Editing Description Logic Ontologies with the Protégé OWL Plugin

Holger Knublauch and Mark A. Musen
Stanford Medical Informatics, Stanford University, CA
holger@smi.stanford.edu, musen@smi.stanford.edu

Alan L. Rector
Medical Informatics Group, University of Manchester, UK
rector@cs.man.ac.uk

Abstract

The growing interest in the Semantic Web and the Web Ontology Language (OWL) will reveal the potential of Description Logics in industrial projects. The rich semantics of OWL provide powerful reasoning capabilities that help build, maintain and query domain models for many purposes. However, before OWL can unfold its full potential, user-friendly tools with a scalable architecture are required. We present the OWL Plugin, an extension of the Protégé ontology development environment, which can be used to define classes and properties, to edit logical class expressions, to invoke reasoners, and to link ontologies into the Semantic Web. We analyze some of the challenges for developers of Description Logic editors, and discuss some of our user interface design decisions.

1 Introduction

The formal underpinnings of Description Logics (DL) [1] are a cornerstone of Semantic Web technology such as the Web Ontology Language (OWL) [8]. DL reasoners can help build and maintain sharable ontologies by revealing inconsistencies, hidden dependencies, redundancies, and misclassifications [7].

While the formal terrain of DLs is now well mapped out and covered by efficient algorithms, a new generation of end-user tools is necessary to put this technology into the spotlight of industrial routine. In contrast to DLs, other modeling paradigms such as Object-Orientation and frames are supported by many professional editing tools that have evolved over many years of industrial use and, as a result, reflect best practices and common design patterns. A good example of such a tool is Protégé [3], a knowledge modeling platform developed at Stanford Medical Informatics with support from a community of thousands of users over almost two decades.

Some of the technologies explored and refined by these tools can be applied to DL. For example, UML diagrams and Protégé's class explorer may serve as a good starting point for DL editors as well. However, there are crucial differences between DLs on the one hand and object-oriented or frame-based systems on the other that require custom-tailored solutions. In this paper, we will focus on four key issues in DL editing tools:

1. *Logical Expressions.* DL languages rely on (potentially deeply nested) logical expressions which can be hard to read, understand, and edit.

2. *Class Descriptions.* Classes in DL ontologies are either primitive or defined. A consistent approach for editing these two modes is needed.

3. *Reasoning.* DL ontologies can be classified (i.e., some relationships between concepts are inferred from the asserted class descriptions). A tool should provide both views, and allow users to compare the differences.

4. *Scalability.* DL ontologies are potentially very complex and large. An ontology editor should simplify navigation and help users to maintain large ontologies through mechanisms such as annotation metadata.

Some solutions for these issues have been explored by existing tools, most notably OilEd [2], developed at the University of Manchester. While OilEd succeeded in making DL technology available to a broader user community, its authors never intended it as a full ontology development environment but rather as a platform for experiments. As a result, OilEd's architecture is neither scalable to really large ontologies nor sufficiently flexible to allow its user interface to be customized.

The developers of Protégé and the OilEd team have recently joined forces in a transatlantic project called CO-ODE [1]. Our goal is to develop a new generation of OWL ontology editing tools, based on Protégé and the experiences collected with OilEd. Our collaboration has lead to the development of the Protégé OWL Plugin, which can be used to define classes and properties, to edit logical class expressions, to invoke reasoners, and to link ontologies into the Semantic Web.

The following sections provide details and design decisions of our current system. Due to space constraints we cannot provide a comprehensive description of the full system, but we focus on the key issues mentioned above. Before we go into these issues, we will start with some background on the Protégé architecture and its general principles.

[1] `http://www.co-ode.org`

2 Protégé and the OWL Plugin

Protégé [3] is an open-source ontology development environment with functionality for editing classes, slots (properties), and instances. The current version of Protégé (2.0) is highly extensible and customizable. At its core is a frame-based knowledge model [5] with support for metaclasses. Other languages such as OWL can be defined on top of this core frame model [6]. The mechanism we have used to represent the OWL metamodel in terms of Protégé frames will be described in a separate paper.

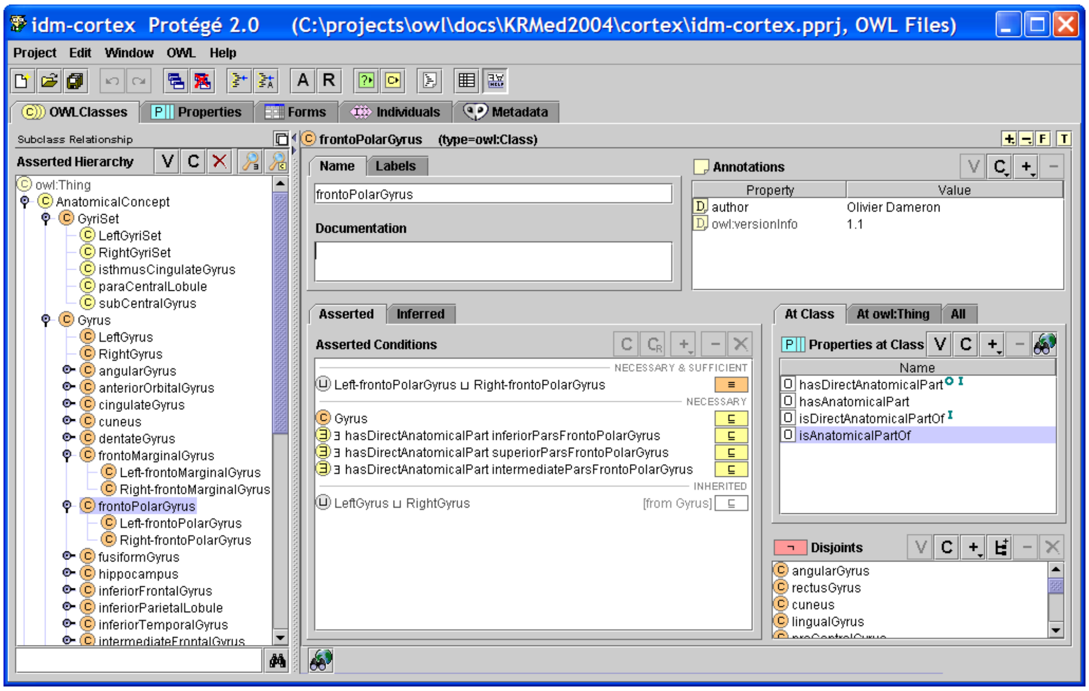
Protégé makes it not only possible to extend the metamodel but also to customize the user interface freely. As illustrated in Figure 1, Protégé's user interface consists of several screens, called *tabs*, each of which displays a different aspect of the ontology in a specialized view. Each of the tabs can include arbitrary Java components. Most of the existing tabs provide an explorer-style view of the model, with a tree on the left hand side and details of the selected node on the right hand side. The details of the selected object are typically displayed by means of forms. The forms consist of configurable components, called *widgets*. Typically, each widget displays one property of the selected object. There are standard widgets for the most common property types, but ontology developers are free to replace the default widgets with specialized components. Widgets, tabs, and back-ends are called *plugins*. Protégé's architecture makes it possible to add and activate plugins dynamically, so that the default system's appearance and behavior can be completely adapted to a project's needs.

The OWL Plugin[2] is a large Protégé plugin with support for OWL. It can be used to load and save OWL files in various formats, to edit OWL ontologies with custom-tailored graphical widgets, and to perform intelligent reasoning based on DLs. As shown in Figure 1, the OWL Plugin's user interface provides various default tabs. The *OWLClasses* tab displays the ontology's class hierarchy, allows developers to create and edit classes, and displays the result of the classification. The *Properties* tab can be used to create and edit the properties in the ontology. The *Individuals* tab can be used to create and edit individuals, and to acquire Semantic Web contents. The *Forms* tab allows to customize the forms used for editing classes, properties and individuals. The *Metadata* tab displays ontology metadata such as namespace prefixes. Ontology builders will typically focus on the OWLClasses tab, which is described in the following sections.

3 Editing Logical OWL Expressions

One of the first and most important decisions in the design of an OWL editor is how to display class expressions in a user-friendly but efficient way. The RDF

[2]`http://protege.stanford.edu/plugins/owl`


idm-cortex Protégé 2.0 (C:\projects\owl\docs\KRMed2004\cortex\idm-cortex.pprj, OWL Files)
Project Edit Window OWL Help
OWLClasses Properties Forms Individuals Metadata
Subclass Relationship
Asserted Hierarchy
frontoPolarGyrus (type=owl:Class)
Name Labels
Annotations
Asserted Inferred
Asserted Conditions
At Class At owl:Thing All
Properties at Class
Disjoints

Asserted Hierarchy:

```
owl:Thing
  AnatomicalConcept
    GyriSet
      LeftGyriSet
      RightGyriSet
      isthmusCingulateGyrus
      paraCentralLobule
      subCentralGyrus
    Gyrus
      LeftGyrus
      RightGyrus
      angularGyrus
      anteriorOrbitalGyrus
      cingulateGyrus
      cuneus
      dentateGyrus
      frontoMarginalGyrus
        Left-frontoMarginalGyrus
        Right-frontoMarginalGyrus
      frontoPolarGyrus
        Left-frontoPolarGyrus
        Right-frontoPolarGyrus
      fusiformGyrus
      hippocampus
      inferiorFrontalGyrus
      inferiorParietalLobule
      inferiorTemporalGyrus
      intermediateFrontalGyrus
```

Name: frontoPolarGyrus

Documentation:

Property	Value
author	Olivier Dameron
owl:versionInfo	1.1

Asserted Conditions:

```
                                                        NECESSARY & SUFFICIENT
Left-frontoPolarGyrus ⊔ Right-frontoPolarGyrus                               ≡
                                                                     NECESSARY
Gyrus                                                                        ⊑
∃ hasDirectAnatomicalPart inferiorParsFrontoPolarGyrus                       ⊑
∃ hasDirectAnatomicalPart superiorParsFrontoPolarGyrus                       ⊑
∃ hasDirectAnatomicalPart intermediateParsFrontoPolarGyrus                   ⊑
                                                                     INHERITED
LeftGyrus ⊔ RightGyrus                                          [from Gyrus] ⊑
```

Properties at Class:

Name
hasDirectAnatomicalPart
hasAnatomicalPart
isDirectAnatomicalPartOf
isAnatomicalPartOf

Disjoints:

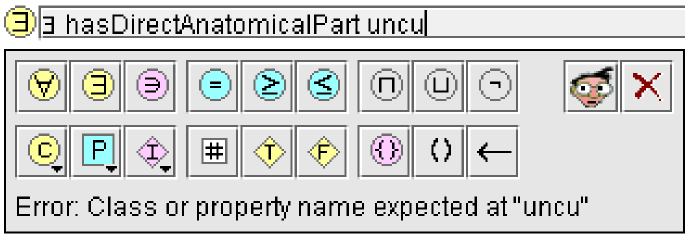
```
angularGyrus
rectusGyrus
cuneus
lingualGyrus
```

Figure 1: The class editor of the Protégé OWL Plugin.

syntax proposed in the OWL specification [8] is clearly too verbose to be of any use here. The OWL Abstract Syntax [9] is much more user-friendly, but still quite verbose. For the OWL Plugin, we chose to use an expression syntax based on standard DL symbols [1], such as ∀ and ⊔. These symbols (Figure 2) allow to the system display even complex nested expressions in a single row.

A trade-off from this syntax is that some characters are not found on standard keyboards. The OWL Plugin provides a comfortable expression editor which allows users to quickly assemble expressions with either the mouse or the keyboard (Figure 2). The special characters are mapped onto keys known from languages such as Java (e.g., `owl:intersectionOf` is entered with the `&` key). To simplify editing, keyboard users can exploit a syntax completion mechanism known from programming environments, which semi-automatically completes partial names after the uses has pressed `tab`. The expression editor is invoked by a double-click on a class expression, and then pops up directly below the expression. For really complex expressions, users can open a multi-line editor in an extra window, which formats the expression using indentation.

The OWL Plugin helps new users to get acquainted with the expression syntax. An English prose text is shown as a “tool tip” when the mouse is moved over the expression. For example, “∃ hasPet Cat” is displayed as “Any object which has a cat as its pet”.


∃ hasDirectAnatomicalPart uncu

Error: Class or property name expected at "uncu"

Figure 2: Protégé provides a comfortable editor for arbitrary OWL expressions.

4 Editing Class Descriptions

Another major design decision for a DL class editor is how to edit the logical class definitions. Protégé users are accustomed to an object-centered view to the interface which has required some effort to adapt to OWL. The distinction between defined and primitive classes simply is not found in frame-style or object-oriented modeling paradigms, and this can compound users' confusion when learning the DL paradigm. In the OWL specification, there is a lack of uniformity between defined classes (classes with necessary & sufficient conditions) and primitive classes (only necessary conditions). Multiple necessary conditions are represented by multiple `rdfs:subClassOf` statements whose intersection is implied, whereas sets of multiple necessary & sufficient conditions are represented by an `owl:equivalentClass` block containing an explicit intersection class. Although logically consistent, experience has shown that many users find the difference confusing. As a result, it was decided that the user interface should not simply reflect the structure suggested by the OWL specification but attempt to provide a clearer more uniform presentation to users.

During the evolution of the OWL Plugin we experimented with several interface designs, partly based on existing tools such as the Protégé core system and OilEd, partly on suggestions from our colleagues and users. OilEd has two modes: one to "partially" define a class with only necessary conditions, the other to "completely" define a class with necessary & sufficient conditions. There is a button to switch between these two modes. While this feature allowed the OilEd developers to provide customized widgets for various kinds of class descriptions (e.g. a widget for only restrictions), it has the disadvantage that users have to maintain separate class axioms in a separate pane for the necessary restrictions of classes that also are "completely" defined by a set of necessary & sufficient conditions. There is no one pane in OilEd in which one can see both the sets of necessary and sufficient conditions and any additional necessary conditions (i.e. axioms taking the defined class as their antecedent.)

As shown in the center of Figure 1, the OWL Plugin solves this problem by means of a list of conditions, organized into blocks of necessary & sufficient, necessary, and inherited (i.e., inferred) conditions. Each of the necessary & sufficient blocks represents a single equivalent intersection class, and only those inherited conditions are listed that have not been further restricted higher up in the hierarchy (e.g., allValuesFrom Animal would not be shown if allValuesFrom Dog were also inferable).

The editor supports drag-and-drop between blocks in the conditions list, and copy-and-paste of expressions. It also supports changing superclasses by dragging a class from one parent to another in the class tree on the left hand side of the window.

In addition to the list of conditions, there is also a custom-tailored widget for entering disjoint classes, which has special support for typical design patterns such as making all siblings disjoint. This rather object-centered design of the OWLClasses tab makes it possible to maintain the whole class definition on a single screen.

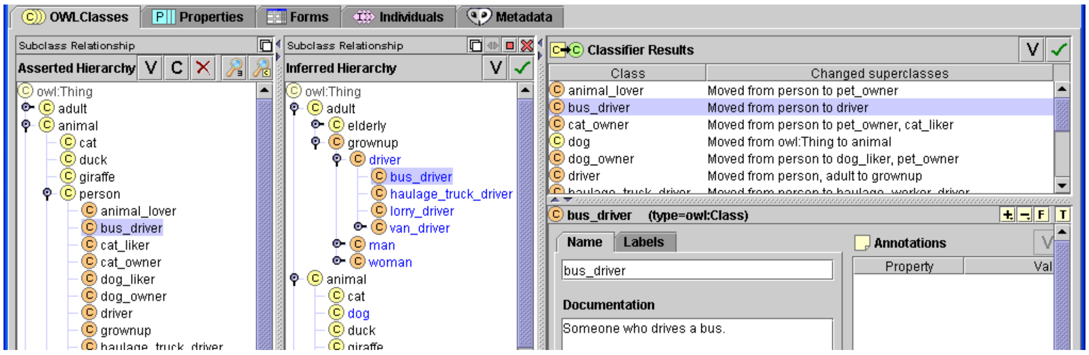
5 Working with Classification

The OWL Plugin provides direct access to reasoners such as Racer [4]. The current user interface supports two types of DL reasoning: Consistency checking and classification (subsumption). Other types of reasoning, such as instance checking, are work in progress.

Consistency checking (i.e., the test whether a class could have instances) can be invoked either for all classes with a single mouse click, or for selected classes only. Inconsistent classes are marked with a red bordered icon.

Classification (i.e., inferring a new subsumption tree from the asserted definitions) can be invoked with the classify button on a one-shot basis. When the classify button is pressed, the system determines the OWL species, because some reasoners are unable to handle OWL Full ontologies. This is done using the validation service from the Jena[3] library. If the ontology is in OWL Full (e.g., because metaclasses are used) the system attempts to convert the ontology temporarily into OWL DL. The OWL Plugin supports editing some features of OWL Full (e.g., to assign ranges to annotation properties, and to create metaclasses). These are easily detected and can be removed before the data are sent to the classifier. Once the ontology has been converted into OWL DL, a full consistency check is performed, because inconsistent classes cannot be classified correctly. Finally, the classification results are stored until the next invocation of the classifier, and can be browsed separately. Classification can be invoked either for the whole ontology, or for selected subtrees only. In the latter case, the transitive closure of all accessible classes is sent to the classifier.

[3] `http://www.hpl.hp.com/semweb/jena2.htm`



Class	Changed superclasses
animal_lover	Moved from person to pet_owner
bus_driver	Moved from person to driver
cat_owner	Moved from person to pet_owner, cat_liker
dog	Moved from owl:Thing to animal
dog_owner	Moved from person to dog_liker, pet_owner
driver	Moved from person, adult to grownup

bus_driver

Documentation

Someone who drives a bus.

Figure 3: Protégé allows users to compare asserted and inferred class relationships. The system displays two hierarchies for asserted and inferred subsumption relationships, and provides a clickable list of the differences between them.

OWL files only store the subsumptions that have been asserted by the user. However, experience has shown that, in order to edit and correct their ontologies, users need to distinguish between what they have asserted and what the classifier has inferred. Many users may find it more natural to navigate the inferred hierarchy, because it displays the semantically correct position of the classes.

The OWL Plugin addresses this need by displaying both hierarchies and making available extensive information on the inferences made during classification. As illustrated in Figure 3, after classification the OWL Plugin displays an inferred classification hierarchy beside the original asserted hierarchy. The classes that have changed their superclasses are highlighted in blue, and moving the mouse over them explains the changes. Furthermore, a complete list of all changes suggested by the classifier is shown in the upper right area. A click on an entry navigates to the affected class. Also, the conditions widget can be switched between asserted and inferred conditions. All this allows the users to quickly analyze the changes.

6 Scalability

DL ontologies may become complex and large. The support for arbitrary class expressions means that DL ontologies typically contain many cross-links among classes, properties, individuals, and even among ontologies. This situation is complicated by the emerging Semantic Web, in which ontology development is distributed between groups around the world. As a result, scalability and support for ontology maintenance are crucial issues in ontology tools.

Protégé supports database storage that is scalable to several million concepts, and provides multi-user support for synchronous knowledge entry. The OWL

Plugin has already been used to edit ontologies with tens of thousands of classes. In support of handling such large ontologies, the OWL Plugin also provides a variety of navigation aids, such as lexical search functions and "find usage" buttons which allow to directly access all classes and properties that somehow reference a given object.

Documentation is essential for large ontologies. Most modeling or programming languages allow the attachment of comments or annotations to document the model's contents and to track provenance and changes. OWL supports this through annotation properties. The OWL Plugin allows to attach annotations to ontologies, properties, individuals, and classes, including anonymous classes. Annotation properties can be edited by means of a specific table widget (in the upper right area of Figure 1). The OWL Plugin allows the user to put arbitrary values into annotations, including complex objects. We are currently optimizing the tool for Dublin Core metadata so that, for example, annotation properties with change dates and authors will be filled in automatically.

7 Discussion

We have provided a brief overview of the OWL Plugin for Protégé. The Plugin explores several innovative approaches for displaying and editing logical expressions, editing class definitions, displaying classification results, and maintaining ontologies. Although we have not performed a formal evaluation of our user interface yet, the feedback from the user community is very encouraging. Many users suggest that the plugin's simple editors and comfortable reasoning interface supports rapid but sustainable ontology evolution. The constructive dialogs on forums such as those of the CO-ODE project and the Protégé discussion list will accelerate the evolution of the user interface with support for additional design patterns and best practices.

Many other groups from around the world are also developing Protégé plugins, including alternative wizard-style editors, and tools to query ontologies, to visualize ontologies graphically, and to manage ontology versions and changes. Furthermore, Protégé provides immediate access to all services for the Jena API, because it internally synchronizes the ontology with a Jena model.

Protégé has a large and rapidly growing community of thousands of users. Providing the community with a widely available, easy-to-use, open-source platform for OWL ontology design has the potential to use the leverage of that user base to bring DL technology into a wider audience. However, to do so effectively, requires thinking carefully about how to provide user interface metaphors with which that community is comfortable. The OWL Plugin aims to retain Protégé's object-centered interface without compromising OWL's DL semantics.

Acknowledgements. This work has been funded by a contract from the US National Cancer Institute and by grant P41LM007885 from the National Library of Medicine. Additional support for this work came from the UK Joint Information Services Committee under the CO-ODE grant. We would like thank our colleagues in Stanford (especially Ray Fergerson and Natasha Noy) and Manchester (especially Sean Bechhofer and Ian Horrocks for their work on OilEd and contribution to the interface discussions), the developers of the Jena library at HP Labs in Bristol, and the many Protégé users around the world for very valuable feedback during the evolution of the OWL Plugin.

References

[1] F. Baader, D. Calvanese, D. McGuineness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[2] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a reason-able ontology editor for the Semantic Web. In *14th International Workshop on Description Logics*, Stanford, CA, 2001.

[3] J. Gennari, M. Musen, R. Fergerson, W. Grosso, M. Crubézy, H. Eriksson, N. Noy, and S. Tu. The evolution of Protégé-2000: An environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[4] V. Haarslev and R. Moeller. Racer: A core inference engine for the Semantic Web. In *2nd International Workshop on Evaluation of Ontology-based Tools (EON-2003)*, Sanibel Island, FL, 2003.

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

[6] N. Noy, M. Sintek, S. Decker, M. Crubézy, R. Fergerson, and M. Musen. Creating Semantic Web contents with Protégé-2000. *IEEE Intelligent Systems*, 2(16):60–71, 2001.

[7] A. Rector. Description logics in medical informatics. Chapter in [1].

[8] World Wide Web Consortium. OWL Web Ontology Language Reference. W3C Recommendation 10 Feb, 2004.

[9] World Wide Web Consortium. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation 10 Feb, 2004.