IMPLEMENTATION OF AN APPLICATION ONTOLOGY

A Comparison of Two Tools

Robert Harrison and Christine W. Chan*
Faculty of Engineering, University of Regina, Regina, Saskatchewan, Canada, S4S 0A2
Email: harrisor@uregina.ca, Christine.Chan@uregina.ca

Abstract: This study presents two implementations of an application ontology for the problem domain of selection of remediation technologies for petroleum contaminated sites. The objective of this work is to study design of ontology modeling systems by comparing two implementations of the same application ontology, one on Protégé and another on a prototype Ontology Modeler developed in-house at Energy Informatics Laboratory of University of Regina, Canada. The two tools both aim to document and represent static knowledge of an application domain. The knowledge acquisition and ontology construction phases are discussed and a comparison of the two implementations is also presented.

Key words: Ontology construction, selection of remediation for petroleum contamination

1. INTRODUCTION

A major cost in building knowledge-based systems is construction of the knowledge base. If several application systems on the same domain are to be constructed, the effort needed to build the knowledge bases for the different systems is often duplicated. This effort is often substantial due to the tacit nature of expertise; and the process of acquiring knowledge for building the knowledge bases is known to be a major bottleneck in the development process. A possible solution for the problem is to share any knowledge on a given problem domain that has been acquired among

* Author to whom all correspondence should be addressed.

systems. Four different approaches have been adopted within the Knowledge Sharing Effort sponsored by Air Force Office of Scientific Research, Defense Advanced Research Projects Agency, the Corporation for National Research Initiatives, and the National Science Foundation [1]. Similar to the objective of the "shared, reusable knowledge based working group" [1], the work reported here aims to overcome the barriers to sharing that arise from lack of consensus across knowledge bases on vocabulary and semantic interpretations in domain models. This approach results in fostering the evolution of sharable ontologies for particular domains, as well as in developing tools and infrastructure needed to facilitate creation and reuse of domain-oriented ontologies.

The objective of this work is to study design of ontology modeling tools by comparing implementations of an ontology developed on Protégé with that on a prototype tool called the Ontology Modeler, constructed at Energy Informatics Laboratory of University of Regina, Canada. Both ontology modeling tools are used for implementation of the same application ontology. A comparison of two implementations of the same application ontology would shed light on the types of features that are useful in an ontology modeling tool, and lay the groundwork for construction of an enhanced version of the Ontology Modeler. The application problem domain used is that of remediation selection for petroleum contaminated sites.

2. BACKGROUND

The Knowledge Acquisition (KA) research community has emphasized for many years knowledge level modeling. This involves knowledge acquisition and analysis of a domain by means of various modeling efforts such as the KADS methodology [2], the generic task approach [3], and components of expertise approach [4]. Within the framework of knowledge-level modeling, two major lines of research have developed. One refines the existing knowledge-level frameworks and emphasizes their formalizations. For example, ML2 has been developed as a formal implemented language based on the KADS methodology [5]. Another line of research aims at developing knowledge level models for a range of tasks and domains in order to uncover generic components, problem solving methods, and ontologies that enable reuse across applications. The objective of the effort is to facilitate knowledge acquisition by providing domain-independent generic models which can guide knowledge engineers in the construction of knowledge models for a particular domain. Specifically, this effort at knowledge modeling can proceed along one of two axis: (1) problem solving

methods, and (2) domain ontology. Briefly, a problem solving method can be seen as an abstract model which provides a means of identifying at each step, candidate actions in a sequence of actions that accomplish some task within a specific domain [6]; while an ontology defines the vocabulary of representational terms with agreed upon definitions in human and machine readable forms [7]. The ontology of a system consists of its vocabulary and a set of constraints on the way terms can be combined to model a domain. All knowledge systems are based on an ontology, which can be implicit or explicit [1].

Ontological analysis as a knowledge modeling technique was first introduced in [8]. Since then, other researchers have emphasized the importance of creating an ontology of a domain [7]. For example, the CommonKADS methodology suggested knowledge categorization in the model of expertise to consist of the two major types of domain theory and control knowledge, the latter includes inference, task, and strategic knowledge (see e.g. Flores-Mendez et al.[5]). In other words, there are four categories of knowledge: domain, inference, task, and strategy. In this paper only the domain knowledge of the sample application domain is represented in the two ontology modeling tools.

3. APPLICATION PROBLEM DOMAIN: SELECTION OF REMEDIATION METHODS FOR PETROLEUM CONTAMINATED SITES

The problem domain involves a selection task. Automation of engineering selection is important for the petroleum industries in which decision for a desired remediation technology at a contaminated site is critical for ensuring safety of the environment and the public. A variety of remediation methods/technologies are available. However, different contaminated sites have different characteristics depending on pollutants' properties, hydrological conditions, and a variety of physical (e.g. mass transfer between different phases), chemical (e.g. oxidation and reduction), and biological processes (e.g. aerobic biodegradation). Thus, the methods selected for different sites vary significantly. The decision for a suitable method at a given site often requires expertise on both remediation technologies and site hydrological conditions.

Most of the remediation technologies are too complex and not easily comprehensible for managers and engineers in industries and government.

Therefore, a decision support system (DSS) for supporting decision-making on site remediation techniques is useful.

4. KNOWLEDGE ACQUISITION FOR DEVELOPMENT OF DECISION SUPPORT SYSTEM

The problem domain in this study involves a vast amount of knowledge and decision tools related to site remediation practices. Factors relevant for treating a site contaminated by petroleum products include information on the site hydrogeology, subsurface contamination, and contaminant transport and conversion. An ontology of the domain can provide the basis for such a system.

Knowledge acquisition involved both interviewing the expert as well as consulting published materials and databases about the hundreds of remediation methods on the market. During the interviews, the expert introduced and explained the concepts and tasks of the problem domain to the knowledge engineer. The process of interviews continued until the knowledge engineer was satisfied that the material was sufficiently clarified. In addition to the human expert, a secondary knowledge source was the commercial database of several hundreds of remediation methods. The database includes descriptions of the remediation methods and the conditions in which they are suitable. Through knowledge acquisition, the domain expert specified the considerations for selection of remediation technologies to include: (1) contaminated site, (2) site hydraulic condition, (3) estimated volume of contaminated soil and groundwater, (4) density of the immisible petroleum contaminant, (5) the immisible contaminants present as free phase or residual phase, and (6) concentration range of chemicals in soil and groundwater. Based on the knowledge acquired and analyzed during knowledge acquisition, an application ontology was configured and implemented on the two tools of Protégé and Ontology Modeler.

5. IMPLEMENTATION OF REMEDIATION SELECTION ONTOLOGY ON PROTEGE

Protégé is an ontology editing tool created by researchers at Stanford University [9]. It is an open-source system programmed in Java. Protégé

has been under development for a number of years with the most recent release being version 3.0.

The Remediation Selection ontology was implemented in Protege through an iterative process of creating a class and then its slots (properties). In Protégé, classes are created and modified through the **Classes** tab. On the left-hand side of the **Classes** tab, there is the **Class Browser**. The **Class Browser** contains a **Class Hierarchy**, which lists the classes contained in the ontology. Initially, the **Class Hierarchy** only contains the class *THING* and its subclass *SYSTEM-CLASS*, which are both required by Protégé. *THING* is the root class of every class in Protégé and *SYSTEM-CLASS* defines base elements such as Class, Slot, Facet, etc. The Remediation Selection ontology's *Media* class is used as an example of how to create classes, slots, and relations.

A class is created by selecting the parent class (in the case of *Media*, the parent is *THING*) in the class hierarchy and then clicking on the **Create Class** button. A window will appear enabling the user to input information about the class. The *Media* class has three sub-classes: *Soil*, *Water*, and *Soil_GroundWater*. Sub-classes are created in the same manner as classes. The class hierarchy for *Media* and its sub classes is shown in Figure 1.

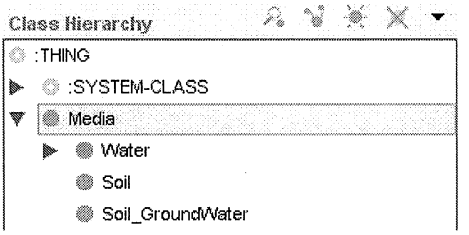

*Figure 1.*Protege Class Hierarchy

The slots for a class are shown in the **Template Slots** window in the **Classes** tab. To create a slot, select a class (eg. *Media*) and then click on the **Create Slot** button. A slot editor window will appear enabling the user to input all the details of the slot. The *Media* class has the slot *site_size*. Details of entering all the *site_size* information is described below. Figure 2 shows the slots window when all the information for the *site_size* slot is entered.

1. In the Name field, type "site_size".

2. *site_size* has three allowed values: small, medium, and large. These values are created by selecting **Symbol** from the **Value Type** field and then using the **Create Value** button.
3. For *site_size*, leave the **Cardinality** fields with the default values.
4. Documentation for the slot goes in the **Documentation** box.
5. The **Default Values** box is for setting one of the allowed values to be the default value for all of the classes and instances that use or inherit the slot. To set the default allowed value to *small*, click on the **Add Value** button above the **Default Values** box, and select *small*.
6. The **Domain** box contains the domains to which this slot belongs. In this case, *site_size* belongs to the *Media* domain.
7. When all information for the slot are entered, click on the **X** in the upper-right corner to close the slot window.

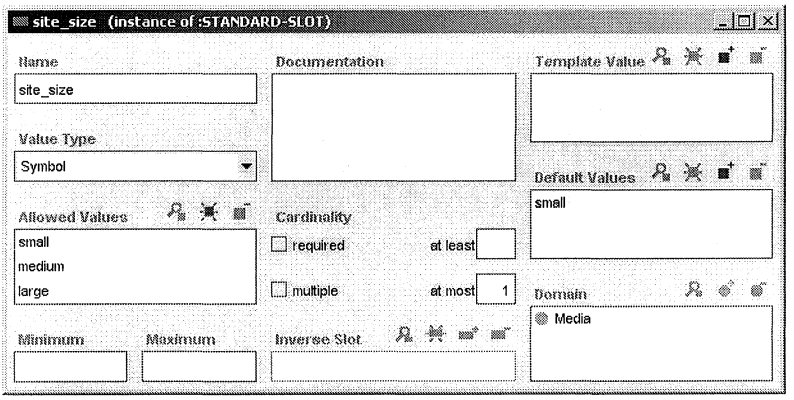

*Figure 2.*Editing site_size slot in Protege

6. DISCUSSION ON THE PROTÉGÉ IMPLEMENTATION

Protégé was not expressive enough and some domain knowledge cannot be entered into the system. For example, *Contaminant.Chemical_Contaminant* has the slot *contamination_weight*, which has different values for each chemical type. When the chemical type is *Benzene_B*, the *contamination_weight* is 0.5. However, the interface does not support input of this sort of knowledge. It is left to the user to ensure that they are using the correct *contamination_weight* with their chosen chemical.

The interface of Protégé can lead the user to incorrectly classify knowledge. For example, the attribute of *contamination_weights* has the values of { 0.5, 0.2, 0.15, 0.15 }, which are floating point values. But, when the user tries to set the value type to float, only the minimum and maximum value constraints can be set, and the value type was set instead to symbol. However, a symbol is not an accurate representation of these values which are really floating point values. The same problem also occurs for multiple integer values.

Protégé can support only parent-child relationships between classes, and other relationships such as the association relationships cannot be defined. For example, the *Media* class has association relationships with the *Remediation*, *Contaminant*, and *Experiment* classes, but these relationships cannot be defined because they are not a parent or child of each other.

7. DESIGN OF THE ONTOLOGY MODELER

The objective of the new ontology modeling system, Ontology Modeler, is to develop a prototype ontology modeling application that provides a visualization of the ontologies and an interface that enables the user to easily manipulate the ontology data. The system was developed with Microsoft Visual C# .Net 2002. The representation of classes and slots in Ontology Modeler is shown in Figure 3.

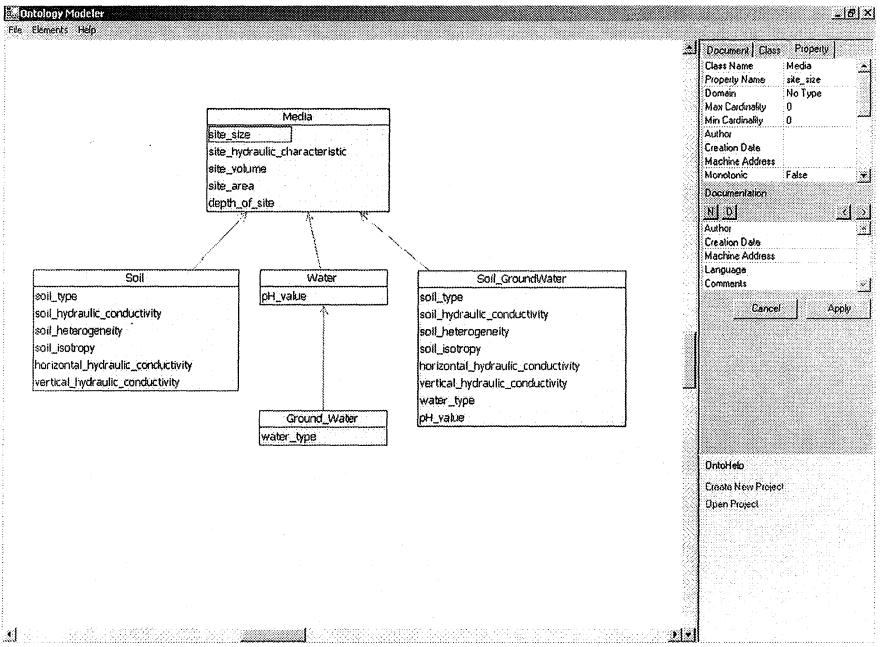

*Figure 3.*Ontology Modeler

7.1 Ontology Representation

The ontologies in Ontology Modeler are represented in the Generic Ontology Representation [10]. The Generic Representation is an object-oriented ontology representation that provides a level of abstraction that makes it easier to manipulate and manage the ontologies. The three main Generic Representation elements discussed in this paper are the document, class axiom, and the data property axiom. A class axiom is similar to a class and a data property axiom is similar to a slot. A document is a structure that contains the ontology's class axioms and data property axioms.

7.2 Ontology Visualization

Ontology visualization can either improve the ontology development process or hinder it, depending on the visualization technique used. One visualization technique is to use a tree structure. Trees, such as the Class Hierarchy in Protégé, are not sufficient for displaying classes with multiple inheritance because they can only show single inheritance. Protégé compensates for this limitation of the tree, by supporting multiple

inheritance in a Superclasses pane. However, with this feature, managing the class hierarchy becomes cumbersome because a class in the Class Hierarchy must be selected first and then the user needs to remember to look at the Superclasses window to see all the parents of a class. It can be concluded that using a tree to visualize an ontology hinders the ontology development process.

A more effective method of visualizing ontologies is to use a graph because it can show multiple inheritance. Due to the similarities between ontologies and object-oriented design, the UML (Trademark of Object Management Group, Inc.) class diagrams are adopted for displaying classes and their properties, and relationships between classes.

The classes of an ontology are displayed in the Viewer, which is the main viewing window. As shown in Figure 3, the top compartment of a class contains the name of the class. The lower compartment of a class contains the names of the properties of the class. Inheritance relationships between classes are shown by lines with arrows pointing to the parent. Clicking on a class will select the class. Classes can be moved by dragging them with the mouse.

7.3 Ontology Editing

The user can modify ontologies with the use of the Viewer and perform the following operations: (1) Add/Delete class axioms, (2) Add/Delete properties, (3) Modify class and property names, and (4) Add/Delete inheritance relationships between classes. However, not all ontology information can be displayed and modified graphically. To support the modification of non-graphical data, the InfoWindow, at the upper-right corner of the screen, can be used. The InfoWindow uses a tabbed user interface, which enables the user to quickly and easily switch between different ontology information types. There are three tabs: Document, Class, and Property. The Document tab contains information defined by the Document class, the Class tab contains information defined by the ClassAxiom class, and the Property tab contains information defined by the DataProperty class. The information contained within the tabs is displayed in a grid for viewing and editing the information. When the user clicks on a class or property in the Viewer, the appropriate tab is displayed and is filled with the information of the clicked class or property. This feature enables the users to focus on their task and not worry about changing tabs or searching through lists of classes or properties.

8. IMPLEMENTATION OF REMEDIATION SELECTION ONTOLOGY ON ONTOLOGY MODELER

Similar to the development process using Protégé, the Remediation Selection ontology was implemented in Ontology Modeler through an iterative process of creating a class and then its properties. To create a class, click Elements->Class and then click in an empty space in the viewer. To change the class name, to *Media*, for example, (1) click Elements->Select, (2) double click on the class, and (3) type to change the name of the class to "Media". Documentation for the *Media* class was created by (1) in the Class tab, click **N** to create new documentation, (2) in the Comments field, type "Media is an object consisting of note 1", and (3) click Apply.

When a class has been created, its properties are created. Properties can be modified through both the main viewer and the Properties tab. The steps for creating the *site_size* property of the *Media* class are listed as follows:

1. With **Elements->Select** enabled, click on the *Media* class.
2. Right-click on *Media* and left-click on **Add Property**.
3. Double-click on the added property.
4. Type to change the name of the property to "site_size".
5. Click outside of the class to save changes.
6. Additional information about the property, such as domain and cardinality, can be input through the **Property** tab.

To show the child-parent relationship between two classes, the user has to draw an arrow pointing from the child class to the parent class. This is performed by selecting **Relation** from the **Elements** menu and then clicking on the child class, *Soil*, and then clicking on the parent class, *Media*.

9. COMPARISON OF TWO IMPLEMENTATIONS: STRENGTHS AND WEAKNESSES

Both Protégé and Ontology Modeler are able to graphically display an ontology. Protégé's tree hierarchy is inadequate for dealing with multiple inheritance. Protégé contains the plugins, OntoViz and Jambalaya, for visualizing ontologies as a graph. However, both plugins have problems. OntoViz gave Java errors while creating a graph and the interface for Jambalaya is not user friendly. To create new nodes and arcs in Jambalaya,

a user needs to use the Class Hierarchy, which suffers from the multiple inheritance problem described above. Ontology Modeler's ontology visualization is an improvement over Protégé's in that it uses a graph and it allows the user to modify and manipulate the classes and relations directly, rather than performing manipulation operations through a separate window. Additionally, the visualization is enhanced with the use of the UML. Neither Protégé nor the current version of Ontology Modeler support association relationships between classes.

Poor labeling is a problem in Protégé. Protégé uses Open Knowledge Base Connectivity (OKBC) for handling the representation of the ontologies [11]. OKBC elements include *:THING*, *:SYSTEM-CLASS*, and *:Documentation*. While these OKBC elements appear in the class and slot hierarchy, they are not a part of the ontology, and should be transparent to the user. The Ontology Modeler uses the Generic Representation, which enables a language neutral user interface to be developed. This is more user-friendly.

Some labels are confusing in the Edit Slot window of Protégé. For a novice user, the difference between the Template Value box and the Default Values box is unclear. The behavior of both boxes is identical as they can both set a default value. The user would only become aware of the differences through trial and error or by reading the online documentation.

Protégé can express more types of data and support representation of more elements of the Remediation Selection ontology. For example, the *site_size* property of the *Media* class has the values of *small*, *medium*, and *large*, which are represented in Protégé. It is not possible to specify these values in Ontology Modeler.

Documenting the knowledge elements is an important feature and both Protégé and Ontology Modeler support documentation of ontology elements. In comparison, Ontology Modeler has better documentation facilities because in addition to comments, it supports the user in recording the author, date, and language of the comments.

10. CONCLUSION AND FUTURE WORK

The objective of this work is to study design of ontology modeling tools by comparing Protégé with a prototype tool called the Ontology Modeler. A comparison of two implementations of the same application ontology

reveals that ontology visualization, support for a wide variety of data types, and the user interface are important features of an ontology modeling tool. This study lays the groundwork for construction of an enhanced version of the Ontology Modeler. Ontology visualization can be further improved by using 3D graphics as more information can be displayed in a 3D environment. Adding support for the object property axiom, a Generic Representation element, will enable users to define more complex properties. Ontology learning techniques can be applied to refine the quality of the generated ontologies. Ontology Modeler will also be enhanced with an ontology management system.

ACKNOWLEDGEMENTS

The generous support of a Strategic Grant from Natural Sciences and Engineering Research Council is gratefully acknowledged. The contributions of current and former students of Energy Informatics Laboratory in this work are also gratefully acknowledged.

REFERENCES

[1] R. Neches, R. Fikes, T. Finn, T. Gruber, R. Patil, T. Senator, and W. R. Swartout, "Enabling Technology for Knowledge Sharing", *AI Magazine*, **12**(3) (Fall 1991) 37-56.

[2] G. Schreiber, J. Breuker, B. Bredeweg, and B. Wielinga, "Modeling in knowledge based systems development", in Boose, J. ,Gaines, B., Linster, M., (eds) *Proceedings of the European Knowledge Acquisition Workshop* (EKAW '88), June 19-23, 1988, Gesellschaft fur Mathematik und Datenverarbeitung, MBH.

[3] B. Chandrasekaran, "Generic tasks in knowledge-based reasoning: high-level building blocks for expert systems design", *IEEE Expert* **1**(3) (1986) 23-30.

[4] L. Steels, "Components of Expertise", *AI Magazine* **11**(2) (1990) 29-49.

[5] R.A. Flores-Mendez, P. van Leeuwen, and D. Lukose, "Modeling expertise using KADS and MODEL-ECS", in *Proceedings of Banff Knowledge Acquisition Workshop '98,* Banff Canada, October 1998.

[6] J. McDermott, "Preliminary steps toward a taxonomy of problem-solving methods", ed. S. Marcus, *Automating Knowledge Acquisition for Expert Systems,* (Boston, MA: Kluwer Academic, 1988) 225-255.

[7] T. Gruber, "A translation approach to portable ontology specifications", in *Proceedings of the seventh Banff Knowledge acquisition Knowledge-Based Systems Workshop '92,* Banff Canada, October 11-16, 1992, paper no.12.

[8] J.H. Alexander, M.J. Freiling, S.J Shulman, S. Rehfuss, and S.L. Messick, "Ontological analysis: An ongoing experiment", International Journal of Man-Machine Studies **26**(4), 473-485.

[9] Protégé, http://protege.stanford.edu.

[10] R. Harrison, D. Obst and C. Chan, "Design of an Ontology Management Framework", accepted for ICCI 2005, Irvine, California, August 8-10, 2005.

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