ABSTRACT

**ONTOSelf+TQ: A Topology Query System for OntoSELF**

by Zhisong Pei

Research in ontology visualization provides software tools for ontology users to understand the structure of and to navigate through the complexity of a large ontology. OntoSELF was previously developed to provide a sophisticated user the ability to combine various weighting and filtering options to create a 3D ontology visualization. The inexperienced user, however, may have difficulty using OntoSELF since it requires mastery of technical details to produce the desired visualization. To allow novice users to easily perform topology understanding tasks on ontologies, the topology query (TQ) system OntoSELF+TQ has been developed. Users of OntoSELF+TQ can specify characteristics and constraints on components of the ontology to produce a visualization that helps them better understand the topology of the ontology. The querying capabilities of OntoSELF+TQ are demonstrated and evaluated through a small user study and system performance evaluations that exercise a wide variety of queries for numerous tasks common to understand ontologies.
OntoSELF+TQ: A Topology Query System for OntoSELF

A Thesis

Submitted to the Faculty of Miami University

in partial fulfillment of
the requirements for the degree of Master in Computer Science

Department of Computer Science and System Analysis

by

Zhisong Pei

Miami University

Oxford, Ohio

2009

Advisor______________________________________

Valerie Cross, PhD.

Reader______________________________________

James Kiper, PhD.

Reader______________________________________

Eric Bachmann, PhD.
# TABLE OF CONTENTS

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

2 Background Concepts ............................................................................................................................. 3
   2.1 Ontologies ........................................................................................................................................... 3
   2.2 Query Language ................................................................................................................................. 5

3 Background Research ............................................................................................................................. 6
   3.1 OntoSELF .......................................................................................................................................... 6
   3.2 OntoCAT .......................................................................................................................................... 8

4 Related Research ..................................................................................................................................... 12
   4.1 OntoQuest ....................................................................................................................................... 12
   4.2 DAG Pattern Query Language ........................................................................................................... 13
   4.3 Graph Task Taxonomy ........................................................................................................................ 16
   4.4 Low-Level Analytic Activity ............................................................................................................. 18
   4.5 Empirical Comparison of Visualization Systems ............................................................................... 19
   4.6 User Experiments with Tree Visualization Systems .......................................................................... 20

5 OntoSELF+TQ Query Design .................................................................................................................. 20
   5.1 Query Language Basics ..................................................................................................................... 20
      5.1.1 Topology Vocabulary ................................................................................................................. 23
      5.1.2 Query Structure .......................................................................................................................... 23
   5.2 SELECTs in TQ .................................................................................................................................. 24
      5.2.1 SELECT with Whole Ontology Context – SELECT Type 1 ....................................................... 24
      5.2.2 SELECT with Specific Node Context – SELECT Type 2 ........................................................... 28
      5.2.3 SELECT with a Nested Query Context – SELECT Type 3 ....................................................... 31
   5.3 OntoSELF+TQ User Interface Design ............................................................................................... 33

6 OntoSELF+TQ Implementation ................................................................................................................. 39
   6.1 Modification Original OntoSELF ...................................................................................................... 39
   6.2 User Interface Processing Architecture ............................................................................................. 42

7 Evaluation .................................................................................................................................................. 45
   7.1 Verification Evaluation ....................................................................................................................... 46
   7.2 Demonstration of Complex Query Capability .................................................................................... 56
      7.2.1 SELECT with the Context of the Whole Ontology ................................................................. 57
      7.2.2 SELECT with Specific Node Context .......................................................................................... 75
7.2.3 SELECT with a Nested Query Context -----------------------------------91
7.2.4 CropCircle Topology Queries ------------------------------------------108
7.3 Topology tasks using Graph Tasks Analytic Activity----------------------110
7.4 User Evaluation --------------------------------------------------------113
8. Conclusion and Future Work ---------------------------------------------119
9. Reference ----------------------------------------------------------------122
Appendices ---------------------------------------------------------------124
  Appendix I VTK Installation ----------------------------------------------124
  Appendix II Human Research Application ----------------------------------126
List of Tables

Table 1 Outline of Select type 1 – Context Whole Ontology .................................................. 25
Table 2 Outline of Select type 2 – Specific Node context .......................................................... 29
Table 3 Timing Comparison with CropCircle ............................................................................. 117
List of Figures

Figure 2-1 A Simple Terrorism Ontology ................................................................. 4
Figure 3-1 OntoSELF User Interface ........................................................................ 7
Figure 3-2 Ontology Analysis tool ............................................................................ 10
Figure 4-1 User interface of OntoQuest .................................................................. 12
Figure 4-2 Sample DAG ....................................................................................... 13
Figure 4-3 Corresponding DAGs to various patterns .............................................. 14
Figure 4-4 Sample fragment of Gene Ontology ..................................................... 15
Figure 5-1 Outline of the classification for graph objects ...................................... 21
Figure 5-2 OntoSELF+TQ User Interface ............................................................... 33
Figure 5-3 File Selection for visualization ............................................................. 34
Figure 5-4 Parallel condition panel for OntoSELF+TQ .......................................... 35
Figure 5-5 Error range input for parallel condition ................................................ 36
Figure 5-6 Error object input for parallel condition ................................................. 36
Figure 5-7 Interface after opening the UMLS ontology file .................................... 37
Figure 5-8 “Nested Query” panel for OntoSELF+TQ ............................................... 37
Figure 5-9 Nested query panel one for OntoSELF+TQ .......................................... 38
Figure 5-10 Nested query panel two for OntoSELF+TQ ......................................... 38
Figure 6-1 UML Class Diagram of OntoSELF+TQ ............................................... 39
Figure 6-2 Processing flow of GUI ......................................................................... 42
Figure 7-1 The overall visualization of UMLS Semantic Network Ontology ........... 45
Figure 7-2 The result for roots query ....................................................................... 46
Figure 7-3 The result for deepest level query ......................................................... 47
Figure 7-4 The result for query - children of “Full_Formed_Anomalical_Structure” .... 47
Figure 7-5 The result for query - descendants of “Spatial_Concept” ....................... 48
Figure 7-6 The result for query - Siblings of “Animal” ........................................... 48
Figure 7-7 The result for query - Parents of “Alga” ................................................. 49
Figure 7-8 The result for query - Ancestors of “Regulation_or_Law” ...................... 49
Figure 7-9 The result for query - Leaves of “Animal” .............................................. 50
Figure 7-10 The result for query - Path(up) of “Occupation_or_Discipline” .......... 50
Figure 7-11 The result for query - Path(down) of “Occupation_or_Discipline” ....... 51
The result for the second path query
Nodes WHERE Ancestors between 3 and 3
Path WHERE Length betweens 2 and 2.

The result for the third leaves query
Children of ―Entity‖ WHERE Hub betweens 7 and max.

The result for the second leaves query
Children of ―Concept_Entity‖ WHERE Children betweens 2 and max and Height between 2 and max

The result for the third path query
No result message for the query

The result for the second path query
Children From Children of Spatial_Concept

The result for the first path query
Level WHERE Width betweens 4 and 4

The result for the second node query

The result for the third node query

The result for the first root query

The result for the second root query

The result for the third root query

The result for the fourth root query

The result for the first path query

The result for the second path query

The result for the third path query

The result for the first level query

The result for the second level query
Figure 7-42  The result for the third path query................................................................. 79
Figure 7-43  The result for the fourth path query .............................................................. 80
Figure 7-44  The result for the first sibling query .............................................................. 81
Figure 7-45  The result for the second sibling query ........................................................... 82
Figure 7-46  The result for the first children query ............................................................. 83
Figure 7-47  The result for the second children query ......................................................... 84
Figure 7-48  The result for the third children query ............................................................ 85
Figure 7-49  The result for the fourth children query .......................................................... 86
Figure 7-50  The result for the query level 7 subtree of “&numerics;Quantity” .................... 87
Figure 7-51  The result for the query children of “&numerics;Quantity” ............................ 88
Figure 7-52  The result for the first subtree query .............................................................. 89
Figure 7-53  The result for the second subtree query .......................................................... 90
Figure 7-54  The result for the first leaves query ................................................................. 91
Figure 7-55  The result for the second leaf query ................................................................. 92
Figure 7-56  The result for the query 7.2.3 I-a ................................................................. 93
Figure 7-57  The result for the query 7.2.3 I-b ................................................................. 94
Figure 7-58  The result for the query 7.2.3 I-c ................................................................. 95
Figure 7-59  The result for the query 7.2.3 I-d ................................................................. 96
Figure 7-60  The result for query 7.2.3I-e ................................................................. 97
Figure 7-61  The result for query 7.2.3 II-a ................................................................. 98
Figure 7-62  The result for query 7.3.3 II-b ................................................................. 99
Figure 7-63  The result for query 7.2.3 II-c ................................................................. 100
Figure 7-64  The result for query 7.2.3 II-d ................................................................. 101
Figure 7-65  The result for query 7.2.3 II-e ................................................................. 102
Figure 7-66  The result for query 7.2.3 II-f ................................................................. 103
Figure 7-67  The result for query 7.2.3 II-g ................................................................. 104
Figure 7-68  The result for query 7.2.3 II-h ................................................................. 105
Figure 7-69  The result for query 7.2.3 II-i ................................................................. 106
Figure 7-70  The interface for level 1 of 7.2.3 III-a ............................................................. 107
Figure 7-71  The main interface for 7.2.3 III-a ................................................................. 107
Figure 7-72  The result for query 7.2.3 III-a ................................................................. 108
Figure 7-73  The result for query 7.2.3 III-b ................................................................. 108
Figure 7-74  The result for the sixth Topology understanding query .................................. 114
Figure 7-75 The result for the seventh Topology understanding query............................. 115
Figure 7-76 The result for the eighth query ................................................................. 116
Figure 7-77 The student’s evaluation result for the bushiest node query ....................... 118
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. James D. Kiper and Dr. Eric Bachmann for their guidance and support.

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. Mufit Ozden, Mrs. Patricia Otto and Mrs. Monica Lea Baxter.

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

Ontologies are becoming increasingly important in fields such as knowledge management, information integration, cooperative information systems, information retrieval and electronic commerce (Baader et al. 2003). A large number of ontologies have been developed for use in various domains. Knowledge engineers are turning to ontology re-use in order to speed up the development process of ontologies and assist with the challenging task of finding and adapting existing ontologies to meet their needs. To help users analyze and understand the details of ontologies, multiple ontology visualization tools have been introduced, some that produce only 2D visualizations and others that perform 3D visualizations.

The focus of this thesis research is to improve one such 3D ontology visualization tool called OntoSELF (Somasundaram 2007) (Yu 2008) by adding a query language interface to allow a user to examine the hierarchical structure and content of an ontology. The original OntoSELF was developed to permit a knowledgeable user to examine the structure of a complex ontology by combining various weighting and filtering settings. Its flexibility permits the user to inspect the ontology’s topology and then further focus on ontology areas that the user finds of special interest based on his or her better understanding of the structure of the ontology. OntoSELF+TQ, the result of this thesis research, adds a topology query (TQ) language and user interface that allows users to more easily take advantage of the powerful 3D visualization offered by OntoSELF and extends OntoSELF with additional visualization features that enhance the users understanding of the ontology structure.

Generally, one could evaluate an ontology through understanding its topology, i.e., its structure. Based on studying the topology of an ontology that is being considered for re-use, users can gain more insights into whether the ontology is a suitable starting point for adaptation for another project. Another aspect of topology understanding is that it has been used in the evaluation of ontology visualization systems in order to compare one visualization system to another (Wang and Parsia 2006). For example, the topology task to find the bushiest (defined as the greatest number of children) child node for a given node was used in the evaluation of CropCircles. A user was given this task and asked to find the answer by viewing and searching a visualization of the ontology. The visualization tool allowed zooming in/out and rotating of the ontology. This approach of using topology understanding to evaluate ontology visualization
software was also considered for the evaluation of OntoSELF. But as the OntoSELF software was being developed, it was discovered that by applying OntoSELF’s weighting and filtering functions on the metrics that it maintains for each node and by using the graph representation of an ontology, determining the answer to such topology understanding tasks could be automatically produced so that the user would not have to visually search through a complex visualization of the ontology using zooming or rotating in order to locate the bushiest child node.

Although OntoSELF has been shown to accomplish topology understanding tasks through its weighted layout and filtering capabilities, it requires the user to thoroughly understand these capabilities in order to successfully combine them to produce the result for a specific topology understanding task. Also, it might not be possible to perform some topology understanding task by a combination of the existing capabilities of OntoSELF. Such limitations reduce OntoSELF’s usefulness as a 3D ontology visualization tool for the inexperienced users. This thesis research has developed a topology query system for ontologies such that users can create English-like queries through a user query template interface. These queries are then translated into the appropriate weighting and filtering criteria based on the given context of the query. Such a query produces a 3D visualization that presents the answer in a transparent manner. The query language eliminates the need for the user to understand the details of OntoSELF’s filtering criteria, multi-step filtering capability, and layout weighting functions in order to produce an answer for a topology understanding task.

The main contribution of this thesis is the development of a topology query language that can be used to analyze and explore the structure of a large ontology which is viewed as a graph. The query language has the ability to focus on different contexts of the ontology and provides three different kinds of SELECT based on the users’ needs. A search of the research literature did not reveal any 3D ontology visualization systems which have incorporated a query language on the objects and attributes of those objects that exist in the 3-D graph representation of an ontology. This thesis provides a novel approach to performing topology understanding tasks as well as overview browsing tasks on a large ontology. It eliminates the need for tedious searching through the visualization of the ontology and instead provides a focus visualization that produces the answer to the topology understanding task.

This thesis is organized as follows: Chapter 2 describes basic concepts for ontologies and the query language. Chapter 3 briefly presents two master theses and their resulting software,
OntoSELF and OntoCAT (Pal, 2006). The OntoSELF software serves as the basis for this thesis and is extended in order to support the topology querying capability on ontologies. Related research is discussed in Chapter 4, including recent query system studies and graph task taxonomies. Chapter 5 presents the design of the query language including the kinds of objects, attributes and values that can be used within a query and describes the user interface used to create the query. An overview of the implementation of OntoSELF+TQ is provided in Chapter 6. An important aspect of this thesis research is the evaluation of OntoSELF+TQ which is presented in Chapter 7. This thesis concludes with Chapter 8 providing a summary and outlining future enhancements that would improve the functionality and features of OntoSELF+TQ.

2 Background Concepts

2.1 Ontologies

In philosophy, ontology is the study of being or existence. In information science, ontology is defined as vocabulary and content theory (Chandrasekaran et al., 1999). An ontology specifies a representation vocabulary which has some domain or subject matter. The representation vocabulary provides a set of terms to represent the facts in such a domain and the body of knowledge using that vocabulary is a collection of facts about a domain. Ontologies are used to identify specific classes of objects and relations that exist in such a domain. The following sample ontology graph from (http://seneca.myweb.uga.edu/GlobalInfoSys/Assignment3/Assignment3.html) provides a small graphical presentation of an ontology.
Figure 2.1 A Simple Terrorism Ontology

From the above graph, this sample ontology has a set of vocabularies which represent a domain of terrorism, such as terrorist organization, person, and terrorist etc. Various domain terms have certain relationships, i.e., “Terrorist_organizaton” “hasMember” “Person.”

The motivation to develop an ontology is that it clarifies the structure of knowledge and enables knowledge sharing. Ontologies enable shared understanding and communication between people with different needs and viewpoints arising from their particular contexts. They create an integration environment for different software tools, and support the design and development of the software systems themselves (Uschold and Gruninger 1996).

According to (Van Heijst et al. 1997), ontologies can fall into two dimensions: the amount and type of structure of the conceptualization and the subject of the conceptualization. Later researchers classify ontologies into intensional and extensional ones (Smith and Welty 2001). An intensional ontology indicates the ontology schema or definitions and an extensional ontology embodies the instances or occurrences of the concepts. Usually a knowledge base is composed of an intensional ontology and its relevant extensional ontology. In this thesis, the query language and its implementation are applicable to the intensional ontology.
2.2 Query Languages

Query languages are computer languages used to retrieve information from databases. The goal of a query task is to find a narrow set of items in a large collection that satisfy a well-understood information need (Marchionini 1995). Generally, query languages are divided into two categories based on whether they are database query languages or information retrieval query languages. For example, the well known structured query language SQL belongs to database query languages. SQL allows a user to describe the data the user wants returned by using a structured format to represent the query. This format includes specific keywords and kinds of objects that the user may specify in the query. The success of SQL has motivated the development of a structured query language that similarly defines the keywords and kinds of objects that may be used to query on the topology of an ontology. The Extensible Stylesheet Language (XSL) belongs to information retrieval query language. XSL uses a "pattern" language for information retrieval. This section briefly describes SQL querying capabilities. The design of the topology query language is described in Chapter 5.

A SQL query is simply constructed by using some query keywords such as SELECT, FROM, WHERE and ORDER BY and following the SQL syntax. The SELECT clause defines what columns or fields one wants to see in the desired results. The FROM clause defines from what table the columns reside in. The WHERE clause defines any special criteria that must be met in order for the data to be selected. The ORDER BY clause defines the order (ascending or descending and what field to order on) by which one wants to display the results. The only two query clauses that are required are SELECT and FROM, but they are almost always accompanied by the WHERE clause to restrict the amount of data retrieved and to present it in an orderly fashion (http://www.thundersoftware.com/churchdb/sqltutorial/sql_tutorial.html).

Summarization of large volumes of data can be achieved by applying aggregation functions, such as SUM, MAX, and AVE etc. Usually these functions are accompanied with SELECT clause. For example, SELECT MAX salary for employees WHERE they have been working for the company less than three years. The capabilities provided by SQL have served as a basis for developing the topology query language since many of the same capabilities are needed such as selection, restriction and summarization.
3 Background Research

A query system to help a user understand the topology of an ontology and to locate interesting concepts and relationships within an ontology requires having accessible as much metric information about the ontology as possible and displaying query result both visually and when needed textually to the user. In this section, the previous research that provides the foundation for developing the querying language and its implementation is described. First the OntoSELF 3D ontology visualization system is briefly overviewed. Then the OntoCAT (Pal 2006) system which produces a metric evaluation of an ontology is described.

3.1 OntoSELF

The newly developed 3D ontology visualization tool OntoSELF (Somasundaram 2007) (Yu 2008) employs additive line drawing algorithm with several weighting and filtering functions to assist users to better understand and analyze a target ontology by using 3D visualization techniques. The interface shown in Figure 3.1 is designed using Java Swing and users can select various options to visualize a target ontology. The selected ontology in OWL format is converted to the graph modeling language (GML) (Himsolt 1997) since that is the language that the graph visualization library (http://www.comp.leeds.ac.uk/djd/graphs/) and the Visualization Toolkit (VTK) (http://www.vtk.org) use for in the visualization process. Multiple ontologies are allowed to be selected at one time and the same options selected by the user are applied on each ontology.
OntoSELF employs several metrics to implement weighting and filtering functions. The weighting metrics are used to affect the position of the node in the 3rd dimension of the layout algorithm. The *Level value* represents the depth of the node in the ontology. A node’s *interval rank* is determined as the midpoint between the depth of the node (distance from the root) and the node’s level from the bottom node (typically a dummy node which all leaf nodes connect to). The level is calculated by subtracting the node’s height from the greatest depth M of the ontology, i.e., interval rank midpoint (depth + (M-height))/2. The *IC value* is the information content of a node with respect to a given ontology. A node’s *extent* is based on the number of descendants the node has. These same metrics may also be used in filtering the nodes that are to be kept in the visualization.

In addition to those metrics, several other parameters are provided for filtering purposes. The *Name* field allows the user to set a context for the other filtering options. The *Descendant* filters based on the number of descendants of a node. The *Children* corresponds to the number of nodes which are direct children of a node. The *Hub Value* is the total in and out degree for a node. The *Subtree Depth* represents depths of a target node’s subtree. It is always used in combination with
the Name filter so that a user can specify the depth of a desired node’s subtree. Certain tasks can be implemented by combining multiple filtering steps. For example, the task to find the number of descendants of a target node needs to be combined with the Name and Descendant filtering options. Name clarifies what the goal node is and Descendant explains which aspect is being investigated. The user can also choose to execute the task either on a single computer or using the cluster computers.

Once the Visualize button is pressed, OntoSELF performs a series of metrics calculations. The result of the execution is a GML file since GML supports attaching arbitrary information to graphs, nodes and edges and is able to emulate almost every other graph format. This file can then be used as the input to the Tool Command Language (TCL) (http://www.tcl.tk) script that executes calls to various routines in the VTK.

In OntoSELF, the 3D Cone layout algorithm is used to visualize the ontology. The input ontology specified in the GML file is first converted to a spanning tree. The tree is then passed to the span layout model for calculating the level within the third dimension for each node and identifying a bend in the edges that connects nodes in non-adjacent levels. The user selected weighting function determines the calculation of the level within the third dimension.

Lastly the tree is passed to the cone layout algorithm which positions the nodes in 3D. Once the placement is done, the aspect ratio (ratio of the length of the longest side to the length of the shortest side of the smallest rectangle covering the graph) is calculated for the current tree for being compacted. At this time the tree can be rendered in VTK for drawing its nodes and edges.

OntoSELF has been evaluated by using two approaches. One is a qualitative approach where visualization on several different ontologies with different properties and sizes are tested. Some of those ontologies are from the biomedical domain such as the Gene Ontology and some are from the physical science domain such as the Nasa JPL Sweet Ontology. The other evaluation was based on an experiment carried out to evaluate the CropCircles ontology visualization system by using several topology understanding tasks. The experimental results demonstrate that OntoSELF can provide purposeful visualization and flexible inspection of large-sized ontologies.

### 3.2 OntoCAT

Metrics can provide a convenient and quick evaluation of an ontology. Such metrics often attempt to discover features of an ontology that are independent of the actual ontology language used to represent (Vrandeˇci´c and Sure 2007), i.e., they are based on the structure of the
ontology when it is viewed as a graph. In (Tartir et al 2005), the metrics are separated into two categories: schema metrics and instance metrics. Schema or intensional metrics evaluate ontology design and its potential for rich knowledge representation. Instance or extensional metrics evaluate the placement of instance data within the ontology and the effective population of the ontology’s knowledge model. Currently OntoSELF is used to visualize the intensional ontologies only. It employs several different intensional metrics to implement weighting and filtering functions that are used in creating ontology visualizations. The ontology consumer analysis tool (Cross and Pal 2006), however, provides a wide range of ontology metrics for users to evaluate a target ontology for both intensional and extensional ontologies.

Intensional metrics focus on classes, subclasses and their properties of an ontology. Extensional metrics are determined based on the actual instances of classes and relationships defined in the intensional ontology. To facilitate user interaction, the metrics could be further divided into size and structural categories. The resulting metrics are, therefore, grouped into four types: size intensional, size extensional, structural intensional and structural extensional.

The size intensional metric is calculated over the entire ontology without concept specification. If a particular concept is chosen, the metric is calculated on the tree specified by a concept such as its root. The structural intensional metric is similar to the size intensional one. The structural extensional metric is calculated on both the specific root concept and the specified relationships. A complete detailed list of all the metrics can be found in (Pal 2006). Here are a few examples in each category.

**Size Intensional:**

\[ i\text{Cnt}(C)(c_j\text{-root}) = \text{the number of classes for the sub-tree at the selected class } c_j . \]

\[ i\text{Cnt}(P)(c_j\text{-root}) = \text{the number of properties defined for the entire sub-tree at class } c_j . \text{ A property may be inherited from its parents. Only new properties are counted for each class.} \]

\[ i\text{Cnt}(R) = \text{the number of relationships defined for the entire intensional ontology. A relationship is a special kind of property that has a class as its range.} \]

**Size extensional:**

\[ e\text{Cnt}(c_j) = \text{the number of object occurrences for class } c_j \]
eCnt(C) = ∑ \(_j\) eCnt(c\(_j\)), the total number of object occurrences in the ontology

eAvgCnt(C) = eCnt(C)/iCnt(C), the average number of occurrences for all classes

eMaxCnt(C) = \(\max_i\{eCnt(c_i)\}\) and identify eMaxCntClass, i.e., the class with the maximum number of occurrences in the ontology

eCnt(r\(_i\)) = the number of occurrences for relation r\(_i\)

**Structural intensional:**

iCnt(Roots) = number of roots in the ontology.

iCnt(leaves(cj-root)) = number of leaf classes of the sub-tree at the selected class cj

iMaxDepth(cj-root) = \(\max_j\{\text{depth}(\text{leaf}_{ij})\}\), the maximum depth of the sub-tree at the selected class c\(_j\) and return the class of the leaf at the maximum depth

**Structural extensional:**

eCnt(roots) = number of root occurrences for all root classes

eCnt(leaves(cj-root )) = number of leaves for all occurrences of class c\(_j\)-root\(_i\).

eCnt(leaves(cj-root (occ)) = number of leaves for specified occurrence of c\(_j\)-root\(_i\).

eMinDepth(cj-root(occ)) = \(\min_i\{\text{depth}(\text{leaf}_{ij} (\text{occ}))\}\), the minimum depth of the sub-tree at the selected root occurrence of root class c\(_j\) and return the leaf occurrence(s) at the minimum depth.

eWidth(depth\(_k\)(cj-root)) = (∑ \(_i\) eWidth(depth\(_k\)(cj-root (occ\(_i\))))), the number of instances at depth k for all occurrences of the selected root class c\(_j\)

A Protégé tab plug-in has been developed for implementing above metrics and its interface is shown in Figure 3.2.
OntoCAT has been used on four different ontologies from varying domains: WordNet, UMLS, UNSPSC, and eCl@ss. OntoCAT produced a variety of metric reports for the user; however, the user could benefit from improvements that would summarize or identify unique features about the ontology and its classes and relationships. A user query interface could be extremely helpful to permit entering a query about the ontology and then provide a specific answer so that the user is not required to set parameters for metric calculations and then look through reports to find the information being sought.

The discussion on OntoSELF and OntoCAT reveal the need for a friendly query interface to permit easier and more understandable use of the numerous capabilities provided in both systems.
4 Related Research

Although some ontology editors provide a few simple metrics such as the number of classes and properties existing in an ontology, the research literature does not provide examples of systems with the objective of answering queries about the topology or metrics of an intensional ontology. Examples of query systems on information stored in the ontology and on graph structures and pattern matching on graphs are described in this section since they have objectives which are similar to this thesis research. Research in graph visualization systems has also provided examples of tasks that are used to evaluate the performance of the system. This section describes research that helped inspire some of the features and capabilities of the OntoSELF+TQ query system on the structure and metrics of an ontology.

4.1 OntoQuest

OntoQuest (Chen et al. 2006) is a system which mainly provides powerful query and reasoning utilities for users to explore the data in a target ontology. It also supports storage for OWL ontologies and bulk editing and updating of instances.

OntoQuest has a friendly interface with a split pane as show in Figure 4.1. On the right side, a scrollable pane is provided for displaying an activity history, such as querying and reasoning tasks. The left side allows users to switch among a set of tabbed panels each assigned a specific functionality. Users who are not familiar with ontology query language (e.g., RDQL) can use the fourth tabbed panel to explore ontologies and issue interrelated queries. Rather than provide a text field for users to type in a query, the interface pops up menus with the appropriate query options for the context. Two sub tabbed panels under the “GuidedExplore” panel list usual starting queries. The left sub panel gathers queries about the ontology metadata, such as version and creator. The right sub panel helps users who do not have knowledge about the content of the ontology to get all the classes, properties, or instances with URIs matching a given string pattern. Based on the initial queries, further exploration can continue with the user being prompted with proper popup menus presenting various query options, such as getSuperClasses, getNeighborInstances, or getSubproperties. For Example, if a user first clicks on “getAllClasses()” on a top menu to initiate the exploration, the user then picks one of interest from the returned classes and right-clicks to pop up a menu presenting queries of the third
category – inquiry for class. Suppose that ‘getInstances()’ was clicked next and all instances of that class are returned. In this case, the user may issue further inquiries for a particular instance, and so on. Most users do not know how to request information at the beginning. One may start with some simple questions which would lead to broad ones. Since those further queries often need the “context” from the previous ones, OntoQuest offers “back” function to allow users to return to any previous context.

![Figure 4.1 User interface of OntoQuest](image)

Besides the queries listed above, OntoQuest provides additional ones which are implemented by calling Jena (http://jena.sourceforge.net) APIs or OWL (http://www.w3.org/TR/2004/REC/owl-features-20040210) APIs to process interesting aggregate operations. For example, the semantics of groupInstancesByPValue (String p) is to group all the instances of a class c by their values of property p. To process this query correctly, all of the instances for each subclass of c need to be gathered. Then these instances need to be sorted and grouped by their p values.

### 4.2 DAG Pattern Query Language

Directed acyclic graphs pattern query language (DQL) (Gupta and Santini 2006) is developed for specially querying a database of directed acyclic graphs (DAG). It applies extraction function for nodes, paths and subgraphs from DAGs, and allows combining various structures for further
construction. Rather than limit querying abilities, DQL aims to provide more powerful and complicated querying facilities to explore a large DAG-structured ontology.

The pattern language DP is the theoretical foundation for DQL. Figure 4.2 shows a DAG (Directed Acyclic Graph) where the pattern \((v=1)\) matches a set of nodes for which the value of the variable is 1. This could be node \([1, 1]\), \([7, 1]\), \([8, 1]\) or \([2, 1]\). The symbol \(\rightarrow\) denotes an edge between left node and right node. For example \((v=3) \rightarrow (v=1)\) matches the edge \([5, 3] \rightarrow [7, 1]\), and \((v = 1) \left[ \neg(v = 2) \right]^* \neg(v = 1)\) where \(^*\) represents Kleene star matches \([1, 1] \rightarrow [3, 2] \rightarrow [7, 1], [1, 1] \rightarrow [3, 2] \rightarrow [2, 1], [1, 1] \rightarrow [3, 2] \rightarrow [4, 2] \rightarrow [8, 1]\).

![Figure 4.2 Sample DAG (Gupta and Santini 2006)](image)

DQL employs the pattern concept where a pattern \(\pi\) is defined by the following rules:

1) A predicate \(C\) is a pattern; in particular \(t\) (the value “true”) is a pattern;
2) If \(\pi_1 \ldots \pi_n\) are patterns, then the following formulas are also patterns:
   i) \(\pi_1 \pi_2\);
   ii) \(\pi_1[-\pi_2]^*\) or \([-\pi_2]^* \pi_1\);
   iii) \(\{\pi_1-, \ldots, \pi_n-\}\) or \(v(\pi_1-, \ldots, \pi_n-\};
   iv) \(\pi_1 | \ldots | \pi_n\);
3) If \(\pi\) is a pattern and \(v\) a variable name, then \(v: \pi\) is a pattern;

These cases are illustrated in Figure 4.3.
Figure 4.3 Corresponding DAGs to various patterns (Gupta and Santini 2006)  
(a) $\pi_1 - \pi_2$, (b) $\pi_1 [-\pi_2](2, 4)$ (2 and 4 represents the second and fourth node), (c) $\tau[-\pi_1, -\pi_2]$ and (d) $\{\pi_1-, \pi_2-\}\nu$

DQL also utilizes the monoid structure. A monoid of type $T$ is a pair $(A, B)$, where $A$ is an associative function of type $T \times T \rightarrow T$ and $B$ is the left and right identity of $A$. For example, $A$ and $B$ are separately $\cup$ and $\{\}$ for a monoid of set($\alpha$). For an expression of the form $\omega \{e| q_1, \ldots, q_n\}$ where $\omega$ denotes monoid and $q$ represents generators which may have one of the following forms:

1) $q_i \equiv x_i \leftarrow A$, where $A$ is a constant or another monoid comprehension

2) $q_i \equiv g \vdash \pi(y_1,\ldots,y_m)$, where $y$'s share the free variables of pattern $\pi$ and $g$ is the collection of variables and constants collected from prior environments of computation (q’s);

3) $q_i \equiv P(y_1,\ldots,y_m)$, where $P$ is a predicate and $y$’s are the free variables of prior environments;

In addition to standard monoids, $\omega$ could be graph monoids. Typical ones include merging two graphs, the smallest or largest graph which contains two subgraphs.

The Gene Ontology (GO) (www.geneontology.org) contains several DAG based components, such as biological processes (BP), molecular functions (MF) and Cellular components (CC). Figure 4.4 shows a fragment of the BP DAG. One may have a number of
different types of queries of this sample DAG. One of them could be “Which biosynthesis processes under lipid biosynthesis are also classified as amine biosynthesis?” By applying DQL, this query can be formulated as $\bigcup \{x | \text{substr(name, “lipid_biosyn”), substr(name, “amin_biosyn”) \# x : substr(name, “biosyn”) } \leftarrow \text{GO}\}$ (# represents $[-t] * - or - [t-]^*)$.

![Sample fragment of Gene Ontology (Gupta and Santini 2006)](image)

**Figure 4.4 Sample fragment of Gene Ontology (Gupta and Santini 2006)**

The planning algorithm which includes operators such as path, merge, apply etc. can be applied to query pattern. The complete algorithm is given as below:

\[
\text{plan}(z : C_1 \rightarrow C_2, g, e) = \\
u_1 = \sigma(g, C_1); \\
u_2 = \sigma(g, C_2); \\
p_{12} = \text{apply[set]}(u_1, \text{fun } x_1 \rightarrow \text{apply[set]}(u_2, \text{fun } x_2 \rightarrow \text{path}(x_1, x_2)); \\
e = \text{apply[set]}(u_1, \text{fun } x_3 \rightarrow (z \rightarrow x_3)) \text{ (where } g \text{ is the input graph and } e \text{ is the environment).}
\]

### 4.3 Graph Task Taxonomy

Currently, a number of graph visualization systems have been developed, and there is a need to have a standard evaluator for those systems. In (Lee et al. 2006), the authors propose a set of tasks commonly encountered while analyzing graph data. Based on these tasks, designers can learn how to improve their systems and consumers can estimate the quality of a tool.
The majority of data tasks are related to basic graph objects, such as nodes, links, paths, graphs, groups, connected components and clusters. Examples of one graph-specific task and two general tasks are as follows:

- Find adjacent nodes of a given node;
- Quickly review the list of items;
- Given multiple sets of nodes, perform set operations on them.

Integrations of Amar’s (Amar et al. 2005) low-level visual analytic tasks with the common graph tasks above can be categorized into the following four groups:

1. Topology-based tasks
2. Attribute-based tasks
3. Browsing tasks
4. Overview tasks

For topology-based tasks, four subtasks, “adjacency”, “accessibility”, “common connection” and “connectivity” are included. “Adjacency” means direct connection and its general descriptions could be “Find the set of nodes adjacent to a node” or “Find a node which has a maximum number of adjacent nodes.” “Accessibility” represents direct or indirect connection and its general descriptions could be “Find the set of nodes accessible from a node” or “Find the set of nodes accessible from a node where the distance is less than or equal to n.” The general descriptions for “common connection” could be “Given nodes, find a set of nodes that are connected to all of them.” The general descriptions for “connectivity” could be “Find the shortest path between two nodes” or “Identify connected components.”

For attribute-based tasks there are the tasks, “on the nodes” and “on the links.” The general descriptions for “on the nodes” could be “Find the nodes having a specific attribute value” or “Review the set of nodes.” The general descriptions for “on the links” could be “Find the nodes connected only by certain types of links for a given node.”

“Follow path” and “revisit”, are under the browsing tasks. “Follow path” means follow a given path. A sample task could be “A user looks into A’s friend B, B’s friend C, and C’s friend D.” The general descriptions for “revisit” could be “Return to a previously visited node.”

For overview tasks, each task can get estimated values quickly. For example, users may be asked to estimate the size of the social network. Sometimes it is more important to estimate the answer than to get an accurate one.
4.4 Low-Level Analytic Activity

Most task taxonomies focus on the design of particular representations rather than the facilitation of user analytic activity. Almost two hundred sample questions from students who analyzed five data sets from different domains and by using an affinity diagramming approach, were used to summarize ten low-level analytic tasks which mainly concern people’s conduct on querying data [Anmar et al. 2005]. The tasks are listed as follows: “retrieve value”, “filter”, “compute derived value”, “find extremum”, “sort”, “determine range”, “characterize distribution”, “find anomalies”, “cluster”, and “correlate”. “Retrieve value” is to find attributes of a set of given cases. “Filter” is to find data cases satisfying some given concrete conditions. “Compute derived value” is to compute an aggregate numeric representation of a set of given data case. “Find extremum” is to find data cases possessing an extreme value of attribute over its range within the data set. “Sort” is to arrange a set of given data cases according to some ordinal metric. “Determine range” is to find the span of values within a set of given data cases. “Characterize distribution” is to characterize the distribution of a group of interesting attributes over the given set of data cases. “Find anomalies” is to identify any anomalies within a given set of data cases with respect to a given relationship or expectation, e.g. statistical outliers. “Cluster” is to find clusters of similar attribute values for a set of given data cases. “Correlate” is to determine useful relationships between the values of two attributes for a set of given data cases.

It is hard to have a taxonomy that covers all queries on a domain. Thus, low-level tasks can be combined to define other tasks. Many queries, however, do not fit into their taxonomy, such as mathematical actions, high-level questions and uncertain criteria. For example, the following question involves the mathematical comparison operation: “Compare the average MPG of American and Japanese cars.” This question utilizes Retrieve Value and Compute Derived Value primitives, respectively, followed by a mathematical comparison operation. Comparing the data requires a mathematical calculation and it can be viewed as being a fundamental cognitive action. Some questions require systems to possess some reasoning functions, and such a query is beyond low-level analysis. Other questions contain uncertain criteria, and those questions are beyond the proposed primitives. For example, the question “What are the characteristics of the most valued customers?” may be answered by supposing the existence of some black-box aggregation function, there may be other ways to answer the question, such as use of a distribution or
clustering method. This question also involves a value judgment that is beyond the proposed primitives.

The proposed taxonomies share some tasks with ones that are recommended by Wehrend and Lewis (Wehrend and Lewis, 1990). However, since data analysis is mainly focused on, several new analytic tasks, such as "find extremum" and "determine range", are developed. The taxonomies also can provide a basis for mapping low-level tasks to high-level tasks; however, how these knowledge precepts can map to lower-level concrete tasks for visualization systems is not examined. Because the sample questions were gotten from the students who were unable to generate advanced low-level tasks, experiments with professional analysts were necessary in order to suggest other kinds of tasks. This research may be useful to building a topology query system for ontologies since these kinds of analytic tasks can be used in characterizing the structure and content of an ontology.

4.5 Empirical Comparison of Visualization Systems

Three different data sets and 82 students are used to compare the three commercial information visualization systems: Eureka, InfoZoom and Spotfire (Kobsa 2001). Through these comparisons, four significant factors related to the quality of a system are proposed.

The aim of the experiment was to determine whether solving tasks in the three systems would differ with respect to solution times and accuracy. The selected tasks are generated by a brainstorming process and covered three domains. The attending students had at least one year of experience working with computers. The experiment assigned a particular visualization system to a special group of students, and students wrote down the required answers on answer sheets. Overall results indicate that all three systems have bad accuracy on those relatively simple tasks. Spotfire users gave the most correct answers with 75% accuracy.

Each system has its own interaction problems. Eureka users are confused by its hidden labels. They also have difficulties with 3+ attributes and correlation. InfoZoom subjects have troubles determining whether a correlation exists between two different attributes. Spotfire users have to spend much time on deciding on the right representation and are confused by using scatterplots.

Detailed analysis of results shows that a visualization system should be concerned with several factors. The two most relevant to this thesis research are: the properties of visualization and the operations that can be performed upon the visualization. These factors suggest that a
query capability for visualization systems is important to improving the users’ interactions with the produced visualizations.

4.6 User Experiments with Tree Visualization Systems

Five tree visualization systems Treemap 3.2, Sequoia View 1.3, Beam Trees, Star Tree Studio 3.0 and Tree viewer along with Windows Explorer serving as the baseline were compared with respect to speed, accuracy and user satisfaction. (Kobsa 2004) Based on the experimental results, several recommendations for visualization system design were recommended.

The fifteen tasks used in the experiment were created by an iterative brainstorming process and categorized into “structure-related tasks” and “attribute-related tasks”. Forty eight students who had at least one year of computer experience were selected to solve those tasks.

Treemap was shown as overall the best visualization system of these tree visualization systems, but video analysis on the tasks indicate that more search functionality is needed. The overall worst performer with respect to quantitative criteria for speed, accuracy, and user satisfaction was BeamTrees. Its very limited functionality includes the display of path and size information. Two extreme instances, Tree Viewer and BeamTrees, show us that query functionality is critical to the quality of a visualization system.

5 OntoSELF+TQ Query Design

A primary objective of OntoSELF+TQ is to allow a user to better understand the content and structure of an ontology. It does this by providing a template driven query interface that provides the user with a methodical yet also fairly flexible approach to query creation. This section describes first the important factors motivating the design of the query interface and then presents the details of the design of the template-driven query interface.

5.1 Query Language Basics

In designing the TQ language, an important early aspect of the design process was finding a comfortable compromise between the time intensive extreme of designing a complex query language such as SQL and the inflexible extreme of simply producing a strict set of report-like queries that did not allow any user creativity to mix, match, and combine features of topology objects. This section describes the important considerations leading to the design of TQ language interface.
5.1.1 Topology Vocabulary

The first task was to determine the vocabulary that the user has available to create a query on the structure of the ontology. Since the subclass/superclass structure of an ontology can be viewed as a hierarchical graph (the network is not considered here), the primary considerations are what kinds of objects, what attributes are associated with them, and what kinds of values can these attributes have. These considerations led to the development of three sub vocabularies for describing 1) objects, 2) attributes for those objects and 3) constraints on the values for those attributes. The detailed usage of this vocabulary is given in section 5.2. Here in this subsection, an overview and high level explanation is first provided.

![Diagram of graph objects classification]

**Figure 5.1 Outline of the classification for graph objects**

Since an ontology can be structured as a graph, the kinds of objects in the query are graph objects. These graph objects can be categorized as individual objects and group objects. Figure 5.1 sketches the outline of the classification details. Individual objects include the two primitive elements of a graph, nodes and links or edges. These individual objects can be associated with other individual objects and based on this association, they form a group object. For example, users often like to examine related nodes to a specific node and such related nodes are usually
referred to by the role they play relative to the specified node such as “parents”, “children”, “descendants”, or “siblings”, etc. Other group objects may have an association because they satisfy a certain property in the ontology, for example, “roots” and “leaves.” Other group objects are related because they make up some subcomponent of the ontology such as “level”, “path”, or “subtree” etc. Most of the standard graph object appellations for ontologies are employed in OntoSELF+TQ.

Graph objects have certain kinds of properties or attributes which are described by the attribute vocabulary. For example, each node in an ontology graph has a level. These attributes can have their values constrained by specific values or terms from the constraint vocabulary. Terms that might be used with level are “max” and “min” so that a query might, for example, find nodes that are at the maximum level in the ontology. All these nodes would have to be leaf nodes, but not all the leaf nodes in the ontology would necessarily be at the maximum level within the ontology. To be consistent, the “min” term can also be used with level so that a query might find nodes that are at the minimum level in the ontology. For this case, all these nodes would have to be root nodes in the ontology and all the root nodes in the ontology would necessarily be at the minimum level within the ontology.

The design of the TQ language permits multiple ways to produce the same query result since this variety provides flexibility to each user based on their background knowledge. Note also that the same vocabulary word can be used in two different contexts. This situation occurs especially with the group object vocabulary. For example the word “level” is used to represent 1) a group object, i.e., the set of all nodes at a specific level and 2) an attribute of a node, i.e., the level at which a particular node occurs within the ontology. Another example which is slightly different is the use of the word “children”. When it is used as a group object, it refers to the set of nodes that represent the children of a specified node. When it is used as the attribute of a node, it can be considered as prefixed with “the number of” and refers to the number of children a specified node has, i.e., not the actual children nodes.

An attribute may also have a descriptive set of terms associated with it. For example, the attribute “level” for a node uses the terms “max”, “min”, “greatest” and “least”. The difference between “max” and “greatest” is that “max” is always in the context of the whole ontology, i.e., it has a single value: the deepest level within the whole ontology. “Greatest” is within the context of a subset of the ontology on which the query is being processed. It represents a
satisfying value within the context of the ontology subset. A similar difference exists between “min” and “least.” The terms “min”, “max”, “greatest” and “least” are also used for the attribute “width” for the group object “level” and for the attribute “length” for the group objects associated with a “path”. It was felt that it would be easier to consistently use these terms instead of introducing terms such as “narrowest” and “widest” for the “width” attribute of the “level” object and “shortest” and “longest” for the “length” attribute of the “path” objects.

5.1.2 Query Structure

The vocabulary provides the basics regarding how the user can identify and describe objects that are to be located and then visualized within the ontology. How this vocabulary can be used to formulate a query is determined by the design of the query structure. One of the most popular and well known query languages for retrieving information is the Structured Query Language, SQL. OntoSELF+TQ has adopted syntax structure similar to SQL; however, the TQ language uses a template based approach to eliminate the need for parsing the query and to provide the user with ease in developing a query.

Following the SQL approach, TQ has three main keywords: SELECT, FROM, and WHERE that create the three clauses of the TQ query. Just as the SQL SELECT statement is used to select data from a SQL database, the SELECT in OntoSELF+TQ is used to select objects from the ontology that are to be part of the visualization result. The SQL FROM clause is used to select data from a particular subset of the database, i.e., a table, instead of the entire database. In OntoSELF+TQ, the FROM clause is used to define a subset of the ontology on which to apply the query, but differs from standard SQL in that the FROM clause is not required. When the FROM clause is not used, the complete ontology is used as context for the query.

The SQL WHERE clause is used to select data conditionally. The data values must satisfy certain constraints specified in the WHERE clause. In OntoSELF+TQ, it serves the same purpose so that the user can select only graph objects whose attributes satisfy the specified values in the WHERE clause. The number of conditions allowed in the WHERE clause is dependent on the type of object being selected. The multiple specified conditions use an AND logical connective. It was felt that the user could easily do multiple queries if an OR condition was needed and that the implementation and design of the user interface would be simpler and easier to understand using only the AND logical connective.
To summarize the query structure in OntoSELF+TQ follows the following format:

\[
\text{SELECT} \ <\text{object(s)}> \ \text{FROM} \ <\text{nested query}> \mid <\text{specific concept}> \ \text{WHERE} \ <\text{condition(s)}> \\
\]

The \text{FROM} clause allows two ways to specify the context within the ontology that is to be used when processing the query. The nested query context permits the creation of another query whose result is to be used as the context of the outer query. The specific concept context locates a concept within the ontology to be used as the context for query. The details of each kind of \text{SELECT} query are provided in section 5.2

5.2 SELECTs in TQ

The following sections present the three kinds of \text{SELECT} statements and proceeds in order of simplest to most complex.

5.2.1 SELECT with Whole Ontology Context – SELECT Type 1

This query allows a user to select objects with the complete ontology as the context for the query. No subset of the ontology is specified.

\[
\text{SELECT} \ <\text{object}> \ \text{WHERE} \ <\text{condition}> \\
<\text{condition}> \rightarrow <\text{attribute}> <\text{operator}> <\text{value}> \ \text{[AND} \ {<\text{condition}>}] \\
\]

Conditions, each on a different attribute can be logically connected. Although relational comparison operators are shown in the table to demonstrate that all these kinds of comparisons can be made, in order to simplify the TQ language, the format used for the \text{attribute} \text{operator} \text{value} structure is provided by simply using the “between” keyword. For example, level >=5 and level <= 8 is expressed as level between 5 and 8 where “between” is interpreted as inclusive. This template feature is also described in the User Interface Section 5.3

Table 5.1 shows the objects which can be selected when the context of the topology query is the complete ontology. Some of these objects further restrict the selection process since they identify a set of objects, i.e., leafs and roots which require certain values for attribute, for example, leafs have zero children. More explanation for each of the objects is provided following the table. A “Yes” in a table cell indicates that the attribute in its column header may
be used in the WHERE clause when selecting the object in its row header. All applicable attributes to a specific object can be combined. For example, the Root object has four applicable attributes from Table 5.1, which are “IC”, “Descendants”, “Height”, and “Children”. Therefore, for the Root object, the number of conditions in the WHERE clause can be up to a maximum of four. The resulting Root objects must satisfy all conditions that are used. The same rule is applicable to all other selectable objects.

<table>
<thead>
<tr>
<th>attribute operator</th>
<th>IC</th>
<th>IR</th>
<th>Level</th>
<th>Descendants</th>
<th>Ancestors</th>
<th>Children</th>
<th>Height</th>
<th>Hub</th>
<th>Length</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard values:</td>
<td></td>
<td></td>
<td></td>
<td>=</td>
<td>&gt;=</td>
<td>=</td>
<td>&gt;=</td>
<td>=</td>
<td>&gt;=</td>
<td>&lt;=</td>
</tr>
<tr>
<td>Objects:</td>
<td></td>
<td></td>
<td></td>
<td>&lt;=</td>
<td>&lt;=</td>
<td>&lt;=</td>
<td>&lt;=</td>
<td>&lt;=</td>
<td>&lt;=</td>
<td>&lt;=</td>
</tr>
<tr>
<td>Node(s)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Roots</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Children</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Leafs</td>
<td>No</td>
<td>All</td>
<td>Yes</td>
<td>Yes but</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>No Descendants</td>
<td></td>
<td>No Height</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Levels</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Subtrees with condition on Subtree root</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Path</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5.1 Outline of Select type 1 – Context Whole Ontology
ATTRIBUTES

IC – Information Content. The IC value is the information content of a concept with respect to a given ontology and is given below:

$$IC_{ont}(c) = \log \frac{(\text{desc}(c) + 1)/\text{max}_{ont}}{\log(1/ \text{max}_{wn})}$$

$$= 1 - \log(\text{desc}(c) + 1)/\log(\text{max}_{ont})$$

where desc(c) is the number of descendants of concept c, and max_{ont} is the maximum number of concepts in the ontology. The IC value is in the interval [0, 1] where 0 represents the information content of the root node and 1 represents the information content of a leaf node.

IR - Interval Rank. The interval rank (IR) for a concept c is the midpoint of the interval [depth(c), M – height(c)] where M is the length of the longest path from top node to bottom node in the ontology.

Level. The level of roots in the original ontology is set to 1 since the original OntoSELF created a dummy root when creating the visualized ontology. The dummy root is considered at level 0. The level for all other nodes is based on the maximum path length from the dummy root node. For example, if the node is both a grandchild and a child of an ontology root which is at level 1, then the node’s level value is set as three, not two. This situation should not occur that frequently in well-designed ontologies, but OntoSELF+TQ must handle all ontologies and the decision was made to use the maximum length from the dummy root.

Groups of Nodes (Descendants, Ancestors, Children, Parents, Siblings). For the attributes using a description which is a group of nodes, this attribute is prefixed by “number of”, i.e., “number of descendants”, “number of parents”, etc. Also note that the siblings attribute provides the total number of siblings for a node based on all its parents. For example, one node could have two different parents. With the first parent, this node might have 3 siblings and with the second parent, the node has 2 siblings. This node then has a total of 5 siblings.

Height. The height value of the node is the highest, (greatest) subtree height for that node with leaf nodes having a height of 0.

26
**Hub value.** The Hub value is the sum of the in-degree and out-degree for a concept. The in-degree is the number of parents and the out-degree is the number of children.

**Length.** The length is an attribute for the path object and is equal to the number of the nodes on that path minus one.

**Width.** The width is the number of concepts at the same level in the ontology.

**OBJECTS:**

**Nodes:** Any node within the ontology graph may satisfy the WHERE criteria. Only length and width attributes are not applicable for a node object. **Visualization:** Only the result concepts nodes meeting the condition criteria are shown along with any connecting edges that might exist between them.

**Roots:** For the root object, the group of concepts in the ontology is automatically restricted to only the root nodes for the entire ontology graph (ignores the dummy root). The level value is fixed. A root node of the ontology graph does not have any parents or siblings; therefore the children value is the same as the hub value. An artificial dummy root is connected to all the roots of the ontology so the information content for the ontology roots is not 0 and the IC attribute can be used in the WHERE condition. **Visualization:** All roots satisfying the condition are displayed.

**Leafs:** For the leaf object, the group of concepts in the ontology is automatically restricted to only the leaf nodes for the entire ontology graph. The IR attribute is based on both distance from the top and distance from the bottom, but distance from the bottom or height is the same for all leaf nodes. Although the IR attribute may be used, the Level attribute or distance from the top is more appropriate for leaf nodes. For leaf nodes, the descendant or children attribute does not apply but the ancestors and parents attributes do. The height attribute does not apply to leaf objects since the leaf subtree height is always 0. Since the hub attribute is the total of number of parents and the number of children, it produces the same result at the parent attribute because a leaf node does not have any children. **Visualization:** All leafs satisfying the condition are displayed and are linked to a dummy node considered at level 0.

**Subtrees:** Selecting a subtree object is similar to selecting a node object. The only difference is in the visualization result. **Visualization:** All subtree roots satisfying the WHERE criteria are...
displayed along with all their descendants. The subtrees are linked to a dummy node considered at level 0.

**Levels:** Selecting level(s) provides all nodes at those levels satisfying the WHERE criteria. **Visualization:** All of the nodes on the satisfied levels will be shown. If no links exists between any of the selected nodes, a dummy node will be introduced and connected to each of nodes.

**Paths:** The only path objects considered are only those paths which start at a root node and end at a leaf node. **Visualization:** All paths satisfying the condition are displayed starting at the actual root (not dummy one).

### 5.2.2 SELECT with Specific Node Context – SELECT Type 2

This query allows the user to give a context by directly selecting a specific node from a list of concept nodes in the ontology.

SELECT <object> FROM <specific node> WHERE <condition>

The node specified in the above query serves as the focus for the query. Table 5.2 shows the objects which can be selected based on the context of a particular concept that interests the user. Some of these objects further restrict the selection process since they identify a set of objects, i.e., Descendants, Children, and Ancestors. An explanation is provided for each of the objects to clarify how the specific node context affects the query. Note that “specific node” may also be referred to as “specific concept” in this discussion since the node in the ontology graph represents a specific concept within the ontology.

**OBJECTS:**

**Children:** Select the children for a specific concept where these children satisfy the applicable conditions. The sibling attribute is not applicable in the condition since it is assumed all children of the common specified node have the same number of siblings. Other siblings through another parent are not included when the context is a specific node. Similarly for the level attribute, it is assumed that all children are only one level down from the common specified node, though the children could actually be on varying levels if multiple parents are considered within the entire ontology. In this case, only the specific node is currently viewed as the parent since the context
of the query is that of specific node. **Visualization**: The specific node and its children which satisfy the conditions in the WHERE clauses are shown with their links to the specific node.

<table>
<thead>
<tr>
<th>attribute</th>
<th>IC operator</th>
<th>IR operator</th>
<th>Level operator</th>
<th>Descendants</th>
<th>Ancestors</th>
<th>Children</th>
<th>Heigh</th>
<th>Hub operator</th>
<th>length operator</th>
<th>Width operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator object</td>
<td>&gt;=</td>
<td>&gt;=</td>
<td>&gt;=</td>
<td>&gt;=</td>
<td>&lt;=</td>
<td>&lt;=</td>
<td></td>
<td>&lt;=</td>
<td>&gt;=</td>
<td>=</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>&lt;=</td>
<td>&lt;=</td>
<td>&lt;=</td>
<td>&gt;=</td>
<td>&gt;=</td>
<td></td>
<td>&lt;=</td>
<td>&lt;=</td>
<td>=</td>
</tr>
<tr>
<td></td>
<td>min</td>
<td>min</td>
<td>min least</td>
<td>min</td>
<td>max</td>
<td>max greatest least</td>
<td></td>
<td>max greater least</td>
<td>min (leaf)</td>
<td>Min (narrowest)</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>max</td>
<td>max greatest least</td>
<td>max</td>
<td>min</td>
<td>(root)</td>
<td></td>
<td>(root)</td>
<td>Max (longest)</td>
<td>(widest)</td>
</tr>
</tbody>
</table>

| Children           | Yes         | Yes         | No              | Yes          | Yes (no siblings) | Yes | Yes | No | No |
| Descendants        | Yes         | Yes         | Yes             | Yes          | Yes | Yes | No | No |
| Siblings           | Yes         | Yes         | No              | Yes          | Yes (no siblings) | Yes | Yes | No | No |
| Parents            | Yes         | Yes         | No              | Yes          | Yes (no siblings) | Yes | Yes | No | No |
| Ancestors          | Yes         | Yes         | Yes             | Yes          | Yes | Yes | No | No |
| Leafs              | No All 1    | Yes         | Yes             | Yes but No Descendants | Yes | No children | No | Heigh t 0 | Yes = parents | No |
| Subtrees with condition on Subtree root | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | No |
| Levels             | No          | No          | No              | No           | No | No | No | No |
| Path(up)           | No          | No          | No              | No           | No | No | No | Yes | No |
| Path(down)         | No          | No          | No              | No           | No | No | No | Yes | No |

**Table 5.2 Outline of Select type 2 – Specific Node context**
OBJECTS (continued):

Descendants: Select the descendants for a specific concept where these descendants satisfy the applicable conditions. Visualization: The specific node with all its descendants that meet the conditions are displayed along with their associated links. For example, to select the descendants of a specific node which are at level 6 or greater, one could specify the query SELECT descendants FROM specific-node WHERE level BETWEEN 6 and 6. The level attribute is as determined for the complete ontology so some children might directly qualify since they are at level 6 and descendants and be directly linked to the specific node. Other children of the specific node might not qualify but some of their descendants might several levels down. A dummy link is provided when needed from the specific node to those descendants that qualify.

Siblings: Select the siblings for a specific concept where these siblings satisfy the applicable conditions. The specific node could be a sibling to several different sets of nodes due to its having multiple parents. Given that each set of siblings shares a common parent with the specific node however, the attribute level and the attribute siblings are not applicable since the association with the common parent is the focus. In that context each set of siblings would be considered as having the same number of siblings (through a common parent) and same level of the specified node. Again in the context of the whole ontology this might not be the case, but the focus here is on the relationship of siblings to the specific node. Visualization: The specific node along with its siblings that satisfy the conditions are displayed. The common parent for each group of siblings is displayed with a link to each of its children concepts. The specific node is always shown along with its siblings.

Parents: Select the parents for a specific concept where these parents satisfy the applicable conditions. The level attribute is not applicable because for the level, it is assumed that all parents are only one level up from the specified node, though in the context of the whole ontology they might be at varying levels. Visualization: The specific concept and its parents that satisfy the condition are displayed. A link from each parent concept to the specific node is included.

Ancestors: Select the ancestors for a specific concept where these ancestors satisfy the applicable conditions. Visualization: The specific concept and its ancestors that satisfy the
condition are shown. As discussed under descendents object, dummy links are provided from the specific node to the ancestor when not all ancestors on the path are included because of failing to satisfy the condition.

**Leaves:** For the leaf object from a specific node, it is exactly as for the first SELECT except that the group of leafs are not all leafs in the ontology but only leafs in the subtree starting at the specific node. **Visualization:** All leafs satisfying the condition are displayed and shown linked to the specific node.

**Subtrees:** Selecting a subtree object from a specific node is very similar to the first SELECT table. The only difference is that only those subtrees existing under the specific node are considered. **Visualization:** All subtree roots satisfying the WHERE condition(s) are displayed along with all their descendants. The subtrees are linked to the specific node.

**Levels:** Select the level for a specific concept where the other concepts are at the same level as the specific node. No attributes are applicable for the WHERE condition since the exact level is that of the specific node. **Visualization:** All concepts at the same level as the specific node concept are displayed. This level is based on the specific node’s level within the complete ontology so that only those nodes having this same level are displayed.

**Paths:** Select the path for a specific concept. Because of the complexity that might exist if a specific concept node has many paths through it, it was decided to break path into the Path(up) component and the Path(down) component. For path down, only those paths that start at the specific node and end at one of its leaf descendants are to be considered. For path up, only those paths that start at the specific node and end at one of its root ancestors are to be considered. Only these paths that satisfy the WHERE condition (the path length) are kept. **Visualization:** Only those paths satisfying the condition are displayed starting or ending at the specific node.

5.2.3 SELECT with a Nested Query Context – SELECT Type 3

This query is the most complex query since it uses as its context the result of another query. This type of query is provide to allow the user flexibility in creating queries that can filter based on a combination of different kinds of objects. All nodes produced by the nested-query result serve as the subset of the ontology to be used as the context for the current query. Both Table
5.1 and Table 5.2 are applicable to this kind of query since the inner query might be a SELECT of type 1, type 2 or type 3. The following format for the select statement is:

\[
\text{SELECT } <\text{object}> \text{ FROM } <\text{nested-query}> \text{ WHERE } <\text{condition}>
\]

The implementation demonstrates this capability by providing up to two levels of nesting, i.e., the <nested-query> itself can contain a <nested-query>. It was felt that for users, two levels of nesting would provide enough flexibility to develop sophisticated queries but higher levels of nesting may become confusing.

To let the user distinguish the different sets of result nodes, different colors are used. Red is used to represent the color of the nodes for the nested or inner query, and green is used to represent the color of nodes for the current or outer query. Blue is used to represent nodes that belong to the results of both queries.

For handling two levels of nested queries, the visualization results are produced in a step-wise manner. The returned concepts for the innermost or second level nested query which satisfy the conditions are processed first and given as input to the next level up query. This inner or first level nested query shows the objects which satisfy its conditions based on starting with the result from the innermost query. The result of the inner query or first level query is given to the outer level query for processing. For example, consider the following query, SELECT Parents FROM (SELECT Parents FROM (SELECT Parents FROM "Activity"))). First, the result of the innermost query “SELECT Parent FROM ‘Activity’” is executed, and for the convenience, the result is called R1. Then the query “SELECT Parents FROM R1” is processed, the returned result is called R2. At this point, the visualization result for “SELECT Parents FROM (SELECT Parents FROM “Activity”))” is shown. The red color nodes represent R1 results and the green color nodes represent R2. There may be some nodes that belong to both R1 and R2 and these nodes are colored as blue. Next the query “SELECT Parents FROM R2” is processed, the returned result is called R3. The final visualization result for the entire query is shown. The red color nodes represent R2 and the green color nodes represent R3. There may be some nodes that belong to both R2 and R3 and these nodes are colored as blue. Section 5.3 describes the user interface for the nested query and how the user can see the results in a step-wise fashion.
The process for the third type query actually repeats the visualization step of the second type query using a specific node context. For the second type query, the object is a single concept node. For the nested query, the FROM clause provides not a single concept node but a set of objects resulting from the processing of the inner query.

5.3 OntoSELF+TQ User Interface Design

This section presents several example snapshots to explain how the user of OntoSELF+TQ specifies an ontology visualization request. Also provided are examples of some of the error processing that is performed when processing the query.

![Figure 5.2 OntoSELF+TQ User Interface](image)

The GUI shown in Figure 5.2 was designed using Java SWT JFace and allows the user to select different queries for the visualization of ontologies. The File menu allows the user to select the ontology that is to be visualized by browsing through their file system. The ontology is represented in OWL format as show in Figure 5.3. The menu also provides “Save as” and “Visualize Saved GML” functions. The “Save as” option allows a user to store the current query result into their desired directory and rename the file. Later the user can apply the “Visualize Saved GML” option to visualize the stored GML file. A GML file consists of a hierarchical key-value list. It is platform independent, and easy to implement. It has the capability to represent arbitrary data structures, and is flexible enough that a specific order of declarations is not needed.
The user can visualize the entire ontology through the “Visualize Ontology” option on the menu. The “Create query” link leads to the query interface. The query interface as shown in figure 5.2 is structured based on the three clauses of the SELECT statement: SELECT, FROM and WHERE. The SELECT clause provides a dropdown box that allows the user to choose the kind of object to be included in the visualization result. The FROM clause is optional and permits two different ways of creation: a nested query or a specific node. For the specific node option, the text area lists all concepts in the ontology. Figure 5.7 shows this text area filled in after the user has opened the UMLS Semantic Network ontology file. The user can perform the first type of SELECT described in section 5.2.1 by choosing the object from the dropdown box in the SELECT clause, ignoring the FROM clause and completing the information for the WHERE clause. The user can perform the second type of SELECT described in section 5.2.2 by enabling the concept list text area and then scrolling down the concept list to choose the specific concept.

The objects in the first dropdown box will depend on which type of query the user tries to make. For example, if the user leaves the concept list unselected, only six objects will be applicable based on the table 5.1, which are “Nodes”, “Roots”, “Leafs”, “Levels”, “Subtrees” and “Path”.

If the user enables “Choose specific concept”, the dropdown box for the SELECT clause is set to the applicable objects that are listed in Table 5.2. These are “Children”, “Descendants”,

Figure 5.3 File Selection for visualization
“Siblings”, “Parents”, “Ancestors”, “Leafs”, “Subtrees”, “Levels”, “Path(up)”, and “Path(down)”. According to the chosen object, the dropdown box for the WHERE clause is set to different attributes. For example, if “Roots” object is selected in the SELECT dropdown box, then the WHERE dropdown box contains four attributes based on table 5.1, which are “IC”, “Descendants”, “Height”, and “Children”. The same rule is applied for other chosen objects.

The user enables the condition button that is in front of “WHERE” clause and sets attributes for filtering for a value in the desired range. If multiple conditions are needed, the user must enable the “Go to next condition” link by checking the box in front of it. Another condition area is displayed as shown in Figure 5.4. If the previous attribute dropdown box has four attributes, here the “Roots” attributes are used again, which are “IC”, “Descendants”, “Height”, and “Children”, and the “Children” attribute is already selected, the dropdown box in the current condition contains only the remaining three attributes, which are “IC”, “Descendants” and “Height”. The same rule is applied to all other object attributes.

Once the user sets up the attributes and the range of values, he or she can click the “Done” button to hide the panel. If the user sets illegal input in the text field, for example, between 3 and 2, the first number is greater than the second number, then when the “Done” button is clicked, an error message will pop up as shown in Figure 5.5. Once the user completes the query and clicks the “Visualize” button, the visualization result is shown automatically. Before processing the query, the system checks all of inputs provided by the user to make sure the query is complete and correct. Other errors besides those previously mentioned, such as choosing “Go to next condition” without selecting a condition attribute or a range are shown in Figure 5.6. If the user wants to request another query, he or she clicks the “Clear” button and the default values are reset.

![Parallel Condition Panel](5.4 Parallel condition panel for OntoSELF+TQ)
Figure 5.5 Error range input for parallel condition

Figure 5.6 Error object input for parallel condition
If the user wants to create a nested query, the “Create query” checkbox is checked. After checking the box, the “Nested Query One” panel as shown in Figure 5.8 is displayed. Note this panel looks like the original query panel except for the “Done” button. If the user only applies one nested query, the “Done” button is hidden on the panel. However, if the user applies more than one level to the nested query, the “Done” button label will be “Visualize”. By clicking it, the user can observe the innermost nested query results. Figure 5.9 shows the panel.

Figure 5.7 Interface after opening the UMLS ontology file

Figure 5.8 Nested Query One panel

Figure 5.9 Nested Query One panel with more than one nested query
Figure 5.8 “Nested Query” panel for OntoSELF+TQ

![Nested Query One](image)

Figure 5.9 Nested query panel one for OntoSELF+TQ

The Nested Query Two display in Figure 5.9 has the same function as described for the panel in Figure 5.7. Theoretically, the nested query and parallel condition can go on one followed by another. In this thesis, however, the number of nested queries is limited to 2 and parallel conditions to ten. As an example, the query “SELECT … FROM (SELECT … FROM (SELECT … FROM ))” has two nested queries. The object Nodes and Subtrees in table 5.1 have ten attributes, which are “IC”, “IR”, “Level”, “Descendants”, “Ancestors”, “Children”, “Parents”, “Siblings”, “Height” and “Hub”. The other objects have less attributes. For example, the Path object only has one attribute “Length”.

![Nested Query One](image)
6. OntoSELF+TQ Implementation

6.1 Modification of Original OntoSELF

The UML class diagram for the OntoSELF+TQ shown in Figure 6.1 includes all of major classes. The diagram is quite similar to OntoSELF; however, the main focus of the implementation is to provide the SQL like querying capabilities and to enhance the possible queries with additional attributes about the graph objects not originally provided in OntoSELF. In Figure 6.1, the code developed specifically for OntoSELF+TQ is shown in Italics.
The GUI is the user interface class and provides the user with the various SELECT options as previously described in section 5.2. The interface includes three main panels, which help the user choose objects either for the entire ontology or the given context of the query. Each panel has nine sub panels which allow the user to restrict the attributes for the objects. Because different objects have different number of attributes restrictions, it may be not necessary to use all of the nine sub panels. For example, the “Level” object for the entire ontology only has two related attributes, “level” and “width” separately.

To begin the visualization process, an ontology must be opened. The open method is called. Inside this method, OWLtoGMLParser translates the input OWL file to GML file. The save method allows the user to save the current query result in a directory of the user’s choice. The saved file later can be re-examined without having to reprocess the query. This function is implemented by the look method. The exit method is responsible ending the application.
Several visualization functions are used to provide the different levels of visualization. The \textit{visual} method is called when the user has selected the “Visualize Ontology” option from the File menu. It produces a .gml file for the entire ontology. The \textit{visual2} method is called for all three \texttt{SELECT} types but only when the nested query contains only one level of nesting. The \textit{visual3} method is called when the query contains more than one level of nesting. The detailed implementation for GUI is described in section 6.2.

The monitor is mainly responsible for initiating and monitoring the processing of the input gml file. It plays a critical role in parallel computation. The \texttt{ProcessGML} class is responsible for calculating attributes for each object and retrieving related objects when necessary. This thesis work contributed five new attributes to be used in the condition part of the WHERE clause: “ancestor”, “parent”, “sibling”, “length”, and “width”. Please refer to section 5.2 for more detailed explanation for these terms. No particular data structure is provided to store “length” value because the length value of path is one less than the number of nodes in the path. The other attributes values are stored separately in the following vectors, \texttt{nodesAncestor}, \texttt{nodesParentCount}, \texttt{nodesSibling}, and \texttt{widthValue}.

The \texttt{getSource} and \texttt{getTarget} methods are used in processing a nested query. For example, given a one-level nested query:

\begin{verbatim}
SELECT Parents FROM (SELECT Nodes WHERE descendants between 35 and 40),
\end{verbatim}

the \texttt{getSource} method parses the inner query “SELECT Nodes WHERE descendants between 35 and 40” with the object parameter Nodes, the attribute parameter descendants and its range (35 to 40). Then the \texttt{getSource} method retrieves all of the nodes satisfy the query and saves them into a vector data structure. The \texttt{getTarget} method takes the vector produced by the \texttt{getSource} method and the object parameter parents from the outer query to produce the final results.

To differentiate the source nodes and target nodes, different colors are used in the VTK results. Red nodes represent the nodes from the \texttt{getSource} method and green nodes represent the nodes from the \texttt{getTarget} method. Nodes belonging to both groups are colored blue. Here is another example of a two-level nested query

\begin{verbatim}
SELECT Children FROM
    (SELECT Siblings FROM
        ...)
\end{verbatim}
(SELECT Nodes WHERE Hub between max and max)).

The getSource method parses the innermost query “SELECT Nodes WHERE Hub between max and max” as object parameter Nodes and attribute parameter Hub and its range (max to max). Then the getSource method retrieves all of the nodes that satisfy the query and saves them into a vector data structure (called v1). The getTarget method takes the vector v1 produced from the getSource method and the object parameter siblings to produce the first level results that are stored in another vector data structure (called v2). The result is visualized by applying the same color strategy discussed before and the user may produce this visualization as explained previously in section 5.3. At this point, the two inner queries have been processed. To get the final results, the getTarget is called again and takes the vector v2 and the object parameter children to produce the final visualization result. The nodes satisfying the object parameter children are colored green and the previous target nodes satisfying the object parameter siblings are colored red.

6.2 User Interface Processing Architecture

The three types of SELECT queries are explained in section 5.2. This section describes in more detail their implementation. The processing flow of GUI is shown as Figure 6.2. The rectangle represents the processes and the parallelograms represent the outputs.
The GUI class first loads all of concepts in the selected ontology in three lists which are introduced in section 5.3.1. There are three main interface panels in OntoSELF+TQ. Only one panel pop ups on the screen and the other two panels are activated if the user chooses to implement the nested queries. All of three lists on the three panels are loaded all of the concepts of the target ontology because the user may process query for the specific concept in the nested query at any level. Now the user can start to create the query. If no nested query or a specific
concept is found, the class sees the query as SELECT type 1 and filters all of objects for the entire ontology. If a specific concept is chosen, the class sees the query SELECT type 2 and filters all of objects related to that node. Then it checks whether the “WHERE” condition has been set. If the answer is yes, it finds all of constraints and keeps the objects which satisfy all of the conditions. The following explanation for nested queries is illustrated using this structure:

```sql
SELECT ... FROM  // Level 0
  (SELECT ... FROM  // Level 1 nesting
    (SELECT ... FROM ...... )  // Level 2 nesting
  )
```

First the GUI “class” in Figure 6.1 determines the level of nesting and begins with the deepest or innermost SELECT. If it is at the second level then since only two level nesting is possible, it must be the first SELECT type described in section 5.1.1 or the second SELECT type described in section 5.1.2. The processing of the innermost SELECT (level 2) returns a set of objects satisfying that SELECT. The getTarget method in ProcessGML class then uses this set as input to the next SELECT (level 1) and repeatedly applies to the level 1 SELECT to each object in the returned set from the level 2 SELECT. It does this by treating each object in the returned set as a “specific-node” context, i.e., it uses the second SELECT type described in section 5.1.2. The result of the level 1 SELECT query is a set of objects that is taken as input to the level 0 SELECT query. The implementation of OntoSELF+TQ is limited to just two levels, but it demonstrates how to process the multiple levels. For example, if there are three levels, first the getSource method gets the level 3 results, and then by using the result the getTarget method returns the result objects for level 2. Repeating the above function until the final result is got.

To allow the user to observe the results easily, the color strategy has been used for different groups of concepts. For convenience, the inner nested query results are called source concepts; the outer query objects are called target concepts. The red color is given to source nodes and the green color is given to target nodes. When nodes belong to both groups; the blue color is given to these concepts. For the concrete example, please refer to the last paragraph in section 6.1.
The query result is saved automatically as the file fsm.gml in the directory C:\VTK\Graphs\Data in Gml file format. The TCL file color.tcl read the default fsm.gml and the file is activated by the java source code Runtime.getRuntime().exec method. The exec method takes the vtk.exe file and color.tcl file as parameters and automatically generates the 3d visualization result. This capability of automatic visualization was not in the original OntoSELF but was added to make it more user-friendly. The user may save the produced gml file for later visualization in the desired directory.

7. Evaluation

The evaluation of OntoSELF+TQ uses four different approaches. The first is a systematic demonstration of many of the querying capabilities of the software in order to verify as much as possible the correctness of the implementation. For this evaluation, a smaller bioinformatics ontology, the UMLS Semantic Network is used. The smaller size of this ontology easily allows a manual check of the correctness of the query results for a variety of test cases. The second approach uses an ontology that has been previously used to evaluate ontology visualization systems, the NASA SWEET (Semantic Web for Earth and Environmental Terminology) Ontology. This evaluation method is provided to demonstrate the wide range of querying capabilities of OntoSELF+TQ using each of the three types of SELECTs. The third method uses existing proposed categories of graph visualization topology understanding tasks as a checklist to determine how well OntoSELF+TQ meets these proposed requirements. The final approach is an evaluation based on conducting a human-study experiment to determine the ease with which users can develop and answer topology understanding queries and to survey their satisfaction with the visualization software.

As part of this evaluation process, numerous example queries are provided and their corresponding visualizations are presented. These visualizations may not be clearly readable at their current viewing of 100%, but the reader may use the zoom feature of either MS Word or Acrobat Reader to enlarge the figures in order to more clearly see the node labels and the edges between the nodes in these figures. Note also that in some images, groups of nodes are circled or individual nodes underline. These markings are not created by OntoSELF+TQ but are provided to better explain the visualization results.
7.1 Verification Evaluation

Verifying as much as possible the correctness of the OntoSELF+TQ system is the baseline evaluation that must be performed before other evaluation can proceed. The user gets to know the structure of the ontology by visualizing different parts of it. If the visualization result is not correct for the entered query, then the software is of no use in helping the user understand the topology of the ontology.

The UMLS Semantic Network ontology (http://semanticnetwork.nlm.nih.gov) has been chosen as the sample ontology. Since UMLS ontology has only 135 concepts and 133 hierarchical relationships, the user can count the values for the various attributes for each concept from the overall visualization result. The sample query tests cover most of the objects and attributes and demonstrates that the OntoSELF+TQ has credible reliability.

![Figure 7.1 The overall visualization of UMLS Semantic Network Ontology](image-url)
The overall visualization of the UMLS Semantic Network is shown in Figure 7.1. By using the mouse, the user can interact with the visualization image for zooming in and out in order to examine the details of the ontology. VTK provides both zooming and rotating capabilities on the produced visualization. Figure 7.1 provides all the concepts and the hierarchical relationships. The root nodes and the lowest level concepts are easily defined and through zooming capabilities different features can be verified. The DummyRoot is not an actual root of the ontology but connects the two ontology roots, Entity and Event. The deepest level has two concepts, Steroid and Eicosanoid. The reader of this electronic version may also verify this by using the zooming capabilities of MS Word and can easily see these two nodes at 200% zoom. The query “SELECT Roots” returns all of the root nodes for UMLS, and the result is shown as in Figure 7.2. The query “SELECT Levels WHERE Level between max and max” returns all of the concepts in the deepest level, and the result is shown as in Figure 7.3. Both of these test cases show the proper functioning of the selection on roots and levels.

For testing other objects and attributes, partial structure of the ontology are focused on. The left image of Figure 7.4 focuses on concepts “Full_Formed_Anatomical_Structure” and its five children concepts, “Cell”, “Gene_or_Genome”, “Tissue”, “Cell_Component” and “Body_Part_Organ_or_Organ_Component”. The query “SELECT Children FROM ‘Full_Formed_Anatomical_Structure’” returns all its children and itself, and the result is shown as the right image of Figure 7.4.

![Figure 7.2 The result for roots query](image)
Figure 7.3 The result for deepest level query

Figure 7.4 The result for query - children of “Full_Formed_Anatomical_Structure”

The left image of Figure 7.5 focuses on concepts “Spatial_Concept” with its seven descendant concepts, “Geographic_Area”, “Body_Location_or_Region”, “Body_Space_or_Junction”, “Molecular_Sequence”, “Nucleotide_Sequence”, “Carbohydrate_Sequence” and “Amino_Acid_Sequence”. The query “SELECT Descendants FROM ‘Spatial_Concept’ returns all its descendants and itself, and the result is shown as the right image of Figure 7.5.
Figure 7.5 The result for query - descendants of “Spatial_Concept”

The left image of Figure 7.6 focuses on the concept “Animal” with its six sibling concepts, “Bacterium”, “Plant”, “Fungus”, “Rickettsia_or_Chlamydia”, “Virus”, and “Archaeon”. The query “SELECT Siblings FROM ‘Animal’” returns all of its Siblings, itself and their parents, and the result is shown in the right image of Figure 7.6.

Figure 7.6 The result for query - Siblings of “Animal”

The left image of Figure 7.7 focuses on the concept “Alga” with its parent concepts, “Plant”. The query “SELECT Parents FROM ‘Alga’” returns all of its parents and itself, and the result is shown in the right image of Figure 7.7.
The left image of Figure 7.8 focuses on concept “Regulation_or_Law” with its three ancestor concepts, “Intellectual_Product”, “Concept_Entity” and “Entity”. The query “SELECT Ancestors FROM ‘Regulation_or_Law’” returns all of its ancestorst and itself, and the result is shown in the right image of Figure 7.8.

The left image of Figure 7.9 focuses on concepts “Animal” with its six leaf concepts, “Fish”, “Human”, “Reptile”, “Bird”, “Amphibian”, and “Invertebrate”. The query “SELECT Leaves FROM ‘Animal’” returns all of its leaf concepts and itself, and the result is shown in the right image of Figure 7.9.
Figure 7.9 The result for query - Leaves of “Animal”

The left image of Figure 7.10 focuses on the concept “Occupation_or_Discipline” with its path up to the root concept “Entity”. The query “SELECT Path(up) FROM Occupation_or_Discipline” returns all of the concepts in the path, and the result is shown in the right image of Figure 7.10.

Figure 7.10 The result for query - Path(up) of “Occupation_or_Discipline”

The left image of Figure 7.11 focuses on the concept “Occupation_or_Discipline” with its path down to the leaf concept “Biomedical_Occupation_or_Discipline”. The query “SELECT Path(down) FROM ‘Occupation_or_Discipline’” returns all of the concepts in the path, and the result is shown in the right image of Figure 7.11.
Figure 7.11 The result for query - Path(down) of “Occupation_or_Discipline”

The above figures provide example test cases to demonstrate the ability of OntoSELF+TQ to correctly select different kinds of objects. The following example test cases demonstrate its ability to conditionally select objects based on attribute values for objects.

Figure 7.5 shows that the number of descendants of “Spatial_Concept” is seven. The query “SELECT Nodes Where Descendants between 7 and 7” returns two concepts, including “Spatial_Concept” and “Organic_Chemical”, and the result is shown in the left image of Figure 7.12. The right image of Figure 7.12 shows the result from the query “SELECT Descendants FROM ‘Organic_Chemical’”. From the image, seven descendants can be easily counted for “Organic_Chemical” concept.

Figure 7.12 The result for query - Nodes WHERE Descendants between 7 and 7
Figure 7.8 has shown that the number of ancestor of “Regulation_or_Law” is three. The query “SELECT Nodes Where Ancestors between 3 and 3” returns a group of concepts, including “Regulation_or_Law”, and the result is shown in Figure 7.13.

![Figure 7.13 The result for query - Nodes WHERE Ancestors between 3 and 3](image)

The left image of Figure 7.14 shows that the marked concepts “Finding”, “Intellectual_Product”, “Organization”, “Group”, and “Idea_or_Concept” are all children of concept “Concept_Entity”. Furthermore, they all have more than two children. The query “SELECT Children FROM “Concept_Entity” Where children between 2 and max” returns the same group of concepts as described above, and the result is shown in the right image of Figure 7.14.

![Figure 7.14 The result for query – Children of “Concept_Entity” WHERE Children betweens 2 and max](image)
The left image of Figure 7.15 shows that the marked concepts “Group”, and “Idea_or_Concept” are all children of concept “Concept_Entity”. Furthermore, they both have five children and are bushiest children for the concept “Concept_Entity”. The query “SELECT Children FROM “Concept_Entity” Where Children between greatest and greatest” returns the two concepts as described above, and the result is shown in the right image of Figure 7.15.

Figure 7.15 The result for query – Children of “Concept_Entity” WHERE Children betweens greatest and greatest

The left image of Figure 7.16 shows that the marked concept “Idea_or_Concept” is the child node of concept “Concept_Entity”. Furthermore, it is the bushiest child and the only child node with height greater than one for the concept “Concept_Entity”. The query “SELECT Children FROM “Concept_Entity” Where Children between greatest and greatest AND Height between 2 and max” returns the concept as described above, and the result is shown in the right image of Figure 7.16.

The left image of Figure 7.17 shows that the marked concepts “Concept_Entity”, and “Physical_Object” are all children of concept “Entity”. However, only “Concept_Entity” has the hub value greater than six. The query “SELECT Children FROM “Concept_Entity” WHERE Hub between 7 and max” returns the concept as described above, and the result is shown in the right image of Figure 7.17.
Figure 7.16 The result for query – Children of “Concept_Entity” WHERE Children between 2 and max and Height between 2 and max

Figure 7.17 The result for query – Children of “Entity” WHERE Hub between 7 and max

The left image of Figure 7.18 shows that the marked concepts “Injury_or_Poisoning”, “Daily_or_Recreational_Activity”, “Language”, “Machine_Activity” and “Group_Attribute” are all leaf concepts. Furthermore, the length through each of these leaves to the root is two and they are the only leaves that have this property. The query “SELECT Path Where length between 2 and 2” returns all of the concepts on those paths and the result is shown in the right image of Figure 7.18.
The left image of Figure 7.19 shows that the level below the root nodes contain four concepts, which are “Concept_Entity”, “Physical_Object”, “Activity” and “Phenomenon_or_Process.” Furthermore, this level is the only one in the UMLS that has four concepts. The query “SELECT Level WHERE width between 4 and 4” returns all of the concepts in such a level and the result is shown in the right image of Figure 7.19.

The left image of Figure 7.20 shows the children of concept “Spatial_Concept”. Only one child concept “Moleque_Squence” that has three children and the other children concepts of “Spatial_Concept” do not have children. The query “SELECT Children FROM (SELECT Children FROM Spatial_Concept)” returns all of children of the children of “Spatial_Concept”
and result is shown in the right image of Figure 7.20. The red color node “Modeque_Squence” is from the result of the inner query and the three green color nodes are the final returned nodes for the complete query.

![Figure 7.20 The result for query – Children From Children of Spatial_Concept](image1)

The examples given above illustrate test cases used to perform a system test of the various capabilities of the OntoSELF+TQ software. System testing cannot guarantee that no bugs exist in the software but does provide a level of quality evaluation of the software. The next section demonstrates the flexibility and the range of querying capabilities of the software through numerous examples of each kind of SELECT on a larger ontology. It proceeds from simple queries to the advanced nested level queries.

### 7.2 Demonstration of complex query capability

The OntoSELF+TQ system offers multiple query choices for the user to explore the structure of an ontology. In this section, examples of using the three different kinds of SELECTs are provided. The first section below focuses on queries that do not use the FROM clause and, therefore, use the complete ontology for the context of the query. The second section provides examples of queries that use the FROM clause to identify a concept within the ontology that is to serve as the focal point of the query. The third section examines the capability of using a nested query within the FROM clause. The context of the outer query is the result of the nested query. The figures provided in this section are snapshots of the screen and in some cases it may be difficult to clearly see the labels on the nodes. The reader is advised that they can use the zoom capabilities of either MS Word or Acrobat depending on what format the document is in to more
clearly see the labeling of the nodes and the edges between the nodes when the figures contain many nodes and edges. The user of OntoSELF+TQ automatically is provided zooming features and the ability to rotate the visualization through VTK to more clearly see the resulting visualization.

The NASA JPL Semantic Web for Earth and Environmental Terminology (SWEET) ontology (http://www.mindswap.org/) is used for the sample ontology for all of the following queries. SWEET includes 1537 concepts and 1519 hierarchical relationships. This ontology was also used in the CropCircle evaluation (Wang and Paris 2006) and the topology tasks used in the CropCircle are also applied in the query complexity evaluation. A summary of these queries is provided in Section 7.2.4.

All the example queries below produce visualization results. Before beginning those examples, however, the case where the user entered query cannot be satisfied is presented. If this situation occurs, a warning message is provided to the user stating that there is no result for the query. For example, the Figure 7.21 is processing the query “SELECT Parents FROM ‘Entity’”. Because the concept “Entity” is a root node, no parents can be returned. The warning message pop ups and tells the user to try another query.

![Figure 7.21 No result message for the query.](image)

### 7.2.1 SELECT with the Context of the Complete Ontology

The evaluation starts with the Nodes objects. The following three queries show that OntoSELF+TQ filters the qualified nodes with various conditions.
I) Node Queries:

   a) SELECT Nodes WHERE Children between max and max.

   This query allows the user to find all concepts in the ontology which have the most number of
   children. The user can determine the concepts within the ontology that are most broadly defined
   at their next level down. The visualization produced by the OntoSELF+TQ is shown in Figure
   7.22. Only the node “Animal” is returned as the query result since it has the most number of
   child nodes in the whole ontology. See Figure 7.49 below which shows that “Animal” has 27
   children concepts.

   ![Figure 7.22 The result for the first node query](image)

   For the next query, the condition has been loosened, and more results should be expected.

   b) SELECT Nodes WHERE Children 10 and max.

   This query allows the user to find all nodes which have at least ten children. The visualization
   produced by OntoSELF+TQ is shown in Figure 7.23 below. In this visualization, edges are
   include if there is an is-a or subclass relationship between any of the nodes. Many concepts in
the ontology meet this condition of having at least 10 children and are returned, including the node “Animal” (Marked).

Figure 7.23 The result for the second node query

The next query is based on the previous query and adds another condition to the WHERE clause.

c) SELECT Nodes WHERE Children between 10 and max AND Level between 3 and 5.

The query allows the user to further refine his search by restricting the results to nodes that in addition to having at least ten children are located at level 3, 4 or 5. The visualization produced by the OntoSELF+TQ is shown below. Compared to Figure 7.23, nearly two third nodes have been filtered out due to the added level constraint.
The next object for evaluation is the Root object. Four queries show that e OntoSELF+TQ filters the qualified roots with various conditions.

II) Roots

a) SELECT Roots

The above query produces all the root concepts and the resulting visualization is shown in Figure 7.25. If one zooms in on this visualization, the user can see that there is an inconsistency in this ontology. For example, one root of the ontology is “LivingThing”. Another root beside “LivingThing” is “MarineAnimal.” An examination of the .owl file for this ontology does indeed show that this is how “MarineAnimal” has been defined. It has not been given a parent class and, therefore, is considered as one of the root classes of the ontology.
The next query adds one condition on the previous query.

b) SELECT Roots WHERE Height between max and max

The query finds the root whose subtree has the maximum height of all the roots in the ontology. In context of the complete ontology it roots the subtree with the greatest height. The visualization produced by the OntoSELF+TQ is shown in Figure 7.26. Only the node “&numerals;Quantity” is returned as the query result since it roots the highest subtree.

Figure 7.25 The result for the first root query
Figure 7.26 The result for the second root query

For the next query, the condition has been loosened, and the more results would be expected.

c) SELECT Roots WHERE Height between 5 and max.

The above query asks all of the roots whose subtree’s height is at least 5. The visualization produced by the OntoSELF+TQ is shown in Figure 7.27. A group of nodes are returned, including the node “&numerics;Quantity” (Marked). All of these nodes root a subtree with a height of at least 5. Comparing to Figure 7.25, almost half of the root nodes are filtered out because of the additional condition in the WHERE clause.
The next query is based on the previous query and adds a condition on the number of children.

d) SELECT Roots WHERE Height between 5 and max AND Children between greatest and greatest.

The above query produces only those roots with a subtree height of at least 5 and the ones that have the most number of children from those satisfying the height condition. The visualization produced by this query is shown in Figure 7.28. The node “KnowledgeDomain” is the query result from among the roots shown in Figure 7.27 because it has the most child concepts.
The next object for evaluation is the Path object. Three queries show that the OntoSELF+TQ filters the qualified Path with various conditions.

III) Paths

a) SELECT Path WHERE Length between 4 and 4

The above query produces all the paths with length 4 that start at a root of the ontology and end at a leaf node. The visualization produced by this query is given in Figure 7.29.
The next query changes the value for the length attribute on the previous query.

b) SELECT Path WHERE length between max and max

The above query produces all the longest paths starting at the roots of the ontology and the result and ending in a leaf node. The result is shown in Figure 7.30. The longest path in the ontology starts at the root node numerics;Quantity and has a length of 8.
Figure 7.30 The result for the second path query

The next query again modifies the value for the length attribute on the previous query.

c) SELECT Path WHERE Length between min and max.

The above query allows the user to see all the paths in the ontology. The visualization produced is shown in Figure 7.31. This figure actually shows an overview for the entire SWEET ontology. All of the paths include all the concepts and is-a relationships for the ontology. For clearly viewing the entire structure of the SWEET, only 450 nodes on the higher level out of total 1537 concepts have labels. The leaves and other lower level nodes do not show the labels in Figure 7.31. The 450 value is set on the tcl file which is run by vtk and produces the visualization result. The value can be changed to any number the user would prefer to.
The next object for evaluation is the Level object. Two queries show that the OntoSELF+TQ filters the qualified levels with various conditions.

IV) Levels

a) SELECT Levels WHERE Width between 10 and 30

The above query produces all the levels which have between 10 and 30 concepts in them. The visualization produced by the OntoSELF+TQ is shown in Figure 7.32. This figure shows two levels in the ontology which are connecte, and both of them have concepts between 10 and 30. The lower level has exactly 10 concepts and the upper level has 27 concepts.
The next query changes the value for the Width attribute on the previous query.

b) SELECT Levels WHERE Width between max and max

The above query allows the user to see the levels which have the most concepts in the ontology. Figure 7.33 shows the widest level (level 3) in SWEET ontology. This level has 437 nodes and all of these concepts are linked to dummy node.
The next object for evaluation is the Subtree object. Three queries illustrate the ability of OntoSELF+TQ to filter Subtree roots with various conditions in the WHERE clause. Selecting the subtree object is similar to selecting the node object except that instead of just displaying the node, it displays the satisfying node as the root of a subtree and also displays all of its descendants.

V) Subtrees

a) SELECT Subtrees WHERE IR between 2.5 and 4.5 AND Siblings between 3 and 3.

The above query finds nodes which have interval rank between 2.5 and 4.5 and three siblings. Since the object being selected is a subtree, the visualization displays all of the descendants of these nodes. This query result is shown in Figure 7.34. It shows all of the qualified subtree roots (marked) with their descendants. All of the subtree roots are linked to the dummy node.

Figure 7.33 The result for the second level query
Figure 7.34 The result for the first subtree query

The next query changes the attributes being used in the WHERE clause.

b) SELECT Subtrees WHERE Level between 5 and 5 AND Descendants between greatest and greatest.

This query displays the subtrees on level 5 which have the most descendants. Figure 7.35 displays the results of this query. It shows that the subtree rooted at “&property;LinearExtent” has the greatest number of descendants compared to all subtrees on level 5.
The next query demonstrates three conditions in the WHERE clause and basically adds the Children attribute and changes the range for the levels attribute.

c) SELECT Subtrees WHERE Children between 3 and 5 AND Level between 3 and 5 AND Descendants between greatest and greatest.

The above query displays the subtrees with the most descendants, which are on levels between 3 and 5 and have between 3 and 5 children. Figure 7.36 shows that the subtree rooted at “&phenomena;HydrospherePhenomena” satisfies the conditions of the WHERE clause.
The next object for evaluation is the Leaves object. Three queries illustrate the ability of OntoSELF+TQ to filter Leaves with various conditions in the WHERE clause.

VI) Leaves

a) SELECT Leaves WHERE Level between 3 and 3.

The above query displays all the leaf nodes on level three. Figure 7.37 displays the results of this query. It shows that 305 leaves exist on level three of the ontology.
The next query adds one more condition on the first one.

b) SELECT Leaves WHERE Level between 3 and 3 AND Parents between 2 and 2.

The above query displays all the leaf nodes on level three, which have just two parents. Figure 7.38 displays the results of this query. It shows that only sixteen leaf nodes out of 305 leaves on level three have exactly two parents.
One more condition is added on to the previous query to create the next query. Even less nodes should be expected in the result.

c) SELECT Leaves WHERE Level between 3 and 3 AND Parents between 2 and 2 AND Siblings between 3 and 4.

The above query displays all the leaf nodes on level three, which have just two parents and 3 or 4 siblings. Figure 7.39 displays the results of this query. It shows only two leaf nodes satisfy this query. Figure 7.38 has two marked leaves nodes correspond to these two nodes.

**Figure 7.38 The result for the second leaves query**
7.2.2 SELECT with Specific Node Context

Using an explicit node as the context of the query restricts the search in that the results are based on starting with the specified node. This evaluation follows the same approach as the previous section by providing several examples for each kind of object with a specific node context.

I) Path(up) and Path(down)

   a) SELECT Path(up) FROM “AnimalProduction”.

This query starts with the concept “AnimalProduction” and proceeds upward on all paths from the concept node until the path stops with a root concept. Figure 7.40 shows the path upward starting with “AnimalProduction and ending in the root node “HumanActivity”.
The next query changes the path direction.

b) SELECT Path(down) FROM “AnimalProduction”

The Path(down) is similar to the path up in that it starts at the specified node but follows the path(s) down until it ends in a leaf concept. The path starts on concept “AnimalProduction” and ends on a leaf node in the ontology. Figure 7.41 shows that five paths exist from the “AnimalProduction” concept node to a leaf node in the ontology.
c) SELECT Path(down) FROM “&numerics;Boundary” WHERE length between 2 and 9.

The above query displays all the paths which starts on concept “&numerics;Boundary” and ends on a leaf node and have a length between 2 and 9 inclusive. Figure 7.42 shows that the paths which start with the node “&numerics;Boundary” and end with leaf concepts “SeaSurface” and “LandWaterSurface” are the longest paths.
The next query changes the value for the length attribute to greatest to demonstrate that this result can also be visualized if the user is interested in just finding the longest paths down from “&numerics;Boundary”.

d) SELECT Path(down) FROM “&numerics;Boundary” WHERE length between greatest and greatest.

The above query displays the longest path(s) from all paths which start on concept “&numerics;Boundary” and end on a leaf node Figure 7.43 shows the visualization and the user can clearly see that there are two longest paths.
Figure 7.43 The result for the fourth path query

The next object for evaluation is the Sibling object. Two queries show that the OntoSELF+TQ filters the qualified siblings with various conditions.

II) Siblings

    a) SELECT Siblings FROM “Crystal” WHERE Hub between 3 and 8.

The above query finds all the siblings of the concept “Crystal” that have a hub value between 3 and 8. Figure 7.44 shows the visualization for this query. Six sibling concepts are found and are linked with their parent concept “FormOfSubstance Organization”. These six siblings must have at least two children to satisfy the hub attribute value of 3 or more.
Figure 7.44 The result for the first sibling query

The next query is similar to the previous one except the value for the hub attribute is changed to greatest.

b) SELECT Siblings FROM “Crystal” WHERE Hub between greatest and greatest.

The above query displays the sibling concepts of the concept “Crystal” with highest hub value among all its siblings. Figure 7.45 displays the produced visualization for this query. Two sibling concepts (marked) are found and are linked with their parent node “FormOfSubstance Organization”. The fact that Figure 7.45 shows concept “Crystal” helps the user see that the relationship between the returned two nodes and “Crystal” is the sibling relationship.
Figure 7.45 The result for the second sibling query

The next object for evaluation is the Children object. Three queries show that the OntoSELF+TQ filters the qualified children with various conditions.

III) Children

a) SELECT Children FROM “&phenomena;EarthSciencePhenomena” WHERE Children between 5 and greatest.

This query allows the user to examine the children of the specified node and to select those children which satisfy the conditions in the WHERE clause. Note that all the conditions in the WHERE clause are applied to the children concepts of “&phenomena;EarthSciencePhenomena”. The condition states that the children of “&phenomena;EarthSciencePhenomena” must have between 5 and the greatest number possessed by any of the children of “&phenomena;EarthSciencePhenomena.” The concept(s) possessing the greatest number of
children are, in other words, the bushiest child(ren) of “&phenomena;EarthSciencePhenomena.” The visualization produced by the OntoSELF+TQ is shown in Figure 7.46.

![Figure 7.46 The result for the first children query](image)

Four children concepts are found and are linked with their parent node “&phenomena;EarthSciencePhenomena”. The one underlined (underlining not produced in by visualization) is the one with the greatest number of children as demonstrated by changing the value for the attribute Children in the next query.

b) SELECT Children FROM “&phenomena;EarthSciencePhenomena” WHERE Children between greatest and greatest.

The above query identifies the bushiest child of “&phenomena;EarthSciencePhenomena” and is shown in the visualization produced by the OntoSELF+TQ in Figure 7.47. It shows the bushiest child of “&phenomena;EarthSciencePhenomena” is “&phenomena;GlobalOscillation”. It is marked in Figure 7.46.
The next query changes the attribute from Children to Descendants.

c) SELECT Children FROM “&phenomena;EarthSciencePhenomena” WHERE Descendants between 20 and greatest.

The above query finds the children concepts of “&phenomena;EarthSciencePhenomena” having at least 20 descendants and up to the most number of descendants that any child node of “&phenomena;EarthSciencePhenomena” has. The visualization produced by the OntoSELF+TQ is shown in Figure 7.48. Only two children of “&phenomena;EarthSciencePhenomena” have at least 20 descendants.
Figure 7.48 The result for the third children query

To show the concept “Animal” has the most children in Figure 7.22. The following query is processed.

   d) SELECT Children FROM “Animal”.

Figure 7.49 shows the result of above query and 27 children concepts are found.
Figure 7.49 The result for the fourth children query

The next object for evaluation is the Subtree object. Two queries are provided to demonstrate using the subtree object with conditions in the WHERE clause.

IV) Subtrees

Before beginning general examples of querying with for subtree objects, examples to illustrate the difference between subtrees and children objects are given. Subtree objects could be any subtree nodes that exist as descendants of the specific concept. For example, the query “SELECT Subtree FROM “&numerics;Quantity” WHERE Level between 7 and 7” returns all the subtree nodes of the concept “&numerics;Quantity”, which are on the level 7. The result is shown in Figure 7.50.
Figure 7.50 The result for the query level 7 subtree of “&numerics;Quantity”

The concept “&numerics;Quantity” is a root node, so its children are on the level two. The query “SELECT Children FROM ‘&numerics;Quantity’” is shown in Figure 7.51. Based on these two figures, no concepts are shown twice except the specific concept “&numerics;Quantity”. These examples illustrate the difference between the subtrees and children objects for the specific node context.
Figure 7.51 The result for the query children of “&numerics;Quantity”.

The next query shows the use of the descendant attribute with the subtree object.

a)  SELECT Subtrees FROM “&phenomena;EarthSciencePhenomena” WHERE Descendants between greatest and greatest.

The above query finds all subtrees of “&phenomena;EarthSciencePhenomena” with the most number of descendants. Logically for a subtree to have the most number of descendants it must be a subtree rooted by one of the children of “&phenomena;EarthSciencePhenomena.” Since the object being selected is a subtree, the visualization requires showing all of descendants. Figure 7.52 show the visualization produced for this query. The returned subtree roots (marked) show that there are two subtrees that tie with the most number of descendants.
The next query adds another condition using the attribute Height.

b) SELECT Subtrees FROM “&phenomena;EarthSciencePhenomena” WHERE
   Descendants between 20 and greatest and Height between greatest and greatest.

The above query finds the subtrees of “&phenomena;EarthSciencePhenomena” with the highest subtree height and the number of the descendants between 20 and the most number descendants. Again the subtrees with the greatest height must be selected from the children concepts of “&phenomena;EarthSciencePhenomena.” The visualization resulting from this query is shown in Figure 7.53. It shows the returned subtree roots (marked) with their descendants. The results are actually the same as the first subtree query. This is because that two subtree roots have the same number of descendants (both have 37 descendants) and the same subtree height.
Figure 7.53 The result for the second subtree query

The next object for evaluation is the Leaf object. Two queries show the use of the WHERE clause with the leaves object.

V) Leaves

a) SELECT Leaves FROM “&property; LinearExtent” WHERE parents between 2 and max.

The above query asks the leaves concepts of “&property; LinearExtent” which have more than one parent. Figure 7.54 shows that two leaves concepts are found and are connected to the “&property; LinearExtent” concept.
This query changes the condition to use the level attribute.

b) SELECT Leaves FROM “&property;LinearExtent” WHERE Level between greatest and greatest.

The above query finds the leaf concepts of “&property;LinearExtent” which are on the deepest level comparing to other leaves concepts of “&property;LinearExtent.” Only one leaf node is found and is linked to the concept “&property;LinearExtent” as shown in Figure 7.55.
7.2.3 SELECT with a Nested Query Context

In the previous section the context is one specific concept in the ontology. Here the capability of OntoSELF+TQ to handle nested queries, which is where the context of the outer query is the result of another inner query. The inner query can be any of the three kinds of SELECTs; therefore, the example queries in this section are not organized on the type of object selected but on the type of the inner query. The evaluation starts with using for the inner query the first kind of SELECT as described in section 7.2.1 where the context is the query is of the whole ontology and, therefore, no FROM clause is used in the inner SELECT.

I) Inner SELECT type 1

a) SELECT Children FROM (SELECT Levels WHERE Level between 5 and 5 ).
The above query finds all children concepts from the concepts of the inner query results, where the inner query returns all the concepts on level 5. The visualization produced by the OntoSELF+TQ for this query is shown in Figure 7.56.

![Figure 7.56 The result for the query 7.2.3 I-a](image)

If one zooms in on the query results in Figure 7.56, it can be seen that the red nodes represent the inner query results and the green nodes represent the outer query results. Some of the node coloring, however, is difficult to see when the node label covers over the node. The next query adds a condition using the attribute Children which is applied to the outer query results.

b) \[
\text{SELECT Children FROM (SELECT Levels WHERE Level between 5 and 5) WHERE Children between 2 and 5.}
\]

The above query finds all of children concepts from the concepts returned by the inner query, i.e., those on level 5, but requires that the children concepts themselves must between 2 and 5 children. Figure 7.57 shows the result of this query with the red nodes representing the inner query result and the green nodes representing the outer query results.
The next query changes the object being selected and the attribute to descendants in the condition on the outer query.

c) SELECT parents FROM

(SELECT Levels WHERE Level between 5 and 5)

WHERE descendants between greatest and greatest

The above query finds all parent concepts of those concepts that are on level 5 and further restricts the parent concepts to those having the most descendants. Figure 7.58 shows the results of this query with the red node representing a concept at level 5 with a green parent node that has the greatest number of descendants of all parent nodes of concepts at level 5.
The following examples continue with more examples of SELECT type 1 but demonstrated path up and path down more fully and a more complex WHERE clause.

d) SELECT Path(up) FROM
    (SELECT Nodes WHERE Hub between 3 and 9 and Siblings between 2 and 3).

The above query finds all paths which start from the root nodes and end on the concepts of the inner query results, where the inner query returns all of the nodes with hub value between 3 and 9 and the number of siblings between 2 and 3. Figure 7.59 shows the results of this query with the red nodes representing the results of the inner query and the green nodes representing the additional nodes needed to show the path up to the root.
The next query adds a condition on the length attribute to the outer query for path up object

e) SELECT Path(up) FROM

(SELECT Nodes WHERE Hub between 3 and 9 and Siblings between 2 and 3)

WHERE Length between 6 and 6.

The above query basically filters further the results from query 7.2.3 I-d by requiring that only those paths up which have a length of 6 are to be shown. Figure 7.60 shows the results. Comparing these results to those in Figure 7.59, one can see that many paths are filtered out because their lengths are not equal to 6.
Next, examples are presented where the inner SELECT is of type 2, that is, the FROM clause is used with a specific node context.

II) Inner SELECT type 2

a) SELECT Path(down) FROM ( SELECT Siblings FROM Cryosphere ) WHERE Length between 2 and 4.

The above query finds all paths of length between 2 and 4 which start from the concepts that are siblings of “Cryosphere” and end on the leaf nodes. Figure 7.61 shows the query result with the red nodes representing the siblings of Cryosphere that root a path down of length between 2 and 4 and the green nodes representing the additional nodes to show the path down.
Figure 7.61 The result for query 7.2.3 II-a

This example illustrates selecting another kind of object besides path objects.

b) SELECT Siblings FROM (SELECT Parents FROM Region).

The above query finds all siblings of the parents of the “Region” concept. Figure 7.62 shows the query results with the red nodes representing the two parent of “Region” and the green nodes representing the siblings of these two parents “SpatialObject” and “&numerics;GeometricalObject_2D”. For the nested query with sibling concept, the result also shows the parent objects. To differentiate the parent object with them, the third color is introduced and the parent concept “&numerics;GeometricalObject” is colored as blue.
Figure 7.62 The result for query 7.3.3 II-b

The next query adds a condition to the outermost query using the children attribute.

\[\text{c) SELECT Siblings FROM ( SELECT Parents FROM Region ) WHERE Children between 2 and 2}\]

The above query finds all of siblings of the parent of “Region” that have exactly two children. Figure 7.63 shows the results of the query with the red nodes representing the parents of “Region” and the green node representing the one sibling that has exactly two children. The parent concept “&numerics;GeometricalObject” is colored blue.
Comparing to Figure 7.62, Figure 7.63 shows that some of the nodes have been filtered out since they did not satisfy the additional constraint on having exactly two children. The next query changes the value for the children attribute.

\[
\text{d) SELECT Siblings FROM ( SELECT Parents FROM Region ) WHERE Children between greatest and greatest}
\]

The above query finds all siblings of the parent concepts of “Region” which have the most children nodes compared to other sibling nodes. Figure 7.64 shows the results of this query. Note that the resulting node in Figure 7.64 differs from that in Figure 7.63 since the condition on the attribute Children is different.
The next query focuses on selecting descendant objects.

e) SELECT Descendants FROM (SELECT Siblings FROM phenomena;Slide).

The above query displays all descendants of the sibling concepts of “phenomena;Slide”. Figure 7.65 shows marked red nodes that represent inner query results as siblings of phenomena;Slide. The green nodes are all the descendants of the siblings.
The next query adds a condition on the interval rank attribute in inner query.

f)  SELECT Descendants FROM

    ( SELECT Siblings FROM phenomena;Slide

    WHERE IR between 3 and 4)

The above query finds all descendants of the sibling concepts of “phenomena;Slide” where those siblings have an interval rank of between 3 and 4. Figure 7.66 shows the results of this query. It can be seen that several sibling nodes have been eliminated from the previous query result shown in Figure 7.65 since they do not meet the condition on the interval rank attribute.
The next query adds one condition to the outer query by restricting the descendants to be at the greatest level for all the descendants.

\[ g) \quad \text{SELECT Descendants FROM (SELECT Siblings FROM phenomena;Slide)}} \]

\[ \text{WHERE Level between greatest and greatest.} \]

The above query finds all descendants at the greatest level for the siblings of “phenomena;Slide.” Figure 7.67 shows the results of this query. It can be seen by comparing these results to Figure 7.65 that only descendants at the greatest level are displayed and directly connected to their appropriate ancestor that is a sibling of “phenomena;Slide.” All other descendants have been filtered out because they do not meet the greatest level condition.
The next query focuses on selecting leaf objects for the outer SELECT while using an inner SELECT of type 2

h) SELECT Leaves FROM

(SELECT Children FROM &numerics;Quantity WHERE IC between 0.6 and 0.8 )

The above query finds all leaves of the children concepts of the concept “&numerics;Quantity” where those children concepts have an information content value of between 0.6 and 0.8. Figure 7.68 shows the query results with the red nodes representing the children nodes of “&numerics;Quantity” that have the required information content and the green nodes representing the leaf nodes reachable from the red nodes.
The next query starts with the previous query but adds one condition on sibling attribute to constrain the returned leaf objects.

i) SELECT Leaves FROM 

(SELECT Children FROM &numerics;Quantity 

WHERE IC between 0.6 and 0.8 )

WHERE Siblings between 4 and 5

The above query finds all leaf nodes with 4 or 5 siblings, and these leaf nodes descend from the children concepts of the concept “&numerics;Quantity” where those children concepts have an information content value of between 0.6 and 0.8. Figure 7.69 shows the result of the query. It can be seen by comparing these results to Figure 7.68 that only the left tree rooted by “&property;BiologicalQuantity” from that figure is included in the results since its leaf nodes have between 4 and 5 siblings.
The next two examples are presented where the inner SELECT is of type 3, that is, the FROM clause uses a nested query.

III) Inner SELECT type 3

This first query uses a two level nested query and WHERE clause on the innermost nested query.

a) SELECT Children FROM

        (SELECT Children FROM ..........level 0

        (SELECT Level Where Level between 6 and 6) ..........level 1

        .............level 2

WHERE Hub between 2 and 4).

The above query finds all children of the children concepts of those concepts at level 6 where their children concepts have a hub value of between 2 and 4. The level 1 plus level 2 query will be processed by clicking “Visualize” button on “Nested Query One” panel (Figure 7.70).
After pressing the button “Visualize” in Figure 7.70, the gml file is generated to represent the result of level1 plus level 2 of the above query. The left picture of Figure 7.72 shows the red nodes as the result from level 2 and the green nodes as the result from level 1.

To get the final result, the user needs to click the “Visualize” button at the bottom of the screen shown in Figure 7.71. Another gml file is generated to represent the final result of level 0. In the right picture of Figure 7.72, the red nodes are taken from the green nodes in the left result and then the green nodes in the right result are their children and the final result of the query. The reason that only two nodes out of four from the left result are shown in the right result is that the other two nodes do not have children.

Figure 7.70 The interface for level 1 of 7.2.3 III-a

Figure 7.71 The main interface for 7.2.3 III-a
The next query shows using a condition on the outermost SELECT and no conditions on the two inner SELECTS.

a)  SELECT Parents FROM

   (SELECT Levels FROM
    (SELECT Parents FROM property;Value ) ) … level 2

   WHERE IR between 2 and 5

The above query finds all parents with an interval rank value of between 2 and 5 where these are parents of nodes at the same levels as the parents of “property;Value.”
The level 2 nested query returns the parent node “property;Description” for the concept “property;Value”. The node “property;Description” is on level two of the ontology. The left image of Figure 7.73 shows all of the nodes on level two. The node “property;Description” is marked and is colored as blue because it belongs to both nested level one result and level two result. The other nodes are colored as green. The right image of Figure 7.73 shows the result of the entire query. The red nodes represent the result from the left image where their IR value is between 2 and 5. The green nodes represent qualified parent nodes.

### 7.2.4 CropCircle Topology Queries

The following topology understanding tasks were employed to evaluate CropCircle visualization software (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 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 node in the ontology that has 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 root nodes whose subtree has a depth of 5 or more are to be selected.

It was also previously shown that OntoSELF (Somasundaram, 2007) could perform those tasks by setting certain filtering and weighting parameters correctly. The objective of OntoSELF+TQ is to provide a query language to allow a user to more easily perform topology understanding tasks instead of having to tediously browse through a complex visualization of a large ontology. All these five tasks have been easily translated to queries and uses as examples in the section 7.2. Their results are shown in Figures 7.47, 7.52, 7.55, 7.23 and 7.27. These queries are further discussed in section 7.3 since they are also used in the user evaluation of OntoSELF+TQ.
7.3 Topology tasks using Graph Tasks Analytic Activity

In section 4.3 a discussion is provided on several categories of visual analytic tasks which are categorized into the following four groups: topology-based tasks, attribute-based tasks, browsing tasks, and overview tasks. OntoSELF+TQ provides functionality in all of these categories of visual analytic tasks. This section demonstrates how OntoSELF+TQ functionality maps to these general categories.

Within the topology-based tasks there are four subtasks, “adjacency”, “accessibility”, “common connection” and “connectivity”.

1) Adjacency subtasks: OntoSELF+TQ clearly is able to handle a variety of adjacency subtasks and provides numerous ways to select the focal node for which adjacency questions are being asked. The examples given in section 4.3 for adjacency are to find the set of nodes adjacent to a node or to find the node(s) that have a maximum number of adjacent nodes. OntoSELF+TQ can handle adjacency queries since a user, for example, can create the query “SELECT Children FROM <a specific node>” and also the query “SELECT Parents FROM <a specific node>” Figure 7.46 illustrates selection of the children adjacency concepts but also adds a restriction on the value for children attribute of those children of the specific node. The user can create the query “SELECT Nodes WHERE Hub between max and max”. This query returns the set of nodes which have the maximum number of adjacent nodes, including parents and children.

2) Accessibility subtasks: The accessibility subtasks can be seen as a repetition of the adjacency task; however, instead of direct adjacent, the connections are indirect. For example, the task to find the set of nodes accessible from a node, the user can create the query “SELECT Path(down) FROM <a specific node>” and the query “SELECT Path(down) FROM <a specific node>” An example of the path down query is provide in Figure 7.41 where the specific node is “AnimalProduction.”

3) Common connection tasks: An example provided for such a task is “Given nodes, find a set of nodes that are connected to all of them.” This category of tasks is not generalized within OntoSELF+TQ but there are examples of queries that do provide a common connection through the is-a links. For example, “SELECT Descendants FROM <specific node> produces all nodes that are connected to the node through its subtree and with
SELECT Ancestors FROM <specific node>, the specific node is connected to all its ancestor nodes.

4) Connectivity tasks: An example provided for such a task is “Find the shortest path between two nodes” or “Identify connected components.” Again, this category of tasks is not generalized within OntoSELF+TQ but there are examples of queries that do provide connectivity. For example, “SELECT PathDown FROM <specific node> WHERE length is between greatest and greatest.” This query does not allow specifying both a begin node and end node as the query using “between two nodes” but the end node is assumed to be one of the leaf nodes farthest away from the specific node.

Attribute-based tasks are those “on the nodes” and “on the links.” Since OntoSELF+TQ has only been developed based using just the one kind of relationship, the is-a links, there are no attribute tasks based “on the links”. OntoSELF does provide numerous attribute-based tasks “on the nodes” where attributes are specified by the columns of the SELECT tables based on the type of object being selected. The attribute values for OntoSELF+TQ are associated with characteristics of the concept node within the ontology. The attribute-based task “on the nodes” basically filter out the nodes that do not meet the required attribute values. OntoSELF+TQ does provide this filtering capability through its WHERE clause. For example, the user can create the query “SELECT Nodes WHERE Children between 10 and max”. The query returns all of nodes whose value for the attribute number of children is greater than 9. This example query result can be seen in Figure 7.23. OntoSELF+TQ, however, goes beyond “on the nodes” and even provides attribute-based tasks “on the paths” and “on the levels.” For example, the user can also create the query “SELECT Path WHERE Length between 4 and 4”. The query returns all of the paths with length 4 and the result is shown in Figure 7.29.

OntoSELF+TQ also provides functions to implement filter analytic tasks and extremum and range analytic tasks described in (Amar et al. 2005). The filter tasks are to find data cases satisfying some given conditions on attribute values. OntoSELF+TQ determines extremum values for several characteristics of the ontology, for example, the minimum and maximum number of children, hub value, number of levels, width size of a level and so on. It allows specifying a range value for attributes. For example, using OntoSELF+TQ, the user can create the query “SELECT Roots WHERE Height between 5 and max”. The query returns all of the roots whose subtree height is over than 4 and the result is shown in Figure 7.25. Another query
example “SELECT Levels WHERE Width between 10 and 30” requires all of the nodes on the level where there are at least 10 nodes and at most 30 nodes on those levels. This result is shown in Figure 7.32.

The extremum tasks can be done using the context of the whole ontology or the context of the FROM clause in the SELECT. The ‘max’ or ‘min’ argument is used in the BETWEEN part of the condition when the context is the value determined for the whole ontology and the ‘greatest’ or ‘least’ argument is used in the BETWEEN part of the condition when the context is the result of the FROM clause. By using OntoSELF+TQ, the user can create the query “SELECT Siblings FROM ‘Crystal’ WHERE Hub between greatest and greatest”. The query returns the siblings of the concept “Crystal” whose hub value is the highest compared to just the other siblings of “Crystal”, and the result of this query is shown in Figure 7.45.

For both the general browsing tasks and overview tasks, OntoSELF+TQ provides the SELECT type 1 query where the context is the whole ontology so the user can query about general structure of the ontology. For example, the user can create the following queries to get a general overview of the ontology.

1) SELECT Roots. See Figure 7.25
2) SELECT Leafs. See Figure 7.37 but adds restriction for only level 3 leaves.
3) SELECT Path WHERE length BETWEEN max and max. See Figure 7.30
4) SELECT Level WHERE width BETWEEN max and max. See Figure 7.33

Browsing the whole ontology can be done by visualizing the ontology and using the zooming and rotating features of VTK. OntoSELF+TQ provides the ability to scan through an alphabetical list of all concepts within the ontology to allow the user to explore related (siblings, parents, ancestors, etc.) concepts to a particular concept selectable from the list. This ability provides selective browsing by using a SELECT type 2 where a specific node is used in the FROM clause.

To sum up, the OntoSELF+TQ provides a wide range of capabilities to handle a variety of subtasks within the four categories of visual analytics tasks on the structure of an ontology. These tasks can be performed by OntoSELF+TQ through queries that help the user explore and understand the structure of the ontology.
7.4 User Evaluation

To evaluate the OntoSELF+TQ by actual users besides the developer of OntoSELF+TQ, a human studies experiment was designed. The details of the IRB and the pre and post surveys used in the study can be found in Appendix I. Volunteers from both the undergraduate and graduate students in the Computer Science department were solicited to participate in this study. Two undergraduate students from Computer Science department volunteered. These two students were given a brief tutorial on graph structures and terminology, shown several examples of OntoSELF+TQ SELECT queries and shown various zooming and rotating features of VTK. Both of these students did have some basic ontology knowledge but neither was familiar with graph visualization tools. One student has experience with the SQL query language and the other did not.

Eight topology understanding tasks were given to them in order to evaluate the usefulness and effectiveness of the OntoSELF+TQ.

1. Find the bushiest (most children) child node of “phenomena-EarthSciencePhenomena”
2. Find the largest (most descendants) subtree of “phenomena-EarthSciencePhenomena”
3. Find a deepest node from “property-LinearExtent”
4. Find 3 nodes in the ontology with at least 10 children
5. Find 3 root nodes with a height of at least 5.
6. Find the ancestors of leaf nodes that are on paths of length 5 or more
7. Find the bushiest children of the nodes on the widest level
8. Find the bushiest children of the siblings of the nodes that are most connected (hub) within the whole ontology.

The first five queries are the exact queries used in CropCircles experiments described in (Wang and Parsia, 2006). In the CropCircle evaluation, these queries were presented to the user who was required to look at a general visualization of the ontology and first find the specific node and then determine the answer from examining the visualization. These queries have been previously discussed in section 7.2.4 and their results are shown in Figure 7.47, 7.52, 7.55, 7.23 and 7.27 respectively. They have been converted to a corresponding OntoSELF+TQ SELECT query. The last three queries are added to evaluate the user’s ability to develop nested queries.
The sixth query finds all the ancestors of leaves that are at the end of a path from a root where the path is 5 or longer. This query can be created in OntoSELF+TQ as

\[
\text{SELECT path(up) FROM (SELECT leaves WHERE Level between 6 and MAX).}
\]

The inner query finds leaves that are at level 6 or more (remember level 0 of the ontology is the dummy root) and then the outer query finds the paths up from these leaves. The result is shown in Figure 7.74. The red nodes represent the leaves that are on a path from the root that is 5 or more and the green nodes represent the additional nodes for each path up from those red leaf nodes.

![Image](image.png)

**Figure 7.74 The result for the sixth Topology understanding query**

The seventh query finds the children nodes of the nodes on the widest level where these children nodes have the most children themselves. This query can be created in OntoSELF+TQ as

\[
\text{SELECT children FROM (SELECT Level WHERE width between max and max) WHERE children BETWEEN greatest AND greatest.}
\]
The query result is shown in Figure 7.75 with the red nodes representing the nodes on the level with the greatest width and the green nodes representing the red nodes’ children with the most children.

**Figure 7.75 The result for the seventh Topology understanding query**

The eighth query requires two levels of nested queries. It finds the children nodes of the siblings of the nodes in the ontology that have the maximum hub value, i.e., the maximum number of parents and children combined this query can be written as

SELECT children FROM

(SELECT Siblings FROM

(SELECT nodes WHERE Hub between max and max))

WHERE children BETWEEN greatest AND greatest’

The left image of Figure 7.76 shows the result of the inner query and its nested query. The red node represents the node with the maximum hub value, i.e., “Animal” and the lower green node represents the sibling of “Animal”, i.e., “Plant”. The right image of Figure 7.66 shows the result of the outer query. The red node represents the result from the left image, that is the sibling “Plant” and the green node represents its child with the greatest number of children itself, i.e., “Vegetation.”
The students were given 15 minutes training on the basic structure of the ontology, OntoSELF+TQ tool and VTK visualization tool. After the students took the brief pre-survey, they were asked to complete the above eight topology understanding queries and to record an estimate of the time it took for them to develop each query.

Table 7.1 shows the results of the CropCircle experiment. The time is measured in seconds. The CropCircle time represents the time it took the user to answer the query by browsing a general visualization of the ontology. The time for the finding deepest node is not given below because the error rate was used instead of the time since most users in the CropCircle experiment did not find the deepest node. There are four times recorded for the OntoSELF+TQ. The first two columns describe the time spent for the two students to translate the described task into the appropriate query. Two students had the tutorial on the different time period. The student without the SQL experience is given more tutorial focusing on the query implementation. Especially he is given more detailed explanation on the concept max and greatest. The student with the SQL experience is given more tutorial focusing on the structure of the ontology. The student without the query experience had three correct query answers and the student with the query experience had two correct query answers. However, the first student only generated the first five queries and the second student generated all of the eight queries. The reason for the second student got the lower score was that he confused the definition between max and greatest. But most of the tasks require this knowledge. The experienced user who worked with the SQL query language before and understood the system could quickly create the query. The inexperienced user had
difficulty in creating the queries for the topology tasks. It took much longer time for the inexperienced user to figure out how to translate the general task description into the appropriate query. But once the query is set up, the execution time is very short as indicate by the times in the last column of Table 7.1.

<table>
<thead>
<tr>
<th>Task</th>
<th>CropCircle</th>
<th>OntoSELF+TQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bushiest Child</td>
<td>27.42</td>
<td>19.04</td>
</tr>
<tr>
<td>Largest Subtree</td>
<td>26.09</td>
<td>16.25</td>
</tr>
<tr>
<td>Deepest Node</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>3 Nodes with 10 Children</td>
<td>19.56</td>
<td>6.14</td>
</tr>
<tr>
<td>3 Subtrees of Depth 5</td>
<td>47.90</td>
<td>20.4</td>
</tr>
<tr>
<td>Ancestors of Leaves</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Bushiest Child of Widest Level</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Bushiest Child of Siblings</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 7.1 Timing Comparison with CropCircle (* represents the students got the correct answer on that query)
Both of students gave the right answer for the tasks based on the visualizations produced by the correct queries. From these correct visualizations, they were able to determine the correct answer to task. For the first bushiest child task, the correct query should be “SELECT Children FROM ‘phenomena-EarthSciencePhenomena’ WHERE Children between greatest and greatest”. The second student used max value instead of greatest for the condition attribute constraint. Because the max represents the maximum children for the node of the entire ontology and this node is not the child of “phenomena-EarthSciencePhenomena”, no result is returned. After he found out that he did not apply the right query, instead of changing to greatest value, he gave the actual number for the condition constraint. The result is saved and shown as below:

Figure 7.77 The student’s evaluation result for the bushiest node query

Figure 7.77 includes the right answer (marked) and other incorrect answers. The student picked the node “phenomena-Weather” randomly and wrote it as the result.
On the post-survey, the second student complained that he was confused with attribute values “max” and “greatest”. If he was given more tutorial, he could do a better job. Both students agreed that the visualization result is easy to understand. The fact that both gave correct answers when examining correct visualization results demonstrates this. OntoSELF+TQ eliminates the tedious browsing/searching process and provides convenient visualization results. The initial drawback is learning the basics of the query language, but once learned it can allow the user to easily and quickly gain important insights about the characteristics and structure of the ontology.

8. Conclusion and Future Work

Many tools for the visualization of ontologies have been developed. Many are embedded in ontology editors and are limited to 2D visualizations of various components of the ontology. Recently more research in 3D visualization of ontologies has been pursued (Katifori et al., 2007). The objective of this thesis research is the development of a topology query language that can be used to visually explore in 3D the complex structure of large ontologies, i.e., those with over 1000 nodes outside the scope of an ontology editor. To accomplish this objective, the OntoSELF+TQ system has been designed, implemented and evaluated. OntoSELF+TQ has been developed as an enhancement to the existing 3D ontology visualization software of OntoSELF (Somasundaram 2007) and OntoSELF-v2 (Yu 2008). OntoSELF+TQ represents a unique approach to 3D visualization in that its querying capabilities focus on the ontology as a graph structure. The query language is structured based on the kinds of objects that exist in graphs and on the attributes or characteristics of these objects based on their context within the graph structure.

OntoSELF+TQ provides a SQL-like querying capability over graph objects making up the 3D visualization of the ontology structure. This querying capability allows the specification of a context either through focus on a specific concept node in the ontology or through the result of a previous query, i.e., the nested query context. The ability to handle filtering and extremum analytic tasks suggested in the research literature on visualization of the graph is incorporated into OntoSELF+TQ. OntoSELF+TQ provides flexibility to the end users by allowing them to select various objects and specify multiple attributes of those objects to be visualized. The nested query extends the power of implementation of the query tasks. The user can also save the results produced by OntoSELF+TQ for the future study and review.
A system evaluation tested the implementation of OntoSELF+TQ. The results calculated manually on the UMLS Semantic Network for a set of queries were compared to those produced by OntoSELF+TQ. This evaluation was the first software quality check on OntoSELF+TQ. Next a full feature evaluation was done on OntoSELF+TQ and covers the three types of queries. The results show that OntoSELF+TQ can provide flexible querying for the user to explore and understand the ontology structure. A small user evaluation experiment was conducted to determine how users felt about OntoSELF+TQ. This limited evaluation provided good feedback for the OntoSELF+TQ developer and did show that the users thought it a useful tool although they felt they needed more training on it to become better at developing the queries.

The development of OntoSELF+TQ has resulted in a sound foundation from which to investigate more features for 3D ontology visualization. For example, OntoSELF+TQ currently calculates the values for the attributes of the graph object only in the context of the entire ontology. An option to recalculate “local” attribute values based on considering a nested query resulting as a new subontology could be valuable. The greatest and least constraint values were included to provide a limited form of a “local” filtering option but recalculation of attribute values for a local context is not provided. For example, given a concept “A”, the user may want to view that concept as the root of a sub-ontology and be able to perform visualizations on that sub-ontology and have attribute values recalculated for this local view of concept A as the root of a new ontology and ignore the rest of the ontology. An option could be added to allow the user to select either local or global function when using the FROM clause of the query so when query processing occurs, the correct attribute values could be determined based on the option selected. This feature would make OntoSELF+TQ even more flexible and powerful.

OntoSELF+TQ creates the graph structure using the hierarchical is-a relationships for an ontology. User defined relationships have not been incorporated into the query language. Future enhancements to the ontology query language need to incorporate vocabulary from social network analysis in order to visualize the graph structures created by user defined relationships. The enhanced system should enlist all of the relationships and provide the function to allow the user to select the desired relationships and contextual querying based on the type of relationship included in the visualization. The topology visualization results should be generated based on the chosen relationships.
One of the participants in the user evaluation experiment complained that the larger VTK visualization views were slow to navigate. More research on VTK should be done to improve the above performance. The ability to report actual attribute values in the VTK visualization would be a valuable feature. The mouse tooltips functionality needs to be examined in order to provide this capability to the VTK visualization result. When the user drags the mouse on a node, the attribute values of the node (level, number of children, number of descendants etc.) should be displayed along with the node.

Although a variety of enhancements can be done to improve OntoSELF+TQ, as implemented, it provides a unique approach to 3D ontology visualization that allows the user to query on the ontology structure and get a visualization based on the results of the query. It provides flexibility to each user in that the query can be simple and use the complete ontology for its context when in a general browsing or overview mode for ontology viewing or it can be complex and focused on a smaller component of the ontology when detailed or task oriented ontology viewing is needed. To summarize, OntoSELF+TQ allows the user to create queries with desired objects and constrained attribute values in a structured query language and produces focused ontology visualizations. It can be used to help the user better understand the structure and content of an ontology.
9. Reference


Chen, Li, Martone, Maryann, Gupta, Amarnath, and Fong, Lisa, and Wong-Barnum, Mona (2006). OntoQuest: Exploring Ontological Data Made Easy. Int. Conf. on Very Large Databases (VLDB), Seoul, S. Korea.


Tartir, Samir, Arpinar, I. Budak., Moore, Michael., Sheth, Amit P. and Aleman-Meza, Boanerges (2005). OntoQA: Metric-Based Ontology Quality Analysis. *IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources, Houston, TX, USA.*


Appendices

Appendix I VTK Installation
This appendix includes the details of the VTK installation. The document initially created by Ram Somasundaram was modified to make the process clearer.

Installing VTK 4.4.2

1) Get the VTK 4.4.2 from Dr. Cross’s handout directory and extract it into the C:\VTK directory.

2) Create a folder C:\VTKBin to store the compiled binaries.

3) Get the graph.zip package from Dr. Cross’s handout directory and extract it into the C:\VTK directory. Now you should see the folder Graphs inside the VTK folder.

4) Create a file named “LocalUserOptions.cmake” under the C:\VTK directory. The file should contain just the following line “SUBDIRS(graph)”.

5) Open the “Graphs” folder inside C:\VTK and make the following changes. Open the CMakeList.txt file and comment out line number 78 (ADD_LIBRARY(vtkGraphs ${Graphs_SRCS} ${GraphsInstantiator_SRCS}))

6) Get vtkGraphAttributes.cxx and vtkGraphAttributes.h two files from Dr. Cross’s handout directory and save them in the Graphs folder.

7) Edit the files to remove any #include "CGRAPHSConfig.h" line

8) Edit CMakeLists.txt inside the Graphs folder to add vtkGraphAttributes.cxx under Kit_SRCS

9) Download ActiveTCL8.4.18 and install it in the default path.

10) Download CMake 2.6.2 and install it in the default path.

11) Start CMake, provide the source codes and binaries paths and the compiler to CMake,
   a. The source code path should be the C:\VTK
   b. The binary path should be C:\VTKBin
   c. The compiler is chosen VisualStudio.net 2003.
   d. Select the “show advanced options” check box
12) Make sure turn on
   a. **BUILD_SHARED_LIBS**: this causes the VTK dlls to be built
   b. **VTK_USE_HYBRID**: this causes vtkHybrid.lib/dll to be built
   c. **VTK_USE_PARALLEL**: this causes vtkParallel.lib/dll to be built
   d. **VTK_USE_PATENTED**: this causes vtkPatented.lib/dll to be built
   e. **VTK_WRAP_TCL**: This creates the wrapper for tcl files. (Essentially creates new
dlls that will allow the tcl program to be linked to the actual source)

13) Click the Configure once and it will add few more variable in the advance option. Without
    changing the option click the configure button again and then click the OK button.

14) This will create the binaries in the C:\VTKBin folder

15) Now the binaries have to be compiled and for this open the C:\VTKBin folder

16) Open the **vtk.sln** file in VisualStudio.net 2003

17) Change the build mode to ‘Release’ (The build mode appears as a drop down box below
    the menu bar and default will be set to ‘Debug’ mode. Change this to Release)

18) Select the ‘Build’ option in menu bar and then click ‘Build ALL_BUILD’

19) This will take a solid 45 minutes to build in a 2GB System.

20) The set up is now complete.

21) To test the installation success, select any .tcl file in the folder
    C:\VTK\Graphs\Examples\TCL, open the file with C:\VTKBin\bin\release. If the
    visualization result is pop up, that proves the installation is successful.

22) The OntoSELF+TQ is executed by the color.tcl file. The source code has shown the
    information.

**Notes on VTK execution through TCL**

The `vtkSelectNthBestNodes` function of tcl file sets the number of Nodes that are labeled in
VTK visualization. The number value in OntoSELF+TQ is set to 450 and that value can be
changed. If the number is changed to 300, then only 300 hundred nodes will be labeled on the
higher level. The `vtkLookupTable` function set the colors for the nodes in the visualization. The
last four numbers in nodeTable command represent the weight of RGBA colors.
Appendix II Human Research Application

MIAMI UNIVERSITY
APPLICATION FOR APPROVAL OF RESEARCH INVOLVING HUMAN SUBJECTS
COVER PAGE

All research projects involving human subjects must submit an application for review and approval by the Institutional Review Board for Human Subjects (IRB) or the Psychology Departmental Human Subjects Review Board (DRB) prior to the initiation of the research. Research that will be conducted with external or internal grant funding shall be submitted to the IRB. Research that involves no greater than minimal risk and will be conducted without external or internal grant funding shall be reviewed by the IRB or the DRB. If you have questions, call the Office for the Advancement of Research and Scholarship: (513) 529-3600.

A. PROJECT PERSONNEL

☐ Faculty ☐ Staff ☐ Undergraduate ☑ Graduate

Principal Investigator(s) (PI's) **Full Name**, Department and **UNIQUE ID** (ex: smithjha) Date completed CITI & Practicum


Faculty Advisor (if PI is a student) and Department Date completed CITI & Practicum


Other Personnel who will interact with subjects Date completed CITI

None

B. CONTACT INFORMATION FOR PI and Faculty Advisor (if PI is a student)

Campus or Postal Address(es)

PI:  ☐ GIVE YOUR INFORMATION HERE
Advisor: 201E Benton, Miami University Oxford, Ohio 45056-1601. OFFICE PHONE:(513) 529-0789

PI’s E-mail’s Address peiz@muohio.edu Phone? YOUR PHONE #

Faculty Advisor’s E-mail Address crossv(at)muohio.edu Phone:(513) 529-0789

C. PROJECT TITLE OntoSELF-TQ: A Topology Query System for OntoSELF

D. FUNDING SOURCE Graduate Assistantship ___If external, OARS Proposal Approval Form number

E. PROJECT DATES Beginning __April 2009 Ending __April 2009

F. TYPE OF APPLICATION - NEW PROTOCOL REVIEW

G. Does this project make use of any of the following special types of subjects and/or locations?

Please mark an “X” on the appropriate line:

___Research with Children ___Research with Prisoners ___Research with Pregnant Women and Fetuses in Utero

___Research in Public Elementary and Secondary Schools ___International Research

___Internet Research ___Drug Research in Human Population (FDA Regulated)

In the CITI Training Program <http://www.citiprogram.org>, there is a training module associated with each of the above special cases. Before your IRB application can be approved, you will need to complete any of the above modules as part of your optional modules (if not listed in the required modules) that are associated with your research, you must pass each quiz associated with the module/s.

H. INVESTIGATOR’S ASSURANCE STATEMENT

127
I have read Miami University's policy concerning research involving human subjects and I agree to:

1. Accept responsibility for the ethical conduct of this research study.
2. Obtain approval from the Institutional Review Board or Departmental Review Board prior to changing any procedures.
3. Report to the IRB any complications, adverse reactions or unexpected effects on subjects,
4. Submit an Application for Approval of Continuing Projects within one year, or sooner as specified in the approval letter, describing the current status of the project.

Principal Investigator(s) Zhisong Pei

Date Jan 29, 2009

Faculty Review of Student Projects: I have reviewed and approved the procedures to be used in the project described in this application. I agree to meet with the investigator on a regular basis to monitor study progress and assure that the well being of subjects is adequately safeguarded.

Faculty Advisor Valerie V. Cross

Date Jan 29, 2009

Page 2 of 2 of the IRB Application Cover Page
II. INSTRUCTIONS FOR COMPLETING THE APPLICATION FOR NEW PROTOCOL REVIEW

Prepare and submit a signed original application and the following items to the Office for the Advancement of Research and Scholarship (OARS), 102 Roudebush Hall. (send one copy of your total application in either a word document or a PDF file to: humansubjects@muohio.edu) Number all pages of the application.

___Application Cover Page

___Research Description

___Subject recruitment letters, announcements, and advertisements

___Consent and Assent Form(s), as required for the study

___Copies of questionnaires, survey instruments, interview questions, other supporting materials

RESEARCH DESCRIPTION. Provide the following information in non-technical language. Number responses to correspond with each item. Refer to section III, NOTES, for further discussion of some of these items.

1. Purpose. Describe the nature, purpose and significance of the study, including the specific objectives or aims of the research, and what outcomes are expected, both general and specific.

   The main purpose of this study is to evaluate the user interface for and the results produced by a software tool OntoSELF-TQ that allows novice users to develop queries to better understand the contents and structure of an ontology, which is a knowledge representation scheme for semantic web information.

2. Subject Population. Describe the anticipated number of subjects, age ranges, and where appropriate, gender, ethnic background, and health status. Be sure to identify the use of vulnerable populations such as children (under age 18), prisoners, mentally disabled persons, or economically or educationally disadvantaged persons.

   Ten to fifteen upper level undergraduate or graduate students from the School of Engineering and Applied Science. These students are all older than 18 years of age. The issues of gender, ethnicity and health status are not relevant at all to this study and there are no vulnerable populations.
3. Recruitment and Selection of Subjects. Describe how participants will be recruited and selected. Refer to NOTES in section III.A. Attach a copy of any recruitment materials such as advertisements and letters.

A flyer describing the research project is to be distributed mid-semester in computer science courses including the data structures course and any course having data structures as a prerequisite since these students will have background knowledge needed to be able to perform the tasks using the software tool in order to evaluated the usefulness of the software.

4. Potential Risks and Discomforts. Describe the potential risks and discomforts associated with each research procedure including any physical, psychological, social, legal or other risks, and assess their likelihood and seriousness. Refer to section III.B. RISKS AND BENEFITS. Describe procedures for protecting against or minimizing risks. Describe provisions for insuring necessary medical or professional intervention in the event of adverse effects on the subjects.

There will be no medical or professional risks with this research study.

5. Potential Benefits. Describe the potential benefits to subjects as a result of their participation in this research. Describe any potential benefits to society that may be expected from this research.

The students will be able to see a practical application of their data structure material in the visualization of knowledge describing a real world problem domain. The feedback from the students can help improve the quality of the software tool and improvements made from this evaluation will help potential novice users to more easily use and understand the capabilities of the software.

6. Informed Consent. Describe the process of obtaining and documenting consent from participants, including assent from children under age 18. Refer to section III.C. INFORMED CONSENT.

The instructors in the selected courses are to distribute the flyer advertising the need for student volunteers for this research study. None of the instructors of the selected courses are associated with study. The informed consent form is to be given to each student before participating in the study and he or she will have adequate time to read it over and ask questions before the study is carried out. The students decide whether to participate on their own and are to be explicitly told that their participation is strictly voluntary.
7. **Exempt Status Request.** Certain types of low risk research may be exempt from some of the requirements of the Federal guidelines including signed consent. If you are requesting exempt status certification, list the exempt category from section III.D. EXEMPT STATUS REQUEST. Not applicable.

8. **Research Procedures/Methods.** Provide a step-by-step description of each procedure including:
   
   a. Nature of activities in which the subjects will be engaged;
   
   b. Data gathering instruments, including copies of questionnaires or interview questions;
   
   c. Frequency and length of time involved in each activity and the overall length of participation;
   
   d. Training of persons administering the treatment or collecting the data;
   
   e. Compensation to subjects for their participation. Payment should be reasonable and prorated with partial payment to those who withdraw before the completion of the research;

   The students will be taught how to use the software application on visualizing several example ontologies. They may be asked to answer some questions about the example ontologies based on using the software tool. There are two aspects to this evaluation: one objective based on whether the participants provided correct answers to questions asking for information about the example ontologies and the other subjective based on the students responses to questions about the ease of use and areas that might be improved on the software. The students will be asked about their levels of satisfaction with various features or capabilities of the software.

9. **Research Location.** List all sites where the research will be conducted. When appropriate, provide letters granting permission to recruit participants and conduct research at these locations.

   The research will be held in the computer labs in Benton Hall.

10. **Procedures for Safeguarding Confidentiality of Information.** Indicate whether the project is to be considered as Confidential or Anonymous; Refer to section III.E. CONFIDENTIALITY OF INFORMATION. Describe how participants’ privacy will be protected and confidentiality of data maintained, how long confidential documents and information will be retained after the end of the study, how data will be stored, and who will have access to identifiable data.

   Project is Anonymous since no identifying information will be collected from the student.
11. **Deception.** If the research involves deception, provide rationale and describe debriefing procedures. Refer to section III.F.

*No deception is being used.*

**OntoSELF-TQ: A Topology Query System for OntoSELF**

**Investigators:**

Zhisong Pei. Principal Investigator. Miami University Computer Science & Systems Analysis Department. peiz-at-muohio.edu

**Purpose and Benefits:**

The main purpose of this study is to evaluate the user interface for and the results produced by a software tool OntoSELF-TQ that allows novice users to develop queries to better understand the contents and structure of an ontology, which is a knowledge representation scheme for semantic web information.

**Experiment Procedure:**

In the experiment, you will be assigned a group of visualization problems for an ontology. You will be asked to find the answer for those questions based on the visualization tool. The estimated time for the whole experiment for each participant is 1-2 hours.

Here are the detailed experiment procedures:

a) Experiment subjects will be provided the informed consent form and an initial survey to collect their background information to divide them into experimental groups.
b) Experiment subjects should have a similar background composition.
c) Subjects will answer the visualization question based on the provided software.
d) After they finish the visualization problem, the subjects will be given a post-survey which asks questions about their experience of using the visualization software.
e) After the experiment, the identifying information of experiment subjects and the experiment results will be separated. Experiment results will be evaluated anonymously.
f) The PI will keep the identifying information of the subjects in a secure location and destroy the information at the conclusion of the experiment.

**Other Information:**

Participation of the experiment is completely voluntary and you can withdraw from the experiment at any time. **There are absolutely no harms or penalties for choosing not to participate in this experiment.** If you have any questions about this experiment, please contact Zhisong Pei, peiz@muohio.edu, 513-273-4949. If you have questions about your rights as a participant in a human subject research, you can contact the IRB (Institutional Review Board) at Miami University, 513-529-3600, humansubjects@muohio.edu.
I have reviewed the above information and agree to participate in the experiment including taking the surveys for the experiment. I verify that I am at least 18 years old and am aware that this survey is voluntary. I understand that the results of the experiment will be published and used for future research and study at Miami University and give my permission of for this to happen under the condition that this will be done anonymously and the personal information will be only used by the investigators to analyze the data, not to evaluate the participants.

Printed Name: __________________________

Signature: ___________________________________________ Date:__________________
Contact Information

This contact information is used for experiment scheduling and data analysis. All this contact information from you will be destroyed one year after data collection.

Name: ___________________________

E-mail address: ___________________

Comfortable dates/times when you are likely to be available to do the experiment:

_________________________________

_________________________________

_________________________________

_________________________________
Pre-Survey – Programming and Design Background Evaluation

This purpose of this survey is to collect your background information of programming and system design. The information is used to form experimental group.

Please circle the College Degrees received and fill in the area:

<table>
<thead>
<tr>
<th>Degree</th>
<th>Area: ______________________</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS</td>
<td></td>
</tr>
<tr>
<td>MS</td>
<td></td>
</tr>
<tr>
<td>PhD</td>
<td>Area: ______________________</td>
</tr>
</tbody>
</table>

Degrees in progress: ______________________

Level of experience with querying languages such as SQL

<table>
<thead>
<tr>
<th>None</th>
<th>Some</th>
<th>Moderate</th>
<th>Expert</th>
</tr>
</thead>
</table>

Level of experience with Object-Oriented programming

<table>
<thead>
<tr>
<th>None</th>
<th>Some</th>
<th>Moderate</th>
<th>Expert</th>
</tr>
</thead>
</table>

Level of experience with graph data representation

<table>
<thead>
<tr>
<th>None</th>
<th>Some</th>
<th>Moderate</th>
<th>Expert</th>
</tr>
</thead>
</table>

Level of experience with ontology

<table>
<thead>
<tr>
<th>None</th>
<th>Some</th>
<th>Moderate</th>
<th>Expert</th>
</tr>
</thead>
</table>

Level of experience with visualization tools

<table>
<thead>
<tr>
<th>None</th>
<th>Some</th>
<th>Moderate</th>
<th>Expert</th>
</tr>
</thead>
</table>

List visualization tools you are familiar with:

Have you ever visualized a graph: Yes   No

Please describe your last experience using a data visualization tool and for what purpose.
**Post-Survey Experiment Process Experience**

Please circle one answer for each question:

**Usefulness and Effectiveness of OntoSELF-TQ**

1. **OntoSELF-TQ** made it faster for me to visualize the ontology
   - Strongly Agree
   - Agree
   - Undecided
   - Disagree
   - Strongly Disagree
   Why:

2. **OntoSELF-TQ** made it easier for me to visualize the ontology
   - Strongly Agree
   - Agree
   - Undecided
   - Disagree
   - Strongly Disagree
   Why:

3. **OntoSELF-TQ** is easy to use
   - Strongly Agree
   - Agree
   - Undecided
   - Disagree
   - Strongly Disagree
   Why:

4. If I had **OntoSELF-TQ** available, I would use it for visualizing the ontology
   - Strongly Agree
   - Agree
   - Undecided
   - Disagree
   - Strongly Disagree
   Why:

5. I would like to suggest other people to use **OntoSELF-TQ**
   - Strongly Agree
   - Agree
   - Undecided
   - Disagree
   - Strongly Disagree

What suggestions you have to make **OntoSELF-TQ** more useful or efficient?
Evaluation of Experiment

1. The explanation for the experiment is clear for me to understand the process
   Strongly Agree   Agree   Undecided   Disagree   Strongly Disagree
   Why:

2. The tutorial for OntoSELF-TQ is sufficient for me to understand how it works
   Strongly Agree   Agree   Undecided   Disagree   Strongly Disagree
   Why:

3. The visualization problem description is sufficient for me to understand the problem
   Strongly Agree   Agree   Undecided   Disagree   Strongly Disagree
   Why:

4. I had enough time to finish the experiment
   Strongly Agree   Agree   Undecided   Disagree   Strongly Disagree
   Why:

5. Comments for the experiment to make it better:
Design Problem

Ontologies are being used in various domains such as e-commerce, biomedical, and engineering. Increasingly, there is a need for 3D ontology visualization tools to help developers for the Semantic Web to understand and analyze the complex structures of existing ontologies. To allow novice users to easily perform topology understanding tasks and to query on the structure of an ontology, this thesis is developing a topology query (TQ) system for ontologies called OntoSELF-TQ. The OntoSELF-TQ is to permit the consumers to specify characteristics and constraints on components of the ontology in order to produce a visualized result which helps the user better understand structure or topology of the ontology. The result is to utilize visual cues such as size and color to help spotlight the critical elements that satisfy the query. Even without any knowledge of underlying techniques, a user should be able to easily probe ontological data through OntoSELF-TQ.

The participants in this study are to be given a tutorial on the software and then allowed to ask questions they might have before beginning the experiment. They will be asked to find specific nodes or to describe structure elements of the ontology based on these meeting certain explicit criteria. The participants will be given a set of questions that they must answer using the OntoSELF-TQ software. They will then be asked to answer a set of survey questions about their experience using the software.