OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns

Alan Rector[1] Nick Drummond[1] Matthew Horridge[1] Jeremy Rogers[1]
Holger Knublauch[2] Robert Stevens[1] Hai Wang[1] Chris Wroe[1]

[1] Department of Computer Science, University of Manchester,
Manchester M13 9PL, UK
{rector, ndrummond, mhorridge, jrogers, stevensr, hwang, wroec}@cs.man.ac.uk
[2] Stanford Medical Informatics, Stanford University,
Stanford, CA, USA 94305-5479
holger@smi.stanford.edu

Abstract. Understanding the logical meaning of any description logic or similar formalism is difficult for most people, and OWL-DL is no exception. This paper presents the most common difficulties encountered by newcomers to the language, that have been observed during the course of more than a dozen workshops, tutorials and modules about OWL-DL and it's predecessor languages. It emphasises understanding the exact meaning of OWL expressions – proving that understanding by paraphrasing them in pedantic but explicit language. It addresses, specifically, the confusion which OWL's open world assumption presents to users accustomed to closed world systems such as databases, logic programming and frame languages. Our experience has had a major influence in formulating the requirements for a new set of user interfaces for OWL the first of which are now available as prototypes. A summary of the guidelines and paraphrases and examples of the new interface are provided. The example ontologies are available online.

1 Introduction

1.1 Background

Most people find it difficult to understand the logical meaning and potential inferences statements in description logics, including OWL-DL. While there are several initial guides to ontologies available, e.g. [15, 6, 2, 7] and numerous works on ontological principles, e.g. [3, 4, 14], there is little guidance on how to use OWL-DL or related description logic formalism so as to make effective use of their classifiers (aka "reasoners") and even less on the pitfalls involved in their use. Likewise, few example ontologies on the web make extensive use of inference.

Over the past five years the authors have presented a series of tutorials, workshops and post-graduate modules, teaching people to use OWL-DL and its predecessors effectively. The purpose of this paper is to systematise the knowledge gained about new

users' difficulties in understanding OWL, to present examples which address those misunderstandings and patterns which avoid them.

The most common problems which we address are:

1. Failure to make all information explicit - assuming that information implicit in names is "represented" and available to the classifier.
2. Mistaken use of universal rather than existential restrictions as the default
3. Open world reasoning
4. The effect of range and domain constraints as axioms

To this we can add the additional problems posed by:

1. Trivial satisfiability of universal restrictions – that "only" (allValuesFrom) does not imply "some" (someValuesFrom).
2. The difference between defined and primitive classes and the mechanics of converting one to the other.
3. Errors in understanding common logical constructs.
4. Expecting classes to be disjoint by default.
5. The difficulty of understanding subclass axioms used for implication.

(Note that this paper only concerns issues in defining OWL classes, since this is the strength of OWL-DL and most existing classifiers deal with individuals incompletely or not at all.)

Our experience to date has been with the first generation of tools for OWL-DL and its predecessors, OilEd [1] [3], and with even earlier tools from *Open*GALEN [11][4]. The requirements for the new tools being developed in the Protégé-OWL-CO-ODE environment[5] [5] collaboratively by the authors have been informed by this experience.

Ontologies corresponding to the paper can be found at `http://www.co-ode.org/ontologies`. All tools are available at the URLs for the projects given in the footnotes.

1.2 The tutorials: Pizza, Manchester House Style, and "What does it mean"

We have used many example ontologies over the years - vehicles, IKEA catalogues, the University department and course, biomedical examples – but for Western audiences pizzas have proven most successful.[6] They are familiar; they are fun; they are fundamentally compositional. They are concrete and physical; real pizza menus are readily available; and they avoid thorny ontological issues involved in abstract notions such as "ideas", "causation", "agency", etc. Nonetheless, they are rich enough to illustrate key issues. Constructing correct definitions of pizzas from a menu and for a "vegetarian pizza" so that the correct pizzas are classified as "vegetarian" turns out to be a surprisingly challenging exercise.

[3] http://oiled.man.ac.uk

[4] http://www.opengalen.org

[5] http://protege.Stanford.edu → plugins → backends → OWL; http://www.co-ode.org

[6] For some non-western audiences, alternatives are required but are outside the scope of this paper

The style presented here is unashamedly the *Manchester House Style* – what we consider good practice. We do not claim that it is the only way to model in OWL-DL, but we do claim that it is one proven, effective way. The central feature of the style is "normalisation" described in detail in [10]. Note that a slightly simplified form of the OWL abstract syntax [8] is used in this paper which uses "and" and "or" rather than "intersectionOf" and "unionOf", as this corresponds more closely to what actually appears in both OilEd and the Protégé -OWL-CO-ODE interfaces.

2 The Basics

Before discussing the serious difficulties encountered by newcomers, it is helpful to introduce the pizza example and some basics - the notion of disjointness of classes, the function of the classifier in testing consistency, the basic notion of descriptions and existential graphs, and the first conventions for paraphrasing OWL. (The full set of conventions for paraphrasing appears at the end of the paper in Table 1).

2.1 Subsumption, Disjointness and the classifier

A simple taxonomy of pizza toppings is shown in Figure 1. The first question for a newcomer to OWL is "What does such a hierarchy actually mean?" By contrast to frame systems, subsumption in OWL means necessary implication, so the hierarchy means that "All Pepperoni is Meat", that "All Meat is a Pizza topping", etc.[7]

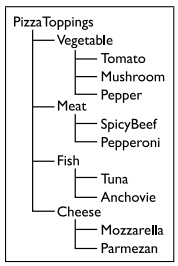


Fig. 1. Pizza hierarchy(Revised disjoint)

Does this mean that Meat and Fish and Vegetables etc. are all different? Can there be anything that is both Meat and Vegetable? Users of many other formalisms would naturally assume that they were different, at least unless they had an explicit common child.

However, in OWL, classes are overlapping until disjointness axioms are entered. This can be illustrated using a procedure that also demonstrates the role of the classifier in checking the consistency of the ontology.

First, a class MeatyVegetable is created that is a subclass of both Meat and Vegetable. This means that all MeatyVegetable are kinds of Meat and also kinds of Vegetable. The classifier is then run first without the disjointness axioms between Meat and Vegetable and then after they have been inserted. Without the disjointness axioms, running the classifier produces no change. However, when the disjointness axioms are added, MeatyVegetable is marked in red in the Protégé-OWL interface, indicating that it is inconsistent or "unsatisfiable". In Protégé -OWL-CO-ODE a note that it has been found unsatisfiable also appears in the list of changes, and a warning is issued during classification.

[7] One must immediately add something like "for purposes of this ontology." (It would be better to add a suffix Topping everywhere, and we normally do so in ontologies, but for this paper it leads to long expressions which tend not to fit on a single line.)

That the notion of a "meaty vegetable" should be inconsistent conforms with users' intuitions from the names of the classes. However, it is critical to understand that this implicit information in their names is unavailable to the classifier. Meat and Vegetable are only recognised by the classifier as disjoint if the disjointness axioms are entered explicitly.

One of the most common errors in building ontologies in OWL has been to omit the disjointness axioms. To help users manage disjointness axioms, the Protégé -OWL-CO-ODE interface makes entering them easy by providing a single button to add or remove disjointness axioms amongst all siblings of a given parent[8] (See Figure 18).

2.2 Properties and existential restrictions

The purpose, however, of OWL is not just to create a concept hierarchy but to describe and define concepts. Therefore we want to 'build' some pizzas. Figure 2 gives a description of the concept – MargheritaPizza[9] – from a Pizza menu as typically constructed by students early on in the course.

The first problem for students is to understand exactly what the description in Figure 2 means. That all MargheritaPizza have Mozzarella and Tomato? That any pizzas having Mozzarella and Tomato are MargheritaPizza? That MargheritaPizza has Mozzarella and Tomato and nothing else? The paraphrase makes the meaning absolutely clear. The italicised words – "*amongst other things*" and "*some*" - are critical.

Note that one of the most common errors made by newcomers to OWL is to use universal (allValuesFrom) rather than existential (someValuesFrom) as the default qualifier. This error is pernicious because the results often appear to work initially with the problems only becoming evident later in the course of developing the ontology. (See also Section 5.3). In teaching we go to great effort to ensure that existential someValuesFrom restrictions are used as the default from the beginning, both through the order of presentation and through the software used, where someValuesFrom is always the default (See also section 5.1).

OWL:
Class (MargheritaPizza partial
  Pizza
    restriction (hasTopping someValuesFrom Mozzarella)
    restriction (hasTopping someValuesFrom Tomato))
Paraphrase:
Margherita pizzas have, *amongst other things*, *some* mozzarella topping and also *some* tomato topping.

Fig. 2. Description and paraphrase of a Margherita Pizza.

[8] In OilEd, by contrast, disjointness axioms have to be entered on a separate tab.
[9] Margherita Pizzas are listed on the menu has having cheese and tomato toppings

3 Definitions and the Open World Assumption

3.1 Defined Classes: "Conceptual Lego"

OWL:
Class(CheeseyPizza complete
  Pizza
    restriction (hasTopping someValuesFrom Cheese))
Paraphrase:
A cheesey pizza is *any* pizza that has, *amongst other things*, *some* cheese topping.

Fig. 3. Initial definition of cheesey pizza

"Primitive" and "defined"classes is one of the major novelties of OWL and Understanding the difference between them is one of the major stumbling blocks for newcomers.

Many potential users are familiar with either frame systems such as Protégé or object oriented programming and analysis and UML. In both formalisms, things can be described by necessary conditions but not as defined classes with sufficiency conditions. However, in OWL, new concepts can be built up from existing concepts by fitting them together in definitions like blocks of Lego. More formally, OWL allows concepts to be defined by sets of necessary and sufficient conditions as shown in Figure 3. We refer to classes for which there are only necessary conditions – marked by the keyword "partial"[10] in the abstract syntax – as "primitive" classes, harking back to an older terminology. Classes for which there is at least one set of necessary and sufficient conditions – marked in the abstract syntax by the keyword "complete" – are referred to as as "defined".[11]

In the paraphrase the meaning of definitions is emphasized by the italicized "*any*" – *i.e.* any pizza that satisfies these conditions will be classified as a "cheesey pizza".

In our experience, the most common accidental error in implementing OWL ontologies is to fail to make a set of restrictions a definition - i.e. to fail to make the definition "complete" rather than "partial", or "necessary and sufficient" rather than just "necessary". It is critical to understand that, in general, nothing will be inferred to be subsumed under a primitive class by the classifier. For example in (Figure 3), if "complete" is replaced by "partial", then nothing will appear under CheesyPizza.

[10] OWL descriptions are sometimes known as "partial definitions" and the keyword "partial", appears in OWL syntax and in OilEd. This usage is seriously misleading and the term "partial definition" a misnomer. Many of the restrictions on primitive, or indeed defined, classes are necessary implications and take no part in any set of necessary and sufficient conditions - i.e. in definitions - "partial" or otherwise.

[11] It is also possible to make the distinction using the difference between "SubClassOf" and "EquivalentClasses", but we usually postpone that discussion until the basics are understood.

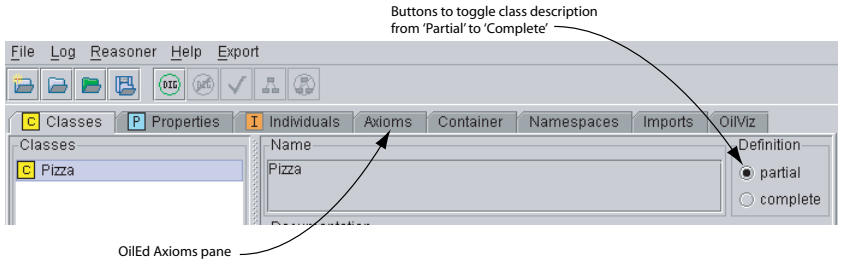


Fig. 4. OilED sets "partial" by default

The first thing, therefore, to check when things fail to be classified under a concept, is whether or not the description of the concept is "defined" or "primitive". In OilEd, a small button in the upper right hand corner of the screen, is used to toggle from "partial" to "complete" and is set to "partial" by default (Figure 4). A major goal for the the new Protégé -OWL-CO-ODE interface has been to make the distinction between primitive and defined classes clearer (See Figure 18).

3.2 Open world reasoning: Vegetarian Pizzas

The biggest single hurdle to understanding OWL and Description Logics is the use of Open World Reasoning. Almost certainly, all systems that newcomers to OWL will have encountered previously use closed world reasoning with "negation as failure" – i.e. if something cannot be found, it is assumed to be absent, *e.g.* databases, logic programming, constraint languages in frame systems, etc. By contrast, OWL uses open world reasoning with negation as unsatisfiability - *i.e.* something is false only if it can be proved to contradict other information in the ontology.

OWL:
Class (VegetarianPizza complete
  Pizza
    complementOf(restriction hasPart someValuesFrom Meat)
    complementOf(restriction hasPart someValuesFrom Fish))
Paraphrase:
A vegetarian pizza is *any* pizza that, *amongst other things*, both does *not* have *some* meat topping and also does *not* have *some* fish topping.

Fig. 5. Correct definitions of Vegetarian Pizza.

This point is dramatically made by attempting to define a VegetarianPizza. Expressing the negation is a problem in itself that is discussed under section 5 "Logical Issues". However, even once a correct logical definition is formulated as in Figure 5, there are surprises.

OWL:
Class (MeatyMargheritaPizza complete
  Pizza
    restriction (hasTopping someValuesFrom Tomato)
    restriction (hasTopping someValuesFrom Mozzarella)
    restriction (hasTopping someValuesFrom SpicyBeef))
Paraphrase:
A meaty margherita pizza is *any* pizza which, *amongst other things*, has *some* tomato topping and also *some* mozzarella topping and also *some* spicy beef topping.

Fig. 6. Definition of a "meaty margherita pizza" which is consistent and will be classified under Margherita pizza even though it has a meat topping.

Given the definitions so far, MargheritaPizza does not classify as VegetarianPizza. There is nothing in their definition that makes it contradictory to add meat or fish toppings. For example the MeatyMargherita pizza defined in Figure 6 is consistent and classifies under MargheritaPizza.

That the definition of MargheritaPizza was inadequate should be clear from the paraphrases in Figures 2 and 6. The rules for paraphrasing require adding "*amongst other things*" and "*any*" specifically so as to capture the open world assumption implicit in all OWL expressions. Clearly, the paraphrase does not correspond to what most restaurant customers would understand from the menu – that a Margherita pizza is a pizza that has mozzarella and tomato toppings and only those toppings. This intuitive understanding is captured formally in the OWL definition in Figure 7. The final restriction is known as a "closure restriction" or "closure axiom" because it closes off the possibility of further additions for a given property. "allValuesFrom" is paraphrased as "*only*", because to say that *all* values come from a given class is the same as saying that values may *only* come from that class.

OWL:
Class(MargheritaPizza complete
  Pizza
    restriction (hasTopping someValuesFrom Tomato)
    restriction (hasTopping someValuesFrom Mozzarella)
    restriction (hasTopping allValuesFrom (Tomato or Mozzarella)))
Paraphrase:
A margherita pizza is *any* pizza which, *amongst other things*, has *some* tomato topping and also *some* mozzarella toppings and also has *only* mozzarella *and/or* tomato toppings.

Fig. 7. Correct version of definition of Margherita pizza with closure restriction.

The phrase "*amongst other things*" in the paraphrase still allows space for a MargheritaPizza to have restrictions involving properties other than hasTopping - *e.g.* to repre-

sent that it is stale, overcooked, chopped into pieces etc. Anything except a margherita pizza with additional toppings.

3.3 Which classes should be defined? Which primitive? How to decide?

A common question from newcomers to OWL is how they should decide which classes to define. The choice of the "skeleton taxonomies" of primitive concepts is a key part of the method of "untangling" discussed in Section 6.2 and in more detail in [13, 12]. However, we suggest three basic heuristics:

- Pragmatic: Do you want things to be classified under the given class automatically?
- Do you want to commit to a definition now? You can always return to the item and change it from primitive to defined later. In fact this is a key part of the methodology we advocate.
- Philosophical. Can you define it completely? There are many things which are "natural kinds" [9] which are virtually impossible to define completely, at least outside a highly technical context - e.g. people, kinds of animals, universities, languages, etc. These are usually best left primitive and merely described. Definition of natural kinds usually turn out to be long and incomplete. Therefore a useful heuristic is that if the definition is getting long or controversial, consider leaving the class as primitive.

4 Domain and Range Constraints and Other Axioms

4.1 Subclass (implication) axioms

OWL allows general expressions to be used in axioms. Like domain and range constraints, axioms are global and do not necessarily appear near the classes affected. On the one hand, the notion that "B is a subclass of A" means "B implies A" emphasizes the meaning of subsumption. On the other, it seems an odd way to express implication, if that is really what is intended. Hence care is required with the paraphrase and improved user interfaces for axioms for Protégé -OWL-CO-ODE are under development.

4.2 Domain and range constraints are axioms

Where most users encounter axioms is in domain and range constrains. In most languages domain and range constraints on properties are simply checked and generate errors if violated. In OWL they are axioms and are used in reasoning, with potentially far-reaching and unexpected effects. They may cause a class to be unsatisfiable or they may cause a class to be "coerced" to be subsumed by another class unexpectedly. For example, if we set the domain of hasTopping to be Pizza, it is the same as entering the axiom in Figure 8.

If we then add a Choc-icecream as shown in Figure 9 there are two possibilities. If Pizza and Icecream are not disjoint[12], then Choc-icecream will be classified as a kind

[12] assusming Choc-icecream is subsumed by Icecream

of Pizza. If, on the other hand, Icecream is disjoint from Pizza, then Choc-icecream will be unsatisfiable. In either case, the reason for the classifier's action is nowhere to be seen in the definition of Choc-icecream, Icecream or Pizza. It must be searched for in the domain restriction on hasTopping. In a large and complex ontology this can be difficult.

Domain constraint
hasTopping domain Pizza
Equivalent axiom
SubClassOf(restriction (hasTopping someValuesFrom owl:Thing) Pizza)
Paraphrase:
Having a topping *imples* being Pizza.

Fig. 8. An axiom stating the domain of hasTopping is Pizza

OWL:
Class (Choc-icecream partial
  restriction(hasTopping someValuesFrom Chocolate))
Paraphrase:
All Choc-icecream have *some* Chocolate topping.

Fig. 9. Description of Choc-icecream

After problems with open world reasoning, difficulties with domain and range constraints are the largest single source of errors and difficulty in our experience with new users of OWL. Furthermore, checking domain and range constraints is more complicated than in other languages, because a class may not satisfy a constraint prior to classification may be inferred by the classifier to do so. Usually, but not always, such behaviour is unintended and indicates an error. Current developments on the Protégé -OWL-CO-ODE tools include options to warn of easily recognised situations in which classification is likely to be affected by domain or range constraints.

5 Common logical issues

Most people learning to use OWL have little or no background in formal logic. In so far as possible, we limit what needs to be known. However, there are a series of issues which users find difficult and cause them to make errors:

1. "Only" (allValuesFrom) does not imply "some" (someValuesFrom).
2. Difference between the linguistic and logical usage of "and" and "or" often cause confusion.

OWL:
Class (EmptyPizza partial
  Pizza
    complementOf (restriction (hasToppings someValuesFrom owl:Thing)))
Paraphrase:
An empty pizza is *any* pizza which, *amongst other things*, does *not* have anything as topping.

Fig. 10. Definitions of an EmptyPizza.

3. Class definitions involving only allValuesFrom can be trivially satisfiable; this is usually the result of error but is easy to miss.
4. It is easy to confuse the representation of "some not" and "not some".

Each issue will be discussed in turn, although confusion over one is often compounded by confusion over another, particularly with respect to issues 1) and 2).

5.1 "Only" does not imply "some": Universal (allValuesFrom) restrictions can be satisfied trivially

The definition for an "EmptyPizza" (Figure 10) satisfies the definition for a Vegetarian pizza - it does not have any Meat or Fish toppings.

There is nothing inconsistent about a restriction that includes allValuesFrom owl:Nothing. It just means that, for the property in question, no values are allowed. Therefore, universal (allValuesFrom) restrictions can be "trivially satisfied" – *i.e.* satisfied by the trivial case in which there is no value at all for the property in question. The only way a universal (allValuesFrom) restriction can be made inconsistent is by there being some, *i.e.* at least one, value which contradicts it.

Note that by contrast, a restriction equivalent to someValuesFrom owl:Nothing is always inconsistent since the definition of owl:Nothing is that no value can be from owl:Nothing.

5.2 Linguistic vs Logical use 'and' and 'or'

In common linguistic usage, "and" and "or" do not correspond consistently to logical conjunction and disjunction respectively. This is a common problem familiar to everyone who uses query languages, the more advanced features of search engines or to anyone who programs. "Find all of the Pizzas containing Fish and Meat " is ambiguous as to whether the request is for pizzas containing both Fish and Meat or either Fish or Meat. In other contexts we disambiguate the expression to use disjunction for "and". In response to instruction to "Find all the meat pizzas and fish pizzas and mark them as spoilt", most people would look for all pizzas which contained either meat or fish. Common though this problem is, it often causes confusion in those learning OWL. Definitions such as the first one in Figure 11 are not uncommon.

5.3 Trivially satisfiable class definitions are easy to miss

Since owl:Nothing is equivalent to any contradiction, confusion over "and" and "or" can lead to definitions which are consistent but only trivially satisfiable. Since they are not flagged as unsatisfiable, such errors often go undetected for some time. Consider the definitions in Figure 11 which are not uncommon in new users' exercises. After running the classifier, newusers are surprised to find ProteinLoversPizza classified under VegetarianPizza as well as under MeatyPizza.

The rules for paraphrasing are designed to minimize these errors. If "A and B" is paraphrased to "both A and also B"[13] and "A or B" is paraphrased to "A and/or B", the confusion is reduced.

The frequency with which we have encountered these errors in practical workshops and modules has motivated debugging options which:

- Check at classification time for universal restrictions with unsatisfiable fillers
- Indicate all unsatisfiable fillers in red in the restriction definition pane even if the restriction, taken as a whole, is satisfiable[14].

OWL:
Class (ProteinLoversPizza complete
  Pizza
    restriction(hasTopping allValuesFrom (Meat and Fish)))
Paraphrase:
A ProteinLoversPizza is *any* Pizza that, *amongst other things*, has *only* topping that are *both* meat *and* also fish.

OWL:
Class (MeatyPizza complete
  Pizza
    restriction(hasTopping allValuesFrom Meat))
Paraphrase:
A MeatyPizza is *any* pizza which, *amongst other things*, has *only* toppings that are Meat.

Fig. 11. Incorrect definition of ProteinLoversPizza which is trivially satisfiable and hence classifies under both MeatyPizza and VegetarianPizza

5.4 Confusion of "some not" and "not some"

It is not uncommon for students to form definitions such as those in Figure 12. Many pizzas classify under VegetarianPizza_wrong since most contain some topping which is not Fish and also contain something which is not Meat. The paraphrase makes the error in the placement of the negation clear. One of the requirements for tools remains to making negation of restrictions as easy as negation of their fillers.

[13] We have even had suggestions for the stronger "simultaneously A and also B"
[14] The second is still in development at time of writing.

OWL:
Class(VegetarianPizzaV4_wrong complete
  Pizza
    restriction(hasTopping someValuesFrom not Meat)
    restriction(hasTopping someValuesFrom not Fish)
Paraphrase:
A vegetarian pizza is *any* pizza which, *amongst other things*, both has *some* topping which is *not* meat and also has *some* topping which is *not* fish.

Fig. 12. . Incorrect definition of vegetarian pizza confusing "some not ..." with "not some ..."

5.5 The benefits of clear definition - "What does it mean: to be a Pizza?"

Up to this point we have put no restriction on what counts as a Pizza. Do all pizzas have to have a base? toppings? Is a pizza base without a topping a Pizza? Is anything with a PizzaBase and PizzaToppings a Pizza? Can we completely define a Pizza?

There is no one right answer to these questions; they depend on our conceptualization of Pizzas. However, most users are reluctant to regard a bare base, or a pizza without a base,as a pizza. So the most common outcome is that shown in Figure 13.

Note that if this definition is used, then the ProteinLoversPizza in Figure 11 is unsatisfiable, because the restriction that all pizzas must have a topping contradicts *restriction*(hasTopping *allValuesFrom* (Meat *and* Fish). Entering definitions and early helps catch errors due to restrictions that would otherwise be trivially satisfiable.

OWL:
Class (Pizza partial
  restriction(hasBase someValuesFrom PizzaBase)
  restriction(hasTopping someValuesFrom PizzaTopping))
Paraphrase:
All pizzas, *amongst other things*, both have *some* base that is a Pizza base and also have *some* topping that is a Pizza topping.

Fig. 13. Description (partial definition) of a pizza

6 Patterns: Values, Value Types, and "Untangling"

6.1 Values and Value Types

The examples to this point have dealt with what various authors call "first class entities", "independent entities", or "sortals" [4, 16], and what we prefer to call "Self-Standing entities" [10] and the relations between them. These correspond roughly to nouns and verbs in ordinary language. However there are many modifiers or "refiners" - roughly adjectives and adverbs in ordinary language - to account for. In object oriented design these are often represented as "attributes" that are entirely internal to objects (as indicated by their appearing inside the box in UML diagrams). It comes as a surprise to

many newcomers to OWL that there is no corresponding distinction between "relation" and "attribute" in the formalism itself. The distinction is left to ontological patterns.

The requirements pattern for values and value types are that: [15]

1. There should be a functional property for each value type.
2. The values for each value type should be disjoint - it should not be possible for something to be Bland and also Hot.
3. The possible values for the value type are exhaustive - so that if we choose to say that the values for Spiciness are Bland, Mild, Medium, and Hot, then those are the only values.

Meeting these requirements requires a sequence of six operations:

1. Create a functional property, e.g. hasSpiciness
2. Create a subclass of value type, e.g. SpicinessValueType
3. Create the individual values as subclass of the value type, e.g. Bland, Mild, Medium, Hot
4. Make the values disjoint using a disjoint axiom
5. Make the values exhaustive by creating a subclass "covering" axiom, e.g. SpicinessVT subclass-of Bland or Mild or Medium or Hot
6. Set the range of the hasSpiciness property to the spiciness value type SpicinessValueType

Students have little trouble understanding these operations in principle, but the number of steps leaves many opportunities for mistakes in practice. One of the first requirements for the Protégé -OWL-CO-ODE interface was that it take users through the steps semi-automatically via a "wizard".

6.2 "Untangling"

The Ontology in Figure 1 is not, in fact, usually the first that students produce. Rather, most students initially produce something more like that shown in Figure 14.

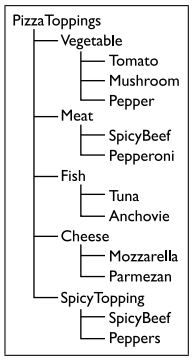


Fig. 14. Tangled First Draft Pizza hierarchy

We advocate a policy in which primitives form a skeleton of pure trees, *i.e.* have exactly one primitive parent. When multiple hierarchies appear in first drafts of the primitive hierarchy, as with SpicyTopping above, they must be *untangled – i.e.* an explicit characteristic found to differentiate the child concepts from all but one of the primitive parents. There are many advantages to this policy, which corresponds closely to traditional Aristotelian notions of "differentia", but the overwhelming practical engineering

[15] Some may question that we represent values as classes rather than individuals. On philosophical grounds, there is an argument to be made for either choice. However, existing reasoners for OWL cannot deal with the all required reasoning using individuals. Therefore, representing values as classes the only practical alternative. (We would also advocate the use of classes on philosophical grounds, but that is an argument for another paper.)

advantage is that it makes it possible to make ontology more modular [10] because the hierarchy of primitives can be split into disjoint branches at any point.

"Untangling" can involve either ordinary relations or value types, but is most simply illustrated with value types. Consider the above SpicenessVT along with an analogously defined FatcontentVT, with values LowFat, MediumFat, HighFat.

It is then only necessary to assign the correct values to the various ingredients and to replace any primitives such as SpicyTopping with a corresponding defined class SpicyTopping. A visualisation of the pre-classification hierarchy, which is a strict tree, and the post classification polyhierarchy including LowFatTopping are shown in Figure 15.

Again, the mechanics can be tedious, so an extension of the value type wizard is being developed to guide users through the process and to warn of possible errors. In ontologies of any size, it is rare that the classifier does not infer new subsumptions missed by students when they created the hierarchy manually.

6.3 Converting primitive classes to defined classes

There is an unexpected complication to the above scenario. In the course of untangling, primitive classes are reformulated as defined classes. However, there may be restrictions in the description[16] of the primitive class that do not form part of its new definition but remain merely necessary implications. For example, the ontology might have contained a restriction that hot ingredients were unsuitable for children as shown in Figure 16. Clearly this restriction is not part of the necessary and sufficient conditions defining SpicyTopping, rather it is a further necessary condition to be inferred whenever something is found to be a SpicyTopping.

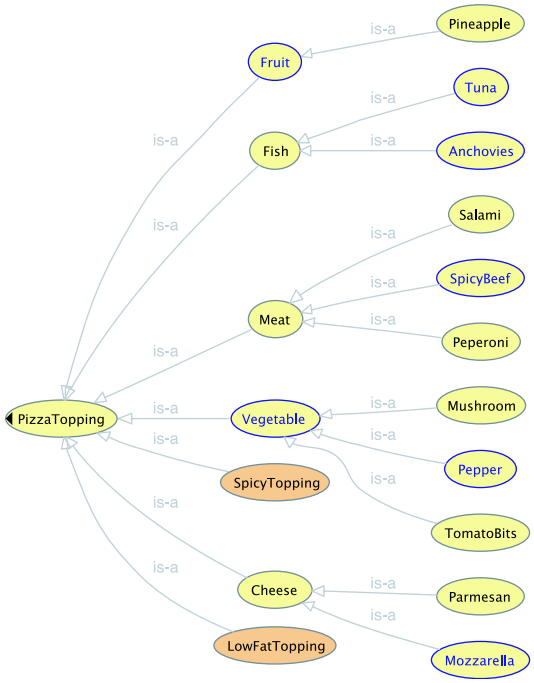
Such residual necessary implications must be converted to subclass axioms, as shown in Figure 16, which requires significant syntactic change. These changes are reflected in the OilEd interface, where subclass axioms are entered on a separate "tab" and not visible when the main class definition tab is selected. A major goal of the Protégé -OWL-CO-ODE interface has been to make this transition easy by placing necessary and sufficient conditions and necessary conditions in adjacent subpanes and allowing drag and drop/cut and paste operations between them (See Figure 18).

7 Summary

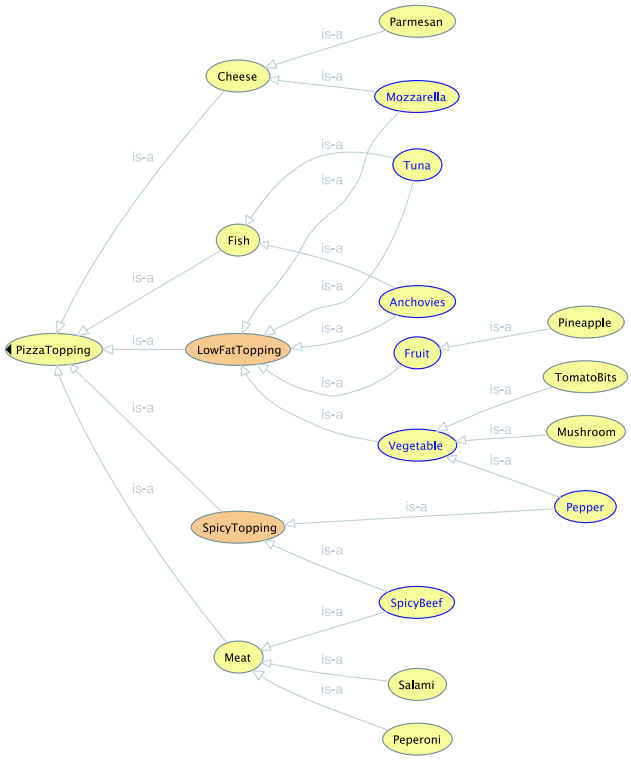
Using any logic-based ontology language presents new users with significant problems, often made worse by details of the language and user interface. The open world assumption and the representation of domain and range constraints as logical axioms run counter to most new users' experience.

In this paper we have described errors commonly made in a series of workshops, tutorials, and teaching modules. A prime goal is to help users understand the precise meaning of OWL-DL through questions and paraphrases. It is a difficult task in natural language generation to specify the paraphrases completely, but the basic forms are summarized in Table 1 with their rationale, and the examples throughout the paper should

[16] For this reason we dislike the use of the phrase "partial definition" and prefer the term "description".



(a) Initial hierarchy



(b) Classified hierarchy

Fig. 15. Pizza hierarchy

OWL:
Class (SpicyTopping partial
not (restriction(isSuitable someValuesFrom SmallChild)))
Paraphrase:
All SpicyToppings are *not* suitable for any small child.
OWL:
Class (SpicyTopping complete
PizzaTopping
restriction(hasSpiciness someValuesFrom Spicy))
SubclassOf (SpicyTopping
not (restriction(isSuitable someValuesFrom SmallChild)))
Paraphrase:
A SpicyTopping is *any* pizza topping which has spiciness value Spicy; all Spicy toppings are not suitable for any small child.

Fig. 16. Conversion of a primitive class with a restriction to a defined class in which the restriction does not form part of the definition requires that the restriction be reformulated as an axiom.

1. Always paraphrase a description or definition before encoding it in OWL, and record the paraphrase in the comment area of the interface.
2. Make all primitives disjoint - which requires that primitives form trees
3. Use *someValuesFrom* as the default qualifier in restrictions
4. Be careful to make defined classes defined – the default is primitive. The classifier will place nothing under a primitive class (except in the presence of axioms /domain/range constraints)
5. Remember the open world assumption. Insert closure restrictions if that is what you mean.
6. Be careful with domain and range constraints. Check them carefully if classification does not work as expected.
7. Be careful about the use of "and" and "or"(*intersectionOf, unionOf*)
8. To spot trivially satisfiable restrictions early, always have an existential (*someValuesFrom*) restriction corresponding to every universal (*allValuesFor*) restriction, either in the class or one of its*superclasses* (unless you specifically intend the class to be trivially satisfiable).
9. Run the classifier frequently; spot errors early

Fig. 17. Brief summary of guidelines

make their usage clear. A brief summary of guidelines for avoiding the most common pitfalls in building ontologies in OWL-DL are given in Figure 17.

This experience is also strongly influencing the design of new user interfaces. A screen shot of the basic Protege-OWL class screen is shown in Figure 18. Preliminary experience is encouraging, but to what extent these new features reduce new users' confusion remains to be proven in practice.

OWL definition	Paraphrase	Rationale
Class(Thing partial ...	All Things ...	Primitive vs Defined
Class(Thing complete parent...	A Thing is any Parent that ...	Defined vs Primitive
(add to all descriptions and definitions)	...amongst other things...	Open world hypothesis
allValuesFrom	only	often misunderstood
someValuesFrom	some	brevity and clarity
and	both... and also	minimise logic errors
not(... and ...)	not all of / not both ... and also	minimise logic errors
not (... or ...)	neither ... nor ...	minimise logic errors
someValuesFrom not	has some ... that are not ...	minimise logic errors
not (someValuesFrom ...)	does not have ... any	minimise logic errors
AllValuesFrom not	has ...no ... / has only ...that are not ...	minimise logic errors
not (allValuesFrom ...)	does not have ... only	minimise logic errors
SubclassOf(A , B)	A implies B	clarify use of subclass for implication

Table 1. Summary of paraphrases. Examples in Figures throughout paper

Acknowledgements.

This work was supported in part by the CO-ODE project funded by the UK Joint Information Services Committee and the HyOntUse Project (GR/S44686) funded by the UK Engineering and Physical Science Research Council and by 21XS067A from the National Cancer Institute. Special thanks to all at Stanford Medical Informatics for their continued collaboration and comments and to the other members of the ontologies and metadata group at Manchester for their contributions and critiques.

References

1. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. Oiled: a reason-able ontology editor for the semantic web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence*, Lecture Notes in Computer Science, pages 396–408. Springer-Verlag.
2. R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, L.A. Resnick, and A. Borgida. Living with classic: When and how to use a kl-one-like language. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the representation of knowledge*, pages 401–456. Morgan Kaufmann, San Mateo, CA, 1991. Good overview. Particularly good straw man section on the arbitrariness of class-instance division.
3. N. Guarino. Understanding, building and using ontologies. *International Journal of Human Computer Studies*, 46:293–310, 1997.

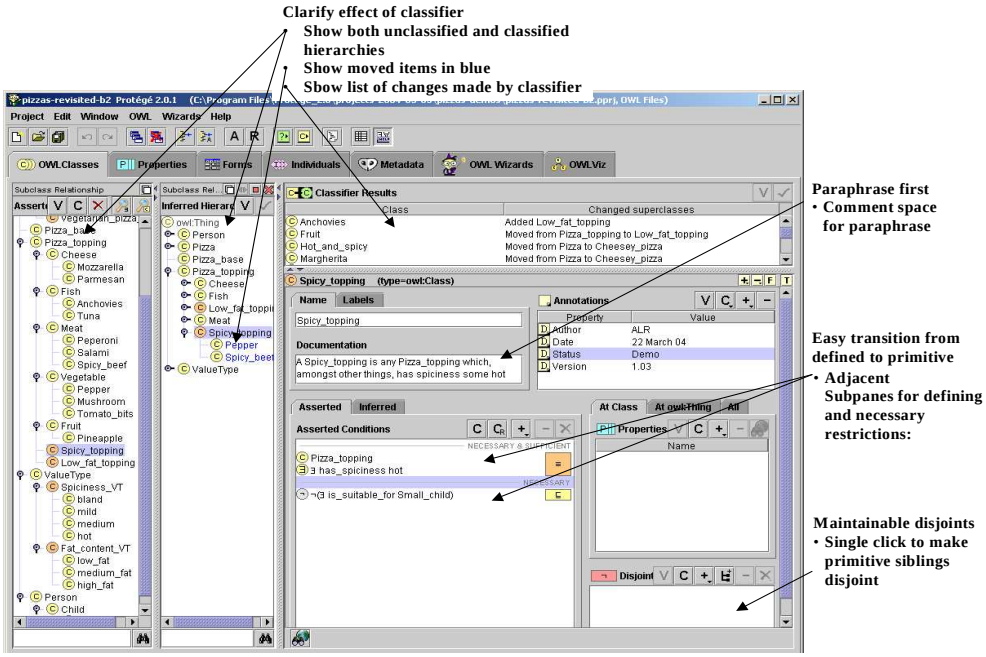


Fig. 18. Main pane of new Protégé-OWL-CO-ODE Interface indicating how requirements identified are met

4. N. Guarino and C. Welty. Towards a methodology for ontology-based model engineering. In *ECOOP-2000 Workshop on Model Engineering*, Cannes, France, 2000.
5. H. Knublauch and O. Dameron a M. A. Musen. Weaving the biomedical semanticweb with the protégé owl plugin. In *Proceedings of KR-MED2004: International Workshop on Formal Biomedical Knowledge Representation*.
6. D.L. McGuinness and A. Borgida. Explaining subsumption in description logics. In *International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 816–821, 1995.
7. N. F. Noy and D.L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics, 2001.
8. P. Patel-Schneider, P. Hayes, and I. Horrocks (Editor). Owl web ontology language semantics and abstract syntax. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/, Feb, 2004.
9. H. Putnam. The meaning of 'meaning. In Keith Gunderson, editor, *Language, Mind and Knowledge*, Minnesota Studies in the Philosophy of Science VII, pages 131–193z. University of Minnesota Press, Minneapolis, MN, 1975.
10. A. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In John Genari, editor, *Knowledge Capture 2003*, pages 121–128, Sanibel Island, FL, 2003. ACM.
11. A. Rector, W. Solomon, W. Nowlan, and T. Rush. A terminology server for medical language and medical information systems, 1994.
12. A. L. Rector. Normalisation of ontology implementations: Towards modularity, re-use, and maintainability. In *Proceedings Workshop on Ontologies for Multiagent Systems (OMAS) in conjunction with European Knowledge Acquisition Workshop*, 2002.

13. Alan L. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In *Proceedings of the international conference on Knowledge capture*, pages 121–128. ACM Press, 2003.
14. S. Staab and A. Maedche. Ontology engineering beyond the modeling of concepts and relations. In R.V. Benjamins, A. Gomez-Perez, N. Guarino, and M. Uschold, editors, *ECAI 2000. 14th European Conference on Artificial Intelligence; Workshop on Applications of Ontologies and Problem-Solving Methods*, 2000.
15. M. Uschold and M. Gruninger. Ontologies: principles, methods and applications. *Knowledge Engineering Review*, 11(2), 1996.
16. C. Welty and N. Guarino. Supporting ontological analysis of taxonomic relationships. *Data and Knowledge Engineering*, 39(1):51–74, 2001.