Weaving the Biomedical Semantic Web with the Protégé OWL Plugin

Holger Knublauch Olivier Dameron Mark A. Musen

Stanford Medical Informatics, Stanford University, Stanford, CA (`http://protege.stanford.edu`)

Abstract

In this document we show how biomedical resources can be linked into a Semantic Web using Protégé. Protégé is a widely-used open-source ontology modeling environment with support for the Web Ontology Language (OWL). With the example domain of brain cortex anatomy we demonstrate how Protégé can be used to build an OWL ontology and to maintain ontology consistency with a description logic classifier. We also show how Protégé can be used to link existing Web resources such as biomedical articles and images into a Semantic Web.

INTRODUCTION

Biomedical Web resources in the existing internet are mainly optimized for use by humans. For example, researchers need to know the "correct" keywords to do a meaningful search using an online publications database. The vision of the Semantic Web [3] is to extend the existing Web with conceptual metadata that are more useful to machines, revealing the intended meaning of Web resources. This meaning could be used by software agents to perform tasks that are difficult with the current Web architecture. For example, an intelligent agent could retrieve semantically related publications, even if they don't contain the "correct" keyword.

Ontologies are a central building block of the Semantic Web. Ontologies define domain concepts and the relationships between them, and thus provide a domain language that is meaningful to both humans and machines. Ontologies are being defined for many biomedical domains, such as anatomy, genetics, and cancer research. The concepts from these ontologies can be used to annotate Web resources. The Web Ontology Language (OWL) [13] is widely accepted as the standard language for sharing Semantic Web contents. Protégé [4, 7] is an ontology development environment with a large community of active users. Protégé has been used for more than a decade to build large-scale biomedical applications. Rather recently, Protégé has been extended with support for OWL, and has become one of the leading OWL tools.

Our goal in this document is to help biomedical projects get started with Semantic Web technology. We first describe the architecture of a typical biomedical Semantic Web application from the domain of brain cortex anatomy. Then we give a short overview of Protégé and its OWL support. We describe how Protégé can be used to define domain classes and properties, and how to use features such as a classifier to maintain semantic consistency. We also briefly introduce the essential features of OWL and their representation in Protégé. Then we show how to link existing Web resources into the Semantic Web, so that they can be accessed by intelligent agents. We end this document with discussion and conclusions.

A BIOMEDICAL SEMANTIC WEB

The current Internet already contains vast amounts of biomedical information resources, such as research articles, images, clinical guidelines, and drug catalogues. Making these resources available in a more structured way is one of the goals of several large-scale ontology development efforts. For example, the goal of the National Cancer Institute's Thesaurus project [5] is to provide a well-defined conceptual model so that cancer-related resources can be structured in a machine-readable way. This conceptual model is an OWL ontology with tens of thousands of classes and dozens of properties.

For the purpose of this paper, we start with a less ambitious example ontology of brain cortex anatomy. Potential use cases of this ontology are teaching, decision support for clinical practice, sharing of neuroimaging data, or semantic assistance for data processing tools. The ontology defines concepts such as `FrontalLobe` and `LeftCentralSulcus`, and specialization, composition and spatial neighborhood relationships. In addition, the ontology also defines the logical characteristics of the concepts. For example, it states that a brain `Hemisphere` is composed of exactly five distinct lobes: one `FrontalLobe`, one `ParietalLobe`, one `TemporalLobe`, one `OccipitalLobe` and one `LimbicLobe`. These concepts and relationships are implemented as OWL classes and properties. They are stored in an OWL file which resides on a publicly accessible Web server. After the ontology has been published on the Web, other OWL ontologies, resources, agents, and services can

link to this file and use the ontology's concepts. For example, a Web repository of MRI scans could provide a collection of image metadata objects that would represent the attributes of the single scans (dimensions, resolution, contents), so that the best images for a specific topic can be retrieved automatically. If the image repository is loosely coupled and distributed over multiple hosts (e.g., multiple hospitals), then each of the servers could provide its own metadata objects. A user searching for a particular scan of a frontal lobe could then invoke an intelligent agent that would crawl through the various repositories to search for the best matches.

Another example of a Semantic Web application would be a context-sensitive search function for research articles. A publication database such as PubMed could provide a Web service that would refer to a conceptual model when providing metadata about articles. It could also rely on this conceptual model to guide and assist query processing. Users could invoke this Web service through a simple client application. The Web service could exploit the definitions from the ontology to widen or narrow the search into concepts that are substantially related to the terms the user has asked for. For example, it could deliver papers about glioma located in the precentral gyrus although the user has only asked for tumors of the frontal lobe, exploiting the background knowledge that a glioma is a kind of tumor and that the precentral gyrus is a part of the frontal lobe.

One of the advantages of shared conceptual models is that they can be reused in various contexts, even some that have not been imagined yet. Finally, the Semantic Web could even be used to point researchers and domain experts into new directions and to reveal cross-links between domains.

These examples illustrate the central role of *ontologies* in Semantic Web applications. Ontologies should adequately represent a domain and allow some kind of formal reasoning. They should be both understandable by humans and processable by software agents. Furthermore, since ontologies will evolve over time, they need to be maintainable. This demands for ontology modeling tools that provide a user-friendly view on the ontology and support an iterative working style with rapid turn-around times. Tools should also provide intelligent services that reveal inconsistencies and hidden dependencies among definitions.

PROTÉGÉ AND THE OWL PLUGIN

Since its beginning in the 1980's, Protégé has been driven by biomedical applications. Protégé started as a rather specialized tool for a specific kind of problem solving [4], but evolved into a very generic and flexible platform for many types of knowledge-based applications and tools from all kinds of domains.

Protégé can be characterized as an ontology development environment. It provides functionality for editing classes, slots (properties), and instances. One of its strengths is that it can automatically generate a user interface from class definitions, and thus can support rapid knowledge acquisition. Protégé supports database storage that is scalable to several million concepts, and provides multi-user support for synchronous knowledge entry.

The current version of Protégé (2.1) is highly extensible and customizable. At its core is a frame-based knowledge model [9] with support for metaclasses. These metaclasses can be extended to define other languages on top of the core frame model [10]. For these other languages, Protégé can be extended with back-ends for alternative file formats. Currently, back-ends for Clips, UML, XML, RDF, DAML+OIL, and OWL are available for download.

Protégé not only allows developers to extend the internal model representation, but also to customize the user interface freely. As illustrated in Figure 1, Protégé's user interface consists of several screens, called *tabs*, which display different aspects of the ontology in different views. Each of the tabs can be filled with arbitrary components. Most of the existing tabs provide a tree-browser view of the model, with a tree on the left 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 allows developers to add and activate plugins arbitrarily, so that the default system's appearance and behavior can be completely adapted to a project's needs.

The OWL Plugin[1] [8] is a complex 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 provide access to reasoning based on description logic. As shown in figure 1, the OWL Plugin's user interface provides various default tabs for editing OWL classes, properties, forms, individuals, and ontology metadata. The following section explains how to use the Classes, Properties and Metadata tabs for the de-

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

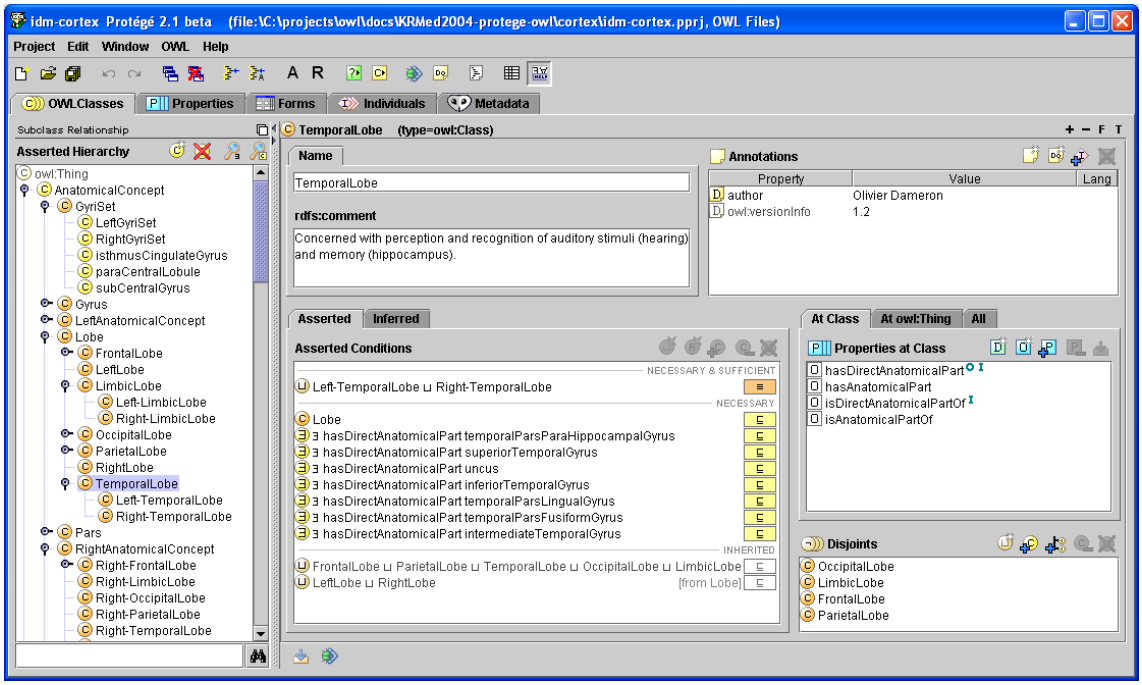

Figure 1: The class editor of the Protégé OWL Plugin.

sign of a biomedical ontology. The section after that introduces how to use the Individuals and Forms tabs for the acquisition of Semantic Web contents.

BUILDING OWL ONTOLOGIES WITH PROTÉGÉ

An OWL ontology can be regarded as a network of classes, properties, and individuals. *Classes* define names of the relevant domain concepts and their logical characteristics. *Properties* (sometimes also called slots, attributes or roles) define the relationships between classes, and allow to assign primitive values to instances. *Individuals* are instances of the classes with specific values for the properties. The Semantic Web can be regarded as a network of ontologies and other Web resources. OWL ontology concepts can have references to concepts in other ontologies. The basic mechanism for this capability is ontology import (i.e., an ontology can import resources from existing ontologies and create instances or specializations of their classes).

In our biomedical example ontology, we have a class called `CentralSulcus` which is defined as a kind of `AnatomicalConcept` that has a measured average depth. Individuals from this ontology would describe specific case data (e.g., a specific left central sulcus of an individual with the value of 23 mm for its depth). For the example ontology, we can import an existing ontology about units, and thus reuse the concepts from other files and support knowledge sharing. Let's take a look at how these elements can be defined in Protégé.

Classes

The most important view in the Protégé OWL Plugin is the OWLClasses tab (Figure 1). This tab displays the tree of the ontology's classes on the left, while the selected class is shown in a form in the center. The upper region of the class form allows users to edit class metadata such as name, comments, and labels, in multiple languages. The widget in the upper right area of the form allows users to assign values for *annotation properties* to a class. Annotation properties can hold arbitrary values such as author and creation date. Ontologies can define their own annotation properties or reuse existing ones such as those from the Dublin Core ontology. In contrast to other properties, annotation properties do not have any formal meaning for external OWL components like reasoners, but they are an extremely important vehicle for maintaining project-specific information. A typical use case for annotation properties in a biomedical field is to assign standardized identifiers such as ICD codes for concepts that describe a disease. Annotation properties, such as the predefined `rdfs:seeAlso`, can also be used to define cross-references between concepts. The OWL Plugin also uses annotation properties to store Protégé-specific information, and to manage "to-do" lists for ontology authors.

Properties

The *Properties* widget of the OWLClasses tab allows users to view and create relationships between classes. It provides access to those properties that could be used by the instances of the current class. The characteristics of a property are edited through the form shown in Figure 2. This form provides a metadata area in the upper part, displaying the property's name, annotations, and so on, similar to the presentation in the class form.

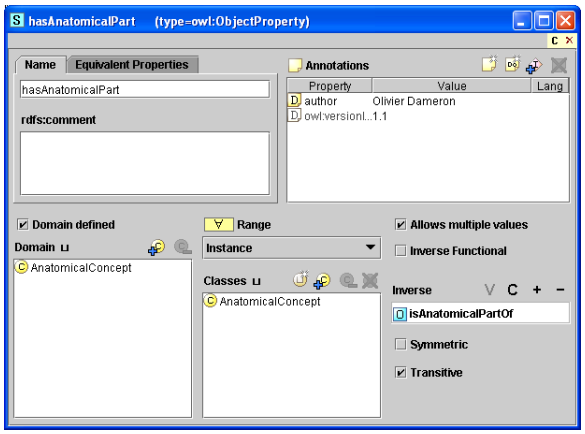

Figure 2: An OWL property form in Protégé.

The available choices in the *Range* drop-down box depend on whether the property is a *datatype property* with primitive values, or an *object property* with references to other classes. For datatype properties, Protégé supports enumerations of symbols (`owl:oneOf`), and all reasonable XML Schema datatypes, grouped into booleans, floats, integers, and string types. For example, the datatype property *`hasMeasuredDepth`* can only take floats as values. Object properties can store references to individuals or classes from the ontology. For example, the object property *`hasAnatomicalPart`* can only take instances of `AnatomicalConcept` as values.

Depending on whether a property is an object or a datatype property, Protégé provides widgets for other property characteristics, such as whether the property is symmetric or transitive. Symmetric properties describe bidirectional relationships (i.e., if A is related to B via property R_s, then B is also related to A). For example, the contiguity relationship is symmetric. A property R_t is transitive if when A is related to B by R_t and B is related to C by R_t, then (A is also related to C by R_t). Part/whole relationships such as *`hasAnatomicalPart`* are usually considered to be transitive.

The *Domain* widget can be used to restrict a property's domain (i.e., the list of classes where the property can be used). Domain restrictions are optional and often left blank in OWL ontologies, because they may slow down some reasoning processes. If a property does not have a domain restriction, then it can be used for instances of any class.

Specialization

OWL has its theoretical foundation in description logic [1]. In description logic, a class is a set of individuals. The concept corresponding to the set of all individuals is usually called *Top* ($\top$), or *Thing*. Whenever the set of the individuals of a class B is a subset of the set of the individuals of a class A, B is said to be a *subclass* of A (noted $B \sqsubseteq A$). B is also said to be a kind of A. All classes are subconcepts of $\top$.

In other words, superclasses define *necessary* conditions for class membership. Conversely, subclasses define *sufficient* conditions for class membership. For example, being a frontal lobe is a necessary condition for being a left frontal lobe: in order to be an instance of `LeftFrontalLobe`, an individual has to be an instance of `FrontalLobe` (and most certainly has to fulfill other requirements). Conversely, being a left frontal lobe is a sufficient condition for being a frontal lobe: every instance of `LeftFrontalLobe` is also an instance of `FrontalLobe` (but there may be other instances of `FrontalLobe` that are not instances of `LeftFrontalLobe`).

It is really important to keep in mind that a subconcept is a subset of individuals. Indeed, it is a common mistake to mix specialization and composition hierarchies. However, defining `UpperLobeOfLung` as a subconcept of `Lung` is erroneous because a lobe of a lung is not a kind of lung, but a part of a lung. Correct subconcepts for lung could be `LeftLung` and `RightLung`.

The specialization principle also implies inheritance of the properties. For instance, if we say that every `Sulcus` has an *`averageDepth`* and that `CentralSulcus` is a subclass of `Sulcus`, then every `CentralSulcus` also has an *`averageDepth`*. Because subclasses are more specific than their superclasses, the range of a subclass may itself be a subclass of the range of the superclass. This is called *property restriction*. For example, we can say that every `Sulcus` has a side in the class `Side`, and that every `LeftSulcus` (subclass of `Sulcus`) has a side `LeftSide` (subclass of `Side`).

In Protégé, the tree widget of the OWLClasses tab is organized according to the subclass hierarchy. We can see that `owl:Thing` (which represents $\top$) is the root of the tree. Protégé users can browse, view, and edit the classes from the tree, create new subclasses, and

move classes easily with drag-and-drop. Direct superclasses are also listed in the Conditions widget, which is described next. The OWL Plugin also allows to navigate and edit ontologies according to other relationships between classes, in particular to visualize the part-of relationships that are so common in biomedical domains.

Logical Class Characteristics

The *Conditions* widget of the OWLClasses tab allows to fully take advantage of OWL's description logic support, and to express conditions on the classes based on property restrictions and other expressions. The syntax used for OWL expressions in Protégé is summarized in table 1.

The key point here is to understand that an expression involving a property and its range such as "∃ *property* Concept" or "∀ *property* Concept" represents a set of individuals, and therefore can be interpreted as a concept. For example, (∃ *hasPart* Lobe) is the set of all the individuals related to at least one instance of Lobe by the *hasPart* relationship (they could also be related to instances of other concepts). Conversely, (∀ *hasPart* Lobe) is the set of all the individuals which are exclusively related to instances of Lobe by the *hasPart* relationship (or which are related to nothing by this relationship). Similarly, the union and intersection of two sets are also sets and can be interpreted as classes. For example, (LeftAnatomicalPart ⊓ Gyrus) represents the set of all left anatomical parts that are at the same time gyri, and (LeftGyrus ⊔ RightGyrus) represents the set of individuals that are instances of either concept. The ¬ operator can be used to define a class of any individual except those from a given class. For instance, ¬LeftSide is the set of all the individuals that are not instance of LeftSide. Finally, OWL also allows to define a class by exhaustively enumerating its instances.

The logical symbols used by the Protégé OWL Plugin are widely used in the description logic community [1]. Their major advantage is that they allow to display even complex class expressions in a relatively compact form. As shown in Figure 3, Protégé provides a convenient expression editor with support for either mouse or keyboard editing. However, some domain experts, especially from rather non-technical domains such as biomedicine, may require some training before they get used to these symbols. For these users, Protégé provides an English prose explanations of an OWL expression when the mouse is moved over it. Our collaborators are also working on alternative editors which support a rather template-based editing metaphor. Protégé's generic form architecture allows to quickly assemble alternative editors into the environment.

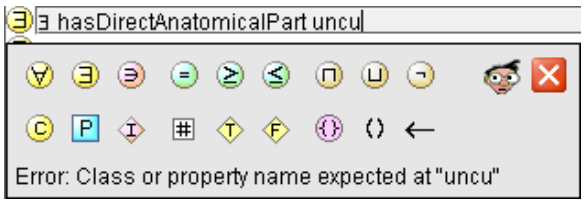

Figure 3: Protégé provides a comfortable editor for arbitrary OWL expressions.

The formal definitions of the OWL primitives can be exploited by reasoners. They compute the specialization relationships (inheritance) between the classes based on their logical definitions. This reasoning support has shown to be a very valuable feature during ontology design, particularly in biomedical domains ([5, 11]). Ontology designers can periodically invoke a reasoner to see whether the logical class definitions meet the expectations, and to make sure that no inconsistency arise.

Necessary conditions. As mentioned above, a necessary condition for an individual to be an instance of a class is to be an instance of all the superclasses of this class. In addition to saying that a class is a subclass of its superclasses, such as FrontalLobe is a subclass of Lobe, necessary conditions allow the specify the properties that the class has to fulfill. This is an important activity when building an ontology, because, we don't want to limit ourselves to saying that a frontal lobe is a kind of lobe; we also want to represent what is specific to the frontal lobe, as opposed to the other lobes. For example, the frontal lobe has to be delimited by the central sulcus, as well as by the lateral sulcus. Therefore, to the original condition FrontalLobe ⊑ Lobe, we can add the two following necessary conditions "FrontalLobe ⊑ (∃ *isDelimitedBy* CentralSulcus)" and "FrontalLobe ⊑ (∃ *isDelimitedBy* LateralSulcus)". These conditions can also hold for other concepts, but an individual that fails to fulfill these conditions cannot be an instance of FrontalLobe.

Necessary and sufficient conditions. Necessary conditions can be interpreted as subset-superset relationships between sets of individuals. Similarly, we may want to represent that two classes have exactly the same instances (they are mutual subclasses of the other). For example, as the left and the right frontal lobe are two kinds of frontal lobe, we have

OWL element	Symbol	Key	Example expression in Protégé
owl:allValuesFrom	∀	*	∀ *hasPart* Lobe
owl:someValuesFrom	∃	?	∃ *hasDirectAnatomicalPart* RectusGyrus
owl:hasValue	∋	$	*hasColor* ∋ yellow
owl:minCardinality	≥	>	*hasSide* ≥ 1 (at least one value)
owl:maxCardinality	≤	<	*hasSide* ≤ 2 (at most two values)
owl:cardinality	=	=	*hasSide* = 1 (exactly one value)
owl:intersectionOf	⊓	&	LeftAnatomicalConcept ⊓ Gyrus
owl:unionOf	⊔	|	LeftGyrus ⊔ RightGyrus
owl:complementOf	¬	!	¬LeftSide
owl:oneOf	{ ... }	{ }	{yellow green red}

Table 1: Protégé uses traditional description logic symbols to display OWL expressions. Property names such as *hasSide* appear in italics. A common naming convention is to use uppercase names such as Lobe to represent classes, while individuals like yellow should be written in lower case.

the following condition: (LeftFrontalLobe ⊔ RightFrontalLobe) ⊑ FrontalLobe. But we also want to say that every frontal lobe is either a left or a right frontal lobe. Therefore, we use a necessary and sufficient condition (LeftFrontalLobe ⊔ RightFrontalLobe) ≡ FrontalLobe, which basically says that if you have a frontal lobe, then it is either a left or a right one (⊒); and that if you have a left or a right frontal lobe, then it is a frontal lobe (⊑). Classes with necessary and sufficient conditions are called *defined* classes (represented by orange icons in Protégé), while classes with only necessary conditions are called *primitive* (yellow icons). The Conditions widget allows to edit either type of conditions, and to copy or move expressions between blocks.

The open world assumption. Description logic make the so-called *open world assumption*, that is what is not said denotes a lack of knowledge (whereas in other contexts such as databases, what is not said is assumed to be false). A direct consequence is that if we don't say explicitly that two classes such as LeftFrontalLobe and RightFrontalLobe are disjoint, then it is perfectly valid for them to have individuals in common. The *Disjoints* widget, in the lower right corner of the OWLClasses tab allows users to represent axioms to control this aspect.

Classification and Consistency Checking

One of the major strengths of description logic languages like OWL is their support for intelligent reasoning. In our context, *reasoning* means to infer new knowledge from the statements asserted by an ontology designer. *Reasoners* are tools that take an ontology and perform reasoning with it. The OWL Plugin can interact with any reasoner that supports the standard DIG interface, such as Racer [6]. Since these reasoners are separate tools we will not discuss their details in this paper, but focus on their application oriented utility. During ontology design, the most interesting reasoning capabilities from these tools are classification and consistency checking.

Classification. Classification is used to infer specialization relationships between classes from their formal definitions. Basically, a classifier takes a class hierarchy including the logical expressions, and then returns a new class hierarchy, which is logically equivalent to the input hierarchy. As illustrated in Figure 4, Protégé can display the classification results graphically. After the user has clicked the classify button, the system displays both the asserted and the inferred hierarchies, and highlights the differences between them.

For example, we defined LeftFrontalLobe as any frontal lobe located in the left hemisphere (LeftFrontalLobe ≡ (FrontalLobe ⊓ LeftAnatomicalConcept)). Therefore, it appears as a direct child of the last two concepts in the asserted hierarchy (Figure 4). Similarly, we also defined LeftLobe as any lobe located in the left hemisphere (LeftLobe ≡ (Lobe ⊓ LeftAnatomicalConcept)). Because the definition of LeftFrontalLobe doesn't mention LeftLobe, these two concepts don't appear to be related. However, after classification, the reasoner infers from FrontalLobe ⊑ Lobe that LeftFrontalLobe is also a subclass of Leftlobe. Note: we could as well have defined LeftFrontalLobe ≡ (FrontalLobe ⊓ LeftLobe), but then we wouldn't have known that it is also a LeftAnatomicalConcept until the reasoner have found out.

This reasoning capability associated with description logic is of particular importance because it allows the

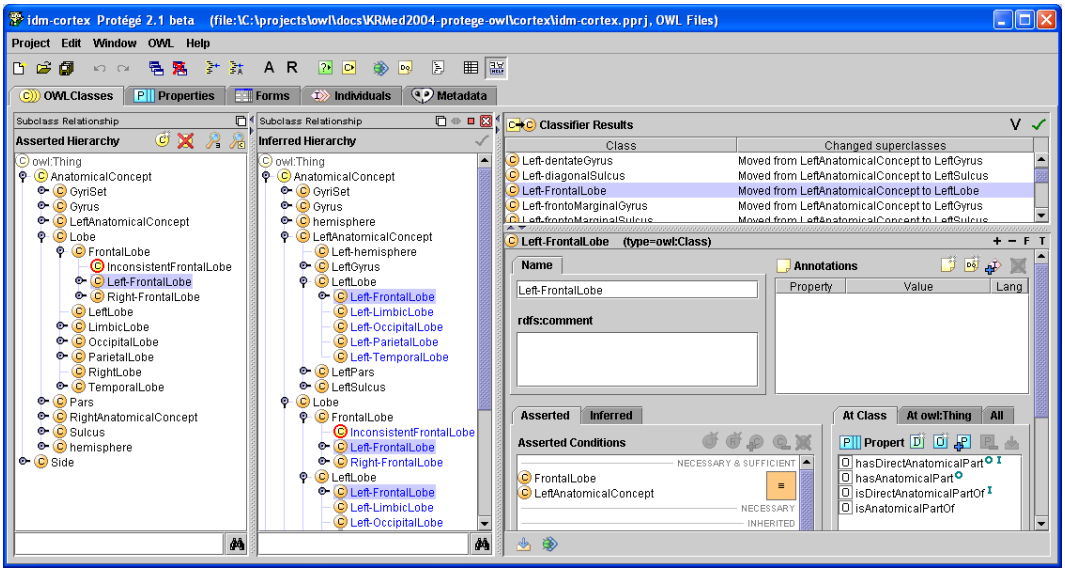

Figure 4: Protégé provides access to description logic classifiers and can display both the asserted and the inferred class relationships.

user to provide intensional definitions for the classes. The specialization relationships become consequences of these definitions, and allow constraints inheritance. Without reasoning capabilities, the approach of creating an ontology is more extensional. It would require to explicitly state every specialization relationships between the concepts (e.g., in the previous example between `LeftFrontalLobe` and `LeftLobe`). This support is especially valuable in the domain of biomedicine, with its deeply nested hierarchies and multi-relationships between almost every part of the anatomy [1, 12]. Using OWL, ontology designers could just add a new concept by describing its logical characteristics, and the classifier would automatically place it in its correct position. Furthermore, it would report the side-effects of adding a new class.

Consistency checking. In addition to providing automatic classification, reasoning capabilities can be exploited to detect logical inconsistencies within the ontology. We could introduce a class `InconsistentFrontalLobe`, which is both a `LeftFrontalLobe` and a `RightFrontalLobe`. Since the last two concepts are defined to be disjoint, the reasoner reports that no individual can be an instance of this class. Clearly, these consistency checks can help tremendously in the construction and maintenance of large biomedical terminologies [12].

OWL Full and OWL DL An important issue with reasoning in OWL is that many reasoners are not able to handle the full expressivity of OWL. The OWL specification distinguishes between OWL Full and OWL DL to indicate which language elements are typically tractable for reasoners. Ontologies that use OWL Full elements such as metaclasses cannot be classified. Protégé allows users to edit some OWL Full concepts and provides features to help convert the ontology into OWL DL when a classifier is to be used. However, since OWL Full ontologies can state anything about anything, Protégé does not support the complete OWL Full syntax.

LINKING BIOMEDICAL RESOURCES INTO THE SEMANTIC WEB

This section demonstrates how to use OWL to link biomedical resources into the Semantic Web. In our scenario, OWL ontologies provide the vocabulary for describing the contents of images and scientific articles.

In order to describe biomedical images, we have defined a small image ontology, which basically only defines a single class `Image`, and defines four properties for each image: the integer properties *`hasWidth`* and *`hasHeight`* provide the dimensions of the image, the property *`hasURI`* stores a reference to the image's location, and the property *`hasContents`* can link an `Image` to an OWL class, such as those defined in the brain cortex ontology. These content concepts can later be used by intelligent agents for search purposes. Protégé can now be used to create a new ontology `cortex-images.owl`, which imports the cortex

ontology and the images ontology. The new ontology basically contains instances of the `Image` class, and uses the classes from the cortex ontology as contents values. Whenever concepts are imported from another ontology, Protégé displays them with a prefix such as `cortex:`.

Protégé provides excellent support for the acquisition of instances. As illustrated in Figure 5, the OWL Plugin makes this functionality available through the *Individuals* tab. For each class in an ontology, Protégé generates forms with appropriate widgets to acquire instances of the class. The Individuals tab shows the classes, their instances, and a form for the selected instance. By default, this form will contain default widgets, such as a numeric text field for integer properties and a clickable list for object properties. For example, Protégé has selected a list widget with create, add and remove buttons for the *`hasContents`* property. However, for the *`hasURI`* property, the system has selected a simple text field widget, which is not optimized for displaying images.

Fortunately, Protégé provides a *Forms* tab, which can be used to customize the forms. The Forms tab allows users to move and resize the widgets, and to replace widgets with other suitable ones. In our example, we have replaced the default text field widget for *`hasURI`* with an image widget, so that a preview of the image can be shown below the URI. Protégé's open architecture allows users to add arbitrary Java components as widgets, if the catalogue of default widgets is not sufficient. With a little bit of programming, we could provide a widget that allows users to select an image, and then fills the values of width and height automatically.

After the instances/individuals have been edited, they can be exported onto a Web server, so that agents can find and process them. A simple search agent would crawl through multiple image repositories, and analyze the image ontologies using an OWL parsing library such as Jena[2]. Supplied with a search concept such as `FrontalLobe`, an agent could then retrieve and filter images by their semantic proximity. A very similar approach can be used to implement a repository of scientific articles.

DISCUSSION AND CONCLUSION

Our main goal in this paper was to introduce the Protégé OWL Plugin, and to show that it provides a promising platform for biomedical ontology and Semantic Web projects. The OWL Plugin pioneers user-friendly components for building and reasoning with description logic ontologies. While researchers from the description logic community have managed to create deeply studied maps of their theoretical terrain, we believe it is now time to put languages such as OWL into practice, and thus reveal the strengths and weaknesses of these languages for particular domains in everyday use. Some issues of how to handle description logic in the development of large clinical terminologies have already been discussed by others (e.g., [5, 12, 11]). However, more work is necessary, in particular in training biomedical domain experts to use the rich semantics of OWL.

Some of the advantages of OWL are already obvious. Descriptions logic rely on a well defined semantics which makes modeling not only the structure, but also the meaning of a domain possible. As opposed to other formalisms such as frames [9], description logic allow users to provide intensional definitions for the concepts. As a consequence, ontologies are more compact, less error-prone, and easier to maintain. The precise semantics of description logic makes it possible to perform automatic reasoning. The intensional definitions of the concepts can be exploited by classifiers. Therefore, when adding a new class, one doesn't have to worry anymore about putting it in the right place in the taxonomic hierarchy. Moreover, multiple inheritance is automatically detected and dealt with. Classifiers can detect any logical inconsistencies in a class definition, that would prevent it of having instances. Eventually, reasoners can infer the correct relationships when combining ontologies of related domains, or extending an ontology with context-specific features. This point favors the sharing of common semantic references and their reuse in various contexts. Therefore, we expect OWL to play a key role not only for the Semantic Web, but also for the evolution and sharing of biomedical knowledge.

A final note about other ontology modeling tools. Given the short history of the Semantic Web, there are few other tools available with OWL support. One of the most popular ontology editors beside Protégé is OilEd [2]. From the beginning on, OilEd has been optimized for reasoning with description logic, and has been successfully used for various biomedical ontology projects. However, OilEd's 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 support customized user interface widgets. Furthermore, it suffers from a rather complicating user interface for editing logical expressions. The developers of Protégé and the OilEd team have recently joined forces in a transatlantic project called CO-ODE, which leads to a growing number of extensions for the Protégé OWL

[2] `http://jena.sourceforge.net/`

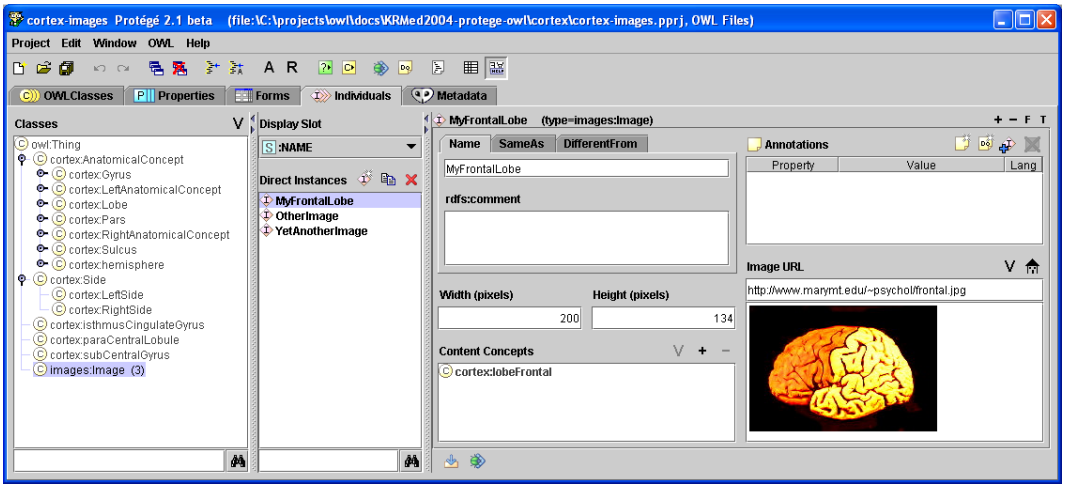

Figure 5: Protégé generates user interfaces to acquire individuals of ontology concepts. This can be used to annotate Web resources such as images for a clinical online repository.

Plugin. Many other groups from around the world are also developing Protégé plugins, including tools which can be used to edit OWL classes and relationships in a visual UML-style diagram. Other large-scale Protégé plugins are being optimized for the OWL Plugin. With its large and rapidly growing community of thousands of users, Protégé has the potential to maintain its position as one of the leading open-source ontology development environments for the Semantic Web.

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. Olivier Dameron is funded by INRIA. Additional support for this work came from the UK Joint Information Services Committee under the CO-ODE grant. Several colleagues and students at SMI were involved in the development of the OWL Plugin, in particular Ray Fergerson and Prashanth Ranganathan. Our partners from Alan Rector's team at the University of Manchester have made very valuable contributions.

References

[1] Franz Baader, Diego Calvanese, Deborah McGuineness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

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

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284:34–43, 2001.

[4] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu. The evolution of Protégé-2000: An environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[5] Jennifer Golbeck, Gilberto Fragoso, Frank Hartel, James Hendler, Bijan Parsia, and Jim Oberthaler. The national cancer institute's thesaurus and ontology. *Journal of Web Semantics*, 1(1), 12 2003.

[6] Volker Haarslev and Ralf Moeller. RACER user's guider and reference manual. `http://www.cs.concordia.ca/~faculty/haarslev/racer`, 2003.

[7] Holger Knublauch. An AI tool for the real world: Knowledge modeling with Protégé. JavaWorld, June 20, 2003.

[8] Holger Knublauch, Mark A. Musen, and Alan L. Rector. Editing description logics ontologies with the Protégé OWL plugin. In *International Workshop on Description Logics*, Whistler, BC, Canada, 2004.

[9] Natalya F. Noy, Ray W. Fergerson, and Mark A. 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.

[10] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson, and Mark A. Musen. Creating Semantic Web contents with Protégé-2000. *IEEE Intelligent Systems*, 2(16):60–71, 2001.

[11] Alan Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In *2nd International Conference on Knowledge Capture (K-CAP)*, Sanibel Island, FL, 2003.

[12] Alan L. Rector. Clinical terminology: Why is it so hard? *Methods Inf. Med.*, 4–5(38):239–52, 1999.

[13] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL Web Ontology Language Guide. `http://www.w3.org/TR/owl-guide/`, 2003.