OilEd: a Reason-able Ontology Editor for the Semantic Web

Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens

Information Management Group, Department of Computer Science,
University of Manchester,
Oxford Road, Manchester M13 9PL, UK
{seanb,horrocks,carole,stevensr}@cs.man.ac.uk,
http://img.cs.man.ac.uk

Abstract. Ontologies will play a pivotal rôle in the "Semantic Web", where they will provide a source of precisely defined terms that can be communicated across people and applications. OilEd, is an ontology editor that has an easy to use frame interface, yet at the same time allows users to exploit the full power of an expressive web ontology language (OIL). OilEd uses reasoning to support ontology design, facilitating the development of ontologies that are both more detailed and more accurate.

1 Introduction

Ontologies have become an increasingly important research topic. This is a result both of their usefulness in a range of application domains [1–3], and of the pivotal rôle that they are set to play in the development of the *Semantic Web*

The Semantic Web vision, as articulated by Tim Berners-Lee [4], is of a Web in which resources are accessible not only to humans, but also to automated processes, e.g., automated "agents" roaming the web performing useful tasks such as improved search (in terms of precision) and resource discovery, information brokering and information filtering. The automation of tasks depends on elevating the status of the web from machine-readable to something we might call machine-understandable. The key idea is to have data on the web defined and linked in such a way that its meaning is explicitly interpretable by software processes rather than just being implicitly interpretable by humans.

To realise this vision, it will be necessary to annotate web resources with *metadata* (i.e., data describing their content/functionality). Standardisation proposals for annotation languages have already been submitted to the World Wide Web Consortium (W3C), in particular RDF (Resource Description Framework) and RDF Schema (see [5] for a discussion of the rôles of these languages and of XML/XML Schema). However, such annotations will be of limited value to automated processes unless they share a common understanding as to their meaning. Ontologies, can help to meet this requirement by providing a "representation of a shared conceptualisation of a particular domain" that can be communicated across people and applications [6].

RDF Schema (RDFS) itself is already recognisable as an ontology/knowledge representation language: it talks about classes and properties (binary relations), range and domain constraints (on properties), and subclass and subproperty (subsumption) relations. However, RDFS is a relatively primitive language (the above is an almost complete description of its functionality), and more expressive power would clearly be necessary/desirable in order to describe resources in sufficient detail. Moreover, such descriptions should be amenable to *automated reasoning* if they are to be used effectively by automated processes.

These considerations have led to the development of OIL [7], an ontology language that extends RDFS with a much richer set of modelling primitives. A similar RDFS based web ontology language called DAML was been developed as part of the DARPA DAML project [8] and the two languages have now been merged under the name DAML+OIL[1]. OIL has a frame-like syntax, which facilitates tool building, yet can be mapped onto an expressive description logic (DL), which facilitates the provision of reasoning services. OilEd is an ontology editing tool for OIL (and DAML+OIL) that exploits both these features in order to provide a familiar and intuitive style of user interface with the added benefit of reasoning support. Its main novelty lies in the extension of the frame editor paradigm to deal with a very expressive language, and the use of a highly optimised DL reasoning engine to provide sound and complete yet still empirically tractable reasoning services.

Reasoning with terms from deployed ontologies will be important for the Semantic Web, but reasoning support is also extremely valuable at the ontology design phase, where it can be used to detect logically inconsistent classes and to discover implicit subclass relations. This encourages a more descriptive approach to ontology design, with the reasoner being used to infer part of the subsumption lattice (see the case study presented in Section 4); the resulting ontologies contain fewer errors, yet provide more detailed descriptions that can be exploited by automated processes in the Semantic Web. Finally, reasoning is of particular benefit when ontologies are large and/or multiply authored, and also facilitates ontology sharing, merging and integration [9]; considerations that will be particularly important in the distributed web environment.

2 Oil and DAML+OIL

The development of OIL resulted from efforts to combine the best features of frame and DL based knowledge representation systems, while at the same time maximising compatibility with emerging web standards. The intention was to design a language that was intuitive to human users, and yet provided adequate expressive power for realistic applications (many early DLs failed on this second count—see [10]).

The resulting language combines a familiar frame like syntax (derived in part from the OKBC-lite knowledge model [11]), with the power and flexibility

[1] see `http://www.daml.org`

of a DL (i.e., boolean connectives, unlimited nesting of class elements, transitive and inverse slots, general axioms, etc.). The language is defined as an extension of RDFS, thereby making OIL ontologies (partially) accessible to any "RDFS-aware" application.

The frame syntax is less daunting to ontologists/domain experts than a DL style syntax, and it facilitates a modelling style in which ontologies start out simple (in terms of their descriptive content) and are gradually extended, both as the design itself is refined and as users become more familiar with the language's advanced features (see Section 4). The frame paradigm also facilitates the construction and adaption of tools, e.g., the OntoEdit and Protégé editors and the Chimaera integration tool are all being adapted to use OIL/DAML+OIL [12, 13, 9].

On the other hand, basing the language on an underlying mapping to a very expressive DL ($\mathcal{SHQ}$) provides a well defined semantics and a clear understanding of its formal properties, in particular that the class subsumption/satisfiability problem is decidable and has worst case ExpTime complexity [14]. The mapping also provides a mechanism for the provision of practical reasoning services by exploiting implemented DL systems, e.g., the FaCT system [15].

OIL extends standard frame languages in a number of directions. One of the key ideas is that an anonymous class description, or even boolean combinations of class descriptions, can occur anywhere that a class name would ordinarily be used, e.g., in slot constraints and in the list of superclasses. For example, in Figure 1 (which uses OIL's "human readable" presentation syntax rather than the more verbose RDFS serialisation), a herbivore is described as an animal that eats only plants or part-of plants. Points to note are that universally quantified (value-type) and existentially quantified (has-value) slot constraints are clearly differentiated, and that the constraint on the eats slot is a disjunction, one of whose components is an anonymous class description (in this case, just a single slot constraint). In addition, it is asserted that the part-of slot is transitive, and that its inverse is the slot has-part. Further details of the language will be given in Section 3, and a complete specification can be found in [7].

```
slot-def part-of
  subslot-of structural-relation
  inverse has-part
  properties transitive
class-def defined herbivore
  subclass-of animal
  slot-constraint eats
    value-type plant OR
      slot-constraint part-of has-value plant
```

Fig. 1. OIL language example

3 OilEd

OilEd is a simple ontology editor that supports the construction of OIL-based ontologies. The basic design has been heavily influenced by similar tools such as Protégé [13] and OntoEdit [12], but OilEd extends these approaches in a number of ways, notably through an extension of expressive power and the use of a reasoner.

However, OilEd is not intended as a replacement for such tools—the current implementation of OilEd is intended primarily as a prototype to test and demonstrate novel ideas, and compromises have been made in the design and implementation. For example, the tool does not provide key functionality for collaborative ontology development such as versioning, integration and merging of ontologies. Similarly, the powerful tailorability and knowledge acquisition aspects of tools such as Protégé have been ignored completely. Rather, the design has concentrated on demonstrating how the frame paradigm can be extended to deal with a more expressive modelling language, and how reasoning can be used to support the design and maintenance of ontologies.

3.1 OilEd Functionality

Basic functionality allows the definition and description of classes, slots, individuals and axioms within an ontology. In general, editing functions are provided through graphical means—mouse driven drop down menus, toolbars and buttons. We will not provide a detailed description of the graphical user interface here, as it is relatively standard (see Figure 2, which provides a screen shot of the editors class definition panel). Instead, we will discuss the novel functionality offered by the tool.

Frame Descriptions The central component used throughout OilEd is the notion of a *frame description*. This consists of a collection of superclasses along with a list of slot constraints. This is similar to other frame systems. Where OilEd differs, however, is that wherever a class name can appear, a recursively defined, anonymous frame description can be used. In addition, arbitrary boolean combinations of frames or classes (using **and**, **or** and **not**) can also appear. This is in contrast to conventional frame systems, where in general, slot constraints and superclasses must be class names.

As well as being able to assert individuals as slot fillers, several types of constraints on slot fillers can be asserted (these kinds of constraint are sometimes called *facets*). These include *value-type* restrictions (all fillers must be of a particular class), *has-value* restrictions (there must be at least one filler of a particular class), and explicit *cardinality* restrictions (e.g., at most three fillers of a given class). Each constraint has a clearly defined meaning, removing the confusion present in some frame systems, where, for example, it is not always clear whether the semantics of a slot-constraint should be interpreted as a universal or existential quantification.

Class Definitions A class definition specifies the class name, along with an optional frame description (see above) and a specification of whether the class

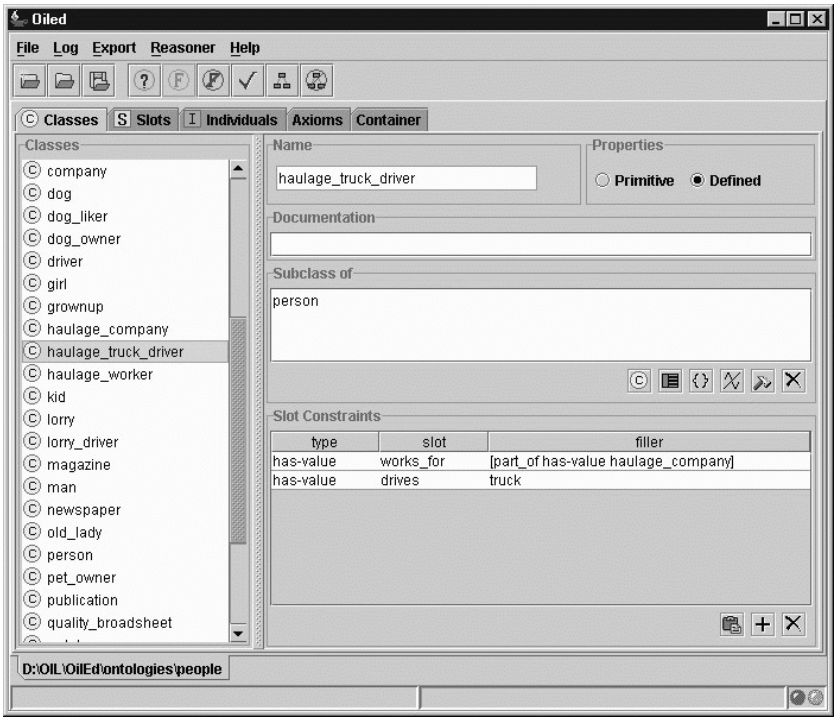

Fig. 2. OilEd Class Panel

is *defined* or *primitive*. If defined, the class is taken to be equivalent to the given description (necessary and sufficient conditions). If primitive, the class is taken to be an explicit subclass of the given description (necessary conditions). In the specification of the OIL language, classes can have multiple definitions. In OilEd, this is disallowed for implementation reasons. Instead classes must have a single definition, but the same effect can be achieved through the use of *equivalence* axioms as discussed below. Ontologies using multiple definitions can be read by the tool. The first definition encountered will be used as the class definition, with any subsequent definitions being translated to the appropriate axioms.

Slot Definitions A slot definition gives the name of the slot and allows additional properties of the slot to be asserted, e.g., the names of any *superslots* or *inverses*. If r is a superslot of s, then any two objects related via s must also be related via r (i.e., $\mathsf{s}(a,b) \rightarrow \mathsf{r}(a,b)$); if r is an inverse of s, then a is related to b via s iff b is related to a via r (i.e., $\mathsf{s}(a,b) \leftrightarrow \mathsf{r}(b,a)$). Domain and range restrictions on a slot can also be specified. For example, we can constrain the relationship parent to have both domain and range person, asserting that only persons can have, and be, parents. As with class descriptions, the domain and range restrictions can be arbitrary class expressions such as anonymous frames or boolean combinations of classes or frames, again extending the expressivity of traditional

frame editors. Note that in this context, the domain and range restrictions are *global*, and apply to every occurrence of the slot, whether explicit or implicit.

A slot r can also be asserted to be transitive (i.e., $\mathsf{r}(a,b)$ and $\mathsf{r}(b,c) \rightarrow \mathsf{r}(a,c)$), functional (i.e., $\mathsf{r}(a,b)$ and $\mathsf{r}(a,c) \rightarrow b = c$) or symmetric (i.e., $\mathsf{r}(a,b) \rightarrow \mathsf{r}(b,a)$).

All assertions made about slots are used by the reasoner, and may induce hierarchical relationships between classes, e.g., as a result of domain and range restrictions.

Axioms Another area where the expressive power of OIL/OilEd exceeds that of traditional frame languages/editors is in the kinds of *axiom* that can be used to assert facts about classes and their relationships. As well as standard class definitions (which are really a restricted form of subsumption/equivalence axiom), OilEd axioms can also be used to assert the *disjointness* or *equivalence* of classes (with the expected semantics) along with *coverings*. A covering asserts that every instance of the covered class must also be an instance of at least one of the covering classes. In addition, coverings can be said to be *disjoint*, in which case every instance of the covered class must be an instance of exactly one of the covering classes.

Again, these axioms are not restricted to class names, but can involve arbitrary class expressions (anonymous frames or boolean combinations). This is a very powerful feature, and is one of the main reasons for the high complexity of the underlying decision problem.

Individuals Limited functionality is provided to support the introduction and description of individuals—the intention within OilEd is that such individuals are for use within class descriptions, rather than supporting the production of large existential knowledge bases (it is supposed that RDF/RDFS will be used directly for this purpose). As an example, we may wish to define the class of Italians as being all those Persons who were born in Italy, where Italy is not a class but an individual.

As the FaCT system does not support reasoning with individuals, they are treated (for reasoning purposes) as disjoint primitive classes. This is not an ideal solution as it does lead to some inferences being lost, in particular those resulting from the interaction between individuals and maximum cardinality constraints. E.g., it would not be possible to infer that Persons who are citizens of Italy, and of no other Country, are citizens of at most one Country. Work is currently underway to extend the FaCT reasoner to deal explicitly with such individuals, so that complete inference can be provided.

Concrete Datatypes Concrete datatypes (string and integers), along with expressions concerning concrete datatypes (such as min, max or ranges) can also be used within class descriptions. However, the FaCT reasoner does not support reasoning over concrete datatypes, and at present OilEd simply ignores concrete datatype restrictions when reasoning about ontologies. The theory underyling concrete datatypes is, however, well understood [16], and work is also in progress to extend the FaCT reasoner with support for concrete datatypes.

The latest DAML+OIL language release uses the XML Schema type system for the definition of data types. These are not fully supported in our current version of OilEd.

3.2 Reasoning

In addition to the extended expressivity discussed above, OilEd's principal novelty is in its use of reasoning to check class consistency and infer subsumption relationships. Reasoning services are currently provided by the FaCT system, but in principal any reasoner with the appropriate functionality/connectivity could be used.

FaCT is a DL classifier that offers sound and complete reasoning (satisfiability, subsumption and classification) for two DLs: $\mathcal{SHF}$ and $\mathcal{SHQ}$. FaCT's most interesting features are its expressive logic (in particular the $\mathcal{SHQ}$ reasoner), its optimised tableaux implementation (which has now become the standard for DL systems), and its CORBA based client-server architecture [15].

The $\mathcal{SHQ}$ language can completely capture OIL ontologies, with the exception of two recently added features: concrete datatypes (strings, numbers, etc.) and named individuals in class descriptions. As mentioned above, individuals can be dealt with by treating them as pairwise disjoint atomic classes (although with some loss of inferential power), while extending FaCT to deal with OIL's concrete datatypes should be relatively straightforward.

FaCT's optimisations are specifically aimed at improving the system's performance when classifying realistic ontologies. These optimisations lead to performance improvements of several orders of magnitude when compared with older DL and modal logic reasoners, and make the use of reasoning support feasable in spite of the discouraging worst case complexity of the underlying decision problem (ExpTime). The performance improvement is often so great that it is impossible to measure precisely as unoptimised systems are virtually nonterminating with ontologies that FaCT is easily able to deal with [15]. Taking a large medical terminology ontology as an example [17], FaCT is able to check the consistency of all 2,740 classes and determine the complete class hierarchy in about 45 seconds of (700MHz Pentium III) CPU time; unoptimised systems have been run for several weeks without their completing even a single class consistency test.

In the current version of OilEd, reasoning is performed on a "single-shot" basis, i.e., at some suitable point the user connects to the reasoner and requests verification of the ontology. Connection is via FaCT's CORBA based client-server interface, which has the advantage that FaCT servers(s) can be running either locally or remotely, and can provide a service to many OilEd users. Moreover, the FaCT system has reasoning engines for both $\mathcal{SHQ}$ and $\mathcal{SHF}$ knowledge bases, and if both services are available the user can choose to connect to the faster $\mathcal{SHF}$ reasoner to verify an ontology that does not include either inverse slots or cardinality constraints. The current implementation simply informs the user if this is appropriate; future enhancements will include automatic selection of an appropriate reasoning service.

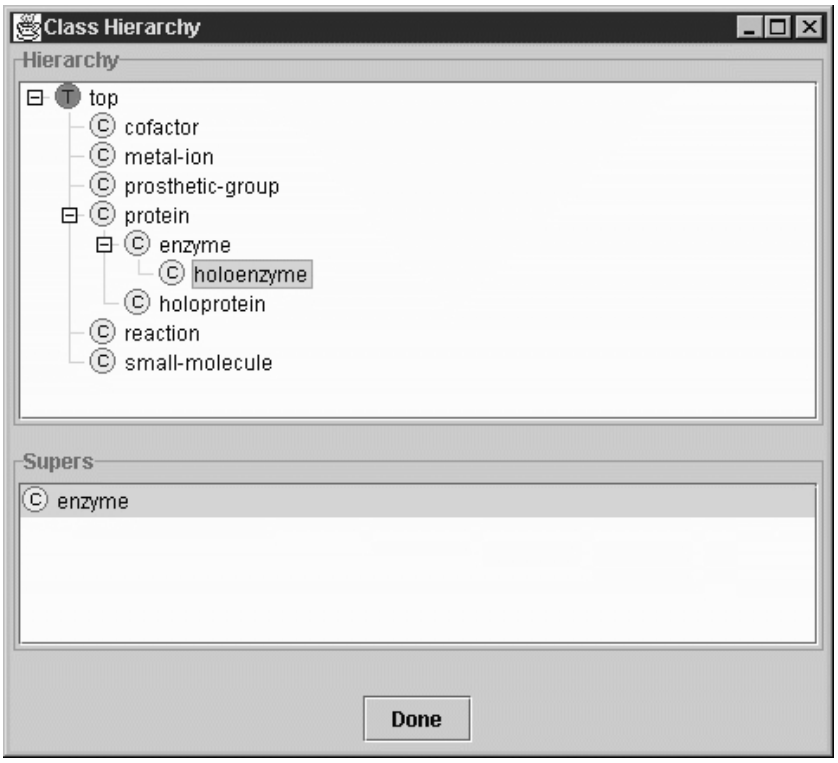

Fig. 3. Hierarchy pre-classification

When verification is requested, the ontology is translated into an equivalent $\mathcal{SHQ}$ (or $\mathcal{SHF}$) knowledge base and sent to the reasoner for classification [18]. OilEd then queries the classified knowledge base, checking for inconsistent classes and implicit subsumption relationships. The results are reported to the user by highlighting inconsistent classes and rearranging the class hierarchy display to reflect any changes discovered. FaCT/OilEd does not provide any explanation of its inferences, although this would clearly be useful in ontology design [19].

Figures 3 and 4 show the effects of classification on (part of) the hierarchy derived from the TAMBIS ontology (see Section 4). When verifying the ontology, a number of new subsumption relationships are discovered (due to the class definitions in the model). In particular we can see that, after verification, holoenzyme is not only an enzyme, but also a holoprotein, and that metal-ion and small-molecule are both subclasses of cofactor.

During subsequent editing, changes to the ontology are not communicated to the reasoner instantaneously, but only when explicitly requested by the user. Future versions of OilEd may incorporate "real-time" reasoning support, but

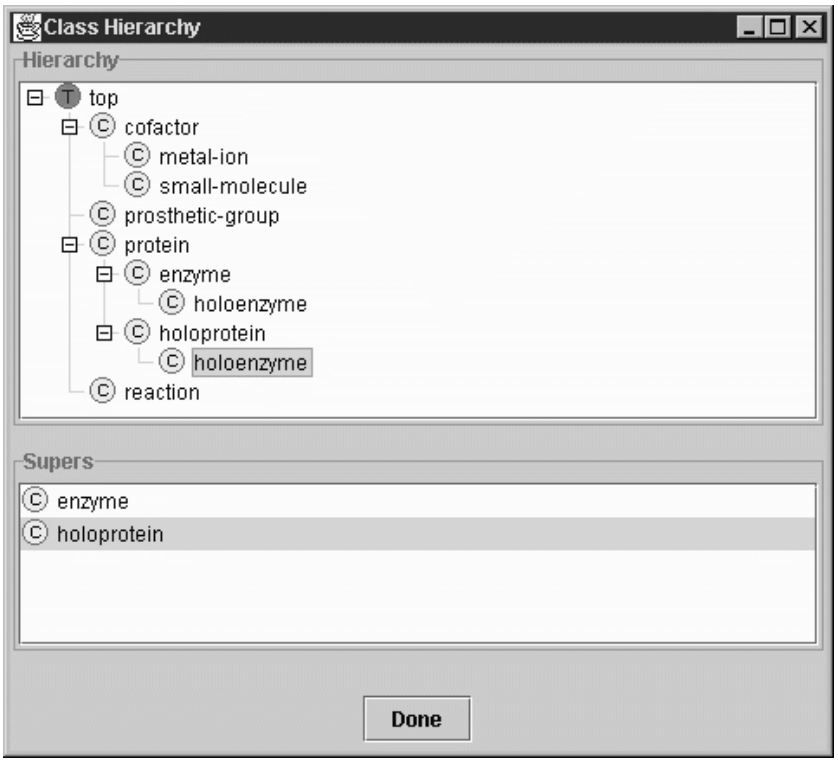

Fig. 4. Hierarchy post-classification

the simple interaction model described here was considered appropriate for the initial prototype.

3.3 Export

Although OilEd is primarily intended as an editor for OIL ontologies, the tool will export to a number of formats. These include OIL Standard (the "human-readable" presentation format for OIL that was used in Figure 1), OIL-RDFS (OIL's standard RDFS serialisation) and DAML+OIL (also RDFS). In addition, ontologies can be exported as HTML, facilitating viewing of the ontology without the tool and class hierarchies generated by the classifier can be exported as graphs for viewing with AT&T's Dotty [2] application.

By exporting ontologies as RDFS, it is envisaged that "RDFS-aware" applications will be able to read and interpret OIL ontologies even if they are not fully "OIL-aware". Of course, such applications would be unable to make use of all of the information in the model, but may be able to use, for example, the subclass hierarchies within the ontology. In order to facilitate this, OilEd allows the

[2] See `http://www.research.att.com/sw/tools/graphviz/`

possibility of explicitly adding all implicit subsumption relationships to the ontology before export, thus making this information available to non-OIL RDFS applications, or even OIL-aware applications that do not employ reasoning.

4 Case Study: the TAMBIS Ontology

The rôle of ontologies in bioinformatics (the discipline of applying computing to molecular biology) has become prominent in the last few years. Ontologies are used as a mechanism for expressing and sharing community knowledge, to define common vocabularies (e.g., for database annotations), and to support intelligent querying over multiple databases [20]. TAMBIS (Transparent Access to Multiple Bioinformatic Information Sources) is a mediation system that uses an ontology to enable biologists to ask questions over multiple external databases using a common query interface. The ontology is central to the TAMBIS system: it provides a model that queries can be formed against, it drives the query formulation interface, it indexes the middleware wrappers of the component sources, and it supports the query rewriting process [21]. The TAMBIS ontology (TaO) covers the principal concepts of molecular biology and bioinformatics: macromolecules; their motifs, their structure, function, cellular location and the processes in which they act. It is an ontology intended for retrieval purposes rather than hypothesis generation, so it is broad and shallow rather than deep and narrow [20].

The TaO was originally modelled in the DL GRAIL [17]. It was subsequently migrated to OIL in order to (a) exploit OIL's high expressivity so as to maintain a better fidelity with biological knowledge as it is currently perceived; (b) use reasoning support when building and evolving complex ontologies where the knowledge is dynamic and shifting; and (c) be able to deliver the TaO as a conventional frame ontology (with all subsumptions made explicit), thus making it accessible to a wider range of (legacy) applications and collaborators.

The approach to developing the ontology was directly influenced by the range of expressivity that OIL affords, and the capabilities of OilEd itself, particularly its reasoning facilities. The modelling philosophy was to be descriptive, i.e., to model properties and allow as much as possible of the subsumption lattice to be inferred by the reasoner. The design methodology was to first construct a basic framework of primitive foundation classes and slots, working both top down and bottom up, mainly using explicitly stated superclasses. The ontology was then incrementally extended and refined by adding new classes, elaborating slot fillers and constraints, and "upgrading" to defined classes wherever possible, so that class specifications became steadily more detailed and more accurate. This process was guided by subsumption reasoning—when elaborating or changing classes, the reasoner could be used to check consistency and to show the impact on the class hierarchy.

Figure 5 shows a (greatly simplified) fragment of the TaO (using OIL's presentation syntax) that we will use to illustrate this methodology.[3] Originally, holoprotein, enzyme and holoenzyme were all primitive classes, with no slot constraints, and an explicitly asserted class hierarchy: holoprotein and enzyme were subclasses of protein, and holoenzyme was a subclass of enzyme. During the extension and refinement phase, the properties of the various classes were described in more detail: it was asserted that a holoprotein binds a prosthetic-group, that an enzyme catalyses a reaction, and that a holoenzyme binds a prosthetic-group. Several of the classes were also upgraded to being *defined* when their description constituted both necessary and sufficient conditions for class membership, e.g., a protein is a holoprotein if and only if it binds a prosthetic-group. This allows the reasoner to infer additional subclass relationships w.r.t. holoprotein, and in particular that holoenzyme is a subclass of holoprotein. This latter relationship probably would have been missed if the ontology had been hand crafted.

class-def protein

class-def defined holoprotein
  subclass-of protein
  slot-constraint binds *has-value* prosthetic-group

class-def defined enzyme
  subclass-of protein
  slot-constraint catalyses *has-value* reaction

class-def defined holoenzyme
  subclass-of enzyme
  slot-constraint binds *has-value* prosthetic-group

class-def defined cofactor
  subclass-of (metal-ion *or* small-molecule)

disjoint metal-ion small-molecule

Fig. 5. Simplified fragment of TAMBIS ontology

The extension and refinement phase also included the addition of axioms asserting disjointness, equality and covering, further enhancing the accuracy of the model. Referring again to Figure 5, our biologist initially asserted that cofactor was a subclass of both metal-ion and small-molecule (a common confusion over the semantics of 'and' and 'or') rather than being either a metal-ion or a small-molecule. Subsequently, when it was asserted that metal-ion and small-molecule are disjoint, the reasoner inferred that cofactor was logically inconsistent, and the mistake was rectified. Modelling mistakes such as these litter bioontologies crafted by hand.

Other advantages derived from the use of OilEd included:

[3] The complete ontology can be found at `http://img.cs.man.ac.uk/stevens/tambis-oil.html`

- The frame-like look and feel of OilEd, and the frame approach of the OIL language, made ontology development much less daunting to our biologist than writing $\mathcal{SHQ}$ logic expressions would have been.
- Clipboard facilities provided by OilEd allowed (parts of) frames to be copied and pasted, making it easy to experiment with new definitions and to maintain a consistent modelling style. E.g., coenzymeA-requiring-oxidoreductase was built by copying nad-requiring-oxidoreductase and changing the constraint on the binds slot from nad to coenzymeA. The reasoner then automatically migrated the class from being a subclass of holoenzyme to being a subclass of coenzyme-requiring-enzyme.
- Class definitions can be as simple as possible yet as complex as necessary. Parts of the TaO are simply primitive frames and slots; other parts are very elaborate and exploit the full expressive power of the OIL language.
- In TAMBIS, the ontology is managed by an ontology server that makes full use of the class definitions, e.g., to classify user generated query classes. However, being able to deliver a static "snapshot" of the ontology in the form of an RDFS taxonomy has proved extremely convenient when working with collaborators who are building ontologies that are in fact simple taxonomies, such as the *Gene Ontology* [22].

5 Conclusion

Ontologies are useful in a range of applications, and will play a pivotal rôle in the Semantic Web, where they will provide a source of precisely defined terms that can be communicated across people and applications. Reasoning with respect to such terms will be important for both the design and deployment of ontologies.

We have presented OilEd, an ontology editor that has an easy to use frame interface, yet at the same time allows users to exploit the full power of an expressive web ontology language (OIL/DAML+OIL). We have also shown how OilEd uses reasoning to support ontology design and maintenance, and presented a case study illustrating how this facility can be used to develop ontologies that describe their domains in more detail and with greater accuracy.

OilEd is a prototype, designed to test and demonstrate novel ideas, and it still lacks many features that would be required of a fully-fledged ontology development environment, e.g., it provides no support for versioning, or for working with multiple ontologies. Moreover, the reasoning support provided by the FaCT system is incomplete for OIL extended with concrete datatypes and individuals, and does not include additional services such as explanation. However, in spite of these shortcomings, OilEd is already sufficiently well developed to be a very useful tool, and to demonstrate the utility of OIL's integration of features from frame, DL and web languages.

References

1. G. van Heijst, A. Schreiber, and B. Wielinga. Using explicit ontologies in KBS development. *Int. J. of Human-Computer Studies*, 46(2/3):183–292, 1997.

2. D. L. McGuinness. Ontological issues for knowledge-enhanced search. In *Proc. of FOIS-98*, 1998.
3. M. Uschold and M. Grüninger. Ontologies: Principles, methods and applications. *K. Eng. Review*, 11(2):93–136, 1996.
4. T. Berners-Lee. *Weaving the Web*. Orion Business Books, 1999.
5. S. Decker *et al.* The semantic web — on the respective roles of XML and RDF. *IEEE Internet Computing*, 2000.
6. T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In *Proc. of Int. Workshop on Formal Ontology*, 1993.
7. D. Fensel *et al.* OIL in a nutshell. In *Proc. of EKAW-2000*, LNAI, 2000.
8. J. Hendler and D. L. McGuinness. The DARPA agent markup language. *IEEE Intelligent Systems*, jan 2001.
9. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proc. of KR-00*, 2000.
10. J. Doyle and R. Patil. Two theses of knowledge representation. *Artificial Intelligence*, 48:261–297, 1991.
11. V. K. Chaudhri *et al.* OKBC: A programmatic foundation for knowledge base interoperability. In *Proc. of AAAI-98*, 1998.
12. S. Staab and A. Maedche. Ontology engineering beyond the modeling of concepts and relations. In *Proc. of the ECAI'2000 Workshop on Application of Ontologies and Problem-Solving Methods*, 2000.
13. W. E. Grosso *et al.* Knowledge modeling at the millennium (the design and evolution of protégé-2000). In *Proc. of KAW99*, 1999.
14. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, pages 161–180, 1999.
15. I. Horrocks. Benchmark analysis with fact. In *Proc. TABLEAUX 2000*, pages 62–66, 2000.
16. F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of IJCAI-91*, pages 452–457, 1991.
17. A. Rector *et al.* The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.
18. S. Decker *et al.* Knowledge representation on the web. In *Proc. of DL 2000*, pages 89–98, 2000.
19. D. McGuinness and A. Borgida. Explaining subsumption in description logics. In *Proc. of IJCAI-95*, pages 816–821, 1995.
20. P.G Baker, C.A Goble, S. Bechhofer, N.W Paton, R. Stevens, and A. Brass. An Ontology for Bioinformatics Applications. *Bioinformatics*, 15(6):510–520, 1999.
21. C. Goble, R. Stevens, G. Ng, S. Bechhofer, N.W. Paton, P.G. Baker, M. Peim, and A. Brass. Transparent Access to Multiple Bioinformatics Information Sources. *IBM Systems Journal*, 40(2), 2001.
22. M. Ashburner *et al.* Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.