OntOWLClean: Cleaning OWL ontologies with OWL

Chris WELTY[1]
IBM Watson Research Center, NY, USA

Abstract. OWL is now very widely used for ontology development and several attempts have been made at incorporating OntoClean analysis into an OWL-based tool. I present here an OWL ontology representing the basic OntoClean distinctions, and a tool and methodology for applying it to OWL ontologies. I briefly touch on the semantic issues implied by using OWL Full syntax to characterize the OntoClean meta-properties as properties of OWL Classes, and how that was solved to employ an off-the-shelf OWL DL reasoner to check the OntoClean constraints on the taxonomy.

Keywords. OWL, OntoClean.

Introduction

OWL is now by far the most common language used to encode formal ontologies for information systems, and after two years is still the only standard. While adoption has been somewhat slow, open-source and free tool support is now available for reasoners, editors, browsers, syntax checkers, viewers, explanation, and numerous others. In some cases the ontologies being developed in OWL are extremely lightweight and scruffy, and do not need extensive analysis or maintenance. In other cases, however, correctly conveying the meaning of terms in an ontology is important and the demand for ontology quality is high, and this has led people to analysis tools such as OntoClean.

OntoClean was first published in a series of papers in 2000 [10,11,12,13]. Since them it has been widely cited, used, and criticized. One of the most common criticisms of OntoClean as a tool for analyzing ontologies is that it is too complicated. Indeed, the semantics of the OntoClean meta-properties were presented using S5 Modal Logic, and many ontology authors are unable to penetrate the axioms. Another, likely consequent, criticism has been that it is not always clear how to assign the meta-properties, and that agreement between even experienced ontology designers is quite low. The most common question we receive by email is, "Is X rigid?" Finally, the use of S5 modal logic for expressing the semantics of OntoClean has led some to conclude, incorrectly, that the computational complexity of any OntoClean analysis tool is too high. I will address each of these issues in this paper.

[1] Corresponding Author: IBM Watson Research Center; 19 Skyline Dr.; Hawthorne, NY. 12540; USA. E-mail: welty@us.ibm.com

1. Understanding the meta-properties

In one recent article, it was claimed that "multiple ontologists assign different meta-properties to the same concept" [1], based on the interesting evaluation work done by Völker et al [2]. This is impossible, of course, the same concept cannot have different meta-properties, in reality the disagreement is due to the same term representing different concepts to different people. In fact, this is precisely the main value of OntoClean, to expose subtle differences in different people's understanding of concepts represented in an ontology. This is the very crux of the issue, a class cannot be both rigid and non-rigid, cannot be both a sortal and non-sortal, etc. If two different analysts adopt conflicting meta-property assignments then they have different interpretations. It is critical to expose such differences as ontologies are supposed to capture a "common understanding" or "shared meaning" of terms.

In our own experience, we have encountered this phenomenon every time we applied the OntoClean analysis, including the very first OntoClean example, which is reproduced in OWL below. We had developed the example drawing specific poorly chosen subsumption relations from existing ontologies and lexical resources, adding a few of our own. The major objective was for the taxonomy to appear reasonable *prima facie*, but to break down under the close scrutiny of the meta-properties. Once we had chosen our classes and taxonomy, we set out to independently to assign the meta-properties. For the class Food, I had chosen Rigid, Independent, Sortal, and Unity. Nicola chose Anti-Rigid, Dependent, Sortal, and Anti-Unity. In the ensuing discussion, I held up a piece of chocolate as an instance of Food and suggested that the only way it could cease to be food would be if it ceased to exist, which I then demonstrated. Nicola's position was that the chocolate was not Food until I ate it.

The key here is not to try and determine who is right – we were both right in the sense that we had applied the meta-properties correctly to reflect what we each thought Food meant – we had to recognize that there are (at least) two ways of thinking about what Food means. I had thought of Food as basically the class of all human edible objects, whereas Nicola thought of Food as a role in a relation, anything can in that sense be Food if it is eaten.

The next step in ontology design would be to determine if both the possible meanings are needed and select one or distinguish them as different classes. In this case we agreed to use Nicola's meaning and dropped the other.

Finally, note that OntoClean doesn't tell us what the class Food means, it allows us to clarify some part of the meaning. Some people have come away from the papers thinking the example ontology is supposed to be authoritative, and that "according to OntoClean" the Person class is Rigid. Thus we so often get questions from people about whether some class in their ontology is rigid or not. The answer is always that *it depends on what you mean*. It's rather a frustrating misunderstanding as the papers repeat the point often: the examples are only examples and it is certainly possible to have an ontology with a class called Person that is not rigid, or (as above) a class called Food that is. What OntoClean does tell us is that these classes mean something different, despite having the same names, because they have different meta-properties.

What has become clear from our experiences is that people need a standard and reusable way to share the meta-property assignments in order to communicate them, discover their disagreements, and take appropriate action.

2. Previous OntoClean Implementations

The semantics of OntoClean were such that a few simple constraints on taxonomic organization fell out:

- A rigid class cannot be a subclass of an anti-rigid class
- A class with unity cannot be a subclass of a class with anti-unity
- All subclasses of a sortal are sortals (sortals are classes that have an identity criterion)
- All subclasses of a dependent class are dependent

The primary challenge in implementing any tool to support OntoClean is that it requires reasoning about the classes and their taxonomic relationships. In logical systems, the taxonomic relationship is often not expressed as a relation but by using implication. In OWL-DL, it *is* expressed as a relation (`rdfs:subClassOf`), but the formal semantics do not treat it as such. Thus the relation (in OWL DL) is privileged and cannot be further refined or restricted (as we would like).

The full version of OWL does allow one to further refine the axiomatization of the subClassOf relation, but such reasoning in general is undecidable [3] and experience with reasoners that implement it is that they are very slow.

In two particular implementations of OntoClean support, the OntoClean constraints were simply encoded in systems that treated the classes as instances and did not consider the instances of the classes themselves: the tools merely pointed out inconsistencies in the taxonomic structure implied by the constraints (e.g. a rigid class subsumed by an anti-rigid class). Thus each class was represented by an individual that "stood in" for it in order to support the OntoClean reasoning. This is analogous to "punning", a technique in logic in which objects are given multiple interpretations depending on certain syntactic or semantic contexts. In a possible new OWL dialect that supports "meta modeling" (i.e. the ability to reason about classes of classes), punning has been proposed as a way to keep the semantic domains of classes and individuals disjoint [4], which is a well understood way to promote decidability, and is a requirement in all OWL-DL reasoners today.

The first OntoClean implementation [5], used LISP-Classic and has not survived. It required generating an a-box version of an ontology which was in no way connected to the original (other than having visibly similar names).

The second implementation [6] uses Protégé PAL constraints and works well within the Protégé system, but the meta-property annotations can not be shared with others who don't use Protégé. Also, it does not work seamlessly with OWL Ontologies, being designed more for use with the built-in OKBC language Protégé was originally designed for.

3. Implementing the OntoClean Constraints in OWL

In order to develop a standard representation of the OntoClean meta-properties and their constraints, and furthermore to enable today's tools to process those constraints in a standard way, I have developed an OWL-DL ontology of the OntoClean meta-properties and constraints, and a simple tool that creates an instance view of an OWL class hierarchy.

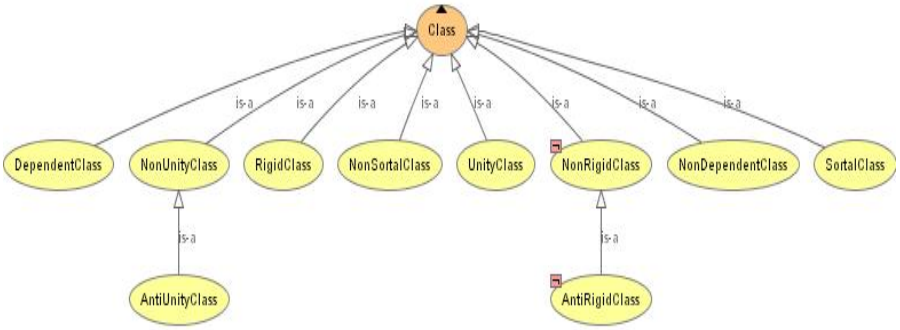


Figure 1. The OntoClean taxonomy of Classes

The OntoClean ontology is shown in Figure 1 and listed below in the OWL abstract syntax (this and the other OWL ontologies shown in this paper are available at the URLs referenced in the Ontology element, with added information like comments):

```
Namespace(rdfs = <http://www.w3.org/2000/01/rdf-schema#>)
Namespace(owl  = <http://www.w3.org/2002/07/owl#>)
Namespace(oc = <http://www.ontoclean.org/ontoclean-dl-v1.1.owl#>)

Ontology( <http://www.ontoclean.org/ontoclean-dl-v1.1.owl>

 Annotation(owl:versionInfo "1.1"@en)

 ObjectProperty(hasSubClass inverseOf(oc:subClassOf)
                domain(Class) range(Class))
 ObjectProperty(oc:subClassOf Transitive inverseOf(hasSubClass)
                domain(Class) range(Class))

 Class(oc:Class complete
  intersectionOf(unionOf(NonSortalClass SortalClass)
                 unionOf(RigidClass NonRigidClass)
                 unionOf(UnityClass NonUnityClass)
                 unionOf(NonDependentClass DependentClass)))

 Class(RigidClass partial oc:Class
   restriction(oc:subClassOf
               allValuesFrom(complementOf(AntiRigidClass))))
 Class(NonRigidClass partial oc:Class)
 Class(AntiRigidClass partial NonRigidClass)
 DisjointClasses(RigidClass NonRigidClass)

 Class(SortalClass partial oc:Class
   restriction(hasSubClass allValuesFrom(SortalClass)))
 Class(NonSortalClass partial oc:Class)
 DisjointClasses(SortalClass NonSortalClass)

 Class(UnityClass partial oc:Class
   restriction(oc:subClassOf
               allValuesFrom(complementOf(AntiUnityClass))))
 Class(NonUnityClass partial oc:Class)
 Class(AntiUnityClass partial NonUnityClass)
 DisjointClasses(UnityClass NonUnityClass)

 Class(DependentClass partial oc:Class
   restriction(hasSubClass allValuesFrom(DependentClass)))
```

```
 Class(NonDependentClass partial oc:Class)
 DisjointClasses(DependentClass NonDependentClass)
)
```

The ontology is quite simple and well within the expressive limitations of OWL-DL and any reasoner I have tried. Pellet [8] lists the ontology as requiring *ALCR*+ expressivity. Note that it is not necessary to encode the modal axioms, just the meta-properties as classes with proper disjointness and constraints on the `oc:subClassOf` relation. In particular, the `oc:subClassOf` relation is defined to be transitive and have an inverse, and the four OntoClean constraints above are expressed as "local range constraints" on the `oc:subClassOf` relation.

The semantics of OntoClean, however, are not expressed in this ontology alone, as one must understand to use it that the domain of the ontology is another ontology; the instances of these classes are OWL classes themselves. To complete the semantics of OntoClean, I have defined a separate ontology that contains two mapping axioms, shown below, that equate `oc:Class` and `oc:subClassOf` to their OWL equivalents:

```
Namespace(rdfs = <http://www.w3.org/2000/01/rdf-schema#>)
Namespace(owl  = <http://www.w3.org/2002/07/owl#>)
Namespace(oc = <http://www.ontoclean.org/ontoclean-dl-v1.1.owl#>)

Ontology( <http://www.ontoclean.org/ontoclean-full-v1.owl>

 Annotation( owl:imports
             <http://www.ontoclean.org/ontoclean-dl-v1.1.owl>)

 SameIndividual(oc:Class owl:Class)
 SameIndividual(oc:subClassOf rdfs:subClassOf)
)
```

This ontology is kept separate for several reasons: I have not yet found a tool that handles it properly (since it uses the OWL and RDFS vocabulary which these tools treat as immutable), and more importantly keeping the basic ontology within DL enables use of existing reasoners to check the OntoClean constraints.

Without support for this kind of meta-modeling, it was necessary to create a simple tool that takes as input an OWL ontology and generates a "view" of that ontology in which all classes are individuals and all `rdfs:subClassOf` relations are `oc:subClassOf` relations, again rather like punning except done explicitly. An important and perhaps subtle point here is that the generated view does not contain new objects, it uses the URIs of the classes in the original ontology as the identifiers for the individuals. The trick is not to import the ontology in which these URIs are labeled as classes, and thus any tool, including reasoners, will load and reason about the data.

Figure 2 shows the infamous OntoClean example, before cleaning, and it is listed below in the OWL abstract syntax:

```
Namespace(rdf  = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Namespace(xsd  = <http://www.w3.org/2001/XMLSchema#>)
Namespace(rdfs = <http://www.w3.org/2000/01/rdf-schema#>)

Ontology( <http://www.ontoclean.org/example-dirty-v1.owl>

 Class(Agent partial Entity)
 Class(AmountOfMatter partial Entity)
 Class(Animal partial Agent PhysicalObject LivingBeing)
 Class(Apple partial Fruit Food)
```

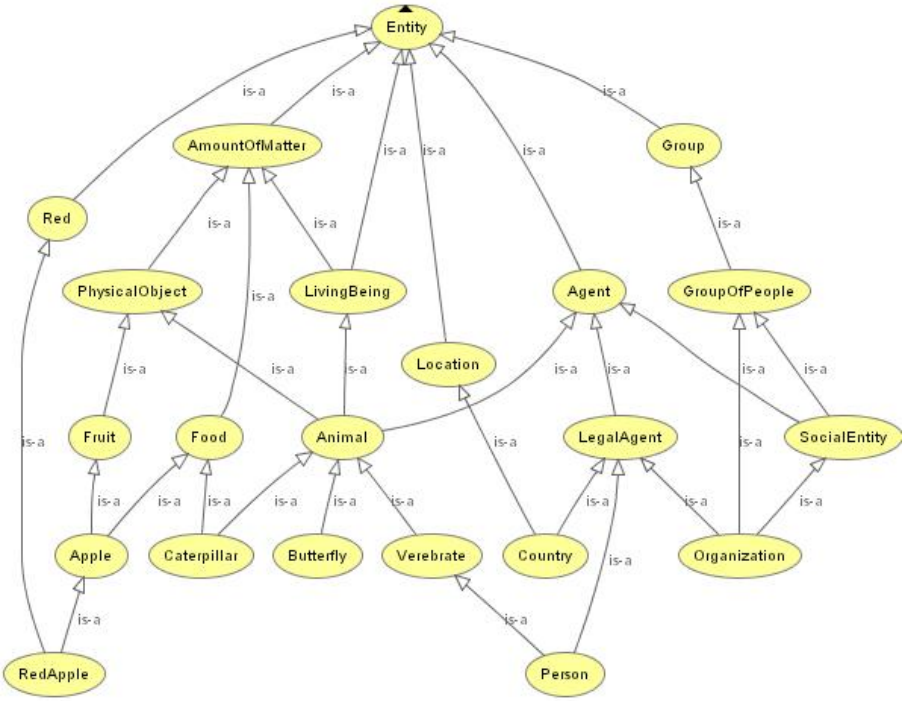


Figure 2. The example before cleaning.

```
 Class(Butterfly partial Animal)
 Class(Caterpillar partial Animal Food)
 Class(Country partial LegalAgent Location)
 Class(Entity partial)
 Class(Food partial AmountOfMatter)
 Class(Fruit partial PhysicalObject)
 Class(Group partial Entity)
 Class(GroupOfPeople partial Group)
 Class(LegalAgent partial Agent)
 Class(LivingBeing partial Entity AmountOfMatter)
 Class(Location partial Entity)
 Class(Organization partial GroupOfPeople SocialEntity LegalAgent)
 Class(Person partial LegalAgent Verebrate)
 Class(PhysicalObject partial AmountOfMatter)
 Class(Red partial Entity)
 Class(RedApple partial Apple Red)
 Class(SocialEntity partial Agent GroupOfPeople)
 Class(Verebrate partial Animal)
)
```

The result of generating the instance view of the example is shown below:

```
Namespace(ont  = <http://www.ontoclean.org/example-dirty-v1.owl#>)
Namespace(oc = <http://www.ontoclean.org/ontoclean-dl-v1.1.owl#>)

Ontology( <http://www.ontoclean.org/example-dirty-v1-punned.owl>

 Annotation( owl:imports
             <http://www.ontoclean.org/ontoclean-dl-v1.1.owl>)

 Individual(ont:Agent type(oc:Class) value(oc:subClassOf ont:Entity))
 Individual(ont:AmountOfMatter type(oc:Class)
            value(oc:subClassOf ont:Entity))
 Individual(ont:Animal type(oc:Class)
```

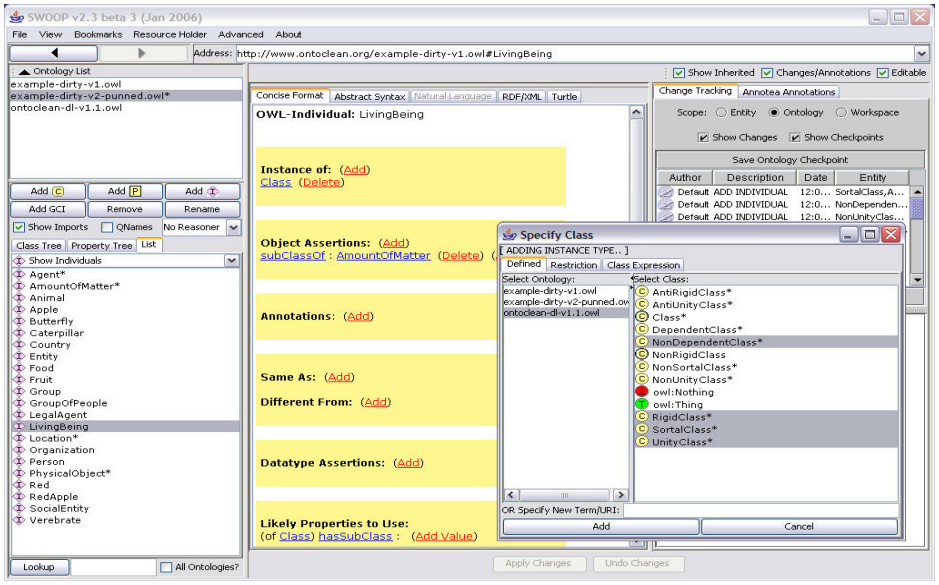
```
            value(oc:subClassOf ont:PhysicalObject))
  Individual(ont:Apple type(oc:Class) value(oc:subClassOf ont:Fruit))
  Individual(ont:Butterfly type(oc:Class)
            value(oc:subClassOf ont:Animal))
  Individual(ont:Caterpillar type(oc:Class)
            value(oc:subClassOf ont:Animal))
  Individual(ont:Country type(oc:Class)
            value(oc:subClassOf ont:LegalAgent))
  Individual(ont:Entity type(oc:Class))
  Individual(ont:Food type(oc:Class)
            value(oc:subClassOf ont:AmountOfMatter))
  Individual(ont:Fruit type(oc:Class)
            value(oc:subClassOf ont:PhysicalObject))
  Individual(ont:Group type(oc:Class) value(oc:subClassOf ont:Entity))
  Individual(ont:GroupOfPeople type(oc:Class)
            value(oc:subClassOf ont:Group))
  Individual(ont:LegalAgent type(oc:Class)
            value(oc:subClassOf ont:Agent))
  Individual(ont:LivingBeing type(oc:Class)
            value(oc:subClassOf ont:AmountOfMatter))
  Individual(ont:Location type(oc:Class)
            value(oc:subClassOf ont:Entity))
  Individual(ont:Organization type(oc:Class)
            value(oc:subClassOf ont:LegalAgent))
  Individual(ont:Person type(oc:Class)
            value(oc:subClassOf ont:LegalAgent))
  Individual(ont:PhysicalObject type(oc:Class)
            value(oc:subClassOf ont:AmountOfMatter))
  Individual(ont:Red type(oc:Class) value(oc:subClassOf ont:Entity))
  Individual(ont:RedApple type(oc:Class) value(oc:subClassOf ont:Red))
  Individual(ont:SocialEntity type(oc:Class)
            value(oc:subClassOf ont:Agent))
  Individual(ont:Verebrate type(oc:Class)
            value(oc:subClassOf ont:Animal))
)
```

Again, take specific note that the ontology does not define any identifiers of its own, each individual is labeled with tags from other ontologies. In the context of FOIS it is worth commenting that, strictly speaking, this is not an ontology at all, just a collection of axioms. However the OWL definition of ontology is a syntactic one – an RDF file that has the `owl:ontology` element in it. In addition, from a semantic web perspective it is not really well-defined what this ontology "means" – should the meaning include the complete definitions of the objects referenced? In practice, most OWL tools do not "follow hyperlinks" to external URIs, and take the "import" annotation to be the only such directive. In this case, every tool I tested will load only the individual view and the imported OntoClean ontology.

4. OntoCleaning using existing tools

The most prominent tools for OWL are Protégé [9] and Swoop [7]. This section provides an informal evaluation of each for the purposes of supporting OntoClean.

Figure 3. Assigning meta-properties in SWOOP.



4.1. Assigning and Checking the MetaProperties

When the individual view is loaded into SWOOP, assigning the meta-properties is done by going to the instance list tab in the left pane and selecting an individual view of one of the classes. Then, in the individual description, click on the "Add" link next to the "Instance Of" heading, which brings up a dialog that lets you select as many classes as desired. Figure 3 shows assigning meta-properties in SWOOP.

When the individual view is loaded into Protégé, the meta-properties can be assigned by switching to the "Individuals" Tab and then selecting one of the inviduals. On the bottom of the individual list is a pane that allows you to add "Asserted Types". Protégé supports dragging an individual from the middle individual list pane to a class on the left class list pane, however this moves the individual rather than copying it so you can only assign one meta-property this way. The rest must be done using the add operation. Figure 4 shows assigning of meta-properties in Protégé.

The individual view in the OWL abstract syntax with all the meta-properties assigned for the example is shown in the appendix.

Both SWOOP and Protégé seem to assume that individuals are defined within an ontology, and do not let you "browse" the individuals in the case where their URIs are external. Protégé is general does not appear to be particularly web-savvy, and does not provide this functionality in general, but SWOOP typically provides clickable links for any URI referenced in an ontology that allows you to, at the least, open up a browser with the URI. This capability is not, however, provided for individuals.

Protégé displays the names of individuals with their full namespaces, but in Pellet they are shown with the namespace stripped off. So rather than `ont:Agent` in Pellet we see just `Agent`. This would be a problem if the analysis was being done on a merge between two ontologies that used the same short name for a class, and visually the namespace prefix helps reinforce that these are externally defined objects.

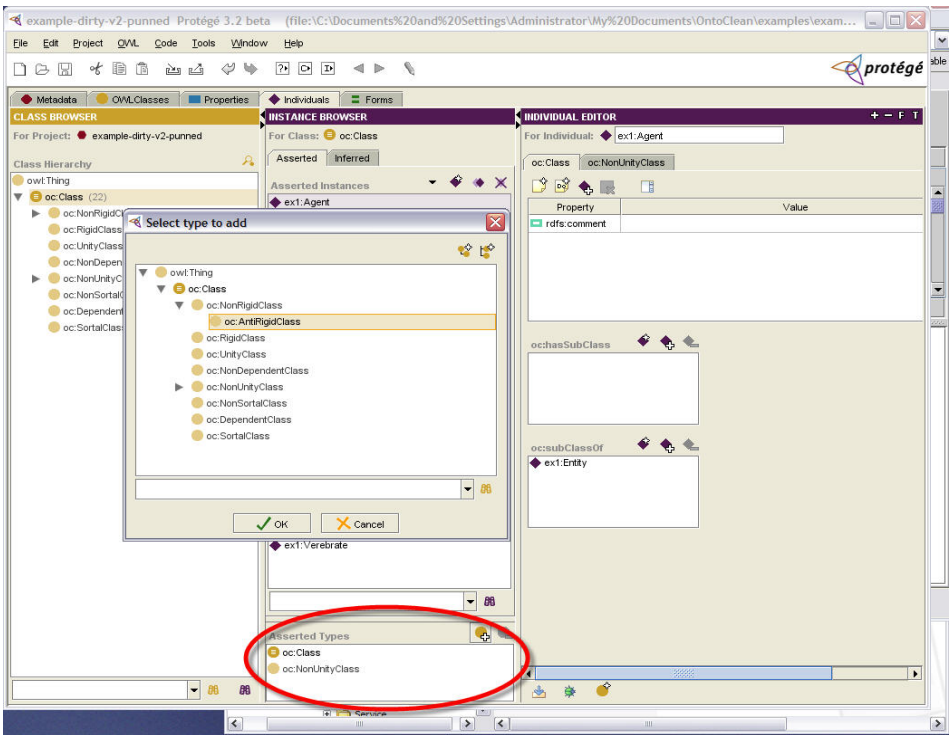


Figure 4. Assigning meta-properties in Protégé.

In general, the SWOOP interface is easier to use for the assignment of meta-properties because it allows you to add all four at once, whereas Protégé requires you assign each meta-property for each individual separately. Dragging seemed a more convenient way to do the assignment, if it could have been used for all four.

In either system, one can assign all the meta-properties and then check the constraints, or turn on a reasoner (Pellet for SWOOP and Racer for Protégé) and have it check as you assign them. There are basically two kinds of violations to be checked for, disjointness (e.g. assigning both Dependent and non-Dependent to a single class) and constraints on the subclass relation (e.g. an anti-rigid class subsuming a rigid one). Technically, all constraints end up being disjointness violations, the distinction is whether the violation arose from the class hierarchy or a primitive assignment.

The interface in Protégé is far better at indicating a meta-property violation, in SWOOP there is no visual indication of a problem, one must know to click on the ontology in the "Ontology List" pane to see if there is a problem. However, in Protégé (really, in Racer) the explanation of the problem is of no use at all, it basically tells you that something is wrong, but not where. Thus you are forced to assign the meta-properties one at a time and run the check after each.

In SWOOP, when you discover a problem the explanation tells you precisely the problem. One can imagine the explanations running rather long if there is a deep taxonomy and a violation occurs between two classes that are very distant, but in general the explanation facility in SWOOP does precisely what is needed, with very little extraneous information. An example of an explanation is shown in Figure 5.

Figure 5. Explaining constraint violations in SWOOP.

4.2. Fixing the Taxonomy

When a violation of the OntoClean constraints is found, the user should either reconsider the meaning of the class and reassign the meta-properties to be consistent, or change the class hierarchy. One can easily experiment with reassignment of the meta-properties in the tools, but if the taxonomy is to be changed the process is somewhat roundabout.

One can experiment in the view to get the hierarchy right by changing the values of the `oc:subClassOf` relation, however the user must keep track of these changes and then manually change them in the original ontology. It would seem that the simple tool used to generate the individual view should be able to "go the other way", i.e. generate the `rdfs:subClassOf` relations from the `oc:subClassOf` relations in the view, but this does not work in general. The problem is twofold:

The generated individual view is a view in a very real sense, it is a monotonic *addition* to the original ontology, it simply adds rdf:type axioms. Any changes to this view must be interpreted the same way, and there is no way to "retract" an assertion in OWL by adding axioms. One can make it inconsistent, but not retract.

The view file itself is not the original ontology in a different form, in particular it does not include properties, nor any axioms on properties, etc., thus a new version of the ontology generated from a modified view would have only the class names and the new subClassOf values.

The simplest way to make changes is to keep an editor open on the original ontology while working in the view. Make changes in the original and regenerate the

view. This does not take very many extra steps than if the changes were made in the view directly, and in theory either Protégé or SWOOP could add buttons that run the view generator to make it even easier.

In SWOOP one can have multiple ontologies open (in the ontology List) so switching between them is easier than in Protégé, which needs to be "run" separately for each ontology open. One can run Pellet separately for each ontology as well, obviously, and this makes it possible to actually view both ontologies on the screen at the same time.

5. Conclusion

One of the obstacles to successfully using OntoClean today is the lack of ability to communicate the meta-property assignments in a standard way, and the lack of tooling to support this assignment and validation of the constraints. I have presented here an OWL ontology of the OntoClean meta-properties that, along with a simple view generator, can be used along with existing off-the-shelf OWL tools to assign the meta-properties and validate the taxonomy. I also briefly surveyed the use of SWOOP and Protégé for meta-property assignment with this ontology, and Pellet and Racer to validate the constraints. The ontology and examples are available on the web.

References

[1] Sleeman, Derek. and Quentin Reul. 2006. CleanONTO: Evaluating Taxonomic Relationships in Ontologies**.** *2006 Workshop on Evaluating Ontologies* (EON 2006). Edinburgh.

[2] Völker, Johanna , Denny Vrandecic, and York Sure. 2005. Automatic Evaluation of Ontologies (AEON). *International Semantic Web Conference 2005* (ISWC-05): 716-731. Galway.

[3] Motik, Boris. 2005. On the Properties of Metamodeling in OWL. *International Semantic Web Conference 2005* (ISWC-05). Galway.

[4] Patel-Schneider, Peter. 2005. *The OWL 1.1 Extension to the W3C OWL Web Ontology Language.* http://www-db.research.bell-labs.com/user/pfps/owl/overview.html

[5] Welty, Chris and Nicola Guarino. 2001. Support for Ontological Analysis of Taxonomic Relationships. *J. Data and Knowledge Engineering.* **39**(1):51-74. October, 2001.

[6] Noy, Natasha. 2003. *The OntoClean Ontology in Protégé.* http://protege.stanford.edu/ontologies/ontoClean/ontoCleanOntology.html

[7] Kalyanpur, Aditya, Bijan Parsia, Evren Sirin, Bernardo Cuenca-Grau, and James Hendler. 2005. Swoop - a web ontology editing browser. *Journal of Web Semantics*, 4(1).

[8] Sirin, Evren, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. In Press. Pellet: A practical owl-dl reasoner. Submitted for publication to *Journal of Web Semantics.*

[9] Knaublock, Holger. 2003. *Protégé-OWL*. http://protege.stanford.edu/overview/protege-owl.html

[10] Guarino, Nicola and Chris Welty. 2000. A Formal Ontology of Properties. In, Dieng, R., and Corby, O., eds, *Proceedings of EKAW-2000: The 12th International Conference on Knowledge Engineering and Knowledge Management.* Spring-Verlag LNCS Vol. 1937:97-112. October, 2000.

[11] Guarino, Nicola and Chris Welty. 2000. Identity, Unity, and Individuality: Towards a formal toolkit for ontological analysis. *In, Horn, W. ed., Proceedings of ECAI-2000: The European Conference on Artificial Intelligence.* Pp. 219-223. Berlin: IOS Press. August, 2000.

[12] Guarino, Nicola and Chris Welty. 2002. Identity and Subsumption. In Rebecca Green, Carol A. Bean, & Sung Hyon Myaeng (Eds.), *The Semantics of Relationships: An Interdisciplinary Perspective.* Pp. 111-125. Dordrecht: Kluwer.

[13] Guarino, Nicola and Chris Welty. 2004. An Overview of OntoClean. In Steffen Staab and Rudi Studer, eds., *The Handbook on Ontologies.* Pp. 151-172. Berlin:Springer-Verlag.

Appendix

```
Namespace(ont  = <http://www.ontoclean.org/example-dirty-v1.owl#>)
Namespace(oc = <http://www.ontoclean.org/ontoclean-dl-v1.1.owl#>)
Ontology( <http://www.ontoclean.org/example-dirty-v1-punned.owl>

 Annotation(imports <http://www.ontoclean.org/ontoclean-dl-v1.1.owl>)

 Individual(ont:Agent type(oc:NonSortalClass) type(oc:AntiRigidClass)
   type(oc:NonUnityClass) type(oc:DependentClass)
   value(oc:subClassOf ont:Entity))
 Individual(ont:AmountOfMatter type(oc:AntiUnityClass)
   type(oc:NonDependentClass) type(oc:SortalClass)
   type(oc:RigidClass) value(oc:subClassOf ont:Entity))
 Individual(ont:Animal type(oc:UnityClass) type(oc:NonDependentClass)
   type(oc:SortalClass) type(oc:RigidClass)
   value(oc:subClassOf ont:PhysicalObject))
 Individual(ont:Apple type(oc:UnityClass) type(oc:NonDependentClass)
   type(oc:SortalClass) type(oc:RigidClass)
   value(oc:subClassOf ont:Fruit))
 Individual(ont:Butterfly type(oc:AntiRigidClass) type(oc:UnityClass)
   type(oc:NonDependentClass) type(oc:SortalClass)
   value(oc:subClassOf ont:Animal))
 Individual(ont:Caterpillar type(oc:AntiRigidClass)
   type(oc:UnityClass) type(oc:NonDependentClass) type(oc:SortalClass)
   value(oc:subClassOf ont:Animal))
 Individual(ont:Country type(oc:AntiRigidClass) type(oc:UnityClass)
   type(oc:NonDependentClass) type(oc:SortalClass)
   value(oc:subClassOf ont:LegalAgent))
 Individual(ont:Entity type(oc:Class))
 Individual(ont:Food type(oc:AntiRigidClass) type(oc:AntiUnityClass)
   type(oc:SortalClass) type(oc:DependentClass)
   value(oc:subClassOf ont:AmountOfMatter))
 Individual(ont:Fruit type(oc:UnityClass) type(oc:NonDependentClass)
   type(oc:SortalClass) type(oc:RigidClass)
   value(oc:subClassOf ont:PhysicalObject))
 Individual(ont:Group type(oc:AntiUnityClass)
   type(oc:NonDependentClass) type(oc:SortalClass) type(oc:RigidClass)
   value(oc:subClassOf ont:Entity))
 Individual(ont:GroupOfPeople type(oc:AntiUnityClass)
   type(oc:NonDependentClass) type(oc:SortalClass) type(oc:RigidClass)
   value(oc:subClassOf ont:Group))
 Individual(ont:LegalAgent type(oc:AntiRigidClass)
   type(oc:SortalClass) type(oc:NonUnityClass) type(oc:DependentClass)
   value(oc:subClassOf ont:Agent))
 Individual(ont:LivingBeing type(oc:UnityClass)
   type(oc:NonDependentClass) type(oc:SortalClass) type(oc:RigidClass)
   value(oc:subClassOf ont:AmountOfMatter))
 Individual(ont:Location type(oc:NonDependentClass)
   type(oc:SortalClass) type(oc:NonUnityClass) type(oc:RigidClass)
   value(oc:subClassOf ont:Entity))
 Individual(ont:Organization type(oc:UnityClass)
   type(oc:NonDependentClass) type(oc:SortalClass) type(oc:RigidClass)
   value(oc:subClassOf ont:LegalAgent))
 Individual(ont:Person type(oc:UnityClass) type(oc:NonDependentClass)
   type(oc:SortalClass) type(oc:RigidClass) value(oc:subClassOf
   ont:LegalAgent))
 Individual(ont:PhysicalObject type(oc:UnityClass)
   type(oc:NonDependentClass) type(oc:SortalClass) type(oc:RigidClass)
   value(oc:subClassOf ont:AmountOfMatter))
 Individual(ont:Red type(oc:NonSortalClass) type(oc:NonDependentClass)
```

```
      type(oc:NonUnityClass) type(oc:NonRigidClass)  value(oc:subClassOf
      ont:Entity))
    Individual(ont:RedApple type(oc:AntiRigidClass) type(oc:UnityClass)
      type(oc:NonDependentClass) type(oc:SortalClass)
      value(oc:subClassOf ont:Red))
    Individual(ont:SocialEntity type(oc:NonSortalClass)
      type(oc:UnityClass) type(oc:NonDependentClass) type(oc:RigidClass)
      value(oc:subClassOf ont:Agent))
    Individual(ont:Verebrate type(oc:UnityClass)
      type(oc:NonDependentClass) type(oc:SortalClass) type(oc:RigidClass)
      value(oc:subClassOf ont:Animal))
   )
```