USC-CIT Technical Report 98-01

Representing and Using Ontologies[1]

Kuhanandha Mahalingam and Michael N. Huhns

January 1998

Center for Information Technology
Department of Electrical and Computer Engineering
University of South Carolina
Columbia, SC, U.S.A. 29208
(803) 777-5921 or kuha@sc.edu
(803) 777-8045 FAX

Keywords: Ontology, Java, Distributed Information Systems

Abstract

This paper discusses the representation of large ontologies, especially in a graphical format. It describes a tool, developed using Java, that incorporates several abstraction mechanisms that can help manage such ontologies. Furthermore, it briefly analyzes possible point-and-click approaches that can be used to form queries on these ontologies.

[1] This research was partially done under a cooperative agreement between the National Institute of Standards and Technology Advanced Technology Program (under the HIIT contract, number 70NANB5H1011) and the Healthcare Open Systems and Trials, Inc. consortium.

1. Introduction

The introduction of the World Wide Web (Web) and the consequent growth of other supporting technologies that, without doubt, have driven the Internet's widespread usage have created many new problems. The most notable problem is how to manage efficiently the volume and heterogeneity of information produced on the Web each day. Further, most of the data produced is no longer just simple text, but also consists of multimedia objects. Multimedia objects require different policies, procedures, and conventions. They originate from and reside on a variety of different operating systems, and, most importantly, are large and complex.

There are many approaches for coping with these problems. One of these approaches is to use ontologies for capturing the knowledge represented by the information sources, so that it can be viewed and manipulated declaratively. However, the construction and management of ontologies is itself problematic, particularly when the ontologies are large and complex, as is common nowadays.

The Java Ontology Editor (JOE)[3], a tool developed at the University of South Carolina, is part of an ongoing project that attempts to find solutions to the above problems by using various abstraction mechanisms. In this paper we are mostly interested in the management and representation of ontologies that are large in terms of the number of concepts and relations that are represented. In the following sections we describe various techniques currently being used for this purpose.

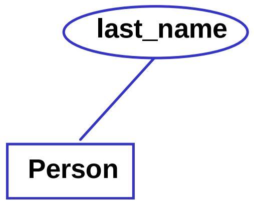
2. Ontologies

An ontology is, fundamentally, a conceptualization or an abstract model of some portion of the world and is described by defining a set of representational terms. Definitions associate the names of entities in a universe of discourse (e.g., classes, relations, functions, or other objects) with formal axioms that constrain the interpretation and well-formed use of these terms [1,2]. Ontologies can be used to organize keywords, database concepts, and even unstructured data produced by multimedia applications, by capturing the semantic relationships among the numerous concepts that define the information space of interest. By using these concepts and relationships, a network structure can be created providing users with a model of the information space for their domain of interest. Due to this attractive feature, ontologies are now heavily used for knowledge sharing in a distributed environment, and several ontology-based information systems have been implemented [4-7].

2.1 Representing Ontologies in Textual Formats

Ultimately all ontologies are stored in a pure textual format of some form or another. A proposed standard for storing ontologies in textual format is the Knowledge Interchange Format (KIF)[8], which is also the most widely used format. In addition to KIF there are several others: Lisp, Clips, Loom, Ontolingua, and LDL formats are the most commonly used ones for representing ontologies.

The following code segments show how three different formats represent the concept ***Person*** and a property called ***last_name***:



KIF
```
(defrelation Person
      (subclass-of Person Agent)
      (class Person))
(deffunction person.last_name
      (function   person.last_name)
      (domain person.last_name Person)
                       (range person.last_name
                 Name)
      (arity person.last_name 2))
```

LOOM
```
(loom:defconcept Person
:context Agent
:is-primitive (:and Agent)
:annotations
((documentation
"A person represents a human")))

(loom:
:context Agent
:is-primitive loom:binary-tuple
:domain Person
:range Name :attributes
      (:single-valued)
:arity 2
:annotations
((documentation
"The last name of Person.")))
```

LDL++
```
frame('Agent_Person'),
name('Agent_Person', 'Person'),
member('Agent_Person', 'Agent'),

slot('Agent_Person_last_name'),
name('Agent_Person_last_name',
     'last_name'),
defrelation person.last_name
domain('Agent_Person_last_name',
       'Agent_Person'),
range('Agent_Person_last_name',
       string),
```

As opposed to graphical representations, textual formats have one very useful advantage. Ontologies represented by means of any of the above mentioned textual formats are easier to port from one operating system to another, since they do not store any platform dependent information in their representations. In a distributed and heterogeneous environment, this feature is very useful for knowledge sharing. Unfortunately, textual formats also exhibit many disadvantages. One is that textual representations such as the above are not so easy to comprehend due to the nature of how the information is presented to users. In other words, unless users are well versed in the format used, they will not be able to read it and understand the representation. If the information is not easily understood, then possible use of it is very limited. We have observed that some of the problems of textual representations can be eliminated by proper use of graphical representations in combination with textual constructs.

2.2 Representing Ontologies in Graphical Formats

It is commonly believed that one of the attractive features of graphical representations is that they are easier to understand than their textual counterparts. Unfortunately, unlike textual formats, graphical formats are harder to port from one operating system to another because system-dependent information often must be stored in addition to the actual knowledge base. However, with the introduction of system-independent development languages such as Java, there is no need for such additional information.

Because of the close relationship between the information spaces and databases, in addition to its simple format, currently most preferred graphical representation method is the popular Entity-Relationship (ER) diagrams. An ER diagram is a graph in which nodes represent entities (or concepts) and arcs represent relationships between concepts. Graphically, entities are shown as rectangles, attributes as ellipses, and relationships as diamonds as shown in Figure 1.

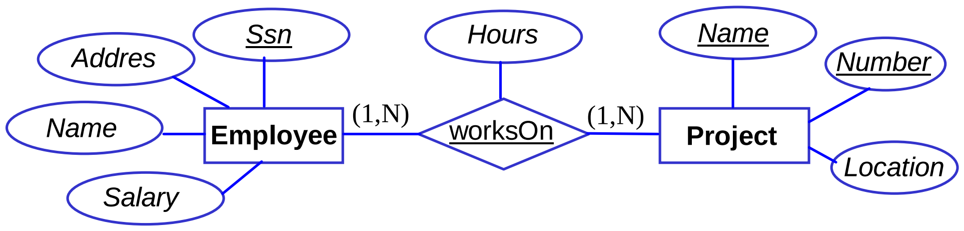


Figure 1. A simple E/R diagram

Looking at Figure 1, it is quite obvious why E/R diagrams are widely used for graphically representing meta-schemas of knowledge bases. E/R diagrams are preferred over others because in their simplest form there can be a one-to-one mapping between an ER diagram and an actual database. For example, we can easily transform the above schema to a database that contains three tables (i.e., Employee, Project, and worksOn), where the first table has four columns (i.e., Name, Address, Ssn, and Salary), the second one has three columns (i.e., Name, Number, and Location), and the third one has four columns (i.e., Ssn, Name, Number, and Hours).

Currently there are several variations that have been built on top of the basic E/R model, but customized for specific application domains. It is also a common practice not to show attributes as ellipses, but instead display them as a list inside the parent entity rectangle.

3. Displaying Large Knowledge Bases

The Internet's widespread usage has generated many new unexpected problems in recent years. Unfortunately, the rate of its growth has made finding solutions to these problems a difficult task. Of all the problems, the one most relevant to ontologies is the task of representing and managing very large information sources, which have become very common. The approach widely used to solve this problem is to develop tools that support various abstraction mechanisms. In the following few sections we will take a look at a few of these abstraction mechanisms in use today.

3.1 Hierarchical Ontologies

Ontologies with a large number of hierarchical relationships can be displayed using various abstract levels.

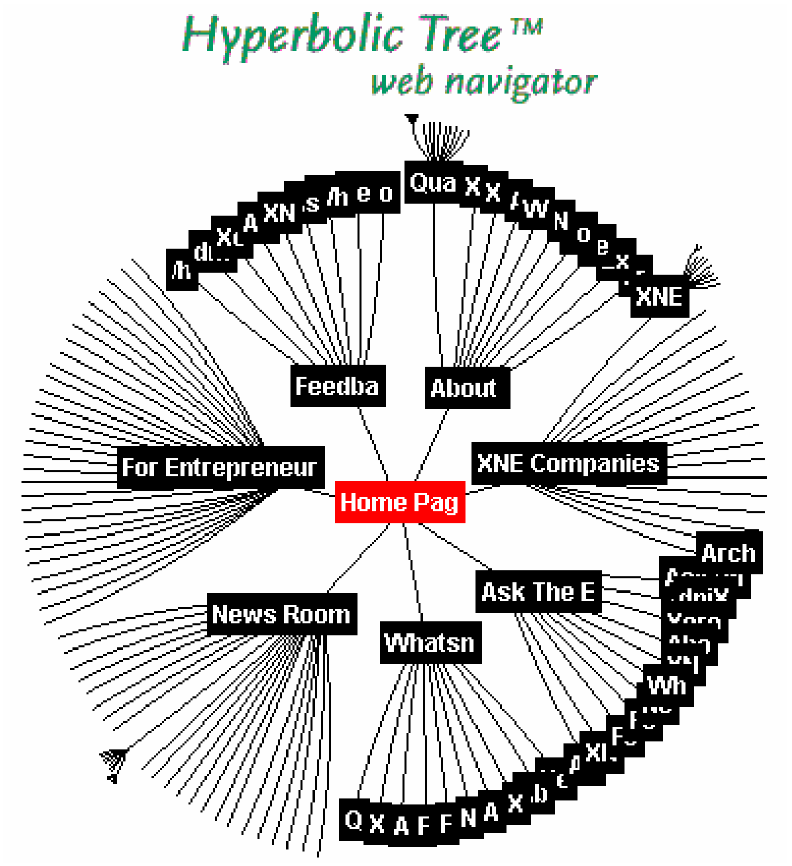


Figure 2. Hyperbolic Tree Display of Inxight Corporation[13]

As a hyperbolic tree: As shown in Figure 2, information can be organized using a hyperbolic data structure. Additional information about any node (especially hidden links) can be made visible by bringing it to the center (i.e., focusing on that node) of the display. In this example, the tool is used to display all WWW documents (or URLs) available at a specific server.

When the user has located a specific URL, they can directly access it by double clicking on the node. This same tool is also ideal for representing any information space that exhibits a large number of hierarchical relationships. However, this tool is not suitable when there are a large number of relationships other than hierarchical. With such relationships it becomes harder to maintain a simple and clear display as shown above due mainly to the increased number of links that are necessary to show each relationship.

An alternative to a hyperbolic tree is the topic graph developed by Alta Vista Corporation [14] (see Figure 3). It is quite similar to the hyperbolic tree, and used as a tool for searching for information in the Internet. Under each of the displayed topics, a list of more specific topics is available (not shown) and, if necessary, the initial query can be further modified to reduce the number of possibilities by selecting a topic from this list.

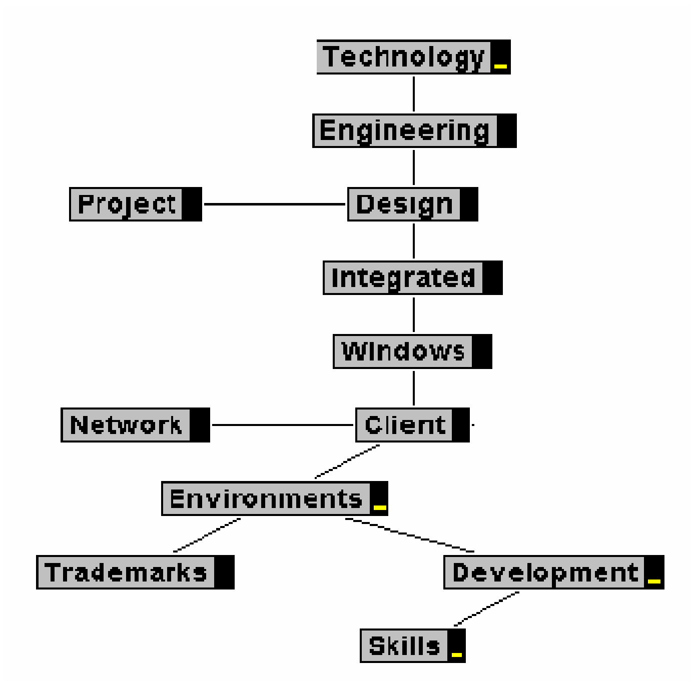


Figure 3. Alta-Vista LiveTopic Graph

As a collapsible and expandable tree: Another alternative is to display the information in a collapsible/expandable tree as shown in Figures 4 and 5. In this format, the user has the freedom to expand only the node of interest and leave the rest in a collapsed state. If necessary, the entire tree can be expanded to get the complete knowledge base. Once again, this type of format is useful only when there are a large number of hierarchical relationships.

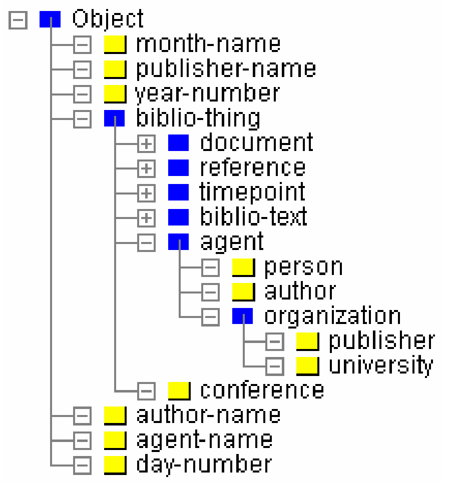


Figure 4. Tree as in MS Windows Explorer Format

3.2 Nonhierarchical Ontologies

When the number of relationships other than hierarchies is larger, the above mentioned tools begin to lose their simple and easy to understand formats. The display becomes cluttered with links between the nodes and eventually the purpose of such displays becomes lost. For such large information spaces, the usual choice is to use graphs and represent these ontologies using the E/R model. Furthermore, since the ontologies are very large, the initial display is usually very abstract giving the user only a model of the layout (see top portion of Figure 6). Here, the colored squares represent entities and relationships (in this example, red ones for hierarchical and yellow one for other types). These tools provide options to progressively zoom into each of the displayed node. For example, from the abstract layout shown at the top of Figure 6, the user can get more detail about a selected node by progressively zooming in; initially, the node names are shown, then attributes of an entity

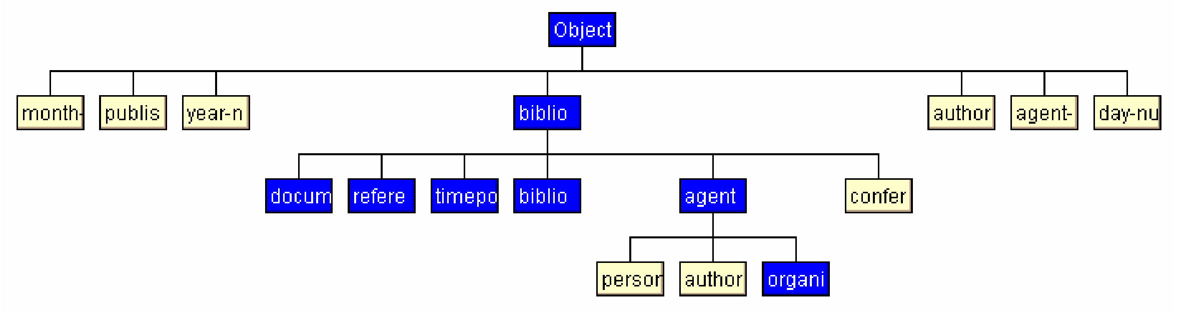


Figure 5. An expandable/collapsible Tree

(if any) are shown (see bottom portion of Figure 6), and further on even more details about each node are shown in Figures 7 and 8.

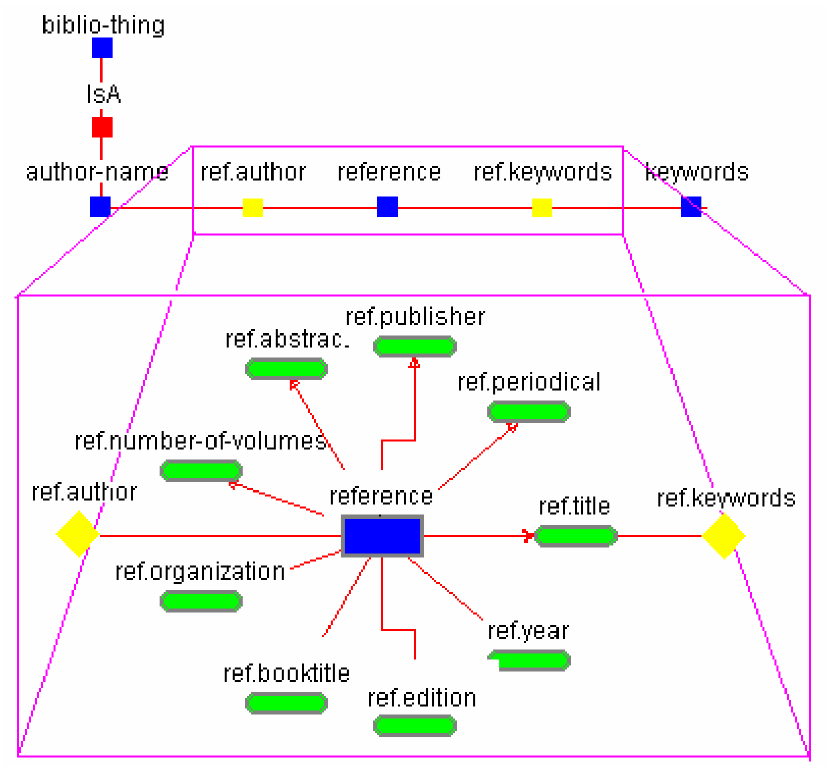
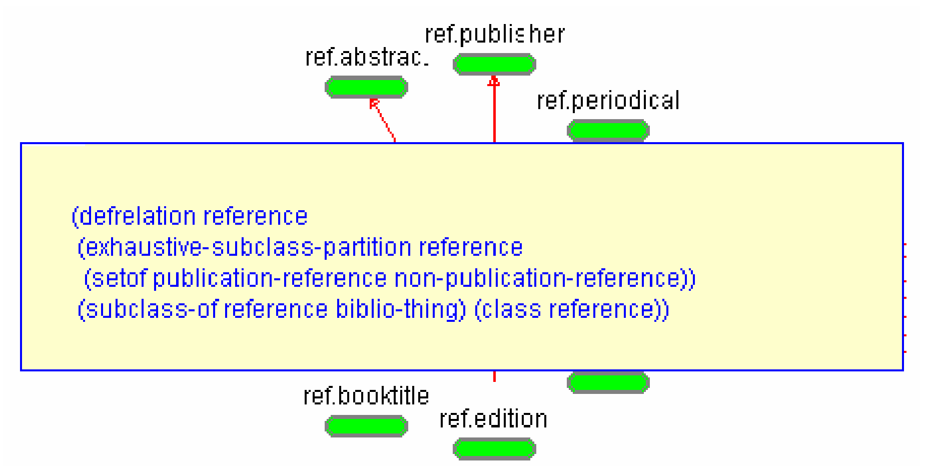
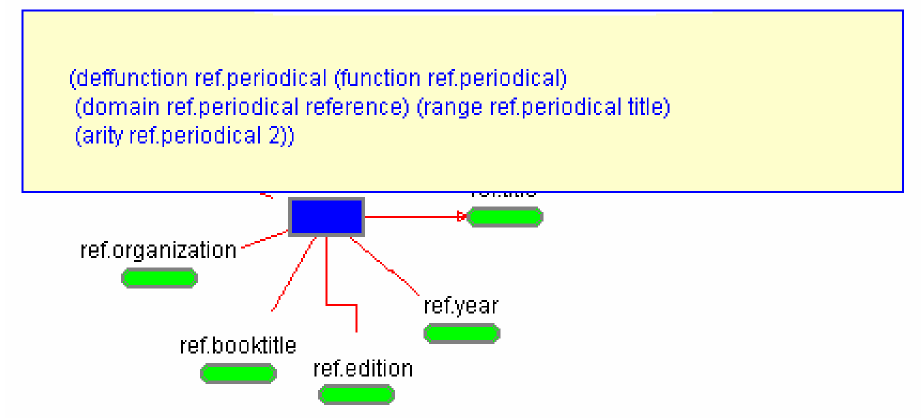


Figure 6. An abstract view of an ontology (the layout) and a zoomed-in view of an entity (reference) showing its attributes



```
(defrelation reference
 (exhaustive-subclass-partition reference
  (setof publication-reference non-publication-reference))
 (subclass-of reference biblio-thing) (class reference))
```

Figure 7. Showing even more detail (the KIF sentence) for an ontology



```
(deffunction ref.periodical (function ref.periodical)
 (domain ref.periodical reference) (range ref.periodical title)
 (arity ref.periodical 2))
```

Figure 8. Detail (the KIF sentence that defined the attribute)

4. Navigating Large Knowledge Bases

Another common problem with large ontologies is locating a desired node or, in other words, navigating through the large number of nodes that might be connected by hundreds of arcs. Usually, users would scroll the display a few times hoping to find the node they are looking for. Although by using this approach the user would eventually find the node, however, it is neither convenient nor efficient. Instead, by using a few abstraction techniques one can speed up this process somewhat.

One such technique is to display the entire ontology, regardless of its size, inside a given window. Unfortunately, if the ontology is quite large then the displayed image usually is unintelligible due mainly to scaling. A solution to this problem is to provide a magnifier – window that displays the area under the mouse pointer at the normal (100%) scale as shown in Figure 9. Using this approach, the user, can easily navigate through the display and locate any node of interest. In addition to this feature, it is also desirable to

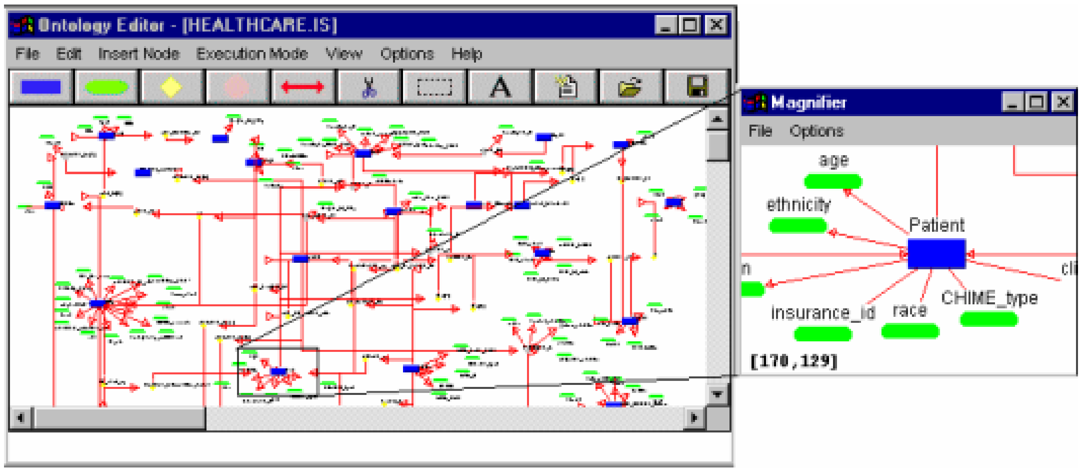

Ontology Editor - [HEALTHCARE.IS]
File Edit Insert Node Execution Mode View Options Help
Magnifier
File Options
age
ethnicity
Patient
insurance_id
race
CHIME_type
[170,129]

Figure 9. Displaying entire ontology within the current window. Magnified view of the selected area is shown on the right.

have an option to search for a given node by its name if the name of the node is already known (see Figure 10).

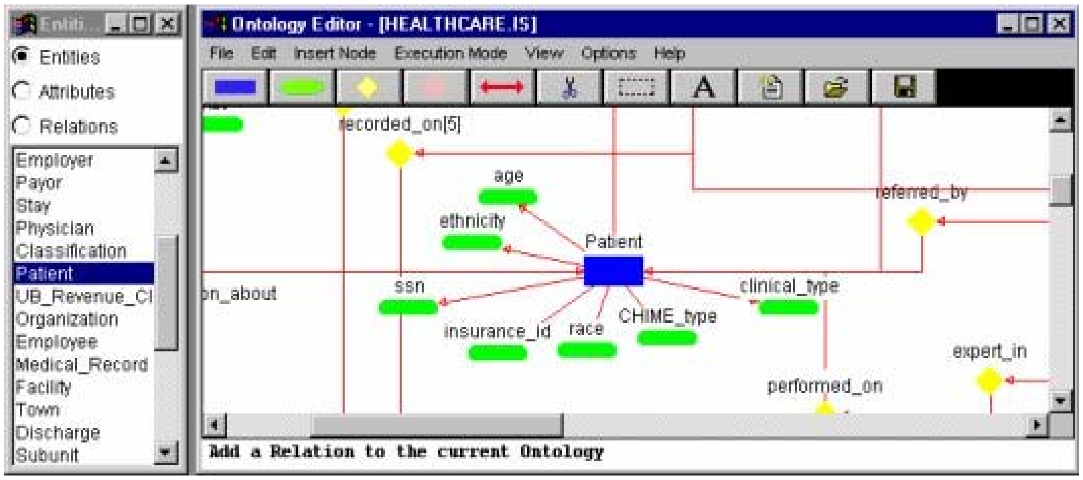


Figure 10. Searching for nodes

5. Building Large Knowledge Bases

In general, developing ontologies is a complicated process and it is currently one of highly debated research issues in the industry. One of many and most widely debated reasons for it is the inevitable problem of incorrectly interpreting a given ontology at a later time. When ontologies are interpreted differently than as designed, they are useless from the knowledge-sharing point of view. Because of this possibility, ontology development must be carefully controlled by means of rules and axioms that can reduce possible misinterpretations by users other than the creator.

This problem is made even more complicated by the loss of information that results in the process of abstraction from the real-world situation to an abstract and inconsistent model. The problem is the differences in our conceptual model of the real world and how each of us interpret them in our day to day activities.

Currently, there are a few tools that are specifically designed for building ontologies that have built in validation mechanisms that guarantee the integrity of the resulting ontologies. Of these, most notable one is the Ontolingua[9] server at Sanford University. The ontology displayed in Figures 4 and 5 is one such ontology built using the Ontolingua server. In addition, there are several other graphical tools that support limited features with regard to ontology development. Our tool, JOE [3], Information Management Tools Suite (IMTS) [10] developed at MCC, and Generic Knowledge Base (GKB) [11] editor are a few such graphical applications that support ontology development on a limited scale. Furthermore, there are several other tools with limited graphical user interface, such as OOHVR [12] from NJIT and others specifically designed for the WWW use, also available.

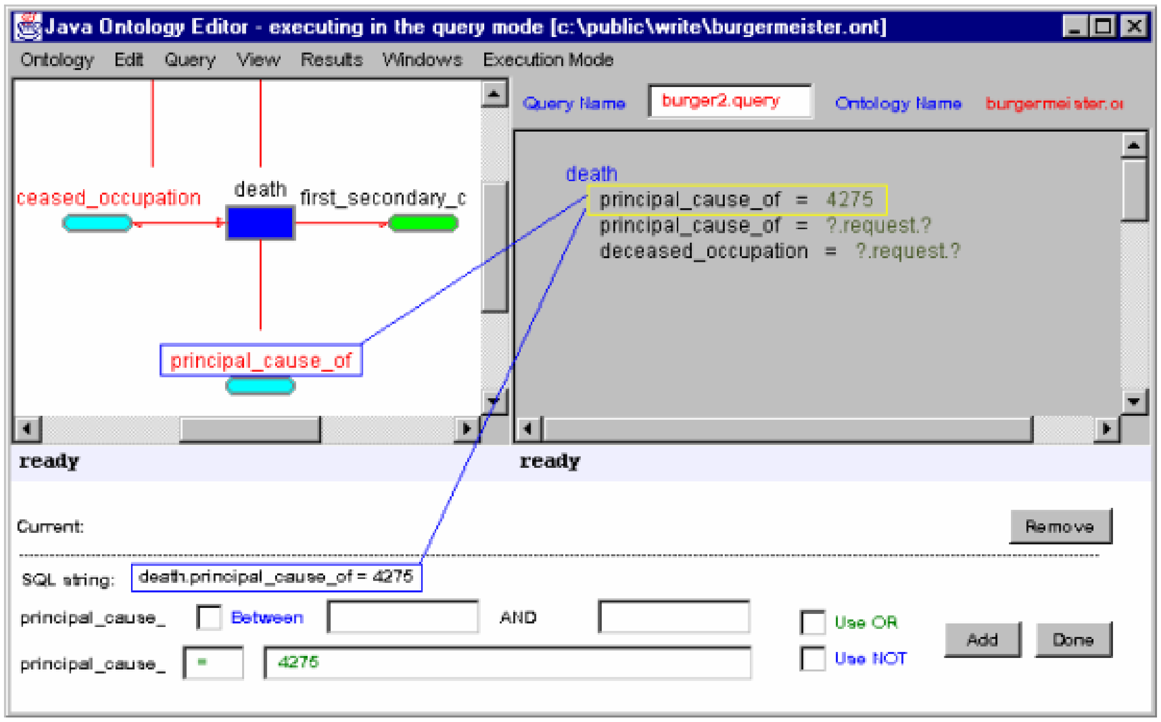
6. Use of Ontologies as a Query Tool

Having access to useful information alone does not guarantee that a user will get accurate and precise information. There should be efficient applications that can help a user get exactly what he or she is looking for in a reasonable amount of time. Towards that end, there are a number of search applications (often referred to as search engines) that were designed specifically for that purpose.

As mentioned before, ontologies are designed to represent meta-schemas of actual knowledge bases that contain real-world instances or facts. Often, in order to access the data, one needs to go directly to the actual database, generate queries, most likely, using some standard query language such as SQL, and then submit it to get the required data. This approach is not convenient due to two reasons:

- First the user must be able to access the database - without proper access privileges, the database is useless.
- Secondly, the user must have knowledge about a query language supported by that specific database.

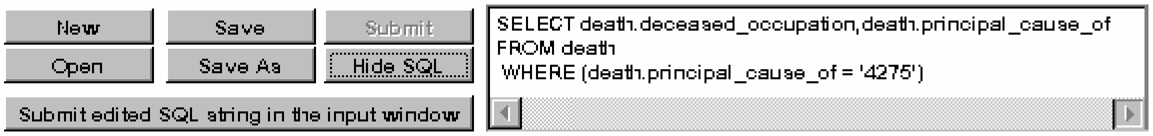
However, by using ontologies, we can find a solution to this problem. One approach is to provide features that support generating queries on the ontology itself. This can be made even simpler by allowing the user to just point-and-click to generate the query at an abstract level in which they are comfortable and let the application do the translation from the abstract query to specific one understood by the actual database (see Figures 11 and 12). There are many advantages in using this approach. Most importantly, the user is not required to have knowledge about query languages. In addition, the user need not worry about access privileges because the application acts as a bridge between the user and the actual database. Furthermore, this approach can easily be expanded to include, in a similar manner, many other databases without affecting the actual users.



death
principal_cause_of = 4275
principal_cause_of = ?.request.?
deceased_occupation = ?.request.?

SQL string: death.principal_cause_of = 4275
principal_cause_
principal_cause_ = 4275

Figure 11. Building Queries



```
SELECT death.deceased_occupation,death.principal_cause_of
FROM death
 WHERE (death.principal_cause_of = '4275')
```

Figure 12. SQL translation of the query displayed in Figure 11

7. Conclusions

In this paper, we looked at some issues related to ontologies. In particular, the problems of managing large ontologies and possible solutions to those were briefly analyzed. In addition, we also briefly explored possible uses for ontologies, especially, use of ontologies for generating queries at an abstract level. The tool described in the paper, the Java Ontology Editor (JOE), is an ongoing process that attempts to incorporate many of the ideas mentioned in this paper. However, it features many of the abstraction techniques mentioned earlier that are suitable for handling large ontologies. In the future, features that safeguard the validity and integrity of the ontologies will be added so that it can become a very useful tool for ontology development in a distributed and heterogeneous environment such as the Internet.

The techniques described above are only a few out of many that are in use. Unfortunately, one technique alone cannot be used for all ontologies. Often, different approaches must be taken in combination to handle a given ontology depending on the type of information it contains. Nevertheless, applications that were developed for the purpose of ontology development should provide at least some of the abstraction methods mentioned above so that they can reduce the burden on the user.

References

[1] T. Gruber. “A translation approach to portable ontologies,” *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220, 1993.

[2] P. D. Karp, K. Myers, and T. Gruber, "The generic frame protocol," in *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, pp. 768--774, 1995.

[3] K. Mahalingam, “The Java Ontology Editor (JOE),” Center for Information Technology, Electrical and Computer Engineering Department, University of South Carolina, 1996. (http://www.ece.sc.edu/Labs/HIIT/html/joe/)

[4] D. Lenat and R. V. Guha, *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project,* Addison-Wesley Publishing Company, Inc., Reading, MA, 1990.

[5] M. Genesereth. “Knowledge Interchange Format,” *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, Cambridge, MA, pages 599-600, Morgan Kaufmann, 1991.

[6] D. Woelk, M. Huhns, and C. Tomlinson. “Uncovering the Next Generation of Active Objects,” *Object Magazine*, vol. 5, no. 4, pp. 32--40, July/August 1995.

[7] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaratnam, Y. Sagiv, Ullman, and J. Widom, “The TSIMMIS Approach to Mediation: Data Models and Languages,” *Proceedings of the NGITS (Next Generation Information Technologies and Systems)*, June 1995.

[8] Knowledge Interchange Format (KIF), Knowledge Sharing Effort Project, Stanford University (http://www-ksl.stanford.edu/knowledge-sharing/kif/)

[9] Ontolingua Server, Knowledge Sharing Effort Project, Stanford University. (http://www-ksl.stanford.edu/knowledge-sharing/ontolingua/index.html)

[10] Information Management Tools Suite(IMTS), Microelectronics and Computer Technology Corporation (http://www.mcc.com/projects/carnot/EMMI.html)

[11] Generic Knowledge Base Editor, SRI International's Artificial Intelligence Center. (http://www.ai.sri.com/~gkb/)

[12] Object Oriented Healthcare Vocabulary Repository, OOHVR Query Interface Project, New Jersey Institute of Technology. (http://object.njit.edu:2000/~newoohvr/JBI/)

[13] Inxight Corporation, Palo Alto, California. (http://www.inxight.com/products/hw/infoseek.htm)

[14] Alta Vista Corporation, Palo Alto, California. (http://www.altavista.digital.com/)

Kuhanandha Mahalingam has recently obtained a Ph.D. degree in the Electrical and Computer Engineering Department at the University of South Carolina. His research interests include ontology-based distributed information systems and software agent technology. He received his BS and MSEE degrees from North Carolina State University. He is a member of IEEE and NSPE.

Michael Huhns is a professor of electrical and computer engineering and director of the Center for Information Technology at the University of South Carolina. Prior to this he was a senior researcher at the Microelectronics and Computer Technology Corporation (MCC) and an adjunct professor in computer sciences at the University of Texas, Austin. Dr. Huhns received the B.S.E.E. degree in 1969 from the University of Michigan, Ann Arbor, and the M.S. and Ph.D. degrees in electrical engineering in 1971 and 1975, respectively, from the University of Southern California, Los Angeles.

Dr. Huhns is a member of Sigma Xi, Tau Beta Pi, Eta Kappa Nu, ACM, IEEE, and AAAI. He is the author of over 100 technical papers in machine intelligence and an editor of the books *Distributed Artificial Intelligence, Volumes I and II* and *Readings in Agents*. His research interests are in the areas of cooperative information systems, DAI and multiagent systems, enterprise integration, and heterogeneous distributed databases. Dr. Huhns is an associate editor for IEEE Intelligent Systems, the Journal of Autonomous Agents and Multi-Agent Systems, and IEEE Internet Computing. He is on the editorial boards of the International Journal of Cooperative Information Systems and the Journal of Intelligent Manufacturing. He is a founder and treasurer of the International Foundation for Multiagent Systems.