

A Tool for Working with Web Ontologies

Aditya Kalyanpur, Bijan Parsia, and James Hendler

University of Maryland, MIND Lab, 8400 Baltimore Ave,
College Park MD 20742, USA

{aditya}@cs.umd.edu

{bparsia}@isr.umd.edu

{hendler}@cs.umd.edu

Abstract. The task of building an open and scalable ontology browsing and editing tool based on OWL, the first standardized Web-oriented ontology language, requires the rethinking of critical User Interface and ontological engineering issues. In this paper, we describe Swoop, a browser and editor specifically tailored to OWL ontologies. Taking a "Web view" of things has proven quite instructive and we discuss some insights into Web Ontologies that we gained through our experience with Swoop, including issues related to the display, navigation, editing and collaborative annotation of OWL ontological data.

1 Introduction

The Web Ontology Language, OWL [1] was approved in February of 2004 as a World Wide Web Consortium (W3C) Recommendation for the publication of ontologies on the World Wide Web – creating a standard language for the publication and exchange of ontological models on the Web. OWL reflects almost ten years of research, experimentation, and small scale deployment of Web ontologies and a number of certain features in its design were made explicitly to help realize the ideal of Web based ontologies, that is, of integrating knowledge representation with the open, global, and distributed hypermedia system of the Web, compatible with the principles of Web architecture design. In this paper we discuss some insights into supporting the use of Web Ontologies that we have gained in building Swoop, an ontology browser and editor, designed specifically for use with OWL and directly supporting the use of Web-based "cultural metaphors" – that is, based on the way people are used to interacting with documents and data in current Web applications.

2 A Web (Ontology) Browser

2.1 OWL

OWL is a standard for representing knowledge on the Web with a focus on both making these documents compatible with Web standards and on being useful for the modeling of knowledge using past research on ontologies and reasoning. OWL comes in three increasingly expressive sublanguages — OWL Lite, DL and Full. The Lite and DL species of OWL are based on description logics, i.e., decidable, class and property oriented subsets of first order logic. OWL Full follows RDF Schema in having a higher order syntax (although first order semantics) — OWL full does not enforce a strict separation of classes, properties, individuals, datatypes or data values. Any entity could be, for example, both a class and an individual. This design was motivated by the Web architecture dictum that “everything is a resource”, thus an individual, and from the general modeling consideration that the choice between whether to represent some aspect of a domain as a class or an individual is not always clear. In a world where people are trying to reuse vocabulary and map between concepts, it seems quite natural to be able to *express* the dual view of certain domain objects as either classes or individuals, and sometimes both.

One characteristic of “Webized” languages, especially Semantic Web languages, is the systematic prevalence of Universal Resource Indicators¹ (URIs) as names for most entities. In OWL, names for classes, properties, individuals, datatypes, etc. are URIs. URIs have a number of useful properties including:

1. For a number of URI schemes, notably http URIs, there is a well developed set of mechanisms for avoiding name collisions, most notably the domain name system (DNS).
2. These mechanisms, especially the DNS, interact with various internet protocols, notably HTTP, to make it very easy to publish and retrieve information associated with a URI.
3. URIs have various degrees of opacity. For example, HTTP imposes relatively few constraints on the semantics of the scheme specific part

¹ A URI is a generalization of the more common URL, roughly composed of a naming scheme or protocol indicator (http, ftp, mailto, etc.) a unique indicator (a domain name space name for http, a mail address for mailto) and a “fragment id” which is a hash mark followed by a set of characters – thus, for example, an owl class called “person” from an ontology on a University server might be named by the URI <http://www.thisuniversity.edu/OntologyLib/csonontology#person>.

of http URIs. The hierarchical structure seen in most http URIs can map directly into a file system (which is a very useful default behavior), but it can also map into queries on a relational database, the object structure of a long running process, or any other Web resource.

4. URIs can work well for end users, who have developed a lot of expertise with using URIs when browsing or authoring.

Web browsers are the ubiquitous way that people use URIs, and, even in authoring tools, the primary mental model people have of URIs is derived from their use in browsers. In designing Swoop, we took the Web browser as our User Interface (UI) paradigm, believing that URIs are central to the understanding and construction of Web Ontologies. We contrast this to other ontology editors such as Protégé [2], OilED [3], and OntoEdit [4] which either are, or were influenced by traditional KR development tools and applications, and do not reflect this “Webiness” in their UI design. In particular, they do not fully support the use of hypertext to drive the exploration and editing of ontologies.

2.2 Hypertextual Navigation

In a Web browser, there are two primary modalities for URIs: manifest and hidden. The address bar is the central mechanism for manifest URIs. URIs must be typed into the address bar, and are always visible there. Browser features such as history drop downs and the use of name completion mean that users need not remember or enter entire URIs, while the address bar requires and abets interaction with raw URIs. The most prominent hidden use of URIs is the hyperlink wherein the URI is address the target of a clickable (in most browsers) region of text (or an image). There are tight links between hidden and manifest URIs. The URIs hidden “in” hyperlinks appear in the address bar after one has followed a hyperlink or may be revealed by mousing over a hot region, retrieved by pop up menu commands (i.e., copy hyperlink), or, viewing the actual HTML source.²

The ecology of Web pages depends on the ease of access to URIs, both hidden (there is no hypertext without hyperlinks!) and manifest. Much Web browsing starts with URIs discovered in non-Web media, from email to billboards and buses. Writing Web pages requires, even in WYSIWIG HTML editors, familiarity with URIs and the ability to secure the right ones.

² Bookmarks are another example of hidden URIs, at least in their most common form. Browsers typically have many ways to review bookmarked URIs.

As the natural habitat of Web ontologies is the Web, Swoop allows the interactions with these using the UI metaphors prevalent on the Web. For loading ontologies, Swoop presents the familiar address bar and the URI for such an ontology can be secured by whatever means, email, google, or, perhaps one day, a billboard or bus.

2.3 Views

It is worth considering the level of detail that needs to be displayed while rendering Web ontological information. While an OWL entity is represented by its URI, it is characterized in a specific context by the axioms dealing with the entity in that context (the document or ontology). Moreover, on the Semantic Web, we expect OWL entities to be characterized by axioms in remote documents. That is, we expect OWL *documents* and OWL ontologies to use *Web* links. When rendering the related axioms or definition of an OWL entity we have take care that the appropriate information is directly presented in an intelligible manner, and that all the known information is naturally accessible. We consider various levels of detail at which information related to an entity can be displayed:

1. Its definition and related axioms (within a single ontology)
2. Axioms relating it to imported entities (from an external ontology)
3. Inferred information (not explicitly stated in the ontology but which is inferred from its definition using an OWL reasoner or otherwise)
4. Semantic consistency information (whether the concept is satisfiable or not, again using an OWL reasoner)
5. Provenance information (source location of a particular axiom, its author, creation date etc)
6. Entity annotations (human readable comments made on the entity)
7. Changes (a log of changes made to the entity definition)
8. Usage of an entity (references in other Semantic Web documents)

Thus, there is an array of entity related information that could be displayed as a single Web document that pertains to any OWL entity. Currently, Swoop supports all but the provenance information and usage (5,8) views listed above, making clear distinctions between the various view types displayed. For instance, inferred axioms are italicized, inconsistent classes have red icons, and changes pending are shown in green (see **Figs. 1 and 2**). The other two open some complex research issues that are being explored by our research group and others.

Orthogonal to the above levels of detail is the syntax (format) used to render the ontology. Currently on the Semantic Web, a wide range of

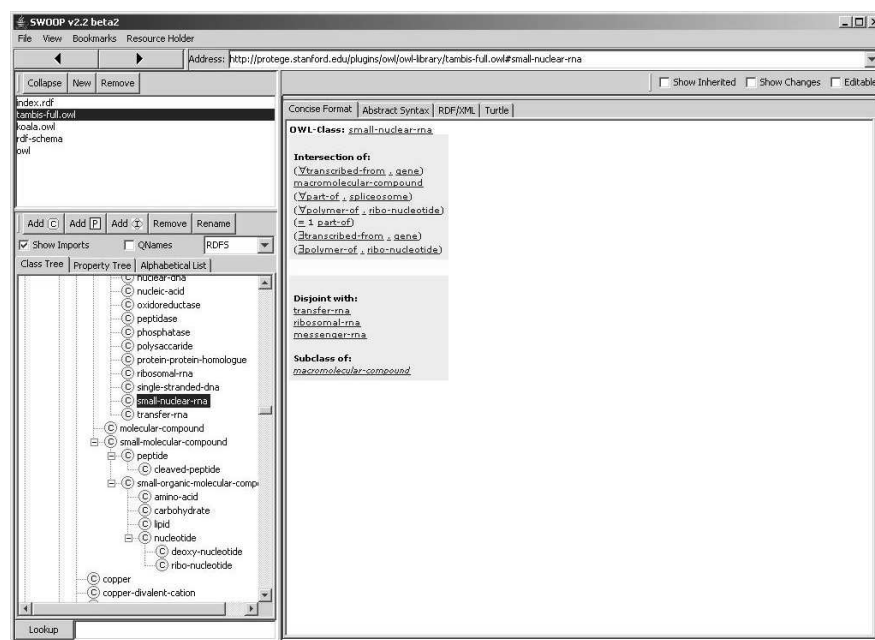


Fig. 1. Web-browser UI reflected in Swoop

OWL presentation syntaxes exist - the raw RDF/XML serialization, the more triple-oriented Turtle language[5], and the OWL Abstract Syntax [6] to name a few. It is important to support as many as possible of these different syntaxes while designing an open, Semantic Web ontology engineering environment. One reason for this is that people tend to have strong biases toward different notations and simply prefer to work in one or another. A second is that some other tool might only consume one particular syntax (with the RDF/XML syntax being the most typical), but that syntax might not be an easy or natural one for a particular user. A third is that it is important to support the “view source” effect, allowing cut and paste reuse into different tools including text editors, markup tools, or other semantic web tools. We have observed that the easy, direct data transformation between any two formats *feels* very powerful to the user, especially if they need to use more than one format for a particular task. The challenge here is that the formats should be treated as similarly as possible, i.e. any task that can be done in one format should be allowed in any other so that people can stick with the syntax they prefer for both browsing and editing.

Swoop uses a plug-in based mechanism for renderers. The architecture supports two types of renderers, a coarse grained type for viewing the ontology as a whole i.e. class/property tree, graphs, lists etc, and a fine grained type for viewing the description of a single ontological entity i.e an OWL class, property or individual. Other levels of granularity can be achieved by filtering out information from the above main types.

All of these formats use URIs (and various URI abbreviations) throughout. Swoop renders those URIs as hyperlinks allowing for essentially the same hypertext based navigation no matter what format is being used. Also, the layout of the ontology and entity renderers resembles a familiar frame-based website viewed through a Web browser (again, see **Fig. 1**). As shown in the figure, a navigation sidebar on the left contains the multiple ontology list, and class/property hierarchies for each ontology, and the center pane contains the various ontology/entity renderers for displaying the core content.

Currently, Swoop bundles in six renderers; two Ontology Renderers —Information and Species Validation; and four Entity Renderers — Concise Format, OWL Abstract Syntax, Turtle and RDF/XML. Besides these, there exists a class/property hierarchy renderer for each ontology, along with an alphabetical list of entities present in the ontology. Here we discuss only the Concise Format renderer, since its motivation, design and subsequent functionality is unique to Swoop.

The Concise Format entity renderer is a non-standard presentation syntax in Swoop (see **Fig. 2**). The idea here is to generate a “Web document” that displays all information related to a particular OWL entity concisely in a single pane. Items are divided into logical groups and rendered in a linear fashion. So taking an OWL Class for example, its OWL enumerations if any i.e. intersectionOf, unionOf and oneOf are listed in one group, while the OWL properties related to it (through domain or range) are listed in another group. Standard Description Logic (DL) operators are used whenever they occur in class expressions to make the representation more concise. Here again, all entity references are made hyperlinks using their URIs as the identifiers. Thus, clicking on an OWL entity link in a particular document causes the view to shift directly to the linked entity’s document. This is in keeping with the look and feel of traditional Web-like viewing and navigation of documents.

2.4 Editing

Editing OWL entities in a multiple ontology engineering environment can be challenging. Some of the issues that arise include:

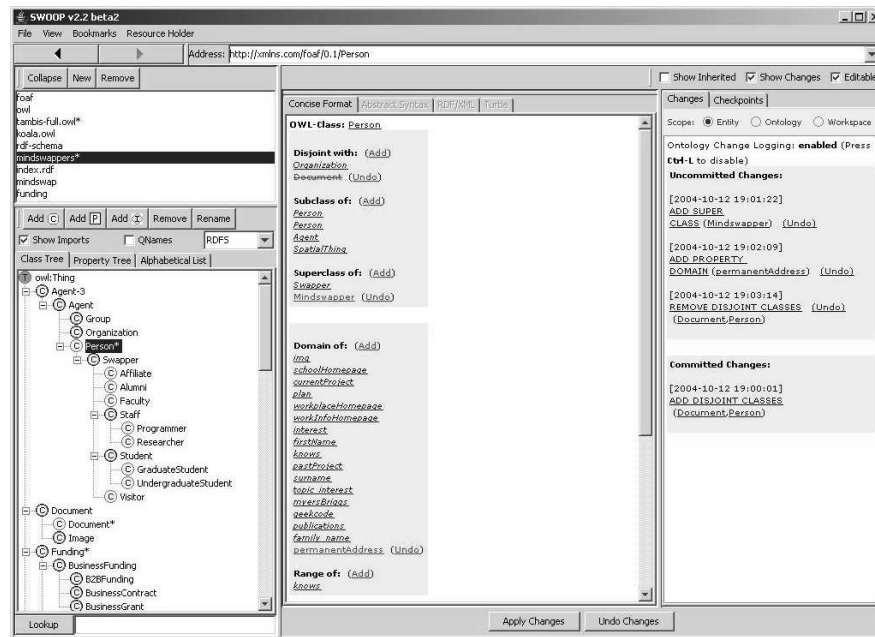


Fig. 2. Editing OWL entities in Swoop (Concise Format View)

1. The scope of a change (should editing be restricted to the local ontology alone or can the imported ontology be (directly or indirectly) altered as well)
2. The types of changes allowed (i.e. atomic vs. composite change strategies as discussed in [7])
3. The level at which changes are made (in the abstract representations or directly in the source code)
4. How to display the effects of changes before they are committed (direct vs. inferred effects on related entity definitions)
5. The degree of rollback possible (for how long changes can be "undo"ne).

Issues 1 and 2 are dealt with in detail in subsequent sections, we consider the remaining here. All ontology editing in Swoop is done inline with the renderer pane. This way context is maintained while editing a particular entity. Also, effects of change on any of the related entities can be easily observed (a single click away) by switching back and forth between the current entity and the related ones by following hyperlinks and use of the history buttons.

Swoop allows ontology editing either at the concise representation level or directly in the code (currently only RDF/XML code editing is

supported). There are some fundamental differences between editing in these two modes. For instance, in the concise format, all information related to an entity is displayed in a single pane. As noted earlier, this information is further subdivided into various logical groups, each of which can be edited separately. The changes enacted in this mode are identifiable, and hence can be recorded, and undone. Also, the axioms related to a particular entity may not be located in a single region of the code. Thus, directly editing all references of a single entity in RDF/XML (for example) might be cumbersome. Moreover, given the arbitrary manner in which the RDF/XML code can be edited, it is not easy to capture and record changes easily. On the other hand, direct code editing can be faster and certain changes can be made easily, for example renaming all references of a single class in the entire ontology can be done using the find/replace functionality of an editor. Given the need for both types of changes, Swoop supports both forms of editing.

Another important consideration in Swoop is the manner in which changes are effected. Swoop provides two options for this: either a change can be applied immediately (upon enacting it), or a set of proposed changes can be set aside and collectively committed at a later stage. While the former approach gives immediate results, the latter has numerous advantages. It speeds up alteration of large ontologies where enforcing multiple changes one at a time would take considerably more time. Additionally, it provides a composite change record which is especially useful for ontology versioning. Finally, it gives a basis for implementing Issue 4 noted above i.e. displaying change effects before they are committed.³

2.5 Searching, comparing, reusing

In a distributed Web ontology setting, numerous engineering tasks such as comparing entities with a view to understanding semantic differences, mapping entities to ensure semantic interoperability or simply reusing entities to prevent reinventing the wheel requires a search/browse process involving disparately located entities. The ontology engineering client can play a big role in making this process efficient.

We take inspiration from the hyperlink based search and cross-referencing utility present in a programming IDE such as Eclipse (<http://www.eclipse.org/>). All named entities in the code are identified and one can easily obtain (and jump directly to) useful related information such as all its references

³ Note: We plan to extend our ontology evolution/versioning framework based on related work such as [8]

in a specific project or working set. This practice is highly beneficial in understanding and debugging code.

During an extended search and browsing routine, the user of Swoop may come across numerous interesting results (OWL entities) that may need to be set aside and revisited. In Swoop we have a provision to store and compare OWL entities via a resource holder panel. Items can be added to this placeholder at any time and they remain static there until the user decides to remove or replace them at a later stage. Upon adding an entity, a time-stamped snapshot of it is saved (with hyperlinks and all), thus providing a reference point for future engineering tasks. These include, but are not limited to, tracking changes made to a particular entity; storing entities for reusing in another ontology; comparing differences in definitions of a set of entities; and determining semantic mappings between a specific pair of entities. (We are working to further improve the resource holder by adding automatic dynamic tracking for selected entities, color coding *diffs* between different entity definitions, and providing support for the editing of mapping terms, such as "owl:equivalentTo" between terms in different resource panes.)

2.6 Why not a website?

In principle, the entire Swoop interface and functionality could have been provided as a Website, or on top of a more full fledged Web browser such as Mozilla. Indeed, a very common first question we get when we show people Swoop is, "Why not do it as a website?" There are several examples of current website-based ontology tools (e.g. Ontosaurus [9], WebODE [10]), and new ones are being developed (e.g. pOWL - <http://powl.sourceforge.net>). However, we have found that using a standard web-based server-client architecture for ontology engineering suffers from being slow (esp. for large ontologies, and depending on network traffic), and cumbersome for maintaining consistency while editing (eg. trapping input errors, changing/deleting objects but reloading from browser cache etc). In addition, such tools can be difficult to extend to new functionalities via plug-in architectures (such as the one used in Swoop). In addition, most website based ontology editors use distinct HTML pages (perhaps dynamically generated) not just for each entity, but for each view of those entities. This indirection puts an uncomfortable distance between the user and the ontology itself. For these reasons, Swoop is developed as a separate Java application that attempts to provide the look and feel of a browser-based application, but with its specialized archi-

itecture designed to optimize OWL browsing and to be extensible via a plug-in architecture.

3 Multiple Ontologies: From Many, Many

OWL's web-based features open up the Web ontology engineering environment to multiple ontologies which can, and often do, refer to each other in a number of ways or share terms. This has ramifications for a number of aspects of ontology-editing that have been largely ignored in many earlier AI-based ontology tools. Swoop assumes the use of multiple ontologies and supports this use in a number of ways.

3.1 Display and Navigation

Being an open multiple ontology engineering environment, Swoop has a *no-holds-barred* approach for pulling in different Web ontologies into its model. Depending on the nature and context of the task being performed, ontologies are brought into Swoop seamlessly i.e no additional user intervention is required and the UI treats all ontologies similarly. For example, consider the scenario in which the user is browsing a particular OWL class, say A, in a Web ontology that has an OWL class B related to it by an axiom (say `rdfs:subClassOf`). Also, B is not defined in the same ontology, instead it has a separate physical Web location and has a number of URIs that share no common prefix with rest of A's URIs. Clicking on the class B hyperlink causes Swoop to directly load the external ontology referenced and select class B in it. Thus, no distinction is made in terms of UI between navigation across entities in a single ontology or those present in multiple ontologies. Also, the back and next buttons can be used to jump between OWL entities in different ontologies on a single click ensuring the familiar Web browser experience.

Besides the aforementioned scenario, there are various other situations which can drive Swoop to load more than one ontology. For example, multiple ontologies can be loaded at any point by entering their Web location URLs in the address bar. Alternately, the bookmarks feature can be used to store, categorize and reload ontologies directly. Finally, if a particular OWL ontology has imported ontologies (defined using `owl:imports`), loading it causes all its imports under transitive closure to be loaded into Swoop directly.

3.2 Living with Imports

The use of `owl:imports` reveals numerous open issues in Web ontology engineering. Two interrelated issues are considered here — UI issues in distinguishing between the definitions and semantics of imported OWL axioms; and editing support for axioms defined in the importing ontology.

Consider the case when an OWL class A is related by an axiom (say `owl:disjointWith`) to another class B. Suppose A and B have been defined in different ontologies, OA and OB respectively, and moreover, OA imports OB. (In OWL, an entity reference is defined in an ontology using `rdf:ID` and it can be further referenced in the same or any other ontology using `rdf:about` - thus allowing cross referencing of terms between ontologies.)

Now, the `owl:disjointWith` axiom can be defined in either ontology OA or ontology OB (or both!). Either way, the semantics of `owl:imports`, and the fact that OA imports OB, ensures the axiom is present in ontology OA. Yet, it is important to display to the user the exact source of axiom definition. This is especially important when the user wishes to *delete* this axiom. Obviously, the axiom cannot be deleted in the importing ontology, instead, the user must delete the axiom at the location at which it is originally defined i.e. imported ontology. Hence, in our case, if the axiom is defined in OB, even though it is displayed in OA as well, it can only be deleted in OB. Swoop needs to make these distinctions since it does viewing and editing axioms inline. Currently, this is accomplished by italicizing all imported axioms (but if an axiom is also local, that overrides).

Also, given that we use the URI of a class as its identifier in a hyperlink, there is an ambiguity of a URI when the class is referenced in different ontologies in terms of what class definition needs to be displayed when the hyperlink is clicked. So consider the above case involving classes A and B but here the `owl:disjointWith` axiom is present in OA and not OB. Now, if the user is viewing the axiomatic definitions of class A and clicks on the hyperlink corresponding to class B, there are two possibilities:

1. Swoop jumps to the class definition B in ontology OB (imported ontology) and here the `disjointWith` axiom is neither defined nor displayed
2. Swoop jumps to the class definition B in ontology OA itself (importing ontology) and here all imported axioms from OB are displayed along with the `owl:disjointWith` axiom

Note how the two views hold different semantics and rightly so, reiterating the point that the meaning of an OWL entity is defined in a specific context (ontology). To solve the URI ambiguity problem, Swoop provides labels next to the hyperlinks as an indicator to the jump location.

3.3 Beyond Imports?

Current research makes it clear that owl:imports is not the last word in combining (or referencing) Web based ontologies (and, in fact, problems with the use of this mechanism were pointed out as part of the OWL documents as an important area for future standardization). Recent work, for example, has been looking at using concepts from foreign ontologies without resorting to the all or nothing approach that owl:imports demands ([11], [12], [13]). We have discovered in Swoop that the problem of “where to go” when following a URI in an OWL document is not unique to owl:imports and arises in many different contexts during the editing of multiple, linked ontologies. Different collections of axioms seem to define (or characterize) different concepts. The RDF(S)/OWL Full view of concepts (or properties) as *entities* which may have varying definitions (and extensions) associated with them in different contexts — even in situations where there is no *disagreement* but mere normal use — is helpful, especially when coupled with some explicit identification mechanism for various definitions. In our work we have observed that the OWL Full view is *more* helpful at the Web infrastructure level than, as far as we can currently see, at the logic level. Classes as instances can be a useful Ontological modeling tool [14], but it might be that in the Semantic Web context, much of their value lies outside their use in *characterizing* a domain. For this reason, Swoop supports OWL Full, and the concise view displays both the class and instance properties of an entity in the same panel. However, these are separated visually to allow the user to more easily identify cases where this occurs.

4 Annotations

When browsing or building ontologies that live on the Web, it is almost as important to have information *about* the ontologies as it is to have the ontologies themselves. OWL allows for the associating of variously structured information with its core entities, e.g., classes and properties. Swoop supports the editing and display of textual or HTML formatted comments, and of photos and other multimedia (both via HTML and independently) as part of ontologies (see **Fig. 3**). Since OWL ontologies can

reference and import other ontologies, one can separate annotations about ontologies from the core ontologies themselves. The Annotea framework [15] takes this idea and provides both a specific RDF based, extensible annotation vocabulary, and a protocol for publishing and finding out-of-band annotations. Swoop uses the Annotea framework as the basis of collaborative ontology development.

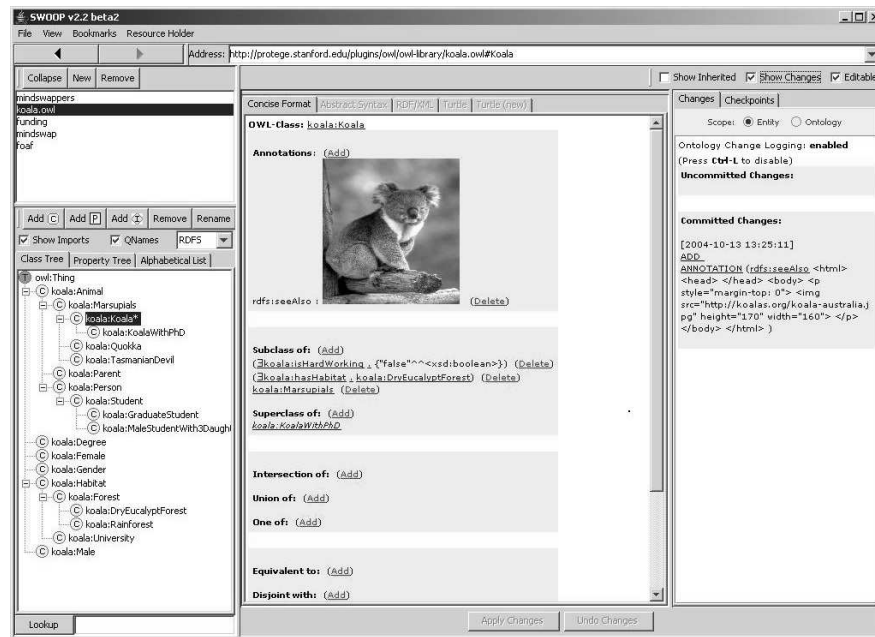


Fig. 3. Annotating OWL Entities - "Prototypical Illustration" of Classes

Annotea support in Swoop is provided via a simple plug in whose implementation is based on the standard W3C Annotea protocols [16] and uses the default Annotea RDF schema to specify annotations. Any public Annotea Server can then be used to publish and distribute the annotations created in Swoop. The default annotation types (comment, advice, example, etc) seem an adequate base for human oriented ontology annotations. One extension we have begun experimenting with is "PrototypicalIllustration", that is, a photo or drawing that represents a typical or canonical instance of the class.

4.1 Change Annotations

We have extended the Annotea Schema with the addition of an OWL ontology for a new class of annotations — ontology changes (similar to [17]). The “Change” annotation defined by the Annotea project was designed to indicate a proposed change to the annotated document, with the proposal described in HTML-marked-up natural language. In our extended ontology, change individuals correspond to specific, undoable changes made in Swoop during editing.

Swoop uses the OWL API [7] to model ontologies and their associated entities, benefiting from its extensive and clean support for changes. The OWL API separates the representation of changes from the application of changes. Each possible change type has a corresponding Java class in the API, which are subsequently applied to the ontology (essentially, the Command design pattern). These classes allow for the rich representation changes, including metadata about the changes.

The Swoop change annotations can be published and retrieved by Annotea servers, or any other annotation distribution mechanism. The retrieved annotations can then be browsed, filtered, endorsed, recommended, and selectively accepted. It is thus possible to define “virtual versions” of an ontology, by specifying a base ontology and a set of changes to apply to it. This is a fairly new addition to Swoop, and we are just beginning to explore the implications of change tracking couple with annotations for the development of large, curated ontologies by collaborative groups of scientists or other ontology definers.

5 Conclusion

We have built a Web (Ontology) browser and editor - Swoop, which takes the standard Web browser as the UI paradigm, believing that URIs are central to the understanding and construction of Semantic Web Ontologies. The familiar look and feel of a browser emphasized by the address bar and history buttons, navigation side bar, bookmarks, hypertextual navigation etc are all supported for web ontologies, corresponding with the mental model people have of URI-based web tools based on their current Web browsers.

All design decisions are in keeping with the OWL nature and specifications. Thus, multiple ontologies are supported easily, various OWL presentation syntaxes are used to render ontologies, and an OWL reasoner can be integrated for consistency checking. A key point in our work is that

the hypermedia basis of the UI is exposed in virtually every aspect of ontology engineering — easy navigation of OWL entities, comparing and editing related entities, search and cross referencing, multimedia support for annotation, etc. — thus allowing the Swoop user to take advantage of the Web-based features of OWL significantly more easily than the user of other ontology-editing tools.

In this paper, we discuss some of the key issues that our work in Swoop has identified as being important in Web Ontology tools. Topics we are currently exploring, not yet implemented in Swoop, are dealing with the ad hoc modification of ontologies by one or more users working on the ontology over time. These are issues exploring the editing of imported ontology data, and the use of annotated ontology change sets for ontology versioning as described above. Currently, we have preliminary solutions for these issues implemented in Swoop, but we are investigating alternate approaches that may be more powerful and better integrated with emerging Web standards. For example, one such approach is the use of the XPointer framework [18] to enable efficient syntactic filtering of ontological code, in order to reduce ontology modification time and effort.

References

1. Dean, M., Schreiber, G.: OWL Web Ontology Language Reference W3C Recommendation. <http://www.w3.org/tr/owl-ref/>. (2004)
2. Noy, N., Sintek, M., Decker, S., Crubezy, M., Fergerson, R., Musen, M.: Creating semantic web contents with Protégé-2000. *IEEE Intelligent Systems* (2001)
3. Bechhofer, S., Horrocks, I., Goble, C., Stevens, R.: OilEd: a reason-able ontology editor for the Semantic Web. *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence* (2001)
4. Sure, Y., Erdmann, M., Angele, J., Staff, S., Studer, R., Wenke, D.: OntoEdit: Collaborative ontology development for the Semantic Web. *Proceedings of the International Semantic Web Conference (ISWC)* (2002)
5. Beckett, D.: Turtle — Terse RDF Triple Language. <http://www.ildt.bris.ac.uk/discovery/2004/01/turtle/>. (2004)
6. Patel-Schneider, P., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax W3C Recommendation. <http://www.w3.org/tr/2004/rec-owl-semantics-20040210/>. (2004)
7. Bechhofer, S., Lord, P., Volz, R.: Cooking the semantic web with the owl api. *Proceedings of the International Semantic Web Conference* (2003)
8. Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-driven ontology evolution management. *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web* (2002)
9. Farquhar, A., Fickas, R., Rice, J.: The Ontolingua server: A tool for collaborative ontology construction. *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge Based System Workshop (KAW95)* (1996)

10. Arpírez, J., Corcho, O., Fernández-López, M., Gómez-Pérez, A.: WebODE: a scalable ontological engineering workbench. First International Conference on Knowledge Capture (K-CAP) (2001)
11. Borgida, A., Serafini, L.: Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics* (2003)
12. Kutz, O., Lutz, C., Wolter, F., Zakharyashev, M.: E-connections of description logics. *Proceedings of the 2003 International Workshop on Description Logics* (2003)
13. Cuenca-Grau, B., Parsia, B., Sirin, E.: Working with multiple ontologies on the semantic web. To appear in *Proceedings of the Third International Semantic Web Conference (ISWC)* (2004)
14. Noy, N.: Representing Classes As Property Values on the Semantic Web, W3C Working Draft. <http://www.w3.org/tr/2004/wd-swbp-classes-as-values-20040721/>. (2004)
15. Kahan, J., Koivunen, M.R., Prud'Hommeaux, E., Swick, R.: Annotea: An open RDF infrastructure for shared web annotations. *Proc. of the WWW10 International Conference* (2001)
16. Swick, R., Prud'Hommeaux, E., Koivunen, M.R., Kahan, J.: Annotea protocols. <http://www.w3.org/2001/Annotea/User/Protocol.html> (2001)
17. Klein, M., Noy, N.: A component-based framework for ontology evolution. *Workshop on Ontologies and Distributed Systems at IJCAI* (2003)
18. DeRose, S., Maler, E., Daniel Jr, R.: XPointer xpointer() Scheme. W3C Working Draft . <http://www.w3.org/tr/xptr-xpointer/>. (2002)