actual information this path can transmit. This measure, regardless of how it is calculated, only reflects the interaction between two actors. It ignores the other actors on that path between the two.

The *betweenness centrality measure* takes into consideration the level of importance one actor might have in the interactions between two nonadjacent actors. Here, the actors in the central positions are those “other” actors who contribute the most to transmitting information between any other pair of actors in the network.

Since there may often be more than one path passing from actor i to actor j, consider for each pair, only the shortest path to transmit information, referred to as a geodesic in
SNA terminology. If there are more than one geodesic, then define $g_{ij}$ as the number of geodesics from $i$ to $j$. If all of these geodesics are equally likely, then the probability of any one of them is simply $1/g_{ij}$. Now, let $g_{ijk}(n_k)$ be the number of geodesics linking actor $i$ to actor $j$ through actor $k$. The betweenness centrality of an actor $k$ is the probability that it is used in passing information from that actor $i$ to actor $j$ and is determined as

$$C_B(n_k) = \sum_{i,j} g_{ij}(n_k) / g_{ij}$$

where the subscript $B$ represents betweenness.

The minimum value for centrality is 0 when an actor does not lie on any geodesic, and the maximum value is $(g - 1)(g - 2)/2$, the number of pairs of actors not including $n_k$.

The standardized betweenness centrality measure becomes:

$$C'_B(n_k) = C_B(n_k) / [(g - 1)(g - 2)/2]$$

which makes the values between 0 and 1.

Another global measure for a network is the diameter which is the maximum geodesic between any two pairs of nodes.

The betweenness centrality measurement offers the most general and powerful way of determining the importance of actors. However, it assumes that actor $i$ only chooses the shortest path to transmit information to actor $j$, and the probability of choosing multiple shortest paths is evenly distributed. It simply omits the fact that an actor which is on geodesic and has large degree may be more expansive than any other actors who are also on geodesic but have a relatively small degrees. The actor with a large degree should have a higher probability to be chosen on the geodesic.

Additionally, the betweenness centrality omits all paths between $i$ and $j$ which have longer length than the shortest paths. This assumption is not necessarily valid. For example, in real world Internet connection, it is possible for one computer to choose a path other than the shortest path to send messages to another computer. Therefore, the betweenness centrality is generalized by the information centrality measure that considers all paths with weights which are dependent on the distances of those paths. The actual algorithm of measuring information centrality of each actor is as follows:

Step 1. Set up a $N \times N$ matrix $M$ as follows:

$$a_{ii} = 1 + \text{sum of values for all lines incident to } n_i$$

$$a_{ij} = 1 \text{ if actor } i \text{ and actor } j \text{ are not adjacent. Otherwise, } a_{ij} = 1 - x_{ij}$$

12
where $x_j$ is the out-degree of actor $i$ to $j$.

Step 2. Get the inverse of $M$, $C$.

Step 3. Calculate two intermediate values:

$$T = \sum_{i=0}^{N} c_{ii}$$
which simply computes the sum of the diagonal entries of the matrix $C$.

$$R = \sum_{j=0}^{N} c_{ij}$$
which simply computes any row sum of the entries in matrix $C$.

Step 4. Calculate the information centrality of each node:

$$C_i(n_i) = \frac{1}{c_{ii} + (T - 2R) / g}$$

Step 5 (optional). Standardize the centrality:

$$C'_i(n_i) = \frac{C_i(n_i)}{\sum_{j=0}^{N} C_j(n_j)}$$
which is in the range $[0, 1]$.

This centrality measure has not been used as much in SNA work as the others although it is not too difficult to calculate. The problem is that determining the inverse of a matrix is not possible when its determinant is 0. This situation can happen when there are isolated nodes within the graph.

### 3.2.2 Prestige

The prestige of an actor in a network focuses on the actor as a recipient of links, i.e., it is related its node's in degree. In other words, if an actor is the recipient of links from many other prestigious actors, its prestige should also be high. The term status has also been used for this measure.

There are numerous prestige measures in SNA and these are typically denoted with a $P$ subscripted by a letter representing the kind of prestige. The simplest is $P_D$, degree prestige which is the indegree of each actor. Relative degree prestige for actor $i$ is defined as the proportion of actors who link to actor $i$. The other prestige measure that is quite frequently used in SNA is the rank prestige $P_R$, which considers the prestige of the individual actors that are linked to the actor for which rank prestige is being determined. In order to quantify this measure, we need to define $P_R(n_i)$ to be the rank prestige of actor
i. The ith column of the sociomatrix provides which actors choose actor i. Therefore, its prestige can be measured by

\[ P_R(n_i) = x_{i1}P_R(n_1) + x_{i2}P_R(n_2) + \ldots + x_{ig}P_R(n_g) \]

where \( x_{ij} \) is the weight of the tie from actor i to actor j, and there are totally g actors in the network. However, at the very beginning, no one’s prestige is known to the other actors. This results in a system of g linear equations with g unknowns. Various approaches can be used to solve this system of linear equations and are categorized based on the constraints they place on the sociomatrix or the system itself.

4 Related Research

This section presents several current papers found in the research literature that address visualizing large heterogeneous networks. The first one describes factors affecting the visualization of large ontologies and is followed by a section that briefly overviews OntoSELF, a 3D ontology visualization system developed for a related thesis completed at Miami University (Somasundaram 2007) and which this thesis research enhanced by adding visual cues, visualization of user-defined relationships, performing SNA with a set of standard SNA metrics and new filtering and abstraction capabilities. The remaining four papers are divided into two categories: visualizing social networks and applying SNA techniques to ontologies. The first two are in the first category and describe social network visualization systems SocialAction and OntoVis.

The goal of SocialAction is to assist users in interactively exploring social networks by systematic and flexible use of the various SNA measures, some being discussed in the previous section. The OntoVis system couples an ontology with a social analysis network in order to use the ontology to perform semantic abstraction by filtering instances based on their concept types. Both of these systems work on extensional knowledge, that is, instance data. The other two papers are very recent and begin to explore the application of SNA techniques to intensional ontologies. The first examines automatic partitioning of concept hierarchies and the other uses SNA techniques to analyze the structure of ontologies.
4.1 On the Visualization of Large Size Ontologies

In (Tzitzikas and Hanaut 2006), features that should be supported in ontology visualization tools and factors affecting the satisfaction with the visual layout of an ontology are described. The focus is on techniques that affect visualizing the structural part of an ontology in order to improve user understanding. Various kinds of rules that may be incorporated in the visualization process, basic considerations for visualizing and understanding large diagrams, and standard techniques used in visualizing large ontologies are all discussed.

Three kinds of rules are important to determining the usefulness of a graph layout: graph-based, semantic-based, and domain-specific. The objective of graph-based drawing rules is to improve the graph’s readability as much as possible. Typical rules include “minimizing the edge crossings, reducing the space occupied by the graph, placing the nodes in an orthogonal grid, displaying symmetries if there are any, and satisfying other aesthetic criteria.” (Tzitzikas and Hainaut, 2006). Semantic-based drawing rules consider the semantics of an ontology when constructing its visualization. For example, the semantics may include the structural relationships such as the taxonomic relationship IS-A and the meronymic relationship part-of. Using such rules usually produces a better layout for small to medium sized ontologies. Domain-specific drawing rules are considered somewhat empirical and result based in a detailed knowledge of the specific domain so that they cannot be translated into standard formal rules.

For large diagrams, the first two kinds of rules by themselves do not address the core visualization problem of too much information contained in a single diagram. These two kinds of rules are still necessary, however, for the purpose of readability and understandability, especially in small to medium size diagrams. The third kind of rule relies on a designer’s experiences and intimate familiarity with the problem domain and normally leads to manually created rules and drawings. These three kinds of rules are all useful but may have mutually conflicting objectives and constraints so that careful integration of them is necessary to optimize the visualization process. Force-directed placement algorithms are appropriate because they permit domain specific rules to be incorporated in the visualization by their expression in the form of a force.
To achieve high-quality understanding of a large-scale ontology, its visualization must capitalize on two important considerations:

1) The basis for understanding the ontology is the labels for its nodes and edges. All other features such as the color of nodes and thickness of edges are used to improve perception, but alone they have no meaning.

2) The barrier for understanding a large ontology is mainly caused by its complexity and not its layout. Even if an optimal layout for 100000 node ontology were created, the complexity of the ontology itself would make it extremely difficult to understand.

The first implies that methods motivated by techniques or research in information retrieval or Web searching are of significance. The second implies methods for generating focused, summarized and abstract views of the graph are extremely valuable.

The three main methods for large ontology visualization in details are:

1) Context-based Browsing allows a user to gradually understand the ontology by focusing the user’s attention to a particular area of the ontology rather than showing the complete ontology. The user may change the focal node by clicking on another node.

2) Filtering is used to eliminate from the ontology visualization the nodes and/or edges not meeting the filtering criteria. The following filtering criteria are examples of the filtering process. Property-based criteria are evaluated upon the property of nodes and/or edges, such as labels. Local connectivity criteria are evaluated upon the connectivity property of nodes, such as outbound number. Global Connectivity criteria is similar to the previous local criteria, but with global connectivity, a set of nodes and edges is considered and not just a single node and/or edge. The complex global connectivity criteria combine multiple global connectivity criteria. Sophisticated Connectivity criteria are often used for filtering large diagrams. Some web searching methods are applied here.

3) Clustering groups some highly related nodes into one node to reduce the number of displayed nodes and edges. Roughly, filtering criteria such as connectivity criterion could also be used for clustering.
4.2 OntoSELF

The OntoSELF system (Somasundaram 2007) provides 3D visualization of the structure of any ontology with various filtering capability based on different ontology metrics and permits parallel computation of ontology metrics as well. OntoSELF assists users to better understand and analyze any ontology by using 3D visualization techniques. However, this system is not simply another Ontology 3D Visualization system, it provides many analysis features as follows:

(1) Visualize Inheritance structure of any ontology;
(2) Various layouts supported based on user-selected weighting functions;
(3) Flexible filtering capabilities based on structural metrics;
(4) Parallel computation of various weighted layouts and filtering options applied on a single ontology, and one weighted layout and filtering option applied to multiple ontologies.

The weighting and filtering capabilities provided by OntoSELF were shown to easily support the following visualization tasks used in previous evaluations of another 2D visualization system CropCircles (Wang and Parsia, 2006)

1) Find the Bushiest Child Node for a given node. Their definition of “bushiest” is the child node of the given node which itself has the most children.
2) Find the Largest Subtree for a given a node. The largest subtree is the child node which itself has the most descendants,
3) Find a Deepest Node of a given node. Any leaf node that has the greatest depth for the subtree rooted at the given node.
4) Find 3 Nodes with at Least 10 Children. Any nodes in the ontology that have an outdegree of 10 or more, i.e., 10 or more children nodes.
5) Find 3 Top-level Nodes that Root a subtree of Depth of at Least 5.

Although OntoSELF handled these visualization tasks, it can still be improved to assist the user to gain a global understanding of a complex ontology with a variety of relationships. Currently only the IS-A and PART-OF relationship types are used in the visualization layout process. A technique that combines the 3D layout using the hierarchical structuring based on these relationship types with other kinds of semantic relationships is needed. Various useful visual cues such as color, size, and labeling are
lacking or used in limited form. Although OntoSELF provides eight different kinds of node filtering for nodes are provided, i.e., name, level, interval rank, information content, direct children, descendants, subtree depth, and hub value, additional metrics determined using SNA techniques such as centrality measures may provide users with other constructive views of an ontology. In addition OntoSELF has a very limited abstraction capability for an ontology. The nodes kept after filtering are connected to one another if paths exist between them through the filtered out nodes. A more sophisticated abstraction mechanism would provide the user with another approach to gaining a high level overview of any ontology that they could then use as a starting point for refinement to look for details of interest. Using semantic abstraction and clustering techniques could improve OntoSELF’s capability to provide an understandable global overview.

4.3 Balancing Systematic and Flexible Exploration of Social Networks

SocialAction (Perer and Schneiderman, 2006) is a social network visualization system that allows users to both systematically and flexibly examine the network. SNA measures are used to produce a variety of visualizations that the user may then refine through iteration by filtering, aggregating, and untangling operations on the network. Aggregation is performed using the link structure to find cohesive subgroups and untangling of the network is done by examining different link types individually. A matrix overview may be used to find patterns across different link types.

Based on the researchers’ discussions with social network analysts, the individual attributes of nodes are ignored and the focus is on a node’s structural characteristics within the network, for example, its position in the network, as isolated or connected to many other nodes. SocialAction presents a variety of the previously described SNA measures such as betweenness centrality, closeness centrality, and degree which may be selected by the user in order to rank nodes by their values for the selected measure. The ranking is displayed in an ordered list, as well as color coded in the node-link network visualization based on the ranking. The user may easily and quickly change the measure and the rankings are recalculated and presented to the user, however, the node position in the network layout remains the same so that the users do not lose their orientation. The
user may also filter out nodes by setting a lower and upper filtering value for the selected measure. Through filtering on rankings of such SNA measures selected by the user, the legibility of the networks improves; thus areas of interest may more easily be identified by the user. Additionally, SocialAction can create a scatterplot for any two selected measures, for example, degree as the x-axis and betweenness centrality as the y-axis with the scatterplot fit to a linear function and color shading used to distinguish those nodes above the line and below the line.

Because one of the main objectives in examining social networks is to find cohesive subgroups of nodes, SocialAction provides the capability of determining subgroups or “communities of interest” based on the link structure. Because the algorithm used to determine subgroups may do so at an ineffective level of granularity, the user can adjust the level of clustering. The user may collapse a subgroup into a single meta-node representing the complete subgroup with the size of the node proportional to the number of nodes in the subgroup. The subgroup can also be treated as its own separate social network on which further analysis can be performed.

Another consideration is social networks may have multiple link types and are referred to as multiplex networks. The example used looks at an attack link type between a terrorism organization and a country as being a different link type based on the year the attack occurred. This approach is using a value associated with a link or relationship such as year occurred to filter out those attack links that do not meet the selected value. In some ways this approach appears as artificial. All the links are attack links but now they are being “specialized” into another type of link based on the year of occurrence. It is not clear how flexible SocialAction is with respect to selecting relationships and values associated with relationships. For example, would SocialAction permit selecting year as an object instead of country and create a relationship “attacked during” between year and terrorism organization and then consider different link types based on the country associated with the “attacked during” relationship.
4.4 Visual Analysis of Large Heterogeneous Social Networks by Semantic and Structural Abstraction

OntoVis (Shen et al., 2006) is a visual analytics tool that performs semantic and structural abstraction and importance filtering to control the complexity of large networks and to make possible analytic reasoning on the network. The objective is to offer the user a global overview of the network yet be able to create detailed views of some isolated groups.

OntoVis uses the intensional ontology which is referred to as the ontology graph to filter and cluster the nodes in the associated extensional ontology which is referred to as the semantic graph. Most of the visualization capabilities are on the extensional ontology. Two types of abstraction are defined, semantic and structural. Semantic abstraction is simply performed by selecting the node types from the ontology for which instances are to be displayed in visualization of the semantic graph. Once a node type or multiple node types are chosen from the ontology graph, all node instances with these types are added to the semantic abstraction. All the other instance nodes that are not included in the abstraction are converted into attribute values of the nodes kept in the abstraction when appropriate, i.e., there exists a relationship link between it and the selected instance. This approach allows the user to incrementally select which additional instances to display based on the chosen relationship to the existing displayed instances. It is not clear from their discussion exactly how a related instance is selected, i.e., just clicking on a displayed attribute value. From the description, it appears that the attribute list of a node may be browsed by the user.

Structural abstraction is performed using two rules the researchers have found to be true in most of the social networks they have worked with. The networks contain many nodes of degree one connecting to specific nodes. The networks also have many duplicate paths between nodes. OntoVis provides the capability of directly removing one-degree nodes and duplicate paths. Finally OntoVis provides importance filtering using measures that are calculated all based on the type of node in the semantic graph. The two primary measures node degree per type of node and its dispersion are used to provide the users with an overall idea about the connectivity of the nodes associated to each type from the
ontology graph. Key actors in the network are typically nodes with large degrees per type and large dispersions.

Typically the visualization is performed on the semantic graph based on information used from the ontology graph. Visualization on the ontology graph, however, may be done based on selecting one of two measures, disparity of connected types and its associated dispersion, which are calculated using the semantic graph but then reflected back in a visualization of the ontology graph. The node type disparity is calculated for each node $i$ of a given type $\alpha$. A small value indicates that the links for a given node are evenly distributed to all types which can connect to nodes of type $\alpha$. The average value over all nodes of type $\alpha$ and its corresponding variance or dispersion is calculated for each type. These values are then scaled based by a summation value. The summation is taken over all types which can connect to type $\alpha$ with the value being summed as the square of the percentage of nodes having a given type. The user selects one of these measures to be visually encoded by the size of the node representing an ontology type in the display of the ontology graph.

The OntoVis system is applied to two cases studies, a movie network compiled from the UCI KDD Archive (http://kdd.isc.uci.edu) and a terrorism network MIPT Terrorism Knowledge Base (http://www.tkb.org). The authors state that for the movies case study OntoVis lead user to “some interesting and useful findings. Specifically, users can use the ontology graph to construct an abstraction that best relates to their analysis objective.” (Shen et al., 2006). For the terrorism network case study, OntoVis aided identification of the most important organizations and was able to analyze their relationships based on location and classification of the organization. The conclusion was that the ontology graph and the abstractions of OntoVis provide strong support for visual analysis.

4.5 Network Analysis as a Basis for Partitioning Class Hierarchies

In this research (Stuckenschmidt, 2005) network analysis techniques are used to analyze ontologies which are viewed as networks of concepts, relations, and instances. These techniques are used to produce information such as important concepts, evaluation of the dependency degree between two concepts, group related concepts, and determine totally
unrelated components of the ontology. The specific task this information can be used in is ontology partitioning into a number of disjoint and covering set of concepts in order to support distributed maintenance, selective reuse and efficient reasoning on ontologies.

The partitioning method used on lightweight ontologies, i.e., those consisting of concept hierarchies has four steps:
1) Create a dependency graph for the source ontology by adding concepts and properties as nodes and relationships as edges.
2) Determine strength of dependencies by using SNA algorithms to calculate degrees of relatedness between concepts based on the graph structure. The strength of relatedness may also incorporate the different types of relationships. In this current paper, the authors only considered one type of relationship, the subclass or is-a relationship and no importance weighting is used.
3) Determine modules by using graph algorithm to find minimal cuts in the strength network. Then use those minimal cuts to split the whole graph into various groups of nodes which are strongly related to each other within its group. It was determined through experimentation that the line island method (Batagelj, 2003) on a relative strength network produced effective results.
4) Optimize the partitioning by assigning leftover nodes to the module that they have the strongest relationship with. Moreover, a merge algorithm is performed to group some smaller modules into larger ones to get a less scattered partitioning.

Several of the used SNA algorithms are described in more detail below.

**Relatedness or Dependency Strength:** To determine the dependency strength between two concepts, the structure of the dependency graph is used. The strength from one concept node to another is based on the number of connections the concept node has. Then the dependency strength of a relationship edge between two concept nodes \( c_i \) and \( c_j \) is determined to be the proportional strengths of the connection between those two over the total number of relationship edges for \( c_i \). It is computed as

\[
  w(c_i, c_j) = \frac{w_{ij} + w_{ji}}{\sum_k w_{ik} + w_{ki}}, \text{ where } w_{ij} \text{ is the weight of the edge from } c_i \text{ and } c_j.
\]
Note that the authors assumed a standard value of 1 for all edge weights and did not take into consideration the type of the relationship since they only were using the is-a relationship. Also they included relationships in both directions so this measure would incorporates edges in both direction and may be considered as looking at edges in a undirected manner; however, the strength of dependency is directional, i.e. \( w(c_i, c_j) \) may be different from \( w(c_j, c_i) \).

**Determining Module using Line Islands:** Based on the dependence strength network calculated, sets of strongly related concepts can be identified to serve as candidates for constructing a component in the partition. An algorithm to compute all maximal line islands is used. A line island is a set of vertices \( I \subseteq C \) in a dependency graph \( G = (C, D, w) \) - where \( C \) is a set of nodes, \( D \) a set of edged representing dependencies between them and \( w \) is a set of weights of the edges if and only if

1) \( I \) induces a connected subgraph of \( G \)

2) There is a weighted graph \( T = (V_T, E_T, w_T) \) such that:
   a) \( T \) is embedded in \( G \)
   b) \( T \) is a maximal spanning tree with respect to \( I \)
   
   Note: edge direction is not considered to determine maximal spanning tree.
   c) the following equation holds:

\[
\max_{\{i,j\} \in D \cap I} w_{ij} < \min_{\{i,j\} \in E_T} w_{kl}
\]

The above equation is used to identify sets of concepts that have a stronger internal dependency to each other than to any other concepts not in the set. An upper and a lower bound on the size of the identified set must be input and then the algorithm assigns an island number to each node in the dependency graph. The assignment of a zero as an island number for a concept means that the concept could not be assigned to an island.

**Improving Partitions by use of the height of line islands:** The common problem of the line island method is that often scattered partitions of very few nodes are created. To diminish this problem, the measure of internal dependencies, or height of an island is introduced as:

\[
\text{height}(P) = \min_{\{i,j\} \in E_T} w_{ij}
\]
The minimal spanning tree $T$ created by the line island algorithm is used to identify the islands that might need merging by finding the overall strength of the internal dependency as the strength of the weakest link in the spanning tree. Since most problematic ones have very strong internal dependencies, they merge any islands with a height of 0.5 or more with other islands. To determine which island to merge with a problematic one $P_1$, all other islands $P_2$ with a height of more than 0.5 are ranked in order of their height and their merging potential with $P_1$ is calculated as:

$$m_R(P_1) = \text{height}(P_2) \cdot \sum_{v_i \in R_1, v_j \in P_2} w_{ij} + w_{ji}.$$

$P_1$ is merged with the island having the highest merging potential. If there are multiple islands with the same merging potential, the one with the highest height is chosen.

This partitioning method is used on the ACM classification of computer science topics and the result compared with information about important computer science topics extracted from the web sites of Dutch Computer Science Departments. The reported precision rate is about 60%.

### 4.6 Semantic Analysis of Ontologies

Although SNA has been used for quite some time in social and behavioral sciences, SNA is beginning to be used in ontology and the Semantic Web research. This paper (Hoser et al., 2006) examines the use of SNA for analysis of ontologies and the Semantic Web. The main focus is the use of various centrality measures to describe the fundamental subject matter and structure of the ontology. These authors coin another term for applying SNA techniques to ontologies and the Semantic Web, SemNA or Semantic Network Analysis. This application of SNA techniques to ontologies is not actually new, however, since many measures for ontology analysis have already been used, some of which are just graph analysis measures, for example node degree. Many of the SNA measures are basically those developed for graph analysis and as such have been applied to ontologies (Cross and Pal, 2007) (Harith et al., 2006).

The following describes how several SNA measures are applied to ontologies and their use on two specific ontologies: the Semantic Web for Research Communities.
(SWRC) (Sure et al., 2006), an ontology that models entities associated with research communities such as persons, organizations, etc, and their relationships and the Suggested Upper Merged Ontology (SUMO) (Pease et al., 2002), a formal public ontology whose goal is to provide a framework that defines basic and universal concepts that can be used in merging varying domain ontologies. SUMO is being used for research and applications in search, linguistics and reasoning.

In order to apply SNA techniques on ontologies, these researchers transform an ontology into a standard format that is more readily analyzed with existing techniques. An ontology can have \( n \) different types of concepts (nodes) and \( k \) different kinds of relationships (edges). In SNA terminology, the graph for the ontology is an \( n \)-mode multigraph with \( k \) edge types which is more complex and difficult to analyze. In their analysis, an ontology graph is greatly simplified into a 1-mode (one node type) and 1-plex (one edge type) network using the following rules:

1. Concepts and properties become nodes.
2. A directed edge \((C1, C2)\) shall be created if \( C1 \) is a direct subconcept of \( C2 \).
3. An edge is added from the domain concept to each of its property nodes.
4. An edge is added from each property node to its range concept node except when the range for the property is scalar-valued or untyped.
5. An edge is added from each subproperty to its superproperty.

Based on these nodes and edges in the resulting graph, an adjacency matrix of the above is constructed with one row and one column for each node and the entry \( a_{ij} \) is 1 if there is an edge from the \( i \)th node to the \( j \)th node. Note from this description the resulting graph is a directed graph since the edges are specifically directed.

The primary SNA measures calculated are in degree, out degree and betweenness centrality density for each no and diameter. Since the network created using their transformation procedure produces a directed network, it is assumed that the density calculated is the directed density. In addition to these standard SNA measures, eigenvalue spectral analysis techniques are used on the adjacency matrices to gain other insights in understanding an ontology. They create a square complex adjacency matrix \( C \) as \( r + si \) where \( r \) is the number of edges from node \( m \) to \( n \) (outbound) and \( s \) is the number of edges from node \( n \) to \( m \) (inbound). Through several transformation steps they produce
a matrix that has full rank so that a complete orthogonal basis can be found on which other techniques are applied in order to analyze the network structure. Their description of this process and the resulting interpretation of the eigenvector analysis are somewhat difficult to understand. As they point out analyzing the results of the eigenvector analysis require experience by the user.

One concern we have is that since the transformation process produces a directed graph that only specifies the is-a links, only upward paths from a subconcept to its superconcept (also from a subproperty to a superproperty) are permitted so the betweenness centrality measure does not consider the opposite direction relationship, superconcept “generalizes” subconcept. From our perspective, it seems like this link between a superconcept and a subconcept should be treated as undirected and traversable in either direction.

5 Designing Extensions to OntoSELF

This chapter will discuss the major design issues to extend the OntoSELF-v1. Section 5.1 presents the ontology preprocessing approach of transforming an ontology to a 1-mode 1-plex social network. Section 5.2 describes the rationale for developing four visualization packages. The problem of using a standardized or a customized OWL parser is discussed in section 5.3. The last two sections describe the issues and implementation algorithms for developing centrality and prestige metrics and the semantic and structural abstractions.

5.1 Ontology Preprocessing Approach

As discussed in sections 3.1 and 4.6, an intensional ontology can be viewed as an n-mode (n type of nodes) k-plex (k types of relationships) social network. To process a social network in order to produce centrality and prestige values, the social network must be a 1-mode 1-plex network. Therefore, an intensional ontology must be transformed to a 1-mode 1-plex social network. One of the concerns, however, was the desire to be able to look at both separately and as combined the two major categories of relationships in an intensional ontology: the IS-A or hierarchical relationships which provide the main
structure for the ontology and the user-defined relationships which indicate the user
details of the structure within the ontology.

Another concern was that in some ontologies such as the GO other user-defined
relationships such as the Part-of have been treated as hierarchical structuring
relationships. The majority of the research using the GO for semantic similarity
measurement between GO terms has treated the Part-of relationship equivalently to the
IS-A relationship (Sun 2007). To address both concerns and to provide flexibility for the
user, a design decision was made to allow the user to specify what ontology relationships
to include in the hierarchical category and what ones to include into the user-defined
category.

The IS-A is by default included in the hierarchical category and cannot be removed.
The user can select other user-defined relationships that are to be treated as hierarchical
structuring relationships such as the Part-of relationship. The user can also select which
individual user-defined relationships within the intensional ontology are bundled into the
one edge-type user-defined for the SNA transformation process. The transformation
process assigns the number of relationships in the same category merged on a single edge
to that edge. This design decision allows distinguishing the two kinds of edges with color
cues and the number of merged relationships by edge thickness in the visualization
process. The two categories of edges are further merged into one edge type, a 1-plex
social network when calculating centrality and prestige measures by summing the edge
weights.

5.2 Visualization Packages in OntoSELF

OntoSELF-v1 solely uses the superclass/subclass relationship as the basis of the
edges in its visual presentations. This relationship is hierarchical and permits multiple
parents. It is used in calculating most of the metrics on the nodes and in particular: node
level, information content, interval rank and extent. Any one of these can be selected as
the weighting function to determine the 3rd dimension value for nodes in the layout
algorithm used to display the ontology. The extension to OntoSELF is designed to
include all user-defined relationships in the analysis and visualization as well. As
discussed in the previous section, the design decision was made to allow the user flexibility in specifying what relationships are to be included in each category of relationships. However, the adding of this user-defined category requires considering the various ways these two major categories of relationships may interact in the visualization results.

The superclass/subclass relationship provides an inherent hierarchical structure for the visualization of an ontology. However, the user-defined relationships do not typically provide a structuring mechanism for the ontology. The different purposes for user analysis of an ontology structure may also involve different combination of these two categories of relationships. To support flexible experimenting with different visualization views for different analysis purposes, it was decided that OntoSELF-v2 would support four visualization packages. Package 1, H3D displays an ontology with only hierarchical relationships in 3D visualization. This package is essentially that provided by OntoSELF-v1 except that SNA metrics are calculated based on a 1-plex network where the edge type is only hierarchical and visual cues enhancements are included. User-defined relationships may be selected by the user for treatment as hierarchical relationships as discussed in section 5.1.

Package 2, U2D views an ontology as a pure social network in 2D visualization by including only user-defined relationships. The user has selection capability on what user-defined relationships from the ontology to bundle into the user-defined category. Initially the 3D VTK visualization capability was being used for this package with only user-defined relationships; however, some initial experiments early on using each of the four existing weighting functions and also the four centrality and prestige measures as weighting functions for the 3rd dimension indicated that this was not a useful visualization layout since there is no inherent 3rd dimension representing a hierarchical structure for a social network. The artificial 3rd dimension, in fact, cluttered the whole visualization because of the many line crossings between each pair of nodes. In order to improve the visualization for the social network view of user-defined relationships, the JUNG 2D visualization capability is used for this visualization package. The 2D visualization scatters all nodes on a single flat layer with less node overlappings. Additionally, the line crossings are much less than those in 3D visualization. The flexible
JUNG visualization also allows users to drag any edges or nodes to have a customized better layout.

Package 3, HU3D and package 4, HU2D both include both categories of relationships but only differ in the visualization techniques. HU3D uses 3D visualization while HU2D uses 2D visualization. The details of the four visualization packages are discussed in section 6.3.2.

5.3 Standardized vs. Customized OWL Parser

The OWL (Web Ontology Language) language is recommended by W3C (World Wide Web Consortium, http://www.w3.org/) as the standard language for constructing ontologies. Therefore, to extract an ontology from an OWL language, it is necessary to have an OWL parser. There are many standardized OWL parsers recently developed and recommended by some of the ontology editing and knowledge management tools. For example, Protégé (http://protege.stanford.edu/) which is an ontology editor and knowledge-based framework uses Jena (http://jena.sourceforge.net/) as its OWL parser. The CO-ODE (http://www.co-ode.org/) which is an ontology engineering plugin for Protégé uses OWL API (http://owlapi.sourceforge.net/) as its OWL parser. In addition to their proved stability and accuracy, both the Jena and the OWL API are developed in Java and also for Java project development. Due to the flexibility and complexity of OWL language, it was decided to experiment with these two OWL parsers.

To parse an OWL file using Jena, an OntModel class must be created to load an OWL file either from a local disk or from a remote web server. After the complete OWL file has been read in, a built-in data structure holds all parts of the ontology. This built-in data structure can be accessed to provide various needed lists. For example, (1) to get all named classes (any anonymous classes are not included), OntModel’s listNamedClasses method is called. (2) To get subclasses of a particular class, call the listSubClasses method of OntClass. (3) To handle user-defined properties, all functional properties, object properties, inverse properties and transitive properties can be parsed by calling OntModel’s listObjectProperties, listFunctionalProperties, listInverseProperties, and
listTransitiveProperties methods. (4) To link classes to each property, the restriction on a property such as isAllValuesFromRestriction, isSomeValuesFromRestriction, or isHasValueRestriction can be called to get the property’s domain and range classes.

The Jena OWL API is easy-to-use and easily integrated into the OntoSELF system. However, after more testing it was determined that Jena could not handle large OWL files after it was used to parse a 2MB OWL file. The only message returned when using Jena on this large OWL file stated that the input is large. After an online search was conducted on this error message, it was discovered that this problem was due to the complex and memory-consuming built-in data structure used in Jena. Another minor problem of Jena’s OWL API is that it cannot support case-insensitive keyword parsing.

After the experience of using the Jena OWL API, the OWL API 2.1.1 (http://owlapi.sourceforge.net/) was investigated. It is also an open source Java API designed for the OWL Language. The latest version of this API is 2.1.1 which supports OWL 1.1 including OWL-Lite, OWL-DL and some parts of OWL-Full. The latest version is developed as part of the CO-ODE project (http://www.co-ode.org/).

The website for this API (http://owlapi.sourceforge.net/documentation.html), lists 10 simple code examples to demonstrate how to use the API. However, after the first unsuccessful experience with the Jena Framework, a simple code example demonstrating loading and saving a large ontology, specifically the GO OWL file, was experimented with. According to the errors reported, it could not handle large OWL files either. The limit of the memory of this built-in data structure is to build around 10,000 tuples where one tuple represents either a class or a property.

The results from experimenting with both of those two APIs, demonstrated the following two major disadvantages:

(1) Memory limitation of parsing large OWL files. Jena could not parse an OWL file larger than 2MB, while the OWL API could not build an ontology structure having more than 10,000 tuples, where a tuple is either a class or a property.

(2) Comparably slow during parsing. Since both APIs are used to build an ontology structure including all defined relationships and properties in the OWL file, they need to parse and include much information that is not necessary for OntoSELF-v2 to produce its visualizations. Users, therefore, would have to wait for parsing
that unnecessary information. The unnecessary information includes class attributes, comments and so on.

Because of these difficulties with the two OWL parsers, a customized OWL parser was created to handle parsing user defined properties. However, parsing these properties is more complex than only parsing for the subClassOf property and required the new parser to use parser stacks, which is discussed in Appendix II section covering the OWL Parser implementation. The results produced by the customized parser were verified by comparing examples with those parsed by Jena in Protégé.

5.4 Issues of Centrality and Prestige Calculation

Developing the algorithms for calculating the SNA metrics presented some challenges that needed to be overcome. The first is the issue of directed or undirected relationships. For the calculation of these metrics, all relationships are treated as undirected since all of them may be considered as inversed. For example, in the pizza ontology, users defined is_base_of and has_base and they are defined as inverse to each other. Has_spiciness does not have any inverse properties, but actually in real world, each may be considered to have an inverse.

In extensional ontologies for which SNA is typically applied, there are relationships that are not considered symmetrical, for example the relationship likes(a,b) might exist but that does not imply that likes(b,a) also exists. In these situations, it is important to be able to distinguish the direction of the relationship. This is not as important to an intensional ontology since the emphasis on the relationship definition. An inverse relationship is different from a symmetric relationship. For example, the inverse relationship of ‘likes’ is ‘likedby’. As another example, the inverse of ‘employs’ is ‘employedBy’. The link between the two concepts in the intensional ontology is therefore treated as undirected in the SNA analysis.
The formula for calculating **closeness centrality** is simply \( C_C(n_i) = \left( \sum_{j \neq i} d(n_i, n_j) \right)^{-1} \),

where the subscript \( C \) is for closeness, \( N \) is the total number of nodes, and \( d(n_i, n_j) \) is the distance from actor (node) \( i \) to actor (node) \( j \). The distance is the number of edges on the shortest path from actor (node) \( i \) to actor (node) \( j \). It is necessary to have a shortest path algorithm to calculate the shortest paths from one node to all the other nodes.

In the initial implementation, all four measurements were calculated based on the sociomatrix approach. To get the shortest distance matrix for which each entry contains the shortest distance from node \( i \) to node \( j \), the all pair shortest distance algorithm is used. The most famous algorithm solving this problem is Floyd-Warshall algorithm (Floyd 1962). However, the matrix implementation had to be replaced due to the memory limitations when trying to work on large-sized ontologies. The Dijkstra shortest path algorithm (Dijkstra 1959) is instead used to produce an array of shortest paths from one single source node to all the other nodes. A Dijkstra algorithm is provided in the algorithms library of the JUNG framework, but is implemented based on its own Graph interface which did not correspond with the pointer approach used to save memory. More details of the implementation issues are provided in section 6.3.4.

OntoSELF-v2 implements the Dijkstra shortest path algorithm (Dijkstra 1959) to produce an array of shortest paths from one single source node to all the other nodes. However, the capability of visualizing only user-defined relationships in package 2 (U2D) brings about the unconnectivity problem, which means not every node can link to all the other nodes in a graph.

In package 2, the unconnectivity problem occurs, for example, if one social network has two unconnected subgraphs. In this case, the distance between any pair of two nodes in different subgraphs is infinity. If infinity is used as the distance in the formula to calculate the measure, all nodes’ closeness measures become 0. One unconnected node can, therefore, make all nodes have closeness centrality measures equal to 0. An objective of the OntoSELF system is to provide the users with as much information on the ontology so that they may better understand it. Looking at only the user-defined
relationships of an ontology could cause separate connected graphs and, therefore, cause all closeness centralities to become 0. The result of 0 for all nodes does not provide the user with any information about the ontology. Because of this limitation, efforts to develop a within-connected-components closeness centrality measure but still consider the overall ontology were undertaken for such a situation.

The first approach used in experimentations was not adding infinity in the formula. But then this approach produced another problem. If one node has only one path to another node and both of these nodes are unconnected to all the other nodes, then the nodes have closeness measure 1. But there could be many other connected nodes, most of which have their closeness measure much less than 1. Obviously, those two dangling nodes are unconnected to the rest of the ontology; they should have a rather small closeness measure. The approach used is to average the total distance measure. After adding all shortest distances, the resulting sum is divided by the total number of nodes in the graph. Therefore, a closeness centrality based on the connectivity within subontologies yet averaged over the size of the whole ontology is produced. Finally, each node’s closeness measure is divided by the maximum closeness measure for normalization purpose.

From literature research on betweenness centrality algorithms, it was discovered that Brandes’ Faster Betweenness Centrality Algorithm (Brandes 2001) provides the best efficiency with respect to both space and time. It requires only O(n+m) space and O(nm+n^2logn) time on weighted networks, where n is the number of nodes in network and m is the number of edges. The details of this algorithm are discussed in section 6.3.4.

The algorithm used for information centrality given in (Wasserman and Faust, 2007) is easy to implement and has already been described in section 3.2.1. It requires the use of the matrix inversion operation. After some research, a matrix API called Jama (http://math.nist.gov/javanumerics/jama/) was discovered which can efficiently perform matrix inversion. However, the matrix inversion operation cannot be taken if the determinant of the matrix is 0. There are two conditions that can cause the determinant have value 0:

1. When the matrix has a row or column of zeros.

2. When the matrix has two rows or columns equal.
Therefore, the first condition limits the use of information centrality when the ontology graph is unconnected. The ontology could, however, only be unconnected in U2D since hierarchical structuring always assumes a dummy root concept from which all ontology root concepts are connected. The second condition can occur if two nodes in the graph link to all the same nodes. In this case, the information centrality cannot be used. This situation could rarely happen, but it still might happen in all four packages.

From literature research, the Faster Katz Status Score (Foster et al. 2001) algorithm for rank prestige was found to be the most efficient one developed so far since it only requires several one-dimension arrays for calculations and not the huge sociomatrix structure used by other algorithms.

5.5 Abstraction Algorithms

In OntoSELF-v2, two abstraction algorithms are used to cluster nodes into groups, and so produce a high-level abstract view of an ontology. This design idea of two algorithms was inspired by OntoVis (Shen et al., 2006), which demonstrates that both semantic abstraction and structural abstraction can enhance a user’s understanding of a given ontology structure. Their semantic abstraction uses the class type for clustering the instance data, i.e., applied on the extensional ontology. However, in an intensional ontology, this approach provided the idea to consider subclasses as instances of their superclasses. Therefore, OntoSELF-v2’s semantic abstraction is produced by using the ontology’s already hierarchical structure. A user can set a level where all nodes at that level will be considered the roots of all clusters. All their descendants will be assigned to their corresponding clusters.

Using the semantic abstraction approach required solving the problem of multiple inheritance for clustering. If a subclass has multiple parent classes and all of them belong to different clusters, a resolution method is needed to determine which cluster the subclass should be assigned to. The solution was made flexible by deciding which cluster is more appropriate for this class to belong to by determining which root of each cluster is more important in the whole structure. The method used to determine importance can be
selected from any of the ontology metrics. If two roots have the same importance, then one is randomly selected. In general, the clustering algorithm for semantic abstraction uses the characteristic that the nodes at higher levels in the ontology are more abstract concepts in order to divide an ontology semantically into abstract groups.

Initially from the proposal, it was planned to use Line Islands algorithm (Stuckenschmidt, 2005) as the structural clustering algorithm in OntoSELF-v2. It basically uses a minimal cut algorithm to split the graph into clusters of nodes that are strongly related by satisfying that the minimum edge weight within the cluster is larger than the maximum edge weight between the nodes within the cluster and those outside the cluster. The edge weight is calculated by \( w(c_i, c_j) = \frac{w_{ij} + w_{ji}}{w_{ik} + w_{ji}} \) where \( w_{ij} \) is the actual edge weight between node \( i \) and node \( j \). However, we could not find any efficient algorithm implementing this approach.

Through the literature research, an alternative approach for structural clustering, the faster Newman algorithm (Newman 2004) was found to be very efficient. The worst case time complexity is \( O((m+n)n) \), which is almost \( O(n^2) \), where \( m \) is the number of edges and \( n \) is the number of nodes in a network. It uses a quality function to evaluate each possible division of a graph. The quality function is \( Q = \sum_i (e_i - a_i^2) \) where \( a_i = \sum_j e_{ij} \). Here \( e_{ij} \) is the fraction of edges flowing from any nodes in cluster \( i \) to those in cluster \( j \). The \( e_{ii} \) is referred to as the inter-connectivity and the \( e_{ii} \) as the intra-connectivity. The whole quality function \( Q \) is the fraction of edges within a community minus the expected value of the quality if the edges fall at random in the network without considering the community structure; therefore, the higher \( Q \), the better the division. It starts with assigning each node into a different cluster, producing \( n \) different clusters. At the next level, it pairs each two clusters with at least one edge connecting one node in each cluster and calculates the change in \( Q \), which is \( \Delta Q = e_{ij} + e_{ji} - 2a_i a_j \). It only merges the pair with the highest change in \( Q \) at one level. Finally after \( n-1 \) iterations, the best cut in the clustering hierarchy is produced. In general, not only does OntoSELF-v2 use an efficient algorithm with the faster Newman algorithm, but also its quality function is more accurate than the simple minimum cuts.
6 OntoSELF Enhancements for Visual Cues and SNA

The first two sections present an overview description of the basic processing architecture and the user interface of OntoSELF-v2. For complete details on the original OntoSELF, referred to as OntoSELF-v1 in this section, see (Somasundaram, 2007). Section 6.3 describes the major modifications and enhancements made to OntoSELF-v1 in order to provide user defined relationships and SNA processing features.

6.1 Overall Processing Architecture

The UML class diagram containing all major classes in OntoSELF-v2 is shown below. After describing these classes, the processing flow is discussed.
In the above diagram, the GUI is the user interface class developed by SWT (Standard Widget Toolkit, http://www.eclipse.org/swt/) API which is a graphical widget toolkit for Java platform developed by IBM but now maintained by Eclipse Foundation (http://www.eclipse.org/). This API makes the previous JAVA AWT-based tedious GUI development very easy to handle and provides many powerful and easy-to-use components, like Group, List, Text, Label and some other commonly used components. Users have to implement some of these components themselves in Java.awt.
Inside the GUI class, the four column composites represent the four vertical regions in the user interface headed by Ontology Selection, Relationship Selection, Filtering Criteria and Clustering. Please refer the user interface shown in Figure 6-3. The open method is responsible for the layout of various components. To provide the list of relationships that the user selects from when deciding which relationships to include in the visualization (see user interface in figure 6.6), the GUI first calls the OWLtoGMLParser to parse the input OWL or GML file. The details of this class are discussed in section 6.3.3. After the parsing phase and setting of various parameters by the user such as the weighting function to be used for layout, size function to be used in determining node size, filtering criteria for determining which nodes are to be kept, and clustering algorithm for abstraction purposes, the GUI calls the Monitor.

The Monitor is mainly responsible for initiating and monitoring the processing of the input gml file. It plays a very important role in parallel computation. The ProcessGML class is the core processing part. It creates one hashtable called graph for storing hierarchical relationships, i.e., by default the is-a or subclass, and another one called weights to store the edge weights of hierarchical relationships. The SNGraph and SNWeights have the similar purpose of graph and weights, but they are in charge of all user-defined relationships. The various hierarchical metrics, such as level, height, IC value, extent value, interval rank, child count, hub value, and outdegree are calculated in the ProcessGML class. But the four centrality and prestige metrics and clustering algorithms are calculated in SNAnalysis class. The non-arrow line linking these two classes indicates their special relationship. All vector and hashtable field variables in SNAnalysis are pointers pointing to the corresponding variables in ProcessGML. The details of SNAnalysis class and its design issues are discussed in section 6.3.4. The SNAnalysis class is responsible for SN metrics calculation and ontology clustering. The SNVisualization class uses an external API, JUNG2 (http://jung.sourceforge.net/) to implement the 2D visualization for visualization packages 2 and 4. The details of this class and any related design issues are discussed in section 6.3.1.
Figure 6-2 Overall processing flow of OntoSELF
In the above figure 6-2, those squares with thicker borders represent the classes, while the other squares represent the processes. The input/output is represented by parallelograms, and the condition is represented as a diamond. The system starts at the GUI class which creates a user interface and requests the user to select a GML or OWL file and a specific package. However, if the GML file contains only hierarchical relationship, and the user selects package 2 which only accepts all user-defined relationship, a warning message is displayed. After selecting the source file and the package, the GUI calls OWLtoGMLParser to parse the input file and sends a list of property names back to the GUI for display to the user in the Relationship Selection area. If the input is an OWL file, it also sends the parsed GML file to the Monitor class; otherwise, it just forwards the processed GML file to the Monitor class. After the GUI gets the list of property names, it activates some features in the rest of the user interface depending on the package the user selected. After it gets all parameters necessary, it sends all them to the Monitor class. The Monitor class will determine if metrics are calculated in parallel.

Then the ProcessGML class gets all the parameters and reads data from the GML file. It calculates all the hierarchical metrics for each concept in the ontology. Then it sets the SNAnalysis class by passing the memory addresses of the four hashtables and the four centrality and prestige metrics vectors to the SNAnalysis class. The SNAnalysis class creates corresponding pointers pointing to those hashtables and vectors. After calculating all requested SN metrics and clustering the ontology, the changes are also made in the variables of the ProcessGML class. Finally, the ProcessGML class writes the processed GML file out and sends it to either the TCL processor for 3D visualization if the package 1 or 3 is chosen or the SNVisualization class for 2D visualization if the package 2 or 4 is chosen.

6.2 User Interface

Below in Figure 6-3 The User Interface of OntoSELF is the user interface for OntoSELF-v1. OntoSELF-v2 provides more functionality than the original version as
shown by in Figure 6-4 The User Interface for OntoSELF-v2 at the start of execution which illustrates the user interface at the beginning of system execution.

![Figure 6-3 The User Interface of OntoSELF-v1](image1)

![Figure 6-4 The User Interface for OntoSELF-v2 at the start of execution](image2)
The Ontology Selection group contains all necessary tasks of selecting one or multiple ontology files. Users can click the Browse button to open a file dialog to select one OWL or GML file. Multiple ontologies can also be selected by checking the Multiple Ontologies checkbox. The following snapshot shown in Figure 6-5 The User Interface when the File Dialog is opened by clicking the Browse button is the effect of clicking the Browse button.

![Figure 6-5 The User Interface when the File Dialog is opened by clicking the Browse button](image)

The two Label selection radio buttons in Ontology Selection group allow users to choose either the class name or the class id in the OWL file to be the label of each node. If the class id is a long URI, we only keep the part after #. The “Label on top” text area allows users to type the number of nodes for which to display labels. A sorting algorithm orders the nodes based on their size as determined by the selection made by the user in the Size Function group and the top number is taken on that ordered list. For example, if 50 is typed in, then the nodes the top 50 nodes in the ordered list will have their labels displayed.

The Package Selection group allows users to select one of the four packages to visualize the ontology. These packages are described below in section 6.3.2. When pressed, the Open File button starts the parsing and processing of the selected OWL or
GML file and then activates the rest of the user interface. The Execution Type group allows users to select either execute the program in standalone mode or parallel mode.

The following snapshot in Figure 6-6 The User Interface displayed after the user selected a umls-semantic-network.owl file and package 1 for visualization, and clicked the Open File button is taken after the Open File button has been clicked and the pizza.owl file has been parsed.

Figure 6-6 The User Interface displayed after the user selected a umls-semantic-network.owl file and package 1 for visualization, and clicked the Open File button

In the Relationship Selection group, the Unselected Relationships text area stores all relationships that are deselected by the user from the Hierarchical Relationship and the User-defined Relationship text areas. The Hierarchical Relationships text area stores the set of hierarchical relationships and always includes superClassOf relationship. The User-defined Relationships text area stores all user-defined relationships selected by the user. Since this diagram shows the first package which is used to visualize only hierarchical relationships in 3D, the user-defined relationship text area is still inactivated. The hierarchical relationships and the unselected relationships text area are activated. The unselected relationships show those relationships that exist in the pizza ontology. Users can select any relationships in Unselected Relationships text area and click the right arrow button to have them included in the category of hierarchical relationships. The
relationships listed in the hierarchical relationship text area are the ones that then are used in determining the hierarchical node metrics such as level, interval rank, IC value and extent that may be used in the weighting function and also hierarchical metrics for filtering nodes. Any of the selected relationships can also be deselected or removed from the hierarchical relationship text area by clicking the left arrow button. Only the superclass relationship may not be removed since that serves as the basic structuring mechanism for any ontology.

The **Weighting Function** group allows users to select a hierarchical metric to determine the 3rd dimension of each node in the 3d ontology visualization layout. The Size Function group provides 9 metrics that can be used to determine a node’s size in the visualization. The Filtering Criteria group includes all 13 hierarchical and social network metrics to use in filtering nodes. The combined checkbox indicates that if multiple filters are selected they should be combined using “and” filtering, i.e., each specified criterion must be met for the node to be included in the visualization. The Clustering group includes three options, 1) Structural Clustering on Filtered Ontology; 2) Structural Clustering on Whole Ontology; and 3) Semantic Clustering on Whole Ontology. The first option uses structural clustering over the ontology after filtering. The second option also uses structural clustering but over the whole ontology before filtering. The last option uses semantic clustering over the whole ontology before filtering. Semantic clustering requires the user specify at what level in the ontology hierarchy to begin the clustering and what criteria to use to select the final cluster for a node that belongs to multiple clusters. The details of both clustering algorithms are provided in section 6.3.5.

### 6.3 Implementation of SNA Processing Features

First the two visualization software tools VTK and JUNG used in OntoSELF are briefly introduced. VTK (Visualization Tool Kit) has already been used in OntoSELF-v1 to provide its 3D visualization capabilities. JUNG (Java Universal Graph and Network) is a powerful and reliable API supporting flexible 2D network visualization. The four visualization packages under which different analytical tasks are supported are discussed. The new customized OWL to GML parser is described but some implementation details
can be found in the appendix. Finally, the social network analysis and clustering methods provided by OntoSELF-v2 are presented.

6.3.1 Visualization Software Overview

6.3.1.1 VTK - Visualization Tool Kit

VTK (http://www.vtk.org/) performs the processing of visualization by various components called VTK pipelines in a sequential manner. Each pipeline performs some algorithmic operations on the data flowing into it and outputs some new format of data after processing. Those data are called data objects, and the process components are called process objects. The following is a typical pipelining in VTK:

```
Source
  ↓
Filter
  ↓
Actor
  ↓
Renderer
  ↓
Window
```

*Figure 6-7 A Typical VTK pipeline*

In VTK, there are two types of sources, input sources and independent sources. The input sources are used to read some original data from a source file. But those independent sources, like sphere source and cube source, are used to define the information representing a sphere or a cube. The file reader used in OntoSELF is vtkGMLReader, which is specifically used to read GML files. The independent source used is vtkSphereSource. In the VTK pipeline used for OntoSELF-v1 visualization, the
vtkGMLReader reads a GML file and passes the output data structure to the layout classes. When the vtkProgrammableGlyphFilter class is created, it needs the data structure passed from the vtkGMLReader class, but it also needs another source to define how to represent those raw data in visualization. The independent source, vtkSphereSource class informs the filter to represent each node data as a sphere.

Filters are used to receive data from some process object preceding it in the pipeline, to process the data in some way, and then to produce the modified data as output to be used by other process objects. Some filters can even get data from two sources and combine them together as a single output. For example, the vtkProgrammableGlyphFilter, previously mentioned can accept data from both an input source and an independent source. Another process object called the mapper has the same functionality as filters, but differs on the kind of output object to which data is passed. A mapper serves as a transmitter of data to an actor object. A filter object, however, never passes data to an actor but only to another filter or a mapper.

In a VTK pipeline, the actor object plays a very important role as process objects that adjust and control the appearance properties of any physical objects used to visualize the data rendered on the screen. The word actor is used to portray the work a human actor does in a scene. A primary task for an actor is decoration. Actors are often responsible for controlling transparency and color mapping. Renderers and windows are the process objects that actually render the whole visualization on the screen. All actors must be added to a rendering window before they can appear on the screen.

For the details of implementation using VTK, please see appendix I.

6.3.1.2 JUNG - Java Universal Network/Graph Framework

JUNG is a software framework providing a common and extensible language for various tasks of network/graph operations, such as modeling, analysis and visualization. Since it is developed in Java and delivered as a Java API, it is very easy to use as a third-party Java library.

In the most recent version JUNG2, there are two major libraries, JUNG and JUNG3D. The former one is the most matured, powerful and reliable library which contains many useful graph manipulation data structures and algorithms and some
flexible and easy-to-use 2D visualization packages. However, the JUNG3D is the newly developed library; its 3D visualization layouts are not easy to use and it provides very limited development capability for 3D visualization. Since OntoSELF-v1 provides its own powerful 3D visualization system based on VTK, the JUNG library is used for looking at ontologies as a network in 2D graphs and to provide some of the built-in data structures and algorithms for network processing. The JUNG library provides four package categories:

(1) algorithm - This package contains all graph-related manipulation and visualization layout algorithms. For example, the shortestpath package contains all graph algorithms related to calculation of shortest path. The layout package contains all 2D visualization layout algorithms. The cluster package contains some clustering algorithms, such as edge betweenness clustering and voltage clustering. Unfortunately, none of those clustering algorithms fit our needs. The WeakComponentGraphClusterer and WeakComponentVertexClusterer are used to detect unconnected subgraphs in any graph. The BicomponentClusterer class is used to find all biconnected components of an undirected graph. Here a biconnected graph is a graph in which at least two vertices have to be removed to disconnect it. It is a fast algorithm, but its simple implementation is not capable of solving social network clustering problem. The EdgeBetweennessClusterer detects clusters in a social network based on edge betweenness (Newman et al. 2001). However, this algorithm is only capable of removing one edge with the highest betweenness for each iteration. This algorithm requires the user to specify the number of clusters at which to stop the processing.

(2) graph – This package contains all built-in graph data structures. For example, the one OntoSELF uses is SparseMultigraph. There are also some other useful graph data structures in this package including DirectedSparseGraph, UndirectedSparseGraph, SparseTree, DirectedSparseMultigraph, and UndirectedSparseMultigraph. Additionally, some graph selection events and some utility classes are also in this package. The initial implementation of the enhancements to OntoSELF used the DirectedSparseGraph class to serve as the
data structure storing vertex and edge attributes. This design was modified, however to use a hashtable to store this data since the DirectedSparseGraph required too much memory and was unable to store and process the whole GO ontology which contains more than 20,000 nodes. The SparseMultigraph is used only for visualization purpose. The reason for choosing this class instead of others is its extensibility for future work since it can use both directed and undirected edges.

(3) io – This package involves file I/O operations. For example, there are pajekNetReader and pajekNetWriter to manipulate pajekNet-related files. Pajek (http://vlado.fmf.uni-lj.si/pub/networks/pajek/) is a program for large network analysis, and JUNG provides a related reader specifically for Pajek.net files. In OntoSELF, this package is not used.

(4) visualization – This package contains numerous classes related to 2D visualization. It is the richest, the most powerful and the most commonly used package among the others.

The details of the 2D visualization implementation using JUNG framework is described in Appendix I.

6.3.2 Visualization Packages in OntoSELF

The 3D visualization of OntoSELF-v1 solely uses the superclass/subclass relationship as the basis of the edges in its presentations. This relationship is hierarchical and permits multiple parents. It is used in calculating most of the metrics on the nodes and in particular: node level, information content, interval rank and extent. Any one of these can be selected as the weighting function to determine the 3rd dimension value for nodes in the layout algorithm used to display the ontology.

As previously described in section 5.2, providing the ability to visualize both the two kinds of relationships, the original hierarchical and the added user-defined, required considering the various ways these two major categories of relationships may interact in the visualization results. The result is that OntoSELF-v1 provides four packages for
visualizing an ontology depending on the relationship types the user requires to be visualized.

1. **Only hierarchical relationships selected in 3D (H3D).** In this package, subclass relationships are used by default in producing the 3D visualization. In some ontologies such as the GO, however, the “part of” relationship is also treated as a hierarchical relationship. In order to accommodate other user-defined relationships such as “part of” which are considered hierarchical, the user can select them for inclusion in the hierarchical relationship category. The original OntoSELF solely used this package but without the capability to include other user-defined relationships in the hierarchical category. This thesis research adds to the package the enhancement of the four centrality measures and the rank prestige measure that can be used as filtering parameters on nodes. Additionally, the 9 size functions can be selected to determine the node size. The semantic clustering and the structural clustering are also added to enhance the perception of visualization and also to test their usability on an ontology with only hierarchical relationships. The layout of this package is basically that of the original OntoSELF since it looks at an ontology as a hierarchical structure for 3D visualization.

2. **Only user-defined relationships selected for 2D visualization (U2D).** Looking at only the user-defined relationships of an ontology, one can view the ontology as a social network. This view is also the initial motivation for integrating social network analysis methods into OntoSELF. The user can select which user-defined relationships to incorporate into the visualization.

   Like in the first package, the node filtering feature is also provided but only the SNA measures may be used since the hierarchical filtering criteria are not available with a social network view. Clustering for both semantic and structural abstraction are supported to enhance the visualization perception.

   In addition to all those features, unlike the H3D package, users are not allowed to set the hierarchical relationship combination, since this package is purely for social network layout with user-defined relationships. However, the users are allowed to deselect user-defined relationships beyond their interests from the list of all user-defined relationships where all of them are set to be selected as default. Additionally, the weighting functions
are not activated. The size function group only activates the four SNA centrality and prestige functions.

3. Both hierarchical and user-defined relationships in 3D (HU3D). Relationships from both major categories are to be included in the visualization. The user may individually select which relationships to categorize as hierarchical and which to categorize as user-defined and non-hierarchical. OntoSELF is an ontology visualization software system specifically for intensional ontologies. The hierarchical superclass/subclass relationship is the backbone of the structure for intensional ontologies; therefore, to visualize an intensional ontology, its inherent hierarchical structure is still the main structure to visualize. This package is like the first except that in addition to the ontological structure imposed by the hierarchical relationships, the user-defined relationships are also visible. As in the first package, all previous filtering criteria and the four centrality and prestige measures will be used to filter on nodes. Both semantic and structural clustering methods may be applied. For the semantic clustering, the users are allowed to select one of the 11 clustering criteria whose details are presented in the section of Clustering for Abstraction.

Additionally, since both hierarchical and user-defined relationships are included in this package, users are allowed to set the combination of relationships for both categories. But the superclass/subclass relationship is always selected and cannot be deselected in the hierarchical relationship group; otherwise, it becomes a pure social network with user-defined relationships only which is already provided in the H2D package.

4. Both hierarchical and user-defined relationships in 2D (HU2D). This package eliminates the ontological structure imposed by the hierarchical relationship but still includes them as a relationships connecting classes, i.e., they have the same status as user-defined nonhierarchical relationships. This package provides users with another view of the ontology where all relationships are viewed equivalently. Like package 3, users can set the hierarchical and user-defined relationship categories, use all filtering criteria, size function selection and both semantic and structural clustering. But the weighting functions are not activated since the visualization of the ontology does not use the third dimension provided in packages 1 and 3. In the visualization, the two relationship categories are distinguished by edge color. Hierarchical relationships are colored by blue and the user-defined ones are in red.
6.3.3 OWL to GML Parser

For OntoSELF-v1, a customized OWL parser was developed to only extract the Class entities and the SubClassOf relationships from an OWL file since the objective of that research was only the visualization of the hierarchical structure of an ontology based on the SubClassOf relationship. However, in this current thesis research, all user-defined relationships as well as the hierarchical structure of the subClassOf relationship that underlies an ontology definition are to be included. The new OWL parser must handle a more complicated and flexible subset of the OWL language. The term subset is used since the parser, for example, ignores all attributes (those only having simple datatypes as values) defined for a class and equivalent classes and all instances of classes.

All anonymous classes that restrict property values are also ignored, but those properties are added to their concrete subclasses. Function properties, object properties and transitive properties are processed as user defined relationships. As discussed in section 5.4, OntoSELF-v2 considers all relationship properties to be invertible. For example, the property “measures” has its inverse “measured by”. With this assumption, all edges in the ontology are treated as undirected and inverse properties specified in OWL can be ignored and undirected edges can be used in calculation of the SNA metrics.

As discussed in section 5.3, memory limitations and slow execution speed of two standard OWL parsers necessitated the development of a customized OWL parser to handle parsing user defined properties. Parsing these properties is more complex than only parsing for the subClassOf property and required the new parser to use parser stacks. The OWLtoGMLParser class is customized to take an OWL file as an input and produce a basic GML file as output. The details of its implementation are discussed in Appendix II.

6.3.4 Social Network Analysis Implementation

Four centrality and prestige measurements have been implemented and integrated into OntoSELF. They are closeness centrality, betweenness centrality, information centrality and rank prestige. They are used for filtering and semantic clustering of nodes.
representing classes in the visualization. A method exists for each measure in the class called SNAnalysis. During the implementation and testing phases, many bottleneck problems were encountered and solved. Some of these have been discussed in section 5.4 but the more technical ones related to implementation decisions are described here.

In the initial implementation, all four centrality metrics were calculated based on a matrix representation for the data. However, an out of memory error occurred when testing the whole system on the cell.owl sample file which contained only 1,500 nodes. This implementation approach was not sufficient to handle the large size ontologies for which OntoSELF-v1 was designed. The implementation needed to be more memory-efficient. After some modifications in the datatypes for the matrix entries did not solve the problem, next the Graph class from the graph API of the JUNG framework was used to replace all matrices except the sociomatrix created for calculating information centrality. However, the use of the Graph class representation still used too much memory to handle large-sized ontologies.

All arrays and hashtables were changed to pointers pointing to the corresponding arrays and hashtables in ProcessGML class, the core class of OntoSELF-v1. A limitation of this approach is that future extensions to SNAnalysis cannot use pointers to other arrays or hashtables but should only modify the data in the existing ProcessGML arrays and hashtables. The UML diagram of the final version of the SNAnalysis class is shown below:
Figure 6-8  Class definition of SNAnalysis class

This class calculates the four centrality and prestige measurements and performs semantic and structural clustering. Any other social network analysis methods can be easily added to this class based on the hashtables stored in it. In the current version, the whole ontology is represented by four hashtables. The hierarchicalGraph table contains all hierarchical relationships including any user-defined hierarchical ones. However, the multiple relationships in one graph might cause multiple edges between two vertices, so the hierarchicalWeights table is created to store the weight of each edge in hierarchicalGraph table. The SNGraph and SNWeights have the similar goal but only stores all selected user-defined relationships not considered as hierarchical by the user. In package 4, all four tables are used, but in package 2 which only handles user-defined relationships, the hierarchicalGraph and hierarchicalWeights to two empty tables.

The four float arrays are used to store the four SNA measures. The vertices list stores all node ids in the graph. Those node ids may differ from node’s index in vertices, because they are defined in GML files and may not be sequential. The optimalCluster is
used for semantic clustering and structural clustering. The details of the clustering algorithms for semantic and structural abstraction and its motivations are discussed in the following section 6.3.5. The optimalCluster is a list structure for storing the optimal division and contains all clusters. Each cluster is also a list containing the node ids of all nodes in that cluster. Therefore, the whole structure for optimalCluster is a list of a list of Integers. It is developed based on the optimal quality measure in the whole dendrogram created by the clustering algorithm. The clusteringCriteria is used for the semantic clustering only. It points to one of the structural metrics arrays and SNA metrics arrays to assist the semantic clustering. The details are discussed in section 6.3.5.

The filteredList is pointing to the filteredNodesList in ProcessGML class. After the nodesList is filtered in ProcessGML class, the filteredNodesList contains all nodes which fit all the filtering criteria. The danglingNodesList is used to store if each node in vertices is dangling in the whole ontology structure. The danglingFilteredNodesList is used to store if each node in filteredList is dangling in the filtered ontology structure. When clustering the ontology, users can either choose to structurally cluster on filtered graph or on the whole graph and all dangling nodes must be removed before processing.

The closeness centrality measure considers the importance of a node as its closeness or distance to all the other nodes in the graph. The formula is very easy to implement. It is $C_c(n_i) = \left[ \sum_{j=0}^{N} d(n_i, n_j) \right]^{-1}$, where the subscript C is for closeness, N is the total number of nodes, and $d(n_i, n_j)$ is the distance from actor (node) i to actor (node) j. The distance is the sum of the number of edges on the path from actor (node) i to actor (node) j. The first step in its calculation is to determine the shortest distance to all the other nodes for $n_i$. The design issues and problems for implementing the centrality measures have been discussed in above and in section 5.4.

The closeness centrality measure was implemented as defined in (Wasserman and Faust, 2007) using the above formula $C_c$; however, its approach to handling unconnectivity problem was not satisfactory for providing the user with an overall understanding of an ontology when looking at only the user-defined relationships. This design issue has been discussed in section 5.4. The unconnectivity problem only happens in package 2 which handles only user-defined relationships. In all the other 3 packages,
the subclass/superclass relationship always connects all nodes, because there is always a
common ancestor in any ontology, named owl:thing, which corresponds to the Object
class in java. The standard closeness centrality measure is used in all the other packages
except package 2.

The betweenness centrality measure takes into consideration the level of importance
one actor might have in the interactions between two nonadjacent actors. The formula for
calculating it is \( C_B(n_i) = \sum_{j<n_k} g_{ij}(n_k) / g_{ij} \), where the subscript \( B \) represents betweenness
and \( g_{ij}(n_k) \) is the number of geodesics (shortest paths) linking actor (node) \( i \) to actor
(node) \( j \) through actor \( k \) and \( g_{ij} \) is the number of shortest paths between node \( i \) and node \( j \). Since the straightforward implementation is time inefficient, Brandes’ Faster
Betweenness Centrality Algorithm (Brandes 2001) was implemented. The following is
the original Brandes’ algorithm:
**Algorithm 1: Betweenness centrality in unweighted graphs**

\[
C_B[v] \leftarrow 0, \quad v \in V; \\
\text{for } s \in V \text{ do} \\
\quad S \leftarrow \text{empty stack}; \\
\quad P[w] \leftarrow \text{empty list}, \quad w \in V; \\
\quad \sigma[t] \leftarrow 0, \quad t \in V; \quad \sigma[s] \leftarrow 1; \\
\quad d[t] \leftarrow -1, \quad t \in V; \quad d[s] \leftarrow 0; \\
\quad Q \leftarrow \text{empty queue}; \\
\quad \text{enqueue } s \rightarrow Q; \\
\quad \text{while } Q \text{ not empty do} \\
\quad \quad \text{dequeue } v \leftarrow Q; \\
\quad \quad \text{push } v \rightarrow S; \\
\quad \quad \text{foreach neighbor } w \text{ of } v \text{ do} \\
\quad \quad \quad \text{if } d[w] < 0 \text{ then} \\
\quad \quad \quad \quad \text{enqueue } w \rightarrow Q; \\
\quad \quad \quad \quad d[w] \leftarrow d[v] + 1; \\
\quad \quad \quad \text{end} \\
\quad \quad \quad \text{if } d[w] = d[v] + 1 \text{ then} \\
\quad \quad \quad \quad \sigma[w] \leftarrow \sigma[w] + \sigma[v]; \\
\quad \quad \quad \quad \text{append } v \rightarrow P[w]; \\
\quad \quad \text{end} \\
\quad \text{end} \\
\delta[v] \leftarrow 0, \quad v \in V; \\
\text{// } S \text{ returns vertices in order of non-increasing distance from } s \\
\text{while } S \text{ not empty do} \\
\quad \text{pop } w \leftarrow S; \\
\quad \text{for } v \in P[w] \text{ do} \\
\quad \quad \delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w]); \\
\quad \quad \text{if } w \neq s \text{ then } C_B[w] \leftarrow C_B[w] + \delta[w]; \\
\text{end} \\
\end{align*}

**Figure 6-9 Betweenness Centrality Algorithm (Brandes 2001)**

The above algorithm is used for a directed social network. Since all edges are considered undirected, some modifications were made to this algorithm. The distances between nodes are calculated by taking the number of relationships bounded on the edge. Inside this algorithm, \( C_B \) represents the betweenness centrality array. \( V \) is the list of all vertices. For each vertex, the initialization part creates several temporary structures to store intermediate values. It first creates an empty stack and an empty queue. For each vertex, create an empty list to form a list of lists. The two arrays \( \sigma \) and \( d \) are for
calculating shortest paths between nodes. The current vertex is placed in the queue. Within the queue not empty loop, the calculation of shortest paths from the current node to all the others is determined. Those neighbors w of v are found based on undirected edges. The S not empty loop calculates the betweenness centrality measure for the current vertex.

The information centrality measure is the generalized version of the betweenness centrality measure. It considers all paths rather than only the shortest paths. However, it is not as popular as the betweenness centrality measure. The algorithm given in (Wasserman and Faust, 2007) is easy to implement and has already been described in section 3.2.1. The implementation issues have been discussed in section 5.4. OntoSELF-v2 provides an error message stating that this measure cannot be calculated if either of the two conditions as described in section 5.4 occurs.

The rank prestige measure is one of the frequently used prestige measures and is considered the most powerful one. It is recursive in the sense that it incorporates the rank prestige of the individual actors (nodes) that are linked to the actor (node) for which rank prestige is being determined. Section 3.2.2 gives the equation for the rank prestige for actor (node) i. As previously discussed, this equation produces a system of g linear equations with g unknowns. The Faster Katz Status Score (Foster et al. 2001) algorithm is used in OntoSELF-v2 to calculate the rank prestige measure.

6.3.5 Clustering for Abstraction

As explained in section 5.5, OntoSELF provides semantic abstraction based on the superclass/subclass relationship. This relationship serves as the backbone of ontologies. It starts from the most abstract class to every specific concrete class by using inheritance. In each multi-inheritance tree, the more specific classes can be viewed as the instances of those most abstract classes. Therefore, a class at the upper levels can serve to define a semantic root for semantic abstraction cluster and its descendants are assigned into their clusters.

The users are permitted to choose from which level the semantic classes are to be defined. These semantic classes serve as the roots of the clusters. Therefore, the number of clusters generated is equal to the number of classes at that chosen level. However, this
capability involves some restrictions and presented some problems that needed to be solved. The first problem is the multi-inheritance problem. In all ontologies, classes are allowed to inherit multiple parent classes. In this situation, once a subclass has multiple parent classes and all of them belong to different clusters, a resolution method is needed to determine which cluster the subclass should be assigned to. Currently we simply equate this problem to deciding which cluster is more appropriate for this class to belong to, i.e., which root of each cluster is more important in the whole structure. The user is given the flexibility of choosing the evaluation criterion for importance in order to resolve which cluster to assign the class to. The user may select any of the 11 measures: IC value, Number of descendants, Hub value, Interval rank, Child Count, Node Height, Extent Value, Closeness centrality measure, Betweenness centrality measure, Information centrality measure, and Rank Prestige. If a tie exists, then one is randomly selected.

After the whole ontology has been clustered, colors are assigned to each cluster. The current design uses 64 colors for representing at most 64 clusters. The colors can be mapped to a color lookup table, where each integer from 0 to 63 maps to a unique color and two adjacent integers map to two similar colors. Therefore, if there are only several clusters, say 8, the system must scatter the cluster ids to make their corresponding colors as distinguishable as possible to human eyes. First an integer between 0 and 63 is randomly selected to be the first cluster’s id. Then the ids of all the other clusters become are floor (64 / number of clusters) offset to the previous cluster’s id. Inside each cluster, all nodes are colored by the color chosen for the cluster.

All classes at levels higher in the ontology than the level selected for semantic abstraction are classified to the same cluster since it provides the user with the ability to easily locate in the visualization those classes considered most abstract. The capability of varying the level of semantic abstraction can provide users with numerous views that help them better understand the overall ontology structure.

Because the user has the capability of specifying the level of the semantic abstraction, a situation can arise that a level deep in the ontology could be selected that contains, for example, 300 classes, i.e., therefore 300 clusters. The human perception for this many clusters and assigned colors might not be able to distinguish them. For the purpose of this implementation and also to provide better perception, 64 colors are to be
distinguished which implies that only levels with 64 or fewer classes are allowed for user selection. If there are more than 64 nodes at a level the user chooses, the system provides a message to allow the user to enter a new level value.

Semantic abstraction is useful for presenting the organization of the ontology with emphasis on the hierarchical structure produced by its built-in is-a or the superclass/subclass relationships. In social networks, however, different methods of *structural abstraction* have been used. In the OntoVis system (Shen et al., 2006), the structural abstraction is very limited. As previously described, it consists of “cleaning up” operations that directly removing one-degree nodes and duplicate paths. The approach taken in SocialAction (Perer and Schneiderman, 2006) uses a popular approach that identifies “community structure” in networks (Newman 2004). This approach is the basis for the structural abstraction provided in OntoSELF.

For community structure approach, each clustered community often has the property of intense intra-connectivity and loose inter-connectivity. However, to divide a network into communities can be a difficult task. For example, the number of communities in a network is hard to determine. The number of nodes in each community is also hard to tell. However, despite of all those problems, there are many successful clustering algorithms developed and applied to detect communities in social network. Basically, there are two major categories of approaches:

1) Hierarchical Clustering can be further divided into bottom-up and top-down approaches. The bottom-up approach divides all n nodes into n clusters and at the next level, it merge two of all clusters into one. At the top level, there is only one cluster as a whole network. Then the user has to determine at which level to cut the clustering hierarchy. The top-down approach basically proceeds in a similar manner but in the opposite direction, starting out with one cluster and then breaking it down into two or more and then working on the individual clusters to see if they need broken down into multiple clusters.

2) Evaluation clustering uses some edge evaluation to cluster a network by iteratively removing edges. The most successful algorithm is Girvan-Newman algorithm (Girvan and Newman 2001). However, the disadvantage is the large time complexity, approximately O(n^3).
The faster Newman algorithm (Newman 2004) has been tested thousands of times faster than any previous algorithms. Its worst case time complexity is $O((m+n)n)$, which is almost $O(n^2)$, where $m$ is the number of edges and $n$ is the number of nodes in a network. Although classified as a hierarchical bottom-up approach, this algorithm also combines a very excellent evaluation approach to determine at which level to cut the clustering hierarchy. To evaluate a particular division of a network, it employs a quality function $Q = \sum_i (e_{ii} - a_i^2)$ where $a_i = \sum_j e_{ij}$. Here $e_{ij}$ is the fraction of edges flowing from any nodes in cluster $i$ to those in cluster $j$. The $e_{ij}$ is referred to as the inter-connectivity and the $e_{ii}$ as the intra-connectivity. The whole quality function $Q$ is the fraction of edges within a community minus the expected value of the quality if the edges fall at random in the network without considering the community structure; therefore, the higher $Q$, the better the division. Since it follows the hierarchical bottom-up approach, it assigns each node into a different cluster, producing initially $n$ clusters. At the next level, it pairs each two clusters with at least one edge connecting one node in each cluster and calculates the change in $Q$, which is $\Delta Q = e_{ij} + e_{ji} - 2a_i a_j$. It only merges the pair with the highest change in $Q$ at one level. Finally after $n-1$ iterations, the best cut in the clustering hierarchy is produced. The simplified description of the algorithm is as follows:

Step 1. Assign each node into one of the $n$ clusters.
Step 2. Merge each pair of connected clusters. Calculates the change in $Q$.
Step 3. Merge the pair of clusters with the highest change in $Q$, and update $Q$.
Step 4. Continue step 2 and 3 until all clusters have been merged into one cluster.

Find the level having the largest $Q$ and cut the clustering hierarchy, or dendrogram at that level.

The following graph shows the clustering hierarchy, or dendrogram of finding community structure of Zachary’s Karate Club sample data.
Figure 6-10  Example clustering using Newman Algorithm (Newman 2004)

Since this algorithm is the fastest and comparably accurate for detecting community structure, it was used to provide structural abstraction capability in the OntoSELF system. It uses a greedy approach, however, to merge clusters in each level without considering any global perspective. According to experiments with test cases (see section 6.3), this algorithm works well for looking at either the pure social network structure or the hierarchical structure or both of an ontology, i.e., ignoring the hierarchical subClassOf relationship. It is suggested that semantic abstraction be used when the hierarchical structure is the user’s focus of interest and that structural abstraction be used when user-defined relationships are the user’s focus.

7 Evaluation

The first version of OntoSELF, OntoSELF-v1, mainly focused on displaying the hierarchical structure of an ontology based on its superclass/subclass relationship. No visual cues were used to enhance the visualization perception. The addition of user-defined relationships and social network analysis metrics contribute to the need for visual cues in order for the user to better understand the representation of the added information. They also can assist the user in gaining a high level comprehension of the overall hierarchical structure of the ontology by making the significant concepts readily apparent to the user.
An evaluation on the usability of the added visual cues in new version OntoSELF-v2 is given in section 7.1. In section 7.2, the four centrality and prestige metrics on sample ontologies are compared and also combined with those hierarchical metrics in the first version to evaluate their usability in various scenarios. Additionally, the social network analysis metrics integrated into the filtering criteria to create a new importance filtering capability are demonstrated. In section 7.3, the two clustering algorithms are compared and evaluated as to their usability in abstraction over an ontology structure. The visualization over both hierarchical and user-defined relationships presents several distinct characteristics and has been implemented by four visualization packages to provide the user with the flexibility to visualize the ontology with different types of relationships either in 2D or 3D. Section 7.4 evaluates the usability of those packages.

Three sample ontologies with different sizes and domain knowledge are used to evaluate OntoSELF:

- UMLS Semantic Network Ontology
- NASA JPL SWEET Ontology
- Gene Ontology

These three ontologies have sizes ranging from small to medium to large. The UMLS Semantic Network Ontology has only 134 concepts and 96 user-defined relationships. The NASA Sweet JPL Ontology has 1,567 concepts and 109 user-defined relationships. The Gene Ontology has 22,622 concepts and only 2 relationships including superclass/subclass relationship. The NASA Sweet JPL Ontology and the Gene Ontology were used in the evaluation of OntoSELF-v1. The Unified Medical Language System (UMLS) Semantic Network (SN) Ontology (http://semanticnetwork.nlm.nih.gov/) is one of the three UMLS Knowledge Sources. The other two are UMLS Metathesaurus and UMLS Knowledge Source Server (UMLSKS). The UMLSKS is a set of software tools to help users analyze resources in the whole UMLS project. The UMLS Metathesaurus has an enormous size with more than 700,000 concepts. In order to help new users understand the vast knowledge content of the Metathesaurus, the UMLS project group (http://www.nlm.nih.gov/research/umls/) developed the UMLS SN which has higher level abstraction of all concepts in UMLS Metathesaurus. However, even though the UMLS SN is a smaller size ontology is still challenging for users to understand. There are
many ways of abstraction and partitioning of the UMLS SN ontology. The new abstraction capabilities of OntoSELF are compared with those methods proposed in (Chen, 2002).

The NASA JPL Semantic Web for Earth and Environmental Terminology (SWEET) ontology (http://www.mindswap.org/) was developed to provide a common semantic vocabulary for the earth science domain area. It is composed of 12 sub-ontologies, each of which has its specialized domain needs. The Phenomena sub-ontology was used in the first version of OntoSELF for qualitative and topology task evaluations. This sub-ontology serves as an ideal medium-sized ontology to test the usability of all new features as compared with the first version. Additionally, the 109 user-defined relationships make it a good test ontology for the four social network centrality and prestige measures.

The Gene Ontology (GO) (http://www.geneontology.org/) is a controlled vocabulary for all attributes of gene and gene product for any organism. It provides the standard definitions for gene terms used in annotation databases for genes and gene products and avoids the consistency problems caused by various ontologies developed by different organizations for biomedical domain area. It is composed of three sub-ontologies, biological processes, cellular components and molecular functions. Only the standard superclass/subclass relationship and the part-of relationship are defined. The use of GO in this evaluation is for comparison purpose to OntoSELF-v1 and to evaluate the usability of the hierarchical relationship selection feature.

### 7.1 Evaluation on Visual Cues Enhancement

Two visual cues color and size have been used on both nodes and edges. The node size and color are added to all four packages. Section 7.1.1 provides some visualization using the node size to represent one of the 9 metrics. The node color discussed in section 7.1.2 is used to indicate the cluster each node belongs to. Nodes in the same cluster are colored the same. Section 7.1.3 describes how color and size have been used in combination. Section 7.1.4 discusses the edge thickness which is another visual cue used to represent
the number of relationships merged into each edge. This cue is provided since ontology is transformed into 1-mode and 1-plex network, and 1 edge between two nodes in the visualization, therefore, can represent the combination of multiple user-defined relationships between concepts in the ontology. Section 7.1.4 describes the use of edge color to distinguish the two categories of relationships, hierarchical and user-defined relationships.

7.1.1 Node Size

To evaluate the usability of node size in OntoSELF, package 1 using only hierarchical relationships is used to compare the difference between adding size in the current version and the first version without any visual cues. In the real world, objects with a bigger size easily capture the attention of humans compared with smaller-sized objects. The application of size on nodes in OntoSELF can dramatically increase the visualization perception. Node size is used to represent each node’s significance with respect to the selected metric used to determine size. The larger the node is within the visualization, the more significant the node and, thus, through the visual cue of size, users can more easily identify those important concepts in an ontology for the purpose of high level comprehension.

The following examples use the rank prestige metric and the level value for the layout weighting function in their comparison of the visualization produced by OntoSELF-v1 and OntoSELF-v2. Since the effects of comparison using node size determined by any of the other SNA metrics and the first version visualization using equal node size will be similar, we only show the scenario using rank prestige for comparison.
Figure 7-1  3D Visualization of OntoSELF-v1 on UMLS SN Ontology with only superclass/subclass relationship and labeling on the top 15 nodes with the highest rank prestige measures. All nodes have equal sizes.
Figure 7-2  3D Visualization of OntoSELF-v2 on UMLS SN Ontology with only superclass/subclass relationship and labeling on the top 15 nodes with the highest rank prestige measures which are also used to determine the node size.

The above two visualizations label 15 nodes out of 135 nodes. By just looking at the labeled nodes, users cannot tell anything significantly different about the nodes in the figure 7-1 except for their level within the ontology hierarchy. However, from figure 7-2, users can easily identify the node with the highest rank prestige measure, Conceptual Entity. This concept has the most children, several of which also have many of their own children; therefore, those children of Conceptual Entity are also considered important concepts.
After identifying this most important concept in UMLS SN ontology, the user can use the node name filter to focus on the subtree of Conceptual Entity to examine the lower level structure of the ontology from the context of the Conceptual Entity. Additionally, there are less nodes overlappings in figure 7-2 than that in figure 7-1 since most concepts have very small rank prestige measure and, therefore, their nodes occupy much less space.

As an example to see how node size works in a larger ontology, the NASA Sweet JPL ontology is used. It contains 1,403 concepts connected to at least one other node and 1,489 superclass/subclass relationships. The visualization produced by the two versions of OntoSELF is shown below:

![Figure 7-3 3D Visualization of OntoSELF-v1 on NASA Sweet JPL Ontology with only superclass/subclass relationship and labels on the top 25 nodes with the highest rank prestige measures. All nodes have equal sizes.](image)

67
An examination of the two figures 7-3 and 7-4 show that the node overlapping problem is not as serious as the visualization on UMLS SN ontology, but it is much harder for users to distinguish nodes in figure 7-3 than in figure 7-1. However, in figure 7-4, it is very easy for users to notice those most important concepts. For example, the concepts Animal, EcologicalProcess, CoordinateSystem, and KnowledgeDomain all have comparably the largest node sizes. The dummy root, in the SWEET ontology becomes the most important concepts in figure 7-4 since has most children and many of its children also have high rank prestige.
The following two visualizations are on the Gene Ontology which has 21,125 concepts connected to at least one other concept and 29,070 superclass/subclass relationships.

Figure 7-5 3D Visualization of OntoSELF-v1 on Gene Ontology with only superclass/subclass relationship and labels on top 50 nodes with the highest rank prestige measures. All nodes have equal size.
Due to the large number of nodes and edges in the GO, both visualizations have very serious node and label overlappings and edge crossing. In figure 7-6, however, the user can easily identify the concept Protein_complex as the most important as determined by rank prestige measure even in a global view of the Gene Ontology with more than 20,000 nodes.

Using this same visualization, the user can zoom in (shown in figure 7-7) to explore the region around Protein_complex in order to gain a better understanding of the other
significant concepts in this area and their relative importance to the identified Protein_complex concept.

Figure 7-7 The area around Protein_complex after zooming in from Figure 7-6

If the user wants to better understand the local context of the Protein_complex concept and the importance of its descendants, OntoSELF can be used to specifically select the concept Protein_complex as the node name value in the filtering options and again use rank prestige to determine the node size. Figure 7-8 shows the results of this local context visualization from the concept Protein_complex with the top 25 concepts labeled.
Figure 7-8 The local context of Protein_complex after selecting it as the main node in the filtering options based on its identification Figure 7-6 and using rank prestige to determine node size and labeling on the top 25 concepts.

7.1.2 Node Color

The visual cue color is added to nodes in all 4 visualization packages, and is also added to edges in visualization package 4 (HU2D). This section compares the use of node color in OntoSELF-v2 with no node color available in OntoSELF-v1.

Node color is used to represent the cluster to which each node belongs; therefore, all nodes in the same cluster have the same color. The maximum number of clusters allowed in each visualization is 64. This number is chosen as an upper limit because having too
many clusters all with unique colors makes it difficult for the human eye to distinguish the difference in colors very close in the color spectrum. The colors become harder to distinguish as the number of clusters increases.

The following examples include visualizations from both versions of OntoSELF in order to compare the two. Both use the same set of parameters, such as level value as the weighting functions, but the OntoSELF-v2 uses the semantic clustering algorithm to divide the ontology into clusters in order to assign node colors since the visualization produced by OntoSELF-v1 is being compared to the enhanced color cue of OntoSELF-v2. The comparison between semantic and structural clustering algorithms is presented in section 7.3.

The NASA Sweet JPL ontology is used in the two visualization provided in the figures.

Figure 7-9 3D Visualization of OntoSELF-v1 on NASA Sweet JPL Ontology with only superclass/subclass relationship with Node Level as its weighting function.
Figure 7-10  3D Visualization of OntoSELF-v2 on NASA Sweet JPL Ontology with only superclass/subclass relationship. Semantic clustering at Level 0 is used to determine node color. Rank prestige is used to determine cluster selection in case of multiple inheritance.

Both visualizations present the structure of the ontology, however, the node color adds to the user’s ability to distinguish various clusters and to easily identify different subtrees in the structure. The user can more easily determine the number of subtrees from the dummy root in the structure in figure 7-10, but would need a closer inspection in order to determine the number of subtrees in Figure 7-9.
7.1.3 Combination of Node Color and Size

The comparisons between the visualizations produced by the two versions of OntoSELF demonstrate that providing both visual cues of size and color used separately on nodes increases the information that the user can determine from the visualization. These two visual cues may be combined together, and the following example demonstrates how useful their combination is to increase the perception compared to using only one visual cue. The NASA Sweet JPL ontology is used in these visualizations.

Figure 7-11 3D Visualization of the NASA Sweet JPL Ontology with only superclass/subclass relationship. The top 15 nodes with the biggest size based on rank prestige have been labeled but not visual cues added.
Figure 7-12  3D Visualization of NASA Sweet JPL Ontology with only superclass/subclass relationship. The Semantic clustering at Level 0 for node colors is used with rank prestige to determine the cluster when multiple inheritance occurs and the size of the node. The top 15 nodes with the biggest sizes are labeled.

Comparing figure 7-12 which uses both node color and node size to figure 7-11 which simply labels the top 15 nodes and figure 7-10 which uses node color for semantic abstraction reveals that the benefits of using both node color and node size in combination. Using color for semantic abstraction helps users to identify each cluster of highly related concepts but it is hard to tell the significant concepts lying in each cluster. In figure 7-12, the combination of the two visual cues provides the user with more insight into the structure and important concept in the ontology and their relatedness. For example, in the cluster with dark green including the concepts &property;index and
&property;PhysicalQuantity, the user can easily identify these two core concepts in this cluster by just looking at their size. Another example is the concept Animal which is identified as a significant concept to this ontology because of its size is also shown to be semantically related to other important concepts such as Plant.

7.1.4 Edge thickness

Standard SNA typically deals with only one kind of node and one kind of relationship, i.e., the 1-mode and 1-plex network. Ontologies have a standard relationship, the superclass/subclass relationship which is the only kind of relationship that OntoSELF-v1 deals with. OntoSELF-v2 also handles user-defined relationships. But in order to use standard SNA techniques, multiple user-defined relationships between two concepts are all collapsed into one edge between the two nodes representing those concepts in the SNA processing and visualization. OntoSELF-v2 has the ability to transform n different types of relationships to just 2 types of relationships, the hierarchical one and the user-defined one.

Through the transformation process, there might be possible that two nodes have multiple relationships. Edge weights specify the number of relationships being mapped to an edge between two nodes representing concepts in the ontology. In visualization, an edge’s thickness is determined by the edge’s weight. The thicker the edge, the more relationships that have been merged into the edge. However, in all the ontologies being used to test this enhancement, there are few multiple relationships combined into one edge. One reason for this situation is in most of the intensional ontologies, few multiple relationships are defined between the same two classes. Another reason for this situation may have resulted from an implementation decision to include only directly defined relationships between classes and not inherited relationships. Package 2 (U2D) and package 4 (HU2D) use the edge thickness feature.
Figure 7-13  U2D Visualization of NASA Sweet JPL Ontology with all user-defined relationships with rank prestige used to determine node size and structural clustering on the whole ontology used to determine node color. Top 35 biggest nodes are chosen for labeling.

Several pairs of nodes have slightly thicker edges, in figure 7-13 and all of those edges just have weight 2 by looking at the gml file. There are two relationships mapped on those thicker edges. For example, the two labeled light green nodes, waterSurfaceLayers and waterSurface have a thicker undirected edge linking them. The undirected edge means the both of them have relationships pointing out to each other. In order to minimize the number of edges without losing any information, the undirected edge is used to replace a loop between two nodes. An examination of the NASA Sweet JPL ontology’s OWL file shows one relationship isLayerUpperBoundaryOf from WaterSurface to WaterSurfaceLayer. Another relationship isAdjacentTo exists from WaterSurfaceLayer to WaterSurface.
The use of edge thickness is not clearly a strong perceptual visualization cue based on testing with intensional ontologies. It does, however, help users to identify the pairs of nodes with multiple user defined relationships between them. This visual cue may be of more use with extensional ontologies in future enhancements to OntoSELF.

### 7.1.5 Edge Color

The edge color used in package 1 (H3D) and package 3(HU3D) represents the node level. For example, the edges flowing out from the root is always red. For the next levels it goes gradually from light red to orange to yellow to green and finally to blue. This edge coloring is provided by OntoSELF-v1 as a visual cue that helps the user have a better view of the hierarchical structure. Edge colors in package 4 (HU2D) distinguish the two types of relationships, hierarchical and user-defined because the third dimension representing the hierarchical structure does not exist in this package and all edges are treated equally on a flat 2D layer. In the following example, the same ontology is presented with all same parameters but visualized by package 3 and package 4 to analyze the difference in the approaches to distinguish hierarchical and user-defined relationships.
Figure 7-14  HU2D Visualization on UMLS SN Ontology with hierarchical relationship in blue and user-defined relationships in red. Rank prestige determines node size and structural clustering on whole ontology determines node color. The top 15 biggest nodes are labeled.
Figure 7-15  HU3D Visualization of UMLS SN Ontology with hierarchical relationship and user-defined relationships colored according to the source node’s level. Rank prestige determines node size. Structural clustering on whole ontology determines node color. Top 15 biggest nodes are labeled.

The above two visualizations includes superclass/subclass as its hierarchical relationship. The user-defined relationships include "occurs_in", "associated_with" and "co-occurs_with". From figure 7-14, it is easy to distinguish between user-defined and hierarchical relationships by just looking at the only two colors on edges. The user can easily see which concepts have links between them based on user-defined relationships. However, determining which node is the root is difficult since inherent hierarchical structure based on the third dimension is missing. On the contrary, figure 7-15 gives a very clear hierarchical structure and it is easy to find the root of a node and the nodes at
the same level. However, there are many user-defined relationships linking those biggest nodes in the middle portion of the structure. These edges not only create more line crossings, but also degrade the perception of the hierarchical structure at the middle portion.

7.2 Evaluation on Filtering Criteria

Many semantic web developers are searching for the most appropriate ontologies for their applications or for their analysis and comparison purposes. They need to be able to develop a basic comprehension of what the ontology is for and what are its core concepts, that is, they need a high-level comprehension of the ontology. Methods to determine the core concepts of an ontology must be used. For an ontology containing thousands of concepts, the SNA centrality and prestige measures can help users to filter out those concepts not as important to a high level understanding of the ontology in order to provide a clearer visualization that exposes the core concepts of the ontology.
7.2.1 Comparison of the SNA Metrics in Hierarchical Structure

From the above four visualizations, it can be seen that for ontologies using only the superclass/subclass relationship, the closeness and information centrality measures do not significantly distinguish between nodes. On the contrary, betweenness centrality and rank
prestige measures do make noticeable distinctions to indicate concepts considered more core as judged by those measures. The betweenness centrality, however, always makes the root the most important concept since the root node always lies on the shortest paths from any node to any of the other nodes that have the root as their lowest common ancestor. The betweenness centrality algorithm considers those nodes on most of the shortest paths between other nodes as the more important nodes. Additionally, if two nodes are at the same level, the one with more descendants is considered more important because each node is considered a root of a subtree root. A subtree root always lies on the shortest path from one of its descendants to another who have the subtree root as their lowest common ancestor; therefore, the more descendants it has, the more important it is.

The closeness centrality simply calculates for each node the sum of the distances to all other nodes, and takes the reciprocal of the sum. In an ontology with only the hierarchical relationships, the root is always the most important one if measured by closeness centrality because its paths to all the other nodes are always the shortest. To better understand this idea, imagine two leaf nodes at level 10 in the hierarchy. The distance from the root to each node and vice versa is 10, but the distance between the two leaf nodes is 20. As can be seen from the visualization, the centrality measure for each node in the hierarchical structure is not very distinctive.

The information centrality is the generalized form of betweenness centrality. It calculates a node’s importance on every path between two other nodes, not only the shortest paths. This measure for each node in hierarchical structure is not distinctive, because every node i except those leaf nodes are lying on the paths from its descendant nodes to all the other nodes that have the common ancestor is either the ancestor of node i or just node i. The rank prestige is comparably the best measure of importance at least for hierarchical structures. This analysis shows the rationale for the selection of rank prestige for node size the previous evaluation examples.
7.2.2 SNA Metrics for Topology Understanding Tasks

For OntoSELF-v1, five topology understanding tasks were used to evaluate its visualization features (Somasundaram 2007). These tasks were accomplished successfully and effectively by using a variety of hierarchical filtering criteria and weighting functions. The use of the OntoSELF-v2 added features such as size functions and SNA filtering criteria are evaluated on their ability to further support those topology tasks listed below:

1) Find the Bushiest Child Node for a given node. Their definition of “bushiest” is the child node of the given node which itself has the most children.

2) Find the Largest Subtree for a given node. The largest subtree is the child node which itself has the most descendants.

3) Find a Deepest Node of a given node. Any leaf node that has the greatest depth for the subtree rooted at the given node.

4) Find 3 Nodes with at Least 10 Children. Any nodes in the ontology that have an outdegree of 10 or more, i.e., 10 or more children nodes.

5) Find 3 Top-level Nodes that Root a subtree of Depth of at Least 5. Top level nodes are children of OWL:Thing, the artificial root. Only those nodes whose subtree has a depth of 5 or more are to be selected.

The test scenarios are the same as used in evaluating OntoSELF-v1 One of the NASA JPL SWEET ontologies, http://www.onthology.org/bitstream/2178/76/1/phenomena.owl is used.

OntoSELF-v1 can be used to accomplish the first task, find the bushiest child node for a given node, by first filtering the whole ontology based on the node name of the given node and only immediate children are considered. After the user gets the output GML file, it is used as input with IC or information content value as its weighting function for the 3rd dimension in the visualization. Finally the user can compare the levels, or height of the immediate children of the given node. The one with the greatest height, i.e., closest to the given node is the bushiest node since it has the lowest information content which is used to position it in the 3rd dimension. This node is clearly
Global Oscillation as can be seen in the figure 7-17 produced by OntoSELF-v1

Figure 7-17  Find the Bushiest child node of a given node (Somasundaram 2007)

For OntoSELF-v2 an improvement is made on accomplishing this task since only one step is needed. By using child count for determining node’s size level as its weighting function for the layout, and filter on the node name with the given node, the user can identify the bushiest child node. Using the same scenario as for OntoSELF-v1, the given node is &phenomena;EarthSciencePhenomena. The following is the initial visualization by using the previously described parameters
Figure 7-18  The initial visualization for the task of finding the bushiest child node of a given node. The number of children each node has determines the node size.

Since the labels are very long, the user must zoom in on the sub-tree rooted at the given node, &phenomena;EarthSciencePhenomena to produce the following visualization:
Figure 7-19 Zooming in on given node from figure 7.18 for the task of finding the bushiest child node of a given node.

An examination of this zoomed in view clearly shows that for &phenomena;EarthSciencePhenomena with all its children at the same level, the node &phenomena;GlobalOscillation is the largest and therefore, is the bushiest child node of &phenomena;EarthSciencePhenomena.

For the second task, finding the largest sub-tree of a given node, the first version OntoSELF can support it by setting all parameters once in the user interface. The IC is selected as the weighting function and the filter is on the name of the given node. The
immediate child closest to the given node in the visualization is the one that satisfies the request. The following is the initial visualization from (Somasundaram 2007):

![Initial Visualization](image)

**Figure 7-20 Find the Largest Subtree for a given a node (Somasundaram 2007)**

However, OntoSELF-v2 can enhance the visualization result by using the number of descendants as the size function so that the one closest to the given node with the largest size is the one with the largest sub-tree. The visualization is as follows:

![Enhanced Visualization](image)
After studying the last three tasks, it was determined that OntoSELF-v1 has already provided the easiest and most effective way to support them. For example, to find a deepest node of a given node as task 3, the first version OntoSELF provides the easiest way to find one by simply filtering on the node name of the given node and setting both the lower and upper bound to MAX for the level filter. The visualization produced shows the given node linking the deepest nodes in its subtree.

To enhance the support for the 5 topology understanding tasks, OntoSELF-v2 can take full advantage of its size functions. In the first task, the size function simplifies the process by reducing it from two filtering steps to one step with the size function set based
on the number of children. For the second task, the size function is used to make more readily visible the node that is the root of the largest subtree by setting the size function to the number of descendants.

7.3 Evaluation of Abstraction

Two clustering algorithms are provided for abstraction purpose, the semantic abstraction and structural abstraction. Both are available in all four visualization packages. In section 6.3.5, the motivations and details have been discussed. In this section, the usability and effectiveness of each algorithm is examined and the results produced by structural clustering algorithm with the one produced in Cohesive Partitioning algorithm (Chen, 2002) are compared. Clustering for semantic abstraction and for structural abstraction are compared.

7.3.1 Clustering for Semantic Abstraction

Figure 7-10 and figure 7-12 illustrate the use of clustering for semantic abstraction by node color and the combination of node color and size. Here, the effects of two parameters settable by the user on the clustering algorithm are demonstrated and the result of this algorithm is compared with the one in (Chen 2002). In that paper, they use only the superclass/subclass relationship to partition the whole UMLS SN ontology. Our clustering for semantic abstraction also uses superclass/subclass relationship to cluster whole ontology. The visualizations are produced in package 1 (H3D) and 4 (HU3D).
Figure 7-22  H3D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Semantic Clustering at Level 1 with Closeness as clustering criteria is used. The top 35 largest nodes are labeled.
Figure 7-23  HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Semantic Clustering at Level 1 with Closeness as clustering criteria is used. All nodes are labeled.

The top 35 nodes that are largest in size are labeled in figure 7-22 for the H3D visualization in order to avoid many label overlappings. In HU2D shown in figure 7-23, there is enough space for all labels. For purposes of semantic abstraction, the node level is counted from the dummy root at -1 and the actual roots in the ontology are at level 0. As an example, if the user chooses clustering for semantic abstraction at t level 1 on the UMLS SN the nodes at level 1 become the head of four clusters, Conceptual Entity, Physical Object, Phenomenon of Process, and Activity. These can be seen in figure 7-22 and each group of nodes is colored based on the color of the cluster head node. The three nodes at the top two levels are clustered into another group colored by green since those nodes represent the highest level of abstraction of the whole ontology. Now the user may examine the cluster of concepts at the highest level abstraction to gain a better understanding of the overall general content of UMLS SN ontology. After gaining this understanding, the user in the same visualization can zoom in on a particular cluster for semantic abstraction that is of interest to see all its nodes in order to see more details to
examine the completeness of specification for this concept domain. The user can clearly focus on learning the context of the Conceptual Entity cluster. The effect of zooming in on the cluster for the semantic abstraction Conceptual Entity is shown in figure 7-24.

![Visualization Toolkit - Win32OpenGl #1](image)

**Figure 7-24 Zooming in on the visualization in figure 7-21**

In figure 7-23 the visualization produced by HU2D package, uses the node colors to increase the user’s perception for finding each cluster of semantic abstraction and provides focus on the context of each cluster with respect to the others. Additionally, the user can transform and zoom in on the visualization to enlarge the view of a particular cluster. Figure 7-25 illustrates the effect of zooming in on the semantic abstraction Conceptual_Entity.
Figure 7-25  Zooming in on the visualization in figure 7-22.

Figure 7-26 and figure 7-27 illustrate clusters for semantic abstraction for both the H3D and HU2D packages. All user selected parameters remain the same except the level for semantic abstraction is set to 2.
Figure 7-26  H3D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Clustering for semantic abstraction is at Level 2 with Closeness as clustering criteria is used. The top largest 35 nodes are labeled.
Figure 7-27 HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Semantic Clustering at Level 2 with Closeness as clustering criteria is used. All nodes are labeled.

In the above two visualizations, there are totally 21 clusters created by the 20 nodes at level 2. The extra cluster is the result of grouping all nodes at level 0 and 1 into the high level cluster for semantic abstraction. The node colors can assist users in identifying each cluster of nodes, but the 21 different colors make it more difficult for the user to separate of clusters whose colors are close by human perception. Too many colors exceed the human limit of distinguishing colors which is about 3 to 7 colors through our tests. However, smaller clusters can increase the user’s understanding of each cluster. The perception of identifying clusters in figure 7-27 is somewhat better than that in figure 7-26. This comparison illustrates that the too many colors coupled with node overlappings in the 3D visualization degrades human perception of the clusters more than that of the 2D visualization. These examples suggest that the user should apply clustering for semantic abstraction preferably at the higher levels, for example, levels 0 through 2. The number of clusters produced, however, is dependent on the structure of the ontology and
the selection of level by the user is also dependent on different topology understanding tasks and analysis. To get smaller more specific clusters and a large number of clusters, i.e., low level semantic abstraction, the users may want to set the level to a higher number. To obtain high level semantic abstraction, i.e., larger more general clusters and fewer clusters, the user should set the level to a lower number.

7.3.2 Clustering for Structural Abstraction

Newman’s Fast Community Detection algorithm (Newman 2004) is used to cluster on either the whole ontology with selected relationships or on the ontology filtered by various hierarchical and social network metrics. The advantage of this algorithm is its automatic partitioning using heuristic evaluation without any user inputs except the ontology graph. First the results produced by this clustering algorithm on the UMLS SN ontology with only superclass/subclass relationship selected are compared to the cohesive partitioning algorithm (Chen 2002). Then some user-defined relationships are included to see the changes to the clustering results. Finally, the results produced by structural clustering on whole ontology and that on a filtered ontology are compared.

The following are the H3D and HU2D visualizations using the clustering algorithm for structural abstraction on the whole UMLS SN ontology with only superclass/subclass relationship.
Figure 7-28  H3D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Structural Clustering on whole ontology is used. Top 35 largest nodes are labeled.
Figure 7-29 HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Structural Clustering on whole ontology is used. All nodes are labeled.

From the above two visualizations, it is easier for users to distinguish clusters in the HU2D visualization than in the H3D visualization. The reason is similar to the comparison made with clustering for semantic abstraction at level 2 shown in figure 7-26 and 7-27. Too many colors coupled with node overlappings distract the perception. The clustering for structural abstraction does groups nodes in a very meaningful way. For example, the concept Animal is a subclass of concept Organism and using semantic abstraction at level 2 shows in figure 7-25 that they are in the same cluster colored hot pink. But with clustering for structural abstraction the concept Animal is an important concept in its own right since it has many descendant concepts. The clustering algorithm assigns different colors to the Organism cluster (pink) and Animal cluster (green). This result corresponds with the structuring rules and human intuition.

The results of Newman’s algorithm used for structural abstraction is compared with that of the cohesive partitioning algorithm (Chen, 2002). In that paper, each cluster is named by the root of this cluster. In the following table, the name of the root for cluster is
used as the cluster’s name and the number of nodes in each cluster is provided. The cluster name included in both algorithms will be listed first.

<table>
<thead>
<tr>
<th>Cluster id</th>
<th>Newman Clustering on UMLS SN</th>
<th>Cohesive Partitioning on UMLS SN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cluster Name</td>
<td>Number of nodes</td>
</tr>
<tr>
<td>1</td>
<td>Idea of Concept</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>Animal</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Biologically Active Substance</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Physiologic Function</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Fully Formed Anatomical Structure</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>Group</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>Pathologic Function</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>Organism</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>Substance</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>Dummy Root</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>Conceptual Entity</td>
<td>17</td>
</tr>
<tr>
<td>12</td>
<td>Activity</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>Organic Chemical</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Both algorithms have 7 identical clusters. This number exceeds 50% of the number of clusters by structural clustering, which has 13 clusters. A comparison of those clusters that do not exactly match is needed. For example, the Organism cluster in Newman’s clustering has 8 nodes but there are only 6 in the cohesive partitioning. This difference comes from the cohesive partitioning cutting the Organism out with only 2 nodes, Plant, and Alga. In terms of granularity, a cluster with 2 nodes is quite small. The Substance cluster is in both results, but has very different sizes. Just looking at figure 7-29, one can see that the sub-tree rooted at Substance has been divided into three clusters, Substance (13 nodes), Biologically Active Substance (7 nodes), and Organic Chemical (8 nodes). The cluster named Biologically Active Substance exactly matched in both results. The difference is based on how to divide the remaining nodes in the Substance sub-tree. In cohesive partitioning, it is divided into three clusters, Substance (3 nodes), Chemical (16 nodes), and Pharmacologic Substance (2 nodes). Again, in terms of granularity,
Newman’s clustering algorithm produces clusters of larger size for those cluster names in black, there are many clusters in cohesive partitioning that have only 1, 2 or 3 nodes. All these clusters are quite specific for granularity. Although Newman’s clustering algorithm provides a higher granularity for each cluster, it might not always be useful for more detailed granularity analysis.

One user-defined relationship is added into the ontology structure to examine its effects on clustering for structural abstraction. The new relationship is called “has measurement”. The following is the list of concepts related on this relationship:

- Physiologic Function has measurement Quantitative Concept
- Physiologic Function has measurement Organism Attribute
- Physiologic Function has measurement Laboratory or Test Result
- Spatial Concept has measurement Quantitative Concept
- Substance has measurement Laboratory or Test Result

The following is the new visualization for structural abstract:

![Figure 7-30: HU2D Visualization of UMLS SN Ontology with superclass/subclass relationship as the hierarchical relationship and has_measure as the user-defined relationship. Rank Prestige is used to determine the node size. Clustering for](image)
structural abstraction on whole ontology is used. All nodes are labeled. The edges in blue are hierarchical ones. The edges in red are user-defined ones.

The following table of clusters is for figure 7-29 and figure 7-30:

<table>
<thead>
<tr>
<th>Cluster id</th>
<th>Cluster Name</th>
<th>Number of nodes</th>
<th>Cluster Name</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fully Formed Anatomical Structure</td>
<td>6</td>
<td>Fully Formed Anatomical Structure</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Group</td>
<td>6</td>
<td>Group</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Pathologic Function</td>
<td>6</td>
<td>Pathologic Function</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Biologically Active Substance</td>
<td>7</td>
<td>Biologically Active Substance</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Organism</td>
<td>8</td>
<td>Organism</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Organic Chemical</td>
<td>8</td>
<td>Organic Chemical</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>Animal</td>
<td>9</td>
<td>Animal</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>Activity</td>
<td>15</td>
<td>Activity</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>Dummy</td>
<td>20</td>
<td>Dummy</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>Physiologic Function</td>
<td>7</td>
<td>Physiologic Function</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>Substance</td>
<td>13</td>
<td>Substance</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>Idea of Concept</td>
<td>14</td>
<td>Idea of Concept</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>Conceptual Entity</td>
<td>17</td>
<td>Conceptual Entity</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td>Organism</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>Spatial Concept</td>
<td>8</td>
</tr>
</tbody>
</table>
Table 2  Comparison between using superclass/subclass only and using both superclass/subclass and has measure relationship. Cluster name in red means the cluster with the same size appears in both algorithms. Cluster name in blue means the cluster appears in both algorithm but with different sizes.

Since all these red clusters do not contain any of the concepts related on “has measurement” relationship, they are not affected so that only those clusters in blue and black need to be examined. Since the concept Physiologic Function has three “has measurement” edges flowing out of it, its size has been increased which means it becomes more important in the new ontology. Additionally, its cluster size has also been increased with 6 new concepts added. Those 6 new concepts are Quantitative Concept, Laboratory or Test Result, Organism Attribute, Clinical Attribute, Finding, and Sign of Symptom. The former three are directly related to Physiologic Function by the “has measurement” relationship. Since the concept Clinical Attribute is a subclass of Organism Attribute, Organism Attribute is also added to this cluster. The Finding is a superclass of Laboratory or Test Result, and Sign of Symptom is a subclass of Finding. The whole sub-tree rooted at Finding is separated from Conceptual Entity but added to the Physiologic Function cluster. The above changes to the Physiological Function cluster and its surrounding clusters correspond with human intuition. Since the concept Spatial Concept has one “has measurement” edge out of five edges, the clustering algorithm considers it as an important concept and separates it from the Idea of Concept and makes it a new cluster.

Another change is the separation of the hierarchical edge between Substance and Chemical. This separation, however, is not useful since Chemical as a subclass of Substance, should also have the “has measurement” relationship. One of the drawbacks of this clustering is that it does not have the capability of reasoning on the difference in importance of the two kinds of relationships.

Finally, a comparison of clustering for structural abstraction on whole ontology and on filtered ontology is provided based on the following figures.
Figure 7-31  HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size. Clustering for structural abstraction on whole ontology is used. All nodes are labeled. The filtering criterion Rank Prestige is used.

Figure 7-32  HU2D Visualization of UMLS SN Ontology with only superclass/subclass relationship. Rank Prestige is used to determine the node size.
Clustering for structural abstraction on filtered ontology (have rank prestige measure larger than or equal to 0.2) is used. All nodes are labeled.

From the comparison of the above two visualizations the one using clustering on the whole ontology provides more clusters since it preserves the whole ontology and provides a global view. However, the one on the filtered ontology is a good choice for distinguishing important concepts from those that have been included after the filtering process. The use of either one depends on the user’s purpose of analysis, for example a global view or selective view based on concepts satisfying the filtering criteria

7.4 Evaluation of Task Support on Ontologies

In a survey of ontology visualization methods (Katifori 2007), many modern ontology visualization tools are compared based on seven high-level tasks that an information visualization software tool should support (Shneiderman 1996). OntoSELF-v1 provided features to assist the user in topology understanding tasks for the hierarchical structure of an ontology by combining layout weighting with different filtering criteria. OntoSELF-v2 extends to these capabilities by helping users perform additional high-level tasks for understanding, analyzing, and navigating ontologies. The seven high-level tasks present a suite of examples that can be used to evaluate these additional capabilities. The four packages are used to determine how effectively these tasks are supported.

The seven tasks are as follows:

1. Overview: Gain a global view of the whole structure.
2. Zoom: Allow zooming in on any items of interest. Meanwhile, the global view can be retained.
3. Filter: Support filtering out items beyond interests.
4. Details-on-demand: Get details of a selected item.
5. Relate: Support viewing relationships among nodes.
6. History: Keeps a list of historical actions to support tasks like undo and replay.
7. Extract: Allow saving sub-parts of items of interest. But as stated in (Katifori 2007), this task is often supported by ontology management tools, so it is not used.
in visualization evaluation. However, OntoSELF has the ability to support this task.

The UMLS SN ontology is used for the above seven tasks since it includes a large number of user-defined relationships and its size is suitable for easier visual comprehension.

7.4.1 Task 1: Overview

Package 1 (H3D) can support the view of ontology structure with only hierarchical relationships, and package 2 (U2D) supports the view of ontology structure with only user-defined relationships. Both provide a global view of the whole structure with partial information. Package 3 (HU3D) and package 4 (HU2D) are capable of supporting the first task by including all relationships in it. The following set of HU3D visualizations provides several different points of view by rotating the whole visualization. The node level is used to determine the node size and the clustering for semantic abstraction at level 1 with Information Centrality as clustering criteria is demonstrated. The choice of size function and layout weighting function affect the layout of HU3D visualizations. Thorough testing with all size functions revealed that the node height is the better function to assists the layout algorithm in preserving the ontology’s hierarchical structure. Since the task is for overviewing, only top 25 nodes with the highest rank prestige are labeled since too many label overlappings decreases the perception dramatically. Note that a different size function (node level) can be used separately from the metric used to label the selected nodes (rank prestige).
Figure 7-33  HU3D Top-down view of the UMLS SN ontology
Figure 7-34  HU3D 90-degree view of the UMLS SN ontology
Figure 7-35  HU3D 45-degree view of the UMLS SN ontology
The above set of four visualizations demonstrates that the HU3D package preserves the hierarchical structure layout of the ontology. However, the line crossings caused by the many user-defined relationships decrease the perception of the hierarchical layout. HU3D, does, however, provides a good overview of the whole ontology with all relationships included based on its hierarchical structure and the ability to rotate the visualization helps to mitigate the decreased perception from the many edge crossings.

The following HU2D visualization uses the same set of parameters from the above scenario except the number of labels is for the top 100 nodes.
Figure 7-37  HU2D visualization of UMLS SN with all relationships selected. User defined relationships are in red and hierarchical relationships are in blue.

Obviously, the HU2D is not an appropriate package for a complete ontology with many nodes and edges due to limited space in a 2D view and the inability to rotate this view. In addition, the loss of the third dimension makes it hard to find nodes at the same hierarchical level. However, it provides a general overview of a high-level abstraction of the whole ontology and is easy to locate higher level concepts by looking at those largest nodes so that in one sense the user can identify nodes at the same hierarchical level. It is easy to relate those nodes with few edges and also those at the peripheral region of the visualization, but most of the nodes cannot be related by just looking at this visualization. Compared with HU3D, HU2D package is better for low-level exploration of a small number of nodes. But the actual limitation on that number varies depending on the user and the analysis purposes.

The following two overview visualizations have either hierarchical (H3D) or user-defined relationships (U2D).
Figure 7-38  HU3D visualization of UMLS SN with only superclass/subclass relationship as hierarchical relationship. Rank Prestige is used to determine the node size. Clustering for semantic abstraction is used. Top 20 largest nodes are labeled.
Figure 7-39  U2D Visualization of UMLS SN with all user-defined relationships selected. Rank Prestige is used to determine the node size. Clustering for semantic abstraction is used. Top 20 largest nodes are labeled.

H3D provides a clear view of the hierarchical structure of the UMLS SN ontology based on its superclass/subclass relationships but excludes all user-defined relationships. H3D package is a better choice for orienting new users to the basic hierarchical structure of an ontology.

The visualization is much clearer than that produced by HU2D including all relationships, but the middle portion with many line crossing is still quite distracting. Therefore, like HU2D, limited by its 2D space, it is better to view a small group of nodes with fewer nodes for low-level exploiting tasks. But again the actual size limitation depends on the user and the analysis purposes.

7.4.2 Task 2: Zoom

The zooming capability is available as a default in most visualization development software, but there are also different types of zooming facilities. For example, in
OntoSELF, filtering-like zoom capabilities are provided in addition to the ordinary zoom in/out, it also provides. Both types of zooming facilities are available in all four packages.

Figure 7-40  Zooming in to view the local context of the cluster rooted at Conceptual Entity in H3D visualization. Rank Prestige is used to determine the node size. Clustering for semantic abstraction is used. Top 20 largest nodes are labeled.

After zooming in to view the local context of the cluster rooted at Conceptual Entity, the user can view more clearly the specific concepts within the Conceptual Entity cluster. Additionally, it preserves the surrounding environment of the cluster of interest, but loses the global context, because it is a simple zooming facility by enlarging the view of visualization. All nodes and edges beyond the window become invisible.
Figure 7-41  Zooming in to view the local context of the concept Pathologic Function in HU3D visualization.

From the above visualization, the user can view some of the concepts linking with the concept Pathologic Function at the current point of view. But the many line crossings still decrease the user’s perception of tracing edges flowing out of the concept of interest.
Figure 7-42 Zooming in to view the local context of the cluster in dark blue in U2D visualization.

Figure 7-43 Zooming in to view the local context of the upper left portion of the visualization in HU2D
The above visualizations produced by U2D and HU2D have the same disadvantage of those using H3D and HU3D. The global context is lost; however, the user can view the portion of interest much more easily.

The enhancement of the zooming with filtering capability can assist the user in topology understanding task. Based on the combining level filtering criteria and the size functions, several of the 5 topology tasks have been improved as described in section 7.2. By combining the four centrality and prestige filtering criteria, OntoSELF-v2 can do two major categories of zooming with filtering. The first one is the sub-structure filtering based on node name, like the one task used for topology understanding as previously described. But it can also combine one of the four centrality and prestige metrics as the size function and the subtree filtering by node name to view the sub-structure with importance measure on each node. The following set of visualizations produced by the four packages with the subtree rooted at the concept Animal illustrates this capability.

![Figure 7-44 Zooming with subtree filtering based on node name “Animal” in H3D](image)

Figure 7-44 Zooming with subtree filtering based on node name “Animal” in H3D
Figure 7-45  Zooming subtree filtering based on node name “Animal” in HU3D

In the above figures, the weighting function for layout is the node’s level value. The clustering for semantic abstraction is at level 1 with rank prestige as clustering criteria for breaking ties. All nodes are labeled. The filtering criterion is based on the node name called “Animal”. The node size in figure 7-45 is determined by rank prestige, but the node size in figure 7-45 is determined by node level. The H3D gives a better hierarchical view of the local context of the concept Animal by including all its descendants and ancestors. This facility filters out all nodes beyond interest but preserves a global context view. The user can clearly know that the concept Animal is located at level 3 (The dummy root is at level -1, because it is not a concept in the ontology). However, the
disadvantage is that none of the user-defined relationships are included. In figure 7-45, since all relationships are included, the local context of the Animal subtree includes all of its descendants and ancestors. In our design, all the concepts that are directly related to any node in Animal sub-tree are also included as part of the subtree through the user-defined relationships. Since the subtree of the concept Animal only includes 20 concepts, there are few line crossings. HU3D not only provides all relationships in a local context but also preserves the hierarchical layout and a global context view. The hierarchical layout, however, is not as clear as the one produced by H3D. The HU3D is a better choice for low-level exploiting based on zooming with filtering than the H3D visualization.

Since there is no hierarchical relationship included in U2D, the node name filtering feature is disabled. The following is the visualization produced by HU2D using zooming with filtering on the concept Animal. The node size is determined by rank prestige. Since there is no third dimension, the layout weighting function is not needed. The clustering for semantic abstraction is at level 1 with rank prestige as clustering criteria for breaking ties between clusters. All nodes are labeled.

![Figure 7-46 Zooming with filtering based on node name “Animal” in HU2D](image)
The above visualization provides a much clearer view than HU3D of the relationships between each concept. There are no node overlappings and fewer line crossings than those in HU3D. However, the user cannot easily identify which nodes are at the same level due to the loss of the hierarchical layout. But they can still easily trace the hierarchical path from any node to its descendants or ancestors following the blue edges. Between HU2D and HU3D using zooming with filtering, HU2D is appears to be better if the analysis looks at the whole graph as a social network, but HU3D is better if the hierarchical layout needs to be preserved.

7.4.3 Task 3: Filter

Section 7.2 discusses the filtering capability of OntoSELF-v2. More filtering examples combining both SN centrality and prestige metrics and hierarchical metrics are shown to demonstrate its enhanced capabilities.

A useful feature is the ability to determine the core concept within specific levels of the ontology. Several scenarios of this kind of task are presented in this section. Since the rank prestige is the best measure for importance calculation, it is always used for the evaluations in this section. From experimentation on different ontologies, concepts with rank prestige measure greater than 0.2 indicate that they are the more significant or core concepts in the ontology.

The following visualizations illustrate using a compound set of filtering criteria, rank prestige and level value. The objective of the scenario is to find the core concepts and located above or at level 3 since those higher levels indicates a higher abstraction level and can provide a user better understanding of the whole ontology structure and its domain area.
Figure 7-47  H3D visualization of UMLS SN. Concepts at level 3 or higher and having rank prestige greater than 0.2 are displayed. Clustering for semantic abstraction at level 1 with rank prestige as clustering criteria. All nodes are labeled.

In Figure 7-47, most of the concepts at level 2 or higher are core concepts, but many of those at level 3 are not and so they are filtered out. This visualization after filtering highlights those concepts at level 2 with some of their important core sub-concepts which are used to understand the subtrees below level 2 without the clutter of all the concepts at that level. The rank prestige here is calculated based on the superclass/subclass relationships.
Figure 7-48  HU3D visualization of UMLS SN. Concepts at level 3 or higher and having rank prestige greater than 0.2 are displayed. Clustering for semantic abstraction at level 1 with rank prestige as clustering criteria is used. All nodes are labeled.

Figure 7-48 is very different from figure 7-47 although the only difference is the addition of user-defined relationships. Since most of the lower level concepts have more user-defined relationships, they often have higher rank prestige than those at higher level. This set of compound filtering criteria used in HU3D indicates to the user that they may also consider looking at subtrees rooted at the concepts having highest rank prestige measures with user-defined relationships.
Figure 7-49  U2D visualization of UMLS SN ontology. Concepts with rank prestige greater than 0.2 are displayed. Clustering for structural abstraction on the whole ontology is used. All nodes are labeled.

Since U2D visualization in figure 7-49 includes only user-defined relationships, it is impossible to get the compound set of filtering criteria used in the previous two packages. However, the filtering on rank prestige is often meaningful in U2D since it considers the whole ontology as a pure social network. The filtering keeps only those core concepts and their relationships linking them and provides a much clearer view because of the filtering out of concepts not meeting the criteria.
Figure 7-50  HU2D visualization of UMLS SN. Concepts at level 3 or higher and having rank prestige greater than 0.2 are displayed. Clustering for semantic abstraction at level 1 with rank prestige as clustering criteria is used. All nodes are labeled.

Figure 7-50 gives a much clearer view than figure 7-48 because there are only two hierarchical edges and it is not an appropriate structure for 3D hierarchical layout. It also provides a better view of how those selected concepts are linked to each other than that seen using HU3D.

7.4.4 Task 5: Relate

All the above figures show the capability of relating nodes in the visualization. However, the best visualization package for this capability is the package 4 (HU2D). There are two different colors distinguishing the 2 different types of relationships.
Additionally, the edge thickness is added to show the number of same type relationships merged into each edge. Users are also allowed to select the set of relationships to be included in each relationship type. The centrality and prestige measures are determined based on relationships selected by the users.

![HU2D visualization of UMLS SN ontology for relating capability](image)

**Figure 7-51** HU2D visualization of UMLS SN ontology for relating capability

In Figure 7-51, the blue edges belong to hierarchical relationship with only superclass/subclass relationship included. The red edges belong to user-defined relationship with two relationships selected by the user. They are “Constitutes” and “Has Part”. After adding those two relationships, the rank prestige of those concepts linked by these two relationships are also increased. The user can easily identify the two different categories of relationships in figure 7-51.

### 7.4.5 Other Tasks of Details on Demand, History, and Extract

OntoSELF-v2 does not provide this capability of hovering over a node and getting all or requested details of that node. It is a future enhancement discussed in Chapter 7. Since
the current OntoSELF does not provide sufficient interactivity, users cannot work with a
history of user actions in order to do or undo any changes in the visualizations. For
example, it is not supported to delete certain nodes and edges, and then undo this action
to return them to the visualization. However, the resulting visualization may be stored in
a GML file. The user can then apply additional filters over that GML file and get a new
GML file after filtering. All those GML files can be used to produce their corresponding
visualization.

For most visualization tools, the extract task is often not supported since it is often
supported by ontology management tools. This capability can be provided by OntoSELF-
v2 by simply storing the subtree of interest in a GML file. The user can then view this
subtree at any time by using the GML file with VTK.

8 Conclusions and Future Work

OntoSELF-v1 visualization software (Somasundaram 2007) recently developed for
visualizing the standard hierarchical structure of an intensional ontology provides four
weighting functions affecting visualization layout and uses various hierarchical structure
metrics as filtering criteria over nodes. In addition to all these, it also supports parallel
processing of one ontology with multiple views and multiple ontologies for one view
using multiple processors on the Miami Redhawks cluster computer.

OntoSELF-v2 focuses on enhancing visualization perception and supporting high-
level comprehension and low-level exploiting of details as well. Four visual cues are
added to enhance the visualization perception: node color, node size, edge color, and
edge size. Individual user-defined relationships or all of them can be selected by the user
for visualization and social network analysis processing along with the standard IS-A or
superclass/subclass relationship which is the only relationship processed by OntoSELF-
v1. Users have the flexibility to select as many or as few user-defined relationships as
needed by their analysis task and have them integrated with the standard IS-A
relationship to form a category of hierarchical relationships. The user can also deselect
any user-defined relationships beyond interest from the category of user-defined relationship.

The four centrality and prestige metrics from social network analysis are added as part of the filtering criteria for measuring importance of each node. For high-level comprehension purposes, two categories of clustering algorithms are used. One is the clustering for semantic abstraction which allows a user to select all nodes at a certain level as the roots of clusters and all their descendants are assigned into their clusters. If a descendant has multiple parents, a clustering criterion selected by the user is used to determine which one to be assigned to. The clustering criteria include many hierarchical structure metrics and the four centrality and prestige metrics. Clustering for structural abstraction is also provided and is implemented using Newman’s fast algorithm for detecting communities in a social network (Newman 2004). Structural abstraction can be used on only hierarchical relationships, only user-defined relationships or a combination of both categories of relationships.

OntoSELF-v2 adds four visualization packages for different analysis purposes and also for comparing the usability of those packages in different scenarios. Package 1, H3D (Hierarchical relationship in 3D visualization) visualizes the whole ontology with only hierarchical relationships. It is basically a simple extension to OntoSELF-v1 visualization but includes the user’s ability to integrate some user-defined relationships into the category of hierarchical relationship. All weighting functions, size functions, filtering criteria and clustering algorithms are available for this package. As the evaluations show in Chapter 7, this package is a good choice for orienting new users to understand and learn an ontology’s hierarchical structure. By using any of the four centrality and prestige metrics as the size function, the visualization can assist users to easily identify the core concepts in the ontology by finding those nodes with large size even for very large-size ontologies like Gene Ontology which includes more than 20,000 concepts. It is also recommended to use the clustering for semantic abstraction at lower levels to separate the whole ontology into several sub-ontologies. This semantic abstraction often gives a new user a very clear high-level comprehension of the basic hierarchical structure of the ontology.
Package 2, U2D (User-defined relationship in 2D visualization) visualizes any user-defined relationships selected by the user in 2D visualization based on the JUNG framework. Since the standard IS-A relationship is not available; all hierarchical structure metrics are not available for use. Since the focus is on user-defined relationships only, the inherent hierarchical IS-A relationship is lost and thus the hierarchy as the third dimension disappears. This package is capable for analyzing the ontology with user-defined relationship as a pure social network. The evaluation shows that the clustering for structural abstraction is a better choice to give an overall view based on user-defined relationships. However, due to the limited space of a 2D window, this package is suggested to use to structure visualization for a smaller number of nodes with the suggested number being less than 30 nodes. Therefore, U2D is a good choice for low-level detail exploiting.

Package 3, HU3D (Hierarchical and User-defined relationships in 3D visualization) is like H3D but integrates user-defined relationships into the 3D visualization. If the ontology contains many user-defined relationships, the empirical evaluation tells us that the hierarchical layout will be affected by not only the layout weighting function but also the size function used for the node. To retain the best hierarchical view, users should use node level as the size function. However, this selection for node size can limit the user’s ability to find the core concepts of the ontology. The added user-defined relationships not only affect the visualization layout but also affect the centrality and prestige metrics of all the nodes. This visualization can provide a better view of the core concept of the whole ontology with complete set of relationships especially when the rank prestige measure is used for the node size function.

Package 4, HU2D (Hierarchical and User-defined relationships in 2D visualization) has all the capabilities HU3D has but just visualizes the ontology in 2D view. The disadvantage of this package is obviously the loss of the third-dimension hierarchical view. However, it uses edge colors to distinguish the two categories of relationships. The edge thickness is also used to represent the number of relationships mapped onto a directed edge. Both of these edge features are not available in HU3D. However, the limited space of a 2D window restricts its best usability for a structure with a fewer number of nodes, 30 nodes or less.
As a conclusion, both H3D and HU3D are good choices for high-level comprehension based on whether the analysis task required the inclusion of the user-defined relationships. Both U2D and HU2D are good choices for low-level detail exploiting based on whether the hierarchical relationship is needed for analysis.

OntoSELF-v2 can be further improved by adding features that help accomplish the seven tasks an information visualization tool should support. First of all, more interactivity shall be added to support the details-on-demand and zooming tasks. A user should be allowed to select a sub-structure of the whole ontology and zoom in on it to view the local context but without losing the global context. A user should be allowed to hover over a node to view the details of interest. Another interesting and useful enhancement is the ability to merge a cluster of nodes into one node in the visualization and to add edges between the nodes representing the clusters. This capability would simplify the visualization. A mechanism for labeling the merged node is also needed. The merged cluster should also be able to be separated back to a cluster of nodes as a restoring capability.

OntoSELF-v2 can only filter on nodes. Filtering on edges based on edge metrics would also be a useful enhancement. The future extension shall also find some sophisticated layout algorithms for both 3D and 2D visualization. Since OntoSELF-v2 is a powerful tool for supporting topology understanding tasks, an easy-to-use topology query language should be provided to help the user produced the needed visualizations without having to understand all the sophisticated user settable parameters such as weighting, filtering, size, color and abstraction features it provides. An enhancement to translate a user’s visualization query into the set of the appropriate parameter values for these features would be important to increasing OntoSELF-v2’s usability.
References


Appendices

Appendix I Visualization Software Implementation

This appendix includes the details of both 3D and 2D visualization implementations using VTK and JUNG respectively.

3D Visualization Implementation using VTK

OntoSELF visualization is developed under the VTK graph visualization library ([http://www.comp.leeds.ac.uk/djd/graphs/](http://www.comp.leeds.ac.uk/djd/graphs/)). Many objects used in the system are part of this graph library; the most significant one being the vtkGMLReader, where GML is an acronym for Graph Modeling Language (Himsolt, 1997) and is a standard file format for graphs. This format is platform independent and is capable of representing any graph structure, whether a single graph or multi-graph. Typically any ontology is an m-mode n-plex network. As discussed in section 3.1, an m-mode n-plex network contains m types of nodes and n types of edges. As a general graph representation format, GML is a good choice for this purpose. Another advantage of this format is its human-readable format.
Users can easily open this file and read it to find out different information associated with each node and edge. The following pipeline was created for OntoSELF and its SNA extension and has as its basis OntoSELF’s original pipeline structure (Somasundaram 2007). Note in the following that the objects added or modified for this thesis work are indicated in colored boxes.
Below the added objects to the original version as part of this thesis work are described:

vtkAssignAttribute (in glyphing pipeline) – Each node has many attributes. In order to color a node according to one of its attributes, the vtkPolyDataMapper must be informed.

Figure A-1 VTK pipeline for OntoSELF
which attribute to use for the color scheme. The vtkPolyDataMapper associates each
node point with one attribute only. In the enhanced OntoSELF the cluster attribute
specifies which cluster each node belongs to and is used for the color scheme.

Therefore, it extracts one attribute cluster from an array of attributes to form an
attribute scalar. This transformation is necessary for coloring each node or each glyph,
because the vtkPolyDataMapper needs to get data from an attribute scalar to map that
data to the corresponding color in the vtkLookupTable. If an array of attributes is fed to
the vtkPolyDataMapper, it cannot figure out which attribute is used to map to the
vtkLookupTable.

vtkProgrammableGlyphFilter – This filter is used to replace the process object,
vtkGlyph3D, because it allows developers to program and add some methods into this
object. For enhanced OntoSELF, a method sizeProc was added to determine the size of
each glyph. It receives as input one attribute from all attributes and calculates each
sphere’s radius based on that attribute value. The input of this process object is
vtkAssignAttribute. Finally it will produce an output and feed it to vtkPolyDataMapper.

vtkLookupTable (in Glyphing pipeline) – This lookup table is created to map one
attribute value of each glyph to one of the colors in the table. As discussed in section
6.3.5, the number of colors in the table is set to 64.

In addition to these newly added process objects, some process objects have been
updated in order to provide the extensions to OntoSELF:

vtkActor (in Glyphing pipeline) – Since the color of each glyph is decided at the
vtkPolyDataMapper processing stage, the vtkActor in Glyphing pipeline is no longer
responsible for coloring.

vtkPolyDataMapper (in Glyphing pipeline) – Since vtkGlyph3D is no longer used, the
input to this process object is changed to vtkProgrammableGlyphFilter. The filter is
almost the same as vtkGlyph3D, but with a new method added to process each glyph
to determine the size of each node. A node attribute called size exists for each node in
the GML file. The user selects from the various metrics in the size function area of
the OntoSELF user interface to determine the node size. The extended and enhanced
OntoSELF user interface is described in section 6.2.
2D Visualization Implementation using JUNG

The following UML class diagram represents the basic data structure and methods of OntoSELF 2D Social Network Visualization based on JUNG 2D Visualization:

Inside this class, the core structure is the Graph interface, which can be instantiated as SparseMultigraph class in the constructor. Figure A-2 shows the UML for the Graph interface.
Figure A-3 The UML class diagram of the external class edu.uci.ics.jung.graph.Graph composed of the two customized classes, Vertex and Edge.

In the above UML class diagram, the Graph interface is an external class imported from the graph library of JUNG. In the latest version JUNG2, users are allowed to create any classes as vertices and edges. For example, users can use String as a vertex and Integer as an edge, so the corresponding Graph can be created as Graph<String, Integer>.

For the OntoSELF implementation, however, customized classes are used in order to include data necessary for the enhancements; Vertex and Edge are used to store specific GML data. All field variables in the class Vertex except for labelOn have corresponding node attributes in the GML file. The variable labelOn is used to signal whether a node’s label is to be displayed in the visualization. The size, cluster, id, label, and labelOn are currently used in the 2D visualization. The other variables are included for future possible extensions to this research. The variable size is responsible for determining the node’s size. The cluster id is used for coloring the node.

The Edge class has fewer field variables. The src and dest are used to store the node ids of the source node and the destination node. We are not using Vertex type for those
two variables just for the purpose of being consistent with the GML format of edges. The variable weight represents the number of actual relationships from the source node to the destination node. In the visualization, the weight is reflected by the edge thickness. The label has only two values in our current version. They are “hierarchical” and “user-defined”, because we are transforming the whole ontology into a 1-mode 2-plex network. Only two types of edges are allowed. The variable label can be distinguished by color in visualization. The hierarchical edges are colored in blue, and the user-defined edges are colored in red.

In the SNVisualization class, the variable vertices keeps track of all the nodes read from the GML file. This list of vertices is kept because there is no method in Graph which can return a list of all vertices for processing. Since one can find an edge by using the findEdge (Vertex v1, Vertex v2) in Graph interface, it is unnecessary to keep a list of all edges. The variable packageId is the id number of the visualization package used, so it only has value 2 and 4, because only package 2 and 4 are using 2D visualization. Section 6.3.2 describes the four visualization packages provided by OntoSELF. The variable topNum is used to restrict the number of nodes to which labels are attached in the visualization of the ontology.

The method readGML reads data from a GML file and stores them into graph and vertices. The method sizeSort is used for sorting vertices’ size from the largest to the smallest. By doing so, the class can switch on the variable labelOn in Vertex class. The core method in this class is visualize which is responsible for visualizing the graph structure in 2D. Inside this method, the core class is VisualizationViewer, the graph visualization base class from edu.uci.ics.jung.visualization library. It is the container in which to place whatever is to be visualized. Since it inherits from javax.swing.JPanel, a javax.swing.JFrame can be placed inside it to make a display window for the frame.

To visualize a graph, the visualization layout must be determined. For OntoSELF’s 2D visualization packages, ISOMLayout has been selected from edu.uci.ics.jung.algorithms.layout library. It implements a self-organizing map layout algorithm. This layout algorithm was selected after testing with many other layouts.

The CircleLayout simply positions all nodes to form a big circle and puts all edges linking those nodes inside the circle. This layout does not attract users’ perception on the
more important nodes, because all of the nodes are in a circle. There are also many more line crossings. Another two layout algorithms, FRLayout and KKLayout provide very similar layouts as ISOMLayout, but both of them provide constant motion of nodes for very large graph until all nodes have been moved to their best places to provide the best layout. This motion process often takes 5 seconds for FRLayout and more than half minute for KKLayout to find the best layout for a test case with only 135 nodes and 224 edges. This process time would be much greater for an ontology with thousands of nodes.

There are also three tree layouts, BalloonLayout, RadialTreeLayout and TreeLayout. Initially, they were to be used to visualize package 4 which includes both hierarchical and user-defined relationships, however, those three layouts require a Forest class instead of a Graph class to build the layout. The Forest class does not allow multiple parents for all nodes. But in ontologies, multi-inheritance can happen between classes. Therefore, ISOMLayout was used as the layout for package 4. Color is used to distinguish between the hierarchical and user-defined edges.

After the layout has been chosen, the VisualizationViewer class is created by setting the layout as VisualizationViewer<Vertex, Edge> visualizationServer = new VisualizationViewer<Vertex, Edge>(layout). After this instantiation, various functionalities must be assigned to the VisualizationViewer, such as node color, node size, edge color, edge thickness, node labeling and even mouse event control. To implement all those functionalities except mouse event control, the RenderContext class which is part of VisualizationViewer is responsible for them. The RenderContext class assigns default values to all of those features. These values must be modified by using those setter methods in RenderContext. To set the previously listed 5 features, the following setter methods are used: setVertexFillPaintTransformer, setVertexShapeTransformer, setEdgeDrawPaintTransformer, setEdgeStrokeTransformer and setVertexLabelTransformer, respectively. A Transformer class argument must be passed into the setter methods. This Transformer<V, E> class is imported from the external library org.apache.commons.collections15. The main purpose of this class is to apply the type E feature to class of type V. For example, to set edge color, the following Transformer class has been redefined:

Transformer<Edge, Paint> edgePaint = new Transformer<Edge, Paint>() {

141
public Paint transform(Edge e) {
    if(e.getLabel().equalsIgnoreCase(""hierarchical"")) {
        return Color.BLUE;
    } else {
        return Color.RED;
    }
}

Every time applications of certain pattern to Vertex or Edge must be modified, the transform method is redefined in the Transformer class because the RenderContext class calls the transform method of each Transformer class to set the features of Vertex or Edge. The Paint interface is from the java.awt library and defines how the color patterns are to be used in 2D graph. The Color class from the same library is one of its implemented classes. The transform method gets an Edge class argument and checks its label. If the label is “hierarchical”, the edge is colored as blue; otherwise, it is red.

Finally, the mouse event control is set to allow transformation, zooming and picking events in the visualization. The JUNG2 visualization package provides a very simple way to set all those three mouse events by using its DefaultModalGraphMouse class from edu.uci.ics.visualization.control library. This class contains a large collection of plugins for picking and transforming a graph, and also allows extension to other graph mouse events. There are two modes in the class; users have to press certain keys to switch between the picking mode and the transforming mode. In both modes, the zooming in and out functionality is always available by scrolling the mouse wheel. In addition to the zooming, the following transformation activities can also be performed under transformation mode:

(1) Left mouse click + mouse drag = graph panning.

(2) Shift key + Left mouse click + Mouse drag = graph rotation.

(3) Control key + Left mouse click + Mouse drag = graph shearing.

The following simple four-line codes set all the above mouse events:

    DefaultModalGraphMouse gm = new DefaultModalGraphMouse();
gm.setMode(ModalGraphMouse.Mode.TRANSFORMING);
visualizationServer.setGraphMouse(gm);
visualizationServer.addKeyListener(gm.getModeKeyListener());

First a DefaultModelGraphMouse class is instantiated, and then it sets its default mode to TRANSFORMING. The above third line adds it to the VisualizationViewer class. Finally, the default key listener is set to switch between the two modes. To Transforming mode, press the letter key “t”; to Picking mode, press the letter key “p”.

**Appendix II Implementation of OWL to GML Parser**

This appendix discusses the details of the parser data structure and the method it parses an OWL file. One simple example of using parser stack is also included. The memory-saving data structure for holding the basic ontology structure is as follows:

![Figure A-4 Class definition of OWLtoGMLParser class](image)

The description of the attribute part of this class is as follows:

1) nodesList: List<String> -- This list contains the class names for the nodes. Typically the class name itself is used as the label for a node but it may be the case that the label property associated with the class better describes the meaning of the class. The OntoSELF OWLtoGML parser provides the ability to allow users to specify whether the class id or the class label is to be the node’s label written in the GML file. But the class id is still stored in nodesList when parsing OWL files since one class might be defined once but used somewhere else multiple times, and class id is the unique id to check if the class has already been added to the nodesList.
2) `edgesList: List<String>` -- This list contains the type names of edges. It stores a list of properties’ names.

3) `labelList: List<String>` -- This list contains the labels for the classes but is only necessary when the user selects to use the OWL class label as the node’s label in the visualization.

4) `graph: List<Hashtable<Integer, Vector<Integer>>>` -- This list contains hashtables, each of which represents a graph of a particular property type. For example, for property `SubClassOf`, there is a hash table used to store the whole graph of that property type. Inside each hash table, the keys of type Integer represent the source node. The values are formatted as a vector of integers and represent the destination nodes the source node links to. In this list, the order of the hash tables is consistent with the order of the `edgesList`.

The description of the method part of this class is as follows:

1) `readOWL (file: File): void` – This method inputs an OWL file and parses it. It assigns data into the above described data structure while parsing the file. The parsing process recognizes the following keywords (case insensitive): 1) `owl:class`, 2) `rdfs:subclassof`, 3) `owl:objectProperty`, 4) `owl:functionalProperty`, 5) `owl:transitiveProperty`, 6) `owl:Restriction`, 7) `owl:onProperty`, 8) `owl:SomeValuesFrom`, 9) `owl:AllValuesFrom`, 10) `owl:hasValue`, 11) `owl:equivalentClass`, 12) `owl:unionOf`, and 13) `owl:intersectionOf`.

After recognizing one of those keywords, the parser separates the line into blocks. For example, the line “<owl:Class rdf:id=“pizza”></owl:Class>” can be separated into two blocks, `<owl:Class rdf:id=“pizza”>` and `</owl:Class>`. The parser processes each block and takes any necessary actions on the parsing stack. The parsing stack contains each block’s compressed information. Each entry of the stack might contain one of the following pieces of information: 1) class, `class_id`, 2) subclass, `class_id`, 3) restriction, `class_id`, 4) property, `property_name`, and 5) valuesfrom, `class_id`.

The above italicized words represent the integer or string which needs to be filled in. The class id of each of the first three pieces represents the index of the current most outside class in the `nodesList`. The class id of the valuesfrom is the index of the destination class in the `nodesList`. Whenever the parser reads an ending block with the
format </ ... >, it pops the top entry off the stack. Many checks are made to handle unexpected situations and reported with error messages if problems are found in the parsing of the OWL source file. The following example illustrates parsing several lines of an OWL class definition given below in Figure A-5 Snapshot of a simple 7-line OWL code from people+pets.owl:

```xml
<owl:Class rdf:about="#bus+company">
  <rdfs:label>bus company</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#company"/>
  </rdfs:subClassOf>
</owl:Class>
```

**Figure A-5** Snapshot of a simple 7-line OWL code from people+pets.owl

The parsing process is as follows:

Line 1: Encounter a class definition. Check if the nodesList already contains it. If not, add it and get its new index in list; otherwise just get its index in list. Push “class,11” to the parsing stack. Here we assume the new class’ index in nodesList is 11. The parsing stack becomes:

![Figure A-6 The Parsing Stack after reading the first line](image)

Line 2: Reads the label of this class. Add it to labelList if the user has selected class labels to be used for node labels in the visualization.

Line 3: Skip this line, since there is no keyword.

Line 4: Encounter a subclass definition. Get the string on top of the stack and get the class index in the string. Push “subclass,11” to the stack. The new stack is becomes:

![Figure A-7 The Parsing Stack after reading the fourth line](image)

Line 5: Encounter another class definition. Check if it is in the nodesList. If not, add it to the nodesList and get the index; otherwise just get the index. Since this line contains
a complete block of class definition, `<owl:Class rdf:about="#company"/>`. So there is no need to push another string to the stack. But get the top of the stack and know that it is time to create a subclass relationship, so create this relationship in the hashtable.

Line 6: Reads the ending block of rdfs:subClassOf, so check if the top of the stack contains “subclass”. If so, pop the stack; otherwise report an error. The new stack becomes:

![Diagram of the Parsing Stack after reading the sixth line](image)

Figure A-8  The Parsing Stack after reading the sixth line `<rdfs:subClassOf>`

Line 7: Reads the ending block of owl:class, so check if the top of the stack contains “class”. If so, pop the stack; otherwise report an error. The stack is empty.