Building a Reason-able Bioinformatics Ontology Using OIL

Robert Stevens, Ian Horrocks, Carole Goble and Sean Bechhofer
Department of Computer Science
University of Manchester
Oxford Road
Manchester, M13 9PL
United Kingdom
robert.stevens@cs.man.ac.uk

Abstract

Ontologies will play an important role in bioinformatics, as they do in other disciplines, where they will provide a source of precisely defined terms that can be communicated across people and applications.

The Ontology Inference Layer (OIL), is an ontology language that has an easy to use frame feel, yet at the same time allows users to exploit the full power of an expressive description logic. OilEd, an editor for OIL, uses reasoning to support ontology design, facilitating the development of ontologies that are both more detailed and more accurate.

This paper presents a bioinformatics ontology building case study using OilEd to highlight the features of the combination of a frame representation and an expressive description logic.

1 Introduction

Ontologies have become an increasingly important research topic. This is chiefly a result of their usefulness in a range of application domains [van Heijst *et al.*, 1997; McGuinness, 1998; Uschold and Grüninger, 1996] including bioinformatics [Stevens *et al.*, 2001].

Biologists have long had a culture of recording and sharing information. This information has traditionally been stored in natural language form and latterly in natural language annotated databases. There has been a recognition that if these information resources are to continue to play their central role in bioinformatics, they have to become machine understandable.

The automation of tasks depends on elevating the status of the information in resources from machine-readable to something we might call machine-understandable. The natural language annotation of a bioinformatics resource can be processed computationally, but using the knowledge contained in the natural language annotation is difficult. The key idea is to have data in these resources 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 goal, it will be necessary to annotate bioinformatics resources with *metadata* (i.e., data describing their content/functionality). Ontologies are a useful mechanism to provide metadata for various resources. 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 [Gruber, 1993].

There have been several attempts to develop bioinformatics ontologies to exploit this biological information. The Gene Ontology (GO) [The Gene Ontology Consortium, 2000] is a controlled vocabulary for annotating gene products for molecular functions, the biological processes in which it is involved and the cellular locations in which it is found. EcoCyc [Karp *et al.*, 2000] has used an ontology to specify a database schema for the E. *coli*. metabolism, signal trandsduction etc. RiboWeb [Altman *et al.*, 1999] also uses an ontology to describe its data, but also guide its users through analysis of their data. Finally, TAMBIS (Transparent Access to Multiple bioinformatics Information Sources) [Baker *et al.*, 1998] uses an ontology to allow users to query bioinformatics databases. Each of these uses a different knowledge representation system from phrases in GO; to frame based systems in EcoCyc and RiboWeb to a description logic in TAMBIS.

Phrase based vocabularies have the advantage of being easily accessible, but suffer from difficulties in consistency and maintenance. It is common for mistakes to be made in phrase based vocabularies, especially in the maintenance of multiple hierarchies. Frame-based systems have the advantage of an easily accessible and intuitive modelling style, reminisient of an object view of the world (a frame is a class and the slots are attributes. The frame encapsulates the properties of the instances). Such systems, like phrase based vocabularies, are esentially hand-crafted and can suffer from inconsistencies and logical mistakes. The well defined semantics and reasoning support of DLs allow logically consistent ontologies to be maintained. Concepts can be defined in terms of their properties and the reasoning used to classify the concepts based upon those descriptions. When an concept expression is unsatisfyable in terms of the rest of the model, the reasoning support can inform the modeller of his or her mistake. Description logic based ontologies avoid the problems of the hand-crafted ontologies, but suffer from the complexity of the modelling style.

These considerations have led to the development of OIL [Fensel *et al.*, 2000], an ontology language that extends a frame-based like view with a much richer set of modelling primitives[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. Thus a modeller is offered the best of both worlds in both development and deployment of an ontology. 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 valuable in many bioinformatics applications. The most obvious is in formulating, processing and answering queries over bioinformatics databases. There is also a great potential in using reasoning during analysis of novel biological entities.

The reasoning support offered by OIL 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 of consistency, yet provide more detailed descriptions that can be exploited by automated processes in the deployed ontologies. Finally, reasoning is of particular benefit when ontologies are large and/or multiply authored, and also facilitates ontology sharing, merging and integration [McGuinness *et al.*, 2000]; considerations that will be particularly important in the distributed bioinformatics environment.

The modeller, however, is not forced to use reasoning support OilEd can be used to construct hierarchies of terms unadorned by descriptions of properties. In fact, the ontology development described in Section 4, uses a cyclic, two stage approach to ontology development. For a given portion of the ontology, a simple hierarchy of terms is hand-crafted. In the next stage, properties are described for the necessary and sufficient conditions to be a member of that class or concept. The reasoner can then be used to check the descriptions for logical consistency and offer any inferred knowledge (unknown subsumptions) that have been found. The cycle is then repeated for this and other portions of the ontology.

This paper concentrates on this ontology design use of OIL, rather than the use of reasoning in the deployed ontologies. A case study taken from the domain of bioinformatics will be used to highlight the development and management facilities

afforded by the combination of the frame-like syntax of OIL with the expressive power of description logics. First, the

principal features of the language (Section 2) and its associated editor

(Section 3) will be described. Section 4 describes the development of an ontology of molecular biology and bioinformatics using OilEd.

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. These standards, such as RDFS [Brickley and Guha, 2000], make it easier to use ontologies consistently across the web. 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 [Doyle and Patil, 1991]).

The resulting language combines a familiar frame like syntax (derived in part from the OKBC-lite knowledge model [Chaudhri *et al.*, 1998]), with the power and flexibility 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 can 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 adaptation of tools, e.g., the OntoEdit and Protégé editors and the Chimaera integration tool are all being adapted to use OIL/DAML+OIL [Staab and Maedche, 2000; Grosso *et al.*, 1999; McGuinness *et al.*, 2000].

On the other hand, basing the language on an underlying mapping to a very expressive DL ($\mathcal{SHIQ}$) 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 [Horrocks *et al.*, 1999]. The mapping also provides a mechanism for the provision of practical reasoning services by exploiting implemented DL systems, e.g., the FaCT system [Horrocks, 2000].

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,

[1] A similar ontology language called DAML has been developed as part of the DARPA DAML project [Hendler and McGuinness, 2001]. These two languages are soon to be merged under the name DAML+OIL.

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 [Fensel *et al.*, 2000].

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

Figure 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 [Grosso *et al.*, 1999] and OntoEdit [Staab and Maedche, 2000], but OilEd extends these approaches in a number of ways, notably through an extension of expressive power and the use of a reasoner.

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. For example, a class called hydrolase has constraints including catalyses hydrolysis, describing one of the properties of a hydrolase to be the promotion of the reaction called hydrolysis. 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. For example, a gene is has-name gene-name or part-of gene-name – indicating that a gene may be found using its name or part of its name. 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

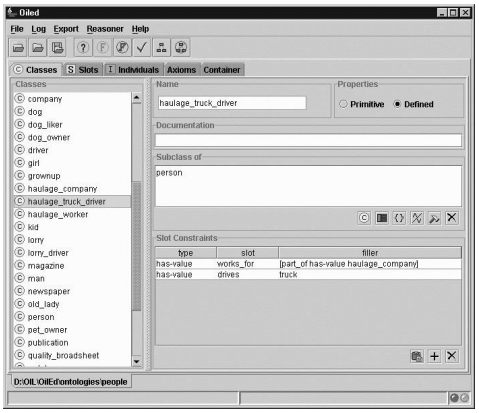

Figure 2: OilEd Class Panel

(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). For instance, it is possible to exactly describe that a G-protein coupled receptor has to have seven and only seven transmembrane regions – otherwise it is not a G-protein coupled receptor. 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 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 not allowed—classes must have a single definition—but the same effect can be achieved through the use of *equivalence* axioms as discussed below.

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 (see Section 4).

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. These axioms, especially the disjointness axiom, are quite heavily used in the case study ontology. It is useful to state explicitly that, for instance, something cannot be both an element and a compound.

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 a (non-biological) 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. The example ontology in Section 4 does not use any individuals. It might, however, be possible to use them to describe individual chemicals within the ontology.

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 underlying concrete datatypes is, however, well understood [Baader and Hanschke, 1991], and work is in progress to extend the FaCT reasoner with support for concrete datatypes. These data types are used in the description of atom in the example ontology (Section 4).

3.2 Reasoning

The editor can be requested to verify an ontology using the FaCT reasoner. When verification is requested, the ontology is translated into an equivalent $\mathcal{SHIQ}$ (or $\mathcal{SHF}$) knowledge base and sent to the reasoner for classification [Decker *et al.*, 2000]. 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 [McGuinness and Borgida, 1995].

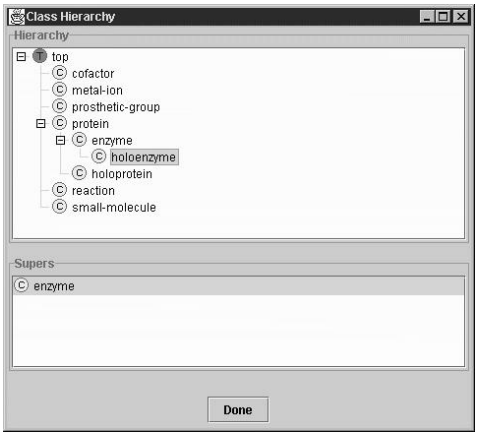

Figure 3: Hierarchy pre-classification

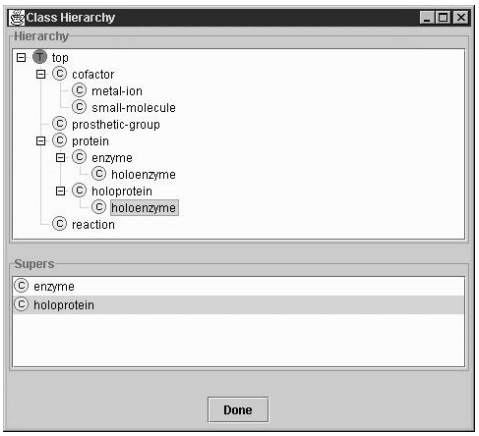

Figure 4: Hierarchy post-classification

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. Note that if the reasoning is not employed, and if the extended expressiveness and advanced features are not used, OilEd will still function as a simple frame editor.

4 Case Study: the TAMBIS Ontology

The role of ontologies in bioinformatics has become prominent in the last few years. Much of biology works by applying prior knowledge to an unknown entity. The complex biolog-

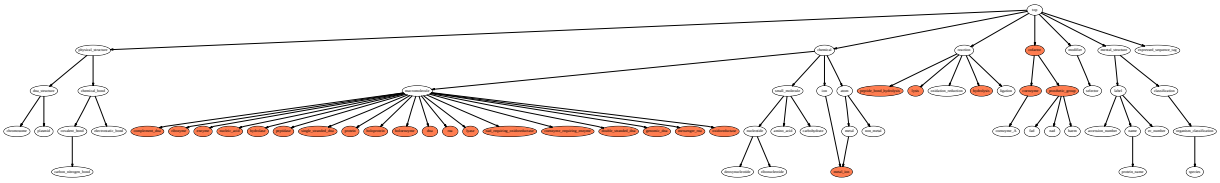

Figure 5: Definitions pre-classification

ical data stored in bioinformatics databases requires knowledge to specify and constrain values held in that database. Ontologies are also 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 [Baker *et al.*, 1999; Stevens *et al.*, 2001].

TAMBIS (Transparent Access to Multiple Bioinformatics 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 over which queries can be formed, it drives the query formulation interface, it indexes the middleware wrappers of the component sources, and it supports the query rewriting process [Goble *et al.*, 2001]. 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 [Baker *et al.*, 1999].

The TaO was originally modelled in the GRAIL DL [Rector *et al.*, 1997]. It was subsequently migrated to OIL in order to (a) exploit OIL's high expressivity, maintaining 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. This was a cyclic activity, with portions of the TaO being described primitively, then in the more descriptive fashion.

In each cycle, the reasoner is used to classify the ontology. The classification can then be viewed (with and without inferred subsumptions) to check the classification against the ontologist's knowledge. The editor allows concepts to be found by name, so recently constructed concepts can be viewed in their context. Logically inconsistent concept expressions (those equivalent to bottom) are higlighted for easy identification of badly formed expressions.

The initial model was very "tree-like", i.e., there were very few classes with multiple superclasses. The primitive portions of the ontology were 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 faithful to the application. 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.

As each cycle of extension of concept definitions ends, the modeller is able to view the use of primitive and defined concepts across the ontology. This view 'zooms' out from the ontology, showing the lattice as dots and arcs, with the dots differentiated according to their being defined or primitive. This enables the modeller to see areas of definition and where definition is lacking. Building-block concepts, that are not central to the use of the ontology, will in all likelyhood remain primitive, but it is useful to spot where definition is lacking; as definition increases the fidelity and justification for the ontology. For instance, the macromolecules within the TaO are highly defined, but the properties, used in the definition of more central concepts remain primitive.

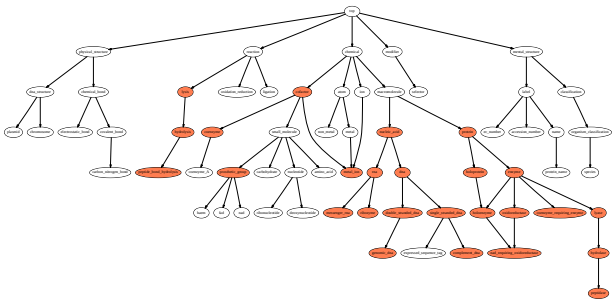

Figure 6: Definitions post-classification

Figures 5 and 6 illustrate this (using a subset of the complete ontology). Figure 5 shows the distrubution of defined concepts throughout the hierarchy before classification[2]. Defined concepts are signified using a darker colour, and we can see that the hierarchy has a very flat structure. In Figure 6,

[2]The hierarchies are generated using OilEd's export functionality, which produces graphs for rendering by AT&T's Graphviz software

we see the situation after classification. The defined concepts have now been organised into a subsumption hierarchy based on their definitions.

Figure 7 shows a (greatly simplified) fragment of the TaO (using OIL's presentation syntax) that we will use to illustrate this methodology.[3]

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

Figure 7: Simplified fragment of TAMBIS ontology

Bioinformatics is the study and analysis of molecular biology – the functions and processes of the products of an organism's genes. The knowledge about molecular biology is contained within numerous data banks and analysis tools. An ontology of bioinformatics therefore needs to support two domains: First, the domain of molecular biology – the chemicals and higher-order chemical structures within a cell and second, to reflect the nature and content of bioinformatics resources.

The TaO was built with both a top-down and bottom-up strategy. A general domain framework was provided into which more detailed molecular biological and bioinformatics concepts could be fitted. As well as this approach, a solid conceptual foundation about chemicals and their structure and behaviour was built. Basic chemicals and their properties are used to describe the more complex biological molecules of interest to bioinformatics, so this is an appropriate approach both from a straightforward content, as well as a modelling, point of view. This involved the description of the different kinds of chemicals (ions, atoms and molecules etc.); their structure, reactions, function and processes in which they act. This general foundation was then used to give the subsequent detailed description of the salient molecular biological concepts that form the bottom-up placement of defined concepts.

The core of the TaO is a description of basic chemical concepts. The various kinds of chemical are defined as children of the concept chemical. These include:

atom The building block of all chemicals. A chemical's behaviour is defined by the number of protons it contains, i.e., its atomic number. Therefore, atom is defined as:

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

class-def defined atom
  subclass-of chemical
  slot-constraint atomic-number
    cardinality 1
    value-type integer
    has-value (*min* 1)

So, atoms may only have one atomic number, which must be an integer greater than or equal to 1. The concepts metal-atom, nonmetal-atom and metalloid-atom are defined to be atoms with the physicochemicalproperty of either metal nonmetal or metalloid respectively.

The concept of carbon has been defined as a kind of atom with atomic number six and the physicochemical property of non-metal. This description of the concept carbon enables it to be automatically placed as a kind of nonmetal-atom. Several other, biologically relevant, atom types have been included in the TaO.

ion An ion is simply a chemical with an electrical charge. It is defined as:

class-def defined ion
  subclass-of chemical
  slot-constraint has-charge
    has-value (*not* 0)

The slot constraint describes that an ion must have an electrical charge and it can only be an electrical charge. It also describes that the value for this charge can be a positive or negative number, but not zero. it would be possible to capture chemical reality further by specifying a minimum cardinality of one – that is, a chemical must have at least one charge to be an ion, but may have more than one charge (a molecule could, for instance, contain both a positive and negative charge).

the chemical ion has two asserted children: cation and anion. Defining cation as a chemical with charge greater-than 0 enables the classifier to place it correctly as a kind of ion. An equivalence axiom can be used to state that cation is a synonym of positive-ion.

Now,divalent-cation (a chemical with two positive charges) can be defined by adding further properties to this slot constraint: That

the filler for has-charge is equal 2, that is, has positive two charges on the chemical.

element An element is a kind of chemical containing only one kind of atom. OIL has the expressive power to constrain the slot atom-type to be equal to only one. Adding the slot constraint atom-type with the value one to atom would also classify atom as an element.

compound A compound is a chemical containing more than one kind of atom. The slot constraint used for element (above) is altered so that the constraint indicates that at least two kinds of atom must be present in this kind of chemical.

molecule A molecule is a kind of chemical containing atoms linked by covalent bonds. The concept covalent-bond was described as a kind of chemical-structure and used to fill the slot contains-bond, with the *has-value* restriction. So, there must be a covalent bond present for it to be classed as a molecule, but other kinds of bond may be present – exactly capturing what we understand of basic chemicals.

Two principal features of the ontology development arise from this chemical core:

1. The need for a framework of primitive concepts, such as metal and properties such as has-charge. These can be used to develop the core of defined concepts at the centre of the TaO. Primitive concepts, as well as those such as chemical itself, are placed within a simple upper level ontology containing physical, mental, substance, structure, function and process. These are extended by their obvious conjunctive forms, e.g., physical-structure.
2. The ability to rapidly extend this chemicals core to another layer of defined chemical concepts, all of which used the previously defined concepts.

The next "layer" of chemical descriptions included:

molecular-compound A chemical containing covalent bonds and more than one type of atom.

elemental-molecule A chemical, such as oxygen (O_2), that contains covalent bonds and only one kind of atom.

metal-ion A kind of atom with an electrical charge.

ionic-compound A kind of chemical containing more than one kind of atom and has an electrical charge.

All these and more were simply defined to be the conjunction of two concepts. For example:

class-def defined metal-ion
  subclass-of metal, ion

class-def defined divalent-cation
  subclass-of chemical
  slot-constraint has-charge
    has-value (*equal* 2)

A concept divalent-zinc-cation can then simply be defined as:

class-def defined divalent-zinc-cation
  subclass-of zinc
    slot-constraint has-charge
      has-value (*equal* 2)

These descriptions of chemicals can be reinforced with the use of axioms. It is not possible to be both an element and a compound, so these two concepts are described as disjoint. This means that if a concept were to be defined with properties of both an element and a compound, it would be found to be inconsistent by the reasoner. Such strict definitions help maintain the consistency and biological thoroughness of the ontology.

An organic-molecular-compound is a molecular compound that contains at least one carbon atom. This, however, is not sufficient to define an organic molecular compound. Carbon dioxide (CO_2) is a molecular compound containing carbon, but is not organic. Thus the property of containing carbon is only a necessary condition for being an organic molecular compound. Again, the ability to be exact with concept descriptions allows the ontology to match chemical and biological knowledge closely and prevent conceptualisations being made that contradict domain knowledge.

Bioinformatics is mainly concerned with organic macromolecular-compounds. Thus, organic molecular compound was split into the biologically useful distinctions of macromolecular-compound and small-molecular-compound. the distinction is one of size and a protein, for example, of over 100 Daltons is usually said to be a macromolecule. Unfortunately the boundary is more complex, a smaller molecule can still be "macro", depending on its context. For this reason, sufficiency conditions were not used in the definition. Useful small organic molecules were simply asserted as primitive concepts underneath small-organic-molecular-compound. These include nucleotide, amino-acid and others useful in describing the properties of biological concepts.

For the purposes of the TaO, macromolecular-compounds are polymers of small-organic-molecular-compounds and are defined as such. Thus, protein is defined as a polymer of amino-acid; nucleic-acid as a polymer of nucleotide and polysaccaride as a polymer of saccaride. A macromolecule can only be a polymer of one kind of small molecule, so the *value-type* restriction is used in the slot constraint. It is only possible to be one of these molecules, so the *disjoint* axiom is used on these macromolecules.

As most of bioinformatics concentrates on the analysis and description of nucleic acids and proteins, much of the TaO's description concentrates in this area. DNA and RNA are both nucleic acids formed from different kinds of nucleotide.

Describing DNA *slot-constraint value-type has-value* deoxy-nucleotide, allows the classifier to correctly place it as a kind of nucleic-acid and capture that DNA can only be a polymer of the deoxy- form of a nucleotide and some of the nucleotide have to be present. The various different kinds of DNA and RNA are distinguished by their function and/or cellular location. Again, as before, other parts of the TaO are used to describe these properties of biological concepts. For example, genomic-dna is dna that is found on a nuclear chromosome, chloroplast chromosome, or mitochondrial chromosome. The slot constraint uses *or* in the filler class expression to describe this:

slot-constraint part-of
  cardinality 1
  value-type
    (nuclear-chromosome *or*
    mitochondrial-chromosome *or*
    chloroplast-chromosome)).

The TaO contains a rich partonomy. The cellular structures, in particular, use the part-of slot and its transitive property to build up this partonomy. For instance, nuclear-chromosome is part-of the nucleus, which itself is part-of the cell. Thus, a nuclear-chromosome is part-of the cell.

These biological-structures and associated partonomy are part of the TaO. Not only are they used in building some of the descriptions of bio-concepts, but are also part of the description of the content of bioinformatics resources.

In the initial description of kinds of protein, holoprotein, enzyme and holoenzyme were originally 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.

Enzyme was removed from the superclass list and replaced with protein; then holoenzyme's properties were described in more detail using slot constraints—in particular, it was asserted that a holoenzyme catalyses a reaction and binds a prosthetic-group. This allows the reasoner to infer not only the subclass relationship w.r.t. enzyme, but also additional subclass relationships w.r.t. holoprotein, and in particular that holoenzyme is a subclass of holoprotein. This latter relationship could have been missed if the ontology had been hand crafted.

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 7, 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.

There are two kinds of cofactor – coenzyme and prosthetic-group. A coenzyme can be either a small molecule or metal ion and binds loosely to a protein. A prosthetic group, on the other hand, is a kind of cofactor that binds tightly to a protein, but can only be a small molecule. Again, OIL is expressive enough to capture these distinctions accurately.

class-def defined prosthetic-group
  subclass-of cofactor *and* (*not* metal-ion)
  slot-constraint binds-tightly
    has-value protein

The slot hierarchy was also used to induce the classification of types of enzyme. For example, reaction (used in the definition of enzyme) has a child lysis. Lysis is the breaking of a covalent bond and hydrolysis is breaking of a covalent bond with water. These two reactions are defined using the following slot definitions:

slot-def lysis-of
  domain reaction
  range covalent-bond

slot-def hydrolysis-of
  subslot-of lysis-of

class-def defined lysis
  subclass-of reaction
  slot-constraint lysis-of
    has-value covalent-bond
    value-type covalent-bond

class-def defined hydrolysis
  subclass-of reaction
  slot-constraint hydrolysis-of
    has-value covalent-bond
    value-type covalent-bond

class-def defined lyase
  subclass-of protein
  slot-constraint catalyses
    has-value lysis
    value-type lysis

class-def defined hydrolase
  subclass-of protein
  slot-constraint catalyses
    has-value hydrolysis
    value-type hydrolysis

A lyase is a protein that catalyses lysis. A hydrolase is a protein that catalyses hydrolysis. As the slot hierarchy describes hydrolysis-of being a subslot of lysis-of, hydrolysis is a child of lysis and consequently, hydrolase is a child of lyase.

Other advantages derived from the use of OilEd included:

- 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 $\mathcal{SHIQ}$ 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* [Ashburner *et al.*, 2000].

5 Conclusion

Ontologies are useful in a range of applications, where they provide a source of precisely defined terms that can be communicated across people and applications. We have used as an example, the initial development of a molecular biology and bioinformatics ontology. Examples from this case study have been used to demonstrate the utility of OIL's integration of features from frame and DL languages. It can be seen from the case study that OIL can support a cyclical ontology development, where incremental moves are made from a primitive, asserted taxonomy to one where concepts are rich with properties. These properties can be used to add richness to the ontology (from inferred knowledge), as well as ensuring the logical consistency and satisfiability of the ontology. Thus, the use of reasoning can be seen to be important for the design and management of ontologies during their development.

OilEd is a prototype development environment for OIL, 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. It is likely that during the development of the TaO that other, or fragments of other, ontologies will be imported into the TaO. 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. Thus, the definitions used for atom and the charge on ions is not used in constructing the classification. Explanation has potential use in both the development and use of a bio-ontology. During development, it will obviously be useful to have explanations of why a concept was unsatisfiable according to the current model. It is a goal for a bio-ontology, such as TaO, to be used in the analysis of novel biological macromolecules. Certain bioinformatics analyses can describe the properties of such molecules. If these could be cast in terms of the TaO, novel concepts generated by such analyses could be classified in the TaO. the use of explanation could significantly guide the use of such analyses.

During this case study, we have presented OIL and 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 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 fidelity.

Acknowledgements: Robert Stevens is supported by BBSRC/EPSRC grant 4/B1012090 and Sean Bechhofer is supported by EPSRC grant GR/M75426.

References

[Altman *et al.*, 1999] R. Altman, M. Bada, X.J. Chai, M. Whirl Carillo, R.O. Chen, and N.F. Abernethy. RiboWeb: An Ontology-Based System for Collaborative Molecular Biology. *IEEE Intelligent Systems*, 14(5):68–76, 1999.

[Ashburner *et al.*, 2000] M. Ashburner *et al.* Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[Baader and Hanschke, 1991] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of IJCAI-91*, pages 452–457, 1991.

[Baker *et al.*, 1998] P.G. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens. TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. An Overview. In *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology*, pages 25–34. AAAI Press, June 28-July 1, 1998 1998.

[Baker *et al.*, 1999] P. Baker *et al.* An ontology for bioinformatics applications. *Bioinformatics*, 15(6):510–520, 1999.

[Brickley and Guha, 2000] D. Brickley and V.R. Guha. Resource description framework schema specification 1.0. W3C Candidate Recommendation, 2000. `http://www.w3.org/TR/rdf-schema`.

[Chaudhri *et al.*, 1998] V. K. Chaudhri *et al.* OKBC: A programmatic foundation for knowledge base interoperability. In *Proc. of AAAI-98*, 1998.

[Decker *et al.*, 2000] S. Decker *et al.* Knowledge representation on the web. In *Proc. of DL 2000*, pages 89–98, 2000.

[Doyle and Patil, 1991] J. Doyle and R. Patil. Two theses of knowledge representation. *Artificial Intelligence*, 48:261–297, 1991.

[Fensel *et al.*, 2000] D. Fensel *et al.* OIL in a nutshell. In *Proc. of EKAW-2000*, LNAI, 2000.

[Goble *et al.*, 2001] C. Goble *et al.* Transparent access to multiple bioinformatics information sources. *IBM Systems Journal*, 40(2), 2001.

[Grosso *et al.*, 1999] W. E. Grosso *et al.* Knowledge modeling at the millennium (the design and evolution of protégé-2000). In *Proc. of KAW99*, 1999.

[Gruber, 1993] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In *Proc. of Int. Workshop on Formal Ontology*, 1993.

[Hendler and McGuinness, 2001] J. Hendler and D. L. McGuinness. The DARPA agent markup language. *IEEE Intelligent Systems*, jan 2001.

[Horrocks *et al.*, 1999] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, pages 161–180, 1999.

[Horrocks, 2000] I. Horrocks. Benchmark analysis with fact. In *Proc. TABLEAUX 2000*, pages 62–66, 2000.

[Karp *et al.*, 2000] P.D. Karp, M. Riley, M. Saier, I.T. Paulsen, S.M. Paley, and A. Pellegrini-Toole. The EcoCyc

and MetaCyc Databases. *Nucleic Acids Research*, 28:56–59, 2000.

[McGuinness and Borgida, 1995] D. McGuinness and A. Borgida. Explaining subsumption in description logics. In *Proc. of IJCAI-95*, pages 816–821, 1995.

[McGuinness *et al.*, 2000] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proc. of KR-00*, 2000.

[McGuinness, 1998] D. L. McGuinness. Ontological issues for knowledge-enhanced search. In *Proc. of FOIS-98*, 1998.

[Rector *et al.*, 1997] A. Rector *et al.* The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.

[Staab and Maedche, 2000] 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.

[Stevens *et al.*, 2001] R. Stevens, C. A. Goble, and S. Bechhofer. Ontology-based knowledge representation for bioinformatics. *Briefings in Bioinformatics*, 2001.

[The Gene Ontology Consortium, 2000] The Gene Ontology Consortium. Gene Ontology: Tool for the Unification of Biology. *Nature Genetics*, 25:25–29, 2000.

[Uschold and Grüninger, 1996] M. Uschold and M. Grüninger. Ontologies: Principles, methods and applications. *K. Eng. Review*, 11(2):93–136, 1996.

[van Heijst *et al.*, 1997] 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.