MANAGING THE KNOWLEDGE NEEDED TO SUPPORT AN ELECTRONIC PERSONAL ASSISTANT

An End-User Friendly Graphical Ontology Editing Tool

Matthias Einig
School of Computer Science/Mathematics, Munich University of Applied Sciences, Lothstraße 34, Munich, Germany

Roger Tagg
School of Computer and Information Science, University of South Australia, Mawson Lakes, SA, Australia

Georg Peters
School of Computer Science/Mathematics, Munich University of Applied Sciences, Lothstraße 34, Munich, Germany

Keywords: Ontology, Graphical Editor, Personal Information Management, Groupware.

Abstract: Today's administrative worker has to handle huge amounts of data of different types, from many different sources and using multiple software tools. The effort in organizing and retrieving this data is often disproportionate to the actual benefit gained. Ontology-based categorization of knowledge has been advocated to provide a common infrastructure to the tools. However, most current software for building and maintaining ontologies is too complicated for the average end-user. This paper describes a prototype ontology editor application that aims to provide a more easily understandable and usable interface.

1 INTRODUCTION

The increasing problem of work and information overload for many information workers has been frequently noted, (e.g. Whittaker, 1996; Patch, 1998). Software solutions such as Groupware (Coleman, 1997; Chaffey, 1998) and Personal Information Management (Jones, 2004; Boardman, 2004) have been introduced. But so far, these solutions fall well short of the service that many managers formerly enjoyed through human secretaries and personal assistants. Our research is geared to providing an Electronic Personal Assistant that can replace – and hopefully improve on – the service previously supplied by human assistants. The beneficiaries would be both individuals and groups, doing either creative work or routine administration – or both.

All such IT support has to be based on a repository of the knowledge, both contextual and structural, through which the human participants view their work. Garshol (2004) discusses a range of approaches for representing such knowledge: metadata, thesauri, taxonomies, topic maps and ontologies.

The work presented in this paper follows the view that ontologies (Davies, 2002; McGuinness, 2004) can provide a good mechanism for combining personal knowledge of individual users with the domain or application knowledge of the groups to which users belong.

This implies that the end-user should be able to manage her/his knowledge directly without significant support from a specialist. Currently, however, managing an ontology tends to be a task of a knowledge management specialist rather than the end-user.

In an attempt to bridge this gap, our work has concentrated on developing a tool, *EzOntoEdit*, which allows the end-users themselves to maintain - their ontology.

Sections 2 and 3 of this paper describe the functional and usability requirements for a suitable tool. Section 4 describes the design and implementation of a prototype. Comments regarding the limitations of this prototype are given in section 5.

2 FUNCTIONAL REQUIREMENTS

Developing any ontology is a dynamic process, starting with an initial ontology that is typically imported from a group or common source. This is later revised, refined, and populated in more detail.

There are a number of software packages now available that allow creation and maintenance of ontologies e.g. Protégé (2005) or AIFB (2005). Most current tools do allow the option of editing in a graphical way, either with a built-in functionality or through separate plug-ins e.g. Jambalaya (Chisel Group, 2005) for Protégé.

Based on an evaluation of these tools we identified four classes of functional requirements for an ontology editor: (1) *editing functions*, (2) *transparency*, (3) *auditing* and (4) *reversibility*.

2.1 Editing Functions

The basic functional requirement is to be able to add, remove and modify entities in each of the main entity types in the OWL Ontology meta-model (W3C, 2005a), namely: (1) *classes*, (2) *hierarchies and other inter-class relationships*, (3) *properties and their domains and ranges*, (4) *instances and their property values*.

2.2 Transparency

It is an essential requirement for the editor to keep the ontology in a consistent, valid state after alterations have been performed. All changes need to be made as transparent as possible to the user.

This may be no problem in small ontologies, but since ontologies tend to grow over time, it becomes difficult to retain an overview over the relationships within the ontology, especially when the ontology is built in a collaborative way.

Consequently, if actions cannot be performed because the editor cannot guarantee the consistency or validity of the ontology, or data might be lost, the user must be warned by the application. Then if the user decides to cancel the operation, no changes will be made and the ontology should stay consistent.

2.3 Auditing

The following aspects of auditing should be considered (Stojanovic, 2002):

- A detailed log should be kept of every change to the ontology to allow manual recovery of lost data and reconstruction of the events which caused the current state of the ontology;
- Additional information to each log entry, like identity of the author of the change, the time of the change and a textual change description should be stored.

Auditing is particularly important as several people may be contributing to a shared ontology. A log also relates to the reversibility requirement (see next section).

2.4 Reversibility

To avoid unintentional data loss when the user makes an unintended change, an *undo* feature is required.

Reversibility implies that the old state of the ontology has to be cached somehow for complete reconstruction of the previous state. This must include the case where a class within a hierarchy is deleted. The tool has to recover the lost instances and property values and return the class hierarchy to its original state.

Reversibility also implies the ability to revert to the previous visual (graphical) state of the work. Previously selected objects should be selected again, and restored objects or properties of objects should be moved into the user's view, so that the user realizes the success of his *undo* operation.

3 USABILITY REQUIREMENTS

According to Shneidermann (2000) usability requirements include: (1) platform independence, (2) native appearance, (3) ease of input method, (4) adjustment to users' preferences and (5) feedback to the user. These are discussed below.

3.1 Platform Independence

One of the key goals defined for this project has been to provide platform independence of the application. The tool should run on all the commonest platforms like Windows, Linux and

MacOSX, using an interface that is familiar on whatever computer the user is used to working.

3.2 Native Appearance

To avoid risk of rejection, a user's interface should behave in tune with the user's computing habits, which includes the computing platform.

But requiring a platform independent application creates the problem that the user interface may no longer be completely uniform. There may be totally different usage concepts on the various platforms, for example the best design of dialog boxes. Although platform independence may be gained by using a programming language like Java, the advantage will be partly lost by the need to implement platform-dependent code fragments into the program.

3.3 Ease of Input Method

In our project we chose to concentrate on a graphical editor since we believe that it suits end-users better than a text based interface. An ideal graphical ontology interface includes:

- Zooming in and out
- Displaying a small context window, which shows the current viewpoint;
- Selective hiding of information;
- Displaying a hierarchy starting from a selected node;
- Displaying instances only of a selected node;
- Searching for objects by name;
- "Magnifying glass", reducing the size of objects the further away they are from the centre;
- Preserving the positions of objects in the view after reloading the ontology.

3.4 Adjustment to Users' Preferences

The appearance and behavioural settings of the software should be open to modification. These include colours, sizes and shapes of objects. Settings should be preserved after exiting the application and restored on the next launch.

3.5 Feedback to the User

The primary types of user feedback are input validation and notification of any necessary additional modifications to the ontology consequent on a user action.

In addition, user assistance is desirable on such matters as:

- Wizards for complex or repetitive tasks , e.g. importing
- Templates for common input formats
- Tips on how to use the tool
- Reviewing the status of the work
- Context-based enabling/disabling of certain functions.

4 PROTOTYPE IMPLEMENTATION

Our prototype EzOntoEdit (Easy Ontology Editor) was developed in Java, but with components enabling it to run as a native application in each of the operating systems Windows, Linux and MacOSX. The Java implementation allows any platform where a Java Virtual Machine (JVM) is provided (Lindholm, 1999). Swing components from the Java Foundation Classes (Sun, 2005) were used to implement the GUI.

4.1 Design Decisions

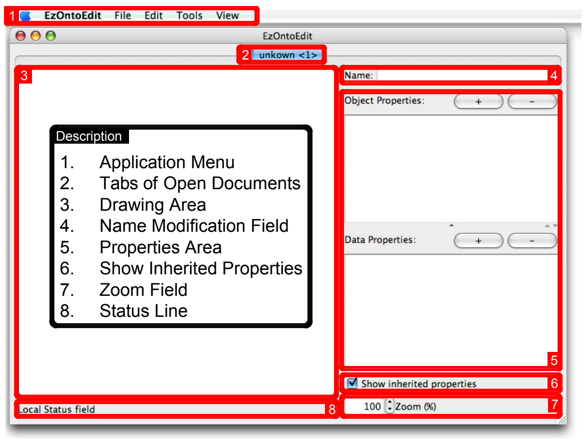
We have used an existing standard ontology format, namely OWL (Web Ontology Language), which is based on the RDF (Resource Description Language) (W3C 2005b).

For parsing OWL files we chose to make use of the Jena Semantic Web Framework (Sourceforge 2005). EzOntoEdit can read and write all formats supported by the Jena Framework.

After considering the use of applets, we decided instead to develop a normal client/server window application. This provides the opportunity to use menus and create separate windows (such as a preferences window).

4.2 Main User Menu

The user interface has been kept as simple as possible. The main menu is shown in Figure 1.



Description
1. Application Menu
2. Tabs of Open Documents
3. Drawing Area
4. Name Modification Field
5. Properties Area
6. Show Inherited Properties
7. Zoom Field
8. Status Line

Figure 1: EzOntoEdit Main Menu.

The application menu (1) allows access to all available functions and is attached to the application (except for MacOSX). The tabs of open documents (2) allow more than one document to be open at a time, with easy switching between them. The drawing area (3) becomes scrollable when the ontology objects stretch beyond the visible region. The name modification field (4) is used to change the name of classes or instances when they are selected. The properties area (5) shows the available object and data properties of a selected class or instance and provides a method to add, delete and modify them. If the checkbox “Show inherited properties” (6) is activated the properties area also shows the properties inherited from classes higher in that class's hierarchy. The zoom field (7) allows changing the current zoom factor of the visible document either by using spin buttons or by typing in the desired % magnification. Finally the status line (8) shows messages related to the currently visible document.

4.3 Importing

EzOntoEdit’s import function allows quick creation of classes and instances from a text list in either CSV (comma separated values) or WSV (whitespace separated values) formatted files. This helps the user to introduce bulk amounts of data into his ontology, rather than enter these as ad hoc additions. The importer provides an option that enables the user, after selecting the file, to choose an existing class in the ontology where the imported object can be added either as subclass or as instance of the class.

4.4 Basic Editing Interface

We envisage that, in most cases, the user of this tool will be editing an ontology that has already been created. However we have included facilities for adding classes, properties and instances at any stage of development.

User facilities for editing are similar to typical modern drawing tools, and allow alternative modes such as choosing from menus, keyboard short cuts etc. Toggling is supported between a class view and an instance view. An instance indicator is used in the class view to show which classes have instances. Validation is performed when the user clicks away from the current selection.

Relations between classes are added by dragging and dropping one class’s icon onto that of the related class (see Figure 2). Relations can also be removed graphically.

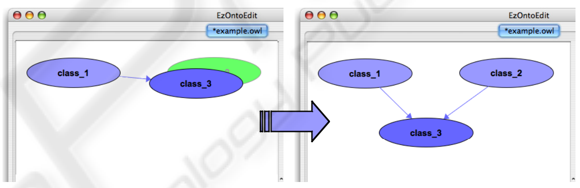


Figure 2: Creating relations by dragging classes.

The application also informs the user of the current view status (i.e. class or instance view) by displaying the corresponding text in the status bar at the bottom after changing the view.

4.5 Validation

A limited amount of input validation has been included in EzOntoEdit. Currently this covers the creation and renaming of classes, instances and properties to assure validity of the object names. The following conditions are checked in the present version of the tool: (1) *name is not empty*, (2) *name starts with a letter a-z or A-Z*, (3) *name contains only alphanumeric characters*, (4) *no resource (class, property or instance) within the ontology has the same name*.

When errors occur on modifying the ontology, dialogs pop up to inform the user about the cause of the error.

When importing, if an object already exists with the same name as an object being imported, an error message is displayed and the importer skips to the next string. The imported names are adjusted automatically to fulfil the naming standards as described in the OWL standard. If an import object starts with a number the character "X" will be added at the beginning of the name, special characters and white spaces (space, tab, new line, etc.) are removed automatically.

4.6 Logging

Currently EzOntoEdit contains basic logging features only. When the logging is switched on the logging messages are written into the file `ezontoedit-<current_date>.log` in the application directory. Four different log levels are used:

- Level 1: Errors, messages that occurred while editing, e.g. a resource exists already;
- Level 2: Information - feedback from successful user actions like add or delete;
- Level 3: Exceptions - application error messages;
- Level 4: Debug information.

All log entries start with a timestamp and their log type, to allow reconstruction of performed actions. As EzOntoEdit does not yet include the ability to manage users, the identification of the author cannot currently be tracked.

4.7 Changing User Preferences

To change the appearance of the user interface to the tool, EzOntoEdit provides a preferences panel. This allows the user to set the (1) *colours for text, background and border of classes and instances*, (2) *colour of arrows*, (3) *height and width of classes, instances and instance indicators*, (4) *position of the instance indicator*, (4) *display shape of classes, instances and instance indicators*.

Any colour can be chosen for the each different state of selection e.g. selected, drop target etc for both classes and instances. Arrow colours can be defined differently for pointing *from* and *to* the selected class.

Currently there are two display shapes available, ovals and rectangles, which can be adjusted in width and height.

After changing the settings the user can preview his adjustments, reset the previous values and discard or accept the changes. The modified and accepted preferences will be reloaded when the application is started the next time.

4.8 Native Look and Feel

A key aspect of this project has been to provide interfaces that follow the guidelines provided by the developers of each of the 3 most common target operating systems, and hence provide interfaces similar to commonly used applications on those platforms.

The Java Virtual Machine (JVM) allows Swing applications to draw themselves with the native controls of the underlying operating system by setting the UIManager to the system look and feel.

Unfortunately there are many other parts of a Java application that do not appear native e.g. common dialogs, keyboard shortcuts, menu arrangement, application bundle etc. Some of these have been addressed in the prototype.

4.9 Packaging the Application

Bundling the application to a platform-dependent native executable file hides from the user the fact that it is a Java application. It also avoids dealing with inconvenient command line instructions or un-aesthetic batch files. This has to be done differently for each operating system.

For MacOSX, there has to be an application bundle (EzOntoEdit.app) which is actually a folder containing the application as a *jar* file, all additional libraries, the icon and some preferences files for storing the meta-data necessary for the application start. For Windows a standard Windows *.exe* executable has to be created out of the application's *jar* file. The necessary libraries cannot be added to the application and therefore have to remain with the executable in a subfolder. On Linux the application has to be started with a shell script, which sets the *classpath* to the required libraries and launches the main executable out of the applications jar-file. Consequently, the files cannot be bundled to a single file. This way of starting would of course also be possible for any other operating system with a Java Virtual Machine, including MacOSX and Windows.

5 EVALUATION

5.1 Significance of the Work

We feel that, although we may not have made any major breakthroughs in ontology editing, we have demonstrated the feasibility of building a relatively lightweight, instinctive graphical tool that can run on any basic hardware and software configuration. Perhaps more significantly, we have addressed the problem of providing a compatible tool for multiple operating systems.

5.2 Limitations

Despite the considerable time spent building this prototype, there are a number of features that have not yet been implemented. These include:

- Class relationships other than specialization
- Displaying a hierarchy from a selected node
- Showing instances of a class as a table
- Moving an instance from one class to another
- Undo and redo of changes
- Copying and pasting between open ontologies
- Searching for nodes by name
- Linking to ontology instance data stored in a relational database
- Changing the names of properties

Although many features could be added, there are arguments for not making the tool too complicated, otherwise it could end up by becoming a tool for specialists again.

In addition to the above list, we feel that the main missing feature in this project is that we have not yet addressed the "lexical dimension". By this we mean the use of an inventory of text strings (words or phrases) by which the concepts in the ontology can be recognized in messages such as emails.

6 CONCLUSION

In this project we have built a new ontology editor that allows easier, more intuitive handling of ontologies, and that also allows sharing of the effort of keeping ontologies up-to-date between users working in groups.

Preliminary end-user tests with EzOntoEdit have confirmed that ontology editing need not necessarily be a complicated task. However, more extensive testing is required before we can reliably establish whether or not non-specialist users will be prepared to invest the effort to maintain their ontologies..

In the longer term, rather than remaining as a stand-alone tool, we feel that EzOntoEdit could be of most benefit as a component that can be integrated with Personal Information Management and Groupware clients in the future.

REFERENCES

AIFB, 2005. OntoEdit website, University of Karlsruhe. http://www.ontoknowledge.org/tools/ontoedit.shtml (accessed 11 Oct 2005)

Boardman, R., 2004. Improving Tool Support for Personal Information Management, PhD Thesis, Imperial College, London. http://www.iis.ee.ic.ac.uk/~rick/¬ thesis/

Chaffey, D., 1998. *Groupware, Workflow and Intranets*, Digital Press.

Chisel Group, 2005. Jambalaya Plugin for Protégé, http://www.thechiselgroup.org (accessed 11 Oct 2005)

Coleman, D., 1997. *Groupware: Collaborative Strategies for Corporate LANs and Intranets*, Prentice Hall, ISBN# 0-13-727728-8.

Davies, J., Fensel, D. and van Harmelen, F., 2002. *Towards the Semantic Web: Ontology-driven Knowledge Management*, Wiley, 2002.

Garshol, L.M., 2004. Metadata? Thesauri? Taxonomies? Topic Maps! – Making Sense of it All. Journal of Information Science, 30(4), pp 378-391, ISSN 0165-5515, CILIP.

Jones, W., 2004. Finders, Keepers? The present and future perfect in support of personal information management. First Monday 9(3). http://www.¬ firstmonday.org/issues/issue9_3/jones/index.html

Lindholm, T. and Vellin, F., 1999. *The Java(TM) Virtual Machine Specification*, Sun/Addison-Wesley.

McGuinness, D.L. and van Harmelen, F., 2004. OWL Web Ontology Language Overview. http://www.¬ w3.org/TR/owl-features/

Patch. K. and Smalley E., 1998. E-mail Overload, Network World, October 26th, http://www.nwfusion¬ .com/netresources/1026email.html

Protégé, 2005. Protégé website, Stanford University. http://protege.stanford.edu (accessed 11 Oct 2005)

Shneiderman, B., 2000. Universal Usability, Communications of the ACM, Vol 43, pp84-91.

Sourceforge, 2005. The Jena Semantic Web Framework, http://jena.sourceforge.net (accessed 11 Oct 2005)

Stojanovic, L. and Motik, B., 2002. Ontology Evolution within Ontology Editors, 13th International Conference on Knowledge Engineering and Management, Madrid, Spain.

Sun, 2005. Java Foundation Classes (JFC) website http://java.sun.com/products/jfc/download.html (accessed 11 Oct 2005)

Whittaker S. and Sidner C., Email Overload: Exploring Personal Information Management of Email, Lotus Development Corp., http://dis.shef.ac.uk/steve¬ whittaker/emlch96.pdf

W3C, 2005a. Web Ontology Language (OWL), http://www.w3.org/2001/sw/WebOnt/ (accessed 11 Oct 2005)

W3C, 2005b. Resource Description Framework, http://www.w3.org/RDF/ (accessed 11 Oct 2005)