Metamodeling in Domain Ontologies

by Mike Bergman - Monday, September 20, 2010

http://www.mkbergman.com/913/metamodeling-in-domain-ontologies/

OWL 2 Has New Options; Useful to SKOS, Too

It is not unusual to want to treat things either as a class or an instance in an ontology, depending on context. Among other aspects, this is known as metamodeling and it can be accomplished in a number of ways. However, the newest version of the Web Ontology Language, OWL 2, provides a neat trick for doing this called "punning". Why one would want to metamodel, how to specify it in an ontology, and why the OWL 2 approach is helpful are described in this post [1].

Why Metamodel?

Lightweight, domain ontologies have been the focus of this ontology series. Domain ontologies are the "world views" by which organizations, communities or enterprises describe the concepts in their domain, the relationships between those concepts, and the instances or individuals that are the actual things that populate that structure. Thus, domain ontologies are the basic bread-and-butter descriptive structures for real-world applications of ontologies.

These lightweight, domain ontologies often have a hierarchical structure for which SKOS (*Simple Knowledge Organization System*) is a recommended starting ontology [2] (see best practices recommendations). A subject concept reference ontology such as UMBEL (*Upper Mapping and Binding Exchange Layer)* [3], which we also recommend, also has a similar structure and a heavy reliance on SKOS in its vocabulary. Because of these structural similarities, ontologies that use SKOS or UMBEL are therefore good candidates for using metamodeling techniques.

To better understand why we should metamodel, let's look at a couple of examples, both of which combine organizing categories of things and then describing or characterizing those things. This dual

need is common to most domains [4]. For the first example, let's take a categorization of apes as a kind of mammal, which is then a kind of animal. In these cases, ape is a class, which relates to other classes, and apes may also have members, be they particular kinds of apes or individual apes. Yet, at the same time, we want to assert some characteristics of apes, such as being hairy, two legs and two arms, no tails, capable of walking bipedally, with grasping hands, and with some being endangered species. These characteristics apply to the notion of apes as an instance.

As another example we may have the category of trucks, which may further be split into truck types, brands of trucks, type of engine, and so forth. Yet, again, we may want to characterize that a truck is designed primarily for the transport of cargo (as opposed to automobiles for people transport), or that trucks may have different drivers license requirements or different license fees than autos. These descriptive properties refer to trucks as an instance.

These mixed cases combine both the organization of concepts in relation to one another and with respect to their set members, with the description and characterization of these concepts as things unto themselves. This is a natural and common way to express most any domain of interest. The practice has been to express these mixed uses in RDFS or OWL Full, which makes them easy to write and create since most "anything goes" (a loose way of saying that the structures are not decidable) [5]. Use of sub-class relationships also enables tree-like hierarchies to be constructed and some minor inferencing (such as one concept is broader than another concept, one of the contributions of SKOS). But such mixed uses do not allow more capable OWL reasoners to be applied, nor for the full power of query or search abstraction to be applied, nor for the ontology to be checked for consistency. These limits may be fine in many circumstances, but their lack does allow structures to evolve that may become incoherent or illogical. If data interoperability is a goal, as it is in our enterprise use cases, incoherent ontologies can not contribute or participate as structures to linking datasets. At most -- and this is the case for much linked data practice -- all that can be done is to make explicit pairwise connections between different dataset objects. This is not efficient and defeats the whole purpose of leveraging schema.

OWL 2 has been designed to fix that (in addition to other benefits [12]). The approach taken by OWL 2 to overcome some of these metamodeling limitations is through "punning" [6]. Recall that objects are named in RDF with URIs (IRIs in OWL 2). The trick with "punning" is to evaluate the object based on how it is used contextually [7]; the IRI is shared but its referent may be viewed as either a class or instance based on context. Thus, objects used both as concepts (classes) and individuals (instances) are allowed and standard OWL 2 reasoners may be used against them. It should be noted, however, that this "punning" technique does not support the full range of possible metamodeling aspects [8]. Like any language, there is a trade-off in OWL 2 between *expressivity* and *reasoning efficiency* [9].

But, for lightweight, domain ontologies where the objective is interoperability across heterogeneous sources -- that is, namely the main objectives of the semantic Web or semantic enterprise -- this trade-off in OWL 2 now appears to be well balanced. Moreover, its automatic detection by tools such as Protégé 4 that use the OWL API also means it is comparatively easy to use and implement.

Relationship to Recommended Best Practices

An earlier chapter in this series presented some best practices for ontology building and maintenance. A fundamental aspect of those recommendations was the desirability of keeping instance data (ABox)

separate from the conceptual structure (TBox) that provides the schema of relationships for those concepts [10]. Fortunately, this approach also integrates well with the metamodeling capabilities in OWL 2. How metamodeling and the ABox-TBox split is accommodated is shown by this diagram, using trucks as an example:

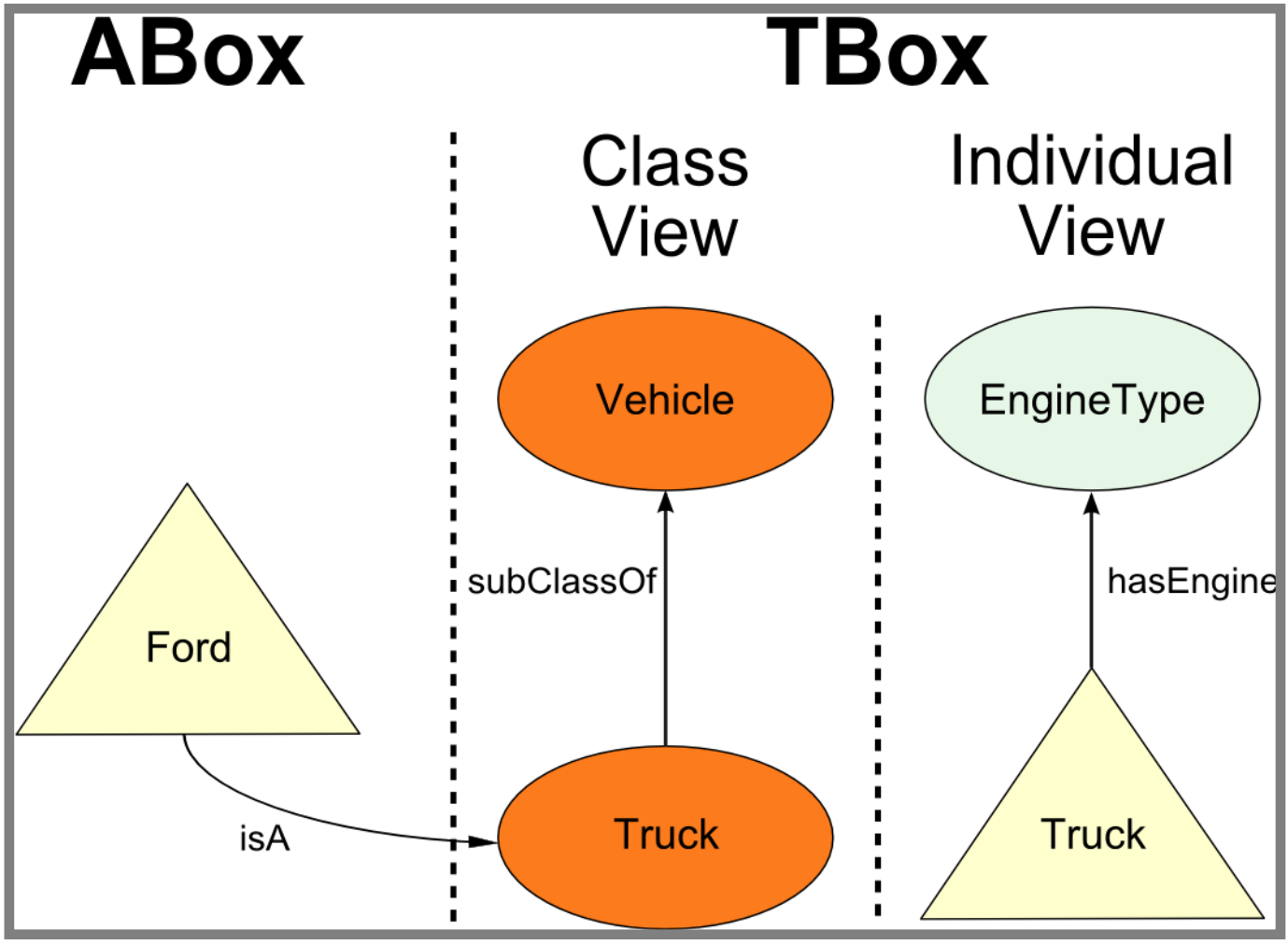


Figure 1. Metamodeling in Domain Ontologies *(click to expand)*

The right-hand side of the diagram shows the two views possible via OWL 2 metamodeling in the TBox. In some cases, we may speak of trucks as a class of vehicle, to which individual members may belong; this is the *class view*. In other contexts, we may want to characterize or make assertions about trucks in our ontology, such as asserting cargo transport or engine type, in which case truck is now represented as an instance (individual) under the *individual view*. These two views in the TBox represent our structural and conceptual description (the "world view") regarding this domain of which vehicles and trucks are a part. Then, when we begin to populate our knowledge base with specific data, we do so via the ABox. In this example, as we add data about the specific brand of Ford trucks and their attributes, we link the Ford instance to the TBox via the Truck class. (Best practice also requires that we model this new attribute structure into the TBox as well, but that is a different topic. ;) .)

How Punning is Triggered in OWL 2

Punning is not triggered by annotation properties. Annotation properties applied to a class merely act as additional description or metadata about that class; the annotation property by definition does not participate in any inferencing or reasoning. You should also know that in OWL 2, certain predicates (properties) such as `label`, `comment` or `description` (among others) are reserved as annotation

properties [11]. You can invoke the OWL 2 punning process directly or via context when your ontologies are processed with the OWL API. The basic rule to follow is:

Any entity declared as a class *and* with an asserted object or data property [15] is punned (metamodeled).

This test is done directly by the OWL API [7]. You can go ahead and test this out with an OWL 2-compliant editor, such as Protégé 4. Here is an example test (in N3 notation): First, begin with some initial declarations:

```
foo:Car a owl:Class .

foo:Animal a owl:Class ;
owl:disjointWith foo:Car .
```

Then, let's describe an object property:

```
foo:isEndangered a owl:ObjectProperty ;
rdf:domain foo:Animal ;
rdf:range bar:SomeSpecies .
```

And define and make an assertion about Apes:

```
foo:Ape a owl:Class ;
foo:isEndangered bar:SomeSpecies .
```

Now, the system begins by testing for punning and other checks, such as:

1. `isEndangered` an annotation property? no
2. what is its domain? `foo:Animal`
3. this will detect and infer:

```
foo:Ape a owl:Class ;
foo:Ape a foo:Animal ;
foo:isEndangered bar:SomeSpecies .
```

1. punning is triggered because non-annotation property has been applied to a class
2. non-annotation properties are now assigned to named individual (which captures individual view part of the TBox above)
3. then, can check for inconsistencies depending on the restriction(s) applied to the `foo:Animal` class.

In this case, no inconsistencies were found. But, let's now add another object (non-annotation) property:

```
foo:hasBrand a owl:ObjectProperty ;
rdf:domain foo:Car ;
rdf:range bar:SomeBrand .
```

And use it to expand our assertions about Apes:

```
foo:Ape a owl:Class ;
foo:isEndangered bar:SomeSpecies ;
foo:hasBrand bar:Ford .
```

And repeat #3:

```
foo:Ape a owl:Class .
foo:Ape a foo:Animal .
foo:Ape a foo:Car ;
foo:isEndangered bar:SomeSpecies ;
foo:hasBrand bar:Ford .
```

Now, inconsistencies are raised in the second #3: So, the consistency check fails, because Ape can not be both an Animal and a Car. While this is clearly a silly example, such checks are quite important as the number of objects and assertions grows in an ontology.

What Does Punning Look Like?

The punning technique works because the IRI for the object ends up being treated as both a concept (class) and an instance (individual). Thus, while the object shares the same IRI, depending on its context, it is evaluated by an OWL reasoner as a different thing (class or individual). The OWL API achieves this by actually writing out the object in both its class view and individual view. Here is an example (in RDF/XML serialization): Input OWL:

```
<owl:Class rdf:about="http://purl.org/ontology/Ape>
<isEndangered>Ape</isEndangered> </owl:Class>
```

Output from Protégé with punning:

```
<!-- http://purl.org/ontology/Ape-->

<owl:Class rdf:about="http://purl.org/ontology/Ape"/>

<!-- http://purl.org/ontology/Ape-->

<owl:NamedIndividual rdf:about="http://purl.org/ontology/Ape">
   <isEndangered>Ape</isEndangered>
</owl:NamedIndividual>
```

Notice the duplicate definition (in RDF/XML) to the `NamedIndividual`. When writing out the ontology, all punned objects are duplicated in a similar manner.

The Beginning of the Transition

OWL 2 and its other general changes [12] have arrived in the nick of time. Not only were we seeing some of the weaknesses in OWL 1 that warranted updating, but we are also now being challenged with regard to how to make linked data and the many datasets in RDF effectively interoperate. Perhaps undecidability and throwing triples to the wind worked OK in the early days of our semantic Web Wild West. But now it is time for the new sheriff to bring order to the emerging chaos. Of course only time will tell, but we believe the design decisions made by the OWL 2 working group were judicious and balanced ones to find that sweet spot between *expressiveness* and *reasoning efficiency* [9]. We also believe that, while useful in its less expressive form [2], that many new domain vocabularies based on SKOS would especially benefit from embracing the OWL 2 metamodeling techniques. But two criticisms still remain. First, tooling support for OWL 2 and the OWL API is weak, as discussed in an earlier chapter. And, as the last chapter discussed, there are not enough practitioners that have yet taken up OWL 2, which means that best practice guidance and exemplars are still limited. Lightweight domain ontologies can greatly benefit from these OWL 2 metamodeling techniques and the OWL RL alternative that also emerged as one of the OWL 2 profile enhancements [13]. Structured Dynamics thinks the growing scale and learning taking place around linked data and RDF datasets is now pointing the way to a necessary transition. And OWL 2 metamodeling should be one of the key components to making our semantic technologies more responsive and effective [14].

[1] This posting is part of a current series on ontology development and tools, jointly developed with Structured Dynamics with co-authorship by Frédérick Giasson. The series began with An Executive Intro to Ontologies, then continued with an update of the prior Ontology Tools listing, which now contains 185 tools. It progressed to a survey of ontology development methodologies. That led to a presentation of a new, Lightweight, Domain Ontologies Development Methodology. That piece was then expanded to address A New Landscape in Ontology Development

Tools, which was followed up by a listing of best practices in domain ontology building and maintenance. This portion completes the series.

[2] Alistair Miles and Sean Bechhofer, eds., 2009. *SKOS Simple Knowledge Organization System Reference*, W3C Recommendation, 18 August 2009. See http://www.w3.org/TR/skos-reference/. Some common SKOS domain predicates include `skos:definition, skos:prefLabel, skos:altLabel, skos:broaderTransitive, skos:narrowerTransitive`.

According to the cited W3C recommendation:

. . . the "concepts" of a thesaurus or classification scheme are modeled [in the base SKOS form] as individuals in the SKOS data model, and the informal descriptions about and links between those "concepts" as given by the thesaurus or classification scheme are modeled as facts about those individuals, never as class or property axioms. Note that these are facts *about* the thesaurus or classification scheme *itself*, such as "concept X has preferred label 'Y' and is part of thesaurus Z"; these are **not** facts about the way the world is arranged within a particular subject domain, as might be expressed in a formal ontology.

Metamodeling and the use of OWL allows the base SKOS form to be expressed as a formal ontology, over which reasoning and inference may occur. Not all SKOS structures may be amenable to this (thesauri and lexical resources such as Wordnet perhaps fall into this category), but some other structures are logical and can be formalized. UMBEL, for example, fits into this category, as do many carefully crafted controlled vocabularies. When used as such, many of the SKOS predicates become OWL annotation properties.

[3] UMBEL (*Upper Mapping and Binding Exchange Layer*) is an ontology of about 20,000 subject concepts that acts as a reference structure for inter-relating disparate datasets. It is also a general vocabulary of classes and predicates designed for the creation of domain-specific ontologies.

[4] In the domain ontologies that are the focus here, we often want to treat our concepts as both classes and instances of a class. This is known as "metamodeling" or "metaclassing" and is enabled by "punning" in OWL 2. For example, here a case cited on the OWL 2 wiki entry on "punning":

People sometimes want to have metaclasses. Imagine you want to model information about the animal kingdom. Hence, you introduce a class a:Eagle, and then you introduce instances of a:Eagle such as a:Harry.

(1) a:Eagle rdf:type owl:Class (2) a:Harry rdf:type a:Eagle

Assume now that you want to say that "eagles are an endangered species". You could do this by treating a:Eagle as an instance of a metaconcept a:Species, and then stating additionally that a:Eagle is an instance of a:EndangeredSpecies. Hence, you would like to say this:

(3) a:Eagle rdf:type a:Species (4) a:Eagle rdf:type a:EndangeredSpecies.

This example comes from Boris Motik, 2005. "On the Properties of Metamodeling in OWL," paper presented at *ISWC 2005*, Galway, Ireland, 2005. For some other examples, see Bernd Neumayr and Michael Schrefl, 2009. "Multi-Level Conceptual Modeling and OWL (Draft, 2 May - Including Full Example)"; see http://www.dke.jku.at/m-owl/most09_22_full.pdf.

[5] A good explanation of this can be found in Rinke J. Hoekstra, 2009. *Ontology Representation: Design Patterns and Ontologies that Make Sense*, thesis for Faculty of Law, University of Amsterdam, SIKS Dissertation Series No.

2009-15, 9/18/2009. 241 pp. See http://dare.uva.nl/document/144859. In that, Hoekstra states (pp. 49-50):

RDFS has a non-fixed meta modelling architecture; it can have an infinite number of class layers because `rdfs:Resource` is both an instance and a super class of `rdfs:Class`, which makes `rdfs:Resource` a member of its own subset (Nejdl et al., 2000). All classes (including `rdfs:Class` itself) are instances of `rdfs:Class`, and every class is the set of its instances. There is no restriction on defining sub classes of `rdfs:Class` itself, nor on defining sub classes of instances of instances of `rdfs:Class` and so on. This is problematic as it leaves the door open to class definitions that lead to Russell's paradox (Pan and Horrocks, 2002). The Russell paradox follows from a comprehension principle built in early versions of set theory (Horrocks et al., 2003). This principle stated that a set can be constructed of the things that satisfy a formula with one free variable. In fact, it introduces the possibility of a set of all things that do not belong to itself . . . .

In RDFS, the reserved properties `rdfs:subClassOf`, `rdf:type`, `rdfs:domain` and `rdfs:range` are used to define both the other RDFS modelling primitives themselves and the models expressed using these primitives. In other words, there is no distinction between the meta-level and the domain.

[6] "Punning" was introduced in OWL 2 and enables the same IRI to be used as a name for both a class and an individual. However, the direct model-theoretic semantics of OWL 2 DL accommodates this by understanding the class `Father` and the individual `Father` as two different views on the same IRI, *i.e.*, they are interpreted semantically as if they were distinct. The technique listed in the main body triggers this treatment in an OWL 2-compliant editor. See further Pascal Hitzler *et al*., eds., 2009. *OWL 2 Web Ontology Language Primer*, a W3C Recommendation, 27 October 2009; see http://www.w3.org/TR/owl2-primer/.

[7] The OWL API is a Java interface and implementation for the W3C Web Ontology Language (OWL), used to represent Semantic Web ontologies. The API provides links to inferencers, managers, annotators, and validators for the OWL2 profiles of RL, QL, EL. Two recent papers describing the updated API are: Matthew Horridge and Sean Bechhofer, 2009. "The OWL API: A Java API for Working with OWL 2 Ontologies," presented at *OWLED 2009, 6th OWL Experienced and Directions Workshop*, Chantilly, Virginia, October 2009. See http://www.webont.org/owled/2009/papers/owled2009_submission_29.pdf; and, Matthew Horridge and Sean Bechhofer, 2010. "The OWL API: A Java API for OWL Ontologies," paper submitted to the *Semantic Web Journal*; see http://www.semantic-web-journal.net/sites/default/files/swj107.pdf. Also see its code documentation at http://owlapi.sourceforge.net/2.x.x/documentation.html.

The main text describes how via "punning" the OWL API supports two parallel views sharing the same IRI, which can enable a concept to operate as either a class or instance depending on context.

[8] Some other metamodeling aspects not supported by "punning" include full multi-level modeling (such as in UML or OMG's model-driven architecture) or linkage with closed-world reasoning.

[9] OWL has historically been described as trying to find the proper *tradeoff between expressive power and efficient reasoning support*. See, for example, Grigoris Antoniou and Frank van Harmelen, 2003. "Web Ontology Language: OWL," in S. Staab and R. Studer, eds., *Handbook on Ontologies in Information Systems*, Springer-Verlag, pp. 76-92. See http://www.few.vu.nl/~frankh/postscript/OntoHandbook03OWL.pdf.

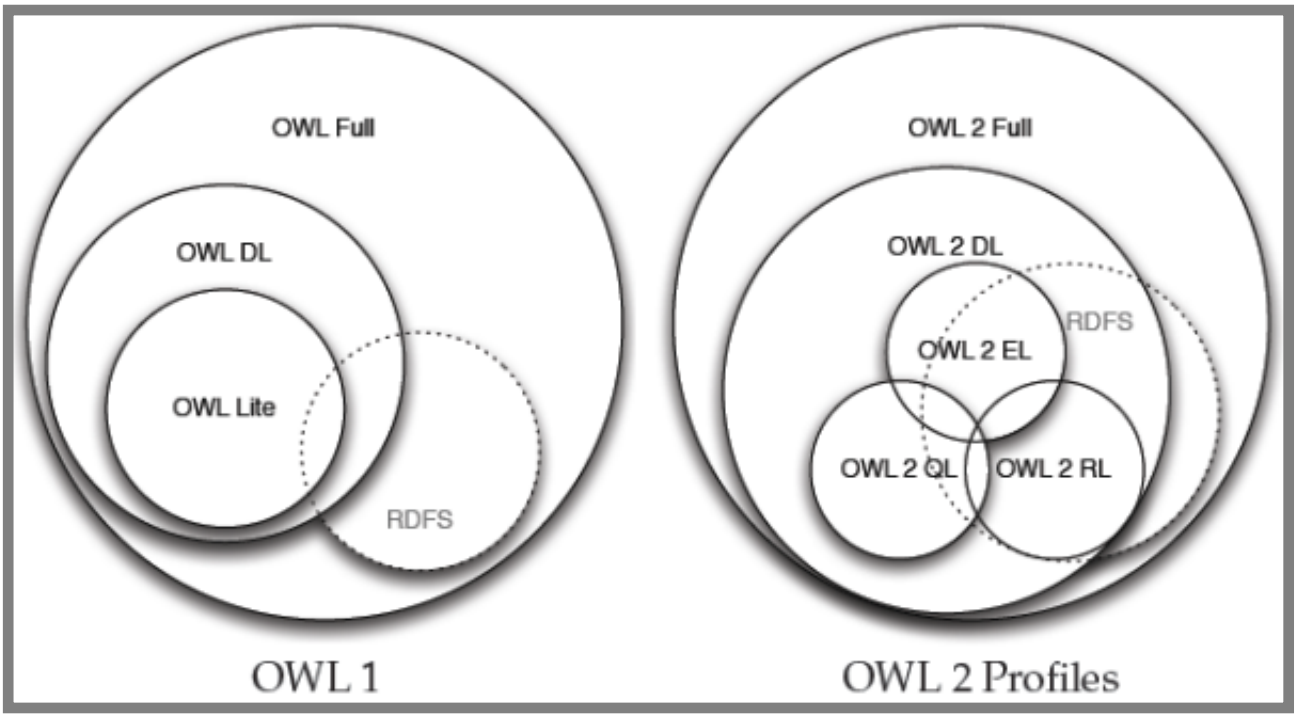
[10] The ***TBox*** portion, or classes (concepts), is the basis of the ontologies. The ontologies establish the structure used for governing the conceptual relationships for that domain and in reference to external (Web) ontologies. The ***ABox*** portion, or instances (named entities), represents the specific, individual things that are the members of those classes. Named entities are the notable objects, persons, places, events, organizations and things of the world. Each

named entity is related to one or more classes (concepts) to which it is a member. Named entities do not set the structure of the domain, but populate that structure. The ABox and TBox play different roles in the use and organization of the information and structure. These distinctions have their grounding in description logics.

[11] For a listing, see http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/#Annotation_Properties. Even if your local ontology defines a sub-property of one of these items, such as foo:myLabel as a sub-property of `rdfs:label`, you are advised to still specifically declare it as an annotation property.

[12] See Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider and Ulrike Sattler, 2008. "OWL2: The Next Step for OWL," see http://www.comlab.ox.ac.uk/people/ian.horrocks/Publications/download/2008/CHMP+08.pdf; and also see the OWL 2 Quick Reference Guide by the W3C, which provides a brief guide to the constructs of OWL 2, noting the changes from OWL 1.

[13] OWL RL is the "rules" profile of OWL 2 and is both decidable and offers additional axiomatic support for metamodeling. As this figure drawn from Hoekstra [Fig. 3-4 in 5] shows comparing OWL 2 to OWL 1, OWL RL provides a subset of decidable description logics:



[14] Metamodeling might be a new concept to you and some of the aspects can certainly be academic. If the references above do not sufficient satisfy your curiosity, you may want to check out some of these other useful references: Birte Glimm, Sebastian Rudolph and Johanna Völker, 2009. "Integrated Metamodeling and Diagnosis in OWL 2," see http://www.comlab.ox.ac.uk/files/3129/paper.pdf; and Nophadol Jekjantuk, Gerd Groener and Jeff. Z. Pan, 2009. "Reasoning in Metamodeling Enabled Ontologies," in Rinke Hoekstra and Peter F. Patel-Schneider, eds., *Proceedings of OWL: Experiences and Directions (OWLED 2009)*; see http://www.webont.org/owled/2009.

[15] In OWL 2, an *object property* is a predicate that defines a binary relationship between two objects (in specific respect to a triple, between a *subject* and an *object*). A *data property* is a predicate that defines a binary relationship between an object an a literal (string or data value). In contrast to object and data properties, annotation properties and reserved OWL and RDF vocabularies are explicitly ***excluded*** from this rule. ***Only*** declared object or data

properties trigger the punning.
