OWL DL / Full Compatability

Peter F. Patel-Schneider, Bell Labs Research

- Model-Theoretic Semantics
- OWL DL and OWL Full Model Theories
- Differences Betwen the Two Semantics
- Forward to OWL 1.1

Model-Theoretic Semantics

- What does the (or a) world look like?
  - There are data values (e.g., strings, integers).
  - There are objects (e.g., Bell Labs, Peter).
  - Objects belong to classes.
  - Objects are related to other objects and data values via properties.
- OWL DL has a fairly standard take on all this.
- RDF and OWL Full have a slightly unusual take.
  - Classes and properties are also objects.
- What does a construct mean?
  - ≥ 2 hasChild
    is the set of objects that are related to at least two other objects by the hasChild property
  - John $\in$ Person John belongs to the Person
- What follows from what?
  - An ontology containing Student $\subseteq$ Person and John $\in$ Student implies John $\in$ Person

Two Model Theories

	OWL DL	OWL Full
objects	disjoint from numbers	includes "everything"
OWL classes	can't contain numbers	contain anything
object properties	can't point to numbers	point to anything
annotations	handled specially	regular facts
rdf:type, owl:Class		objects
syntax		objects

OWL DL Universe:
objects and values (numbers, strings, ...), mutually disjoint
OWL Full Universe:
objects can be values, classes, datatypes, properties, syntax, lists, ontologies

Differences

- Number of objects:
  - In OWL Full there are an infinite number of objects (because objects include the integers).
  - In OWL DL there can be a finite number of objects.
  - This is an *observable* difference (i.e., because of this difference there is an OWL DL ontology that is satisfiable in the OWL DL semantics but unsatisfiable in the OWL Full semantics).
- In OWL Full it is possible to say things like rdf:type owl:sameIndividualAs rdfs:subClassOf.
- In OWL DL classes, properties, and individuals have separate namespaces, so SameIndividuals(Person Student) does not imply EquivalentClasses(Person Student).

OWL 1.1

- OWL 1.1 model theory like the OWL DL model theory.
- Treatment of entity annotations:
  - OWL DL created special "non-objects" as necessary and associated ontololgy facts with them
  - In OWL Full they are just like regular facts

    - OWL 1.1 treats entity annotations as "semantic-free"
- Treatment of axiom annotations:
    - Not in OWL DL or OWL Full
    - OWL 1.1 treats axiom annotations as "semantic-free"
- OWL 1.1 constructs that need reification
    - E.g., negative property assertions
    - How to handle these in OWL Full

Imports

Peter F. Patel-Schneider, Bell Labs Research

- How it works in OWL DL, OWL Full, and OWL 1.1
- See also http://www.w3.org/2007/OWL/wiki/Imports.

Specification in OWL 1.0

OWL DL DL-Style Model Theoretic Semantics
Section 3.4 of the OWL Semantics and Abstract Syntax document:

> [A]n owl:imports annotation also imports the contents of another OWL ontology into the current ontology. The imported ontology is the one, if any, that has as name the argument of the imports construct. (This treatment of imports is divorced from Web issues. The intended use of names for OWL ontologies is to make the name be the location of the ontology on the Web, but this is outside of this formal treatment.)

OWL Full
Section 5.3 of the OWL Semantics and Abstract Syntax document:

> Definition: Let K be a collection of RDF graphs. K is imports closed iff for every triple in any element of K of the form x owl:imports u . then K contains a graph that is the result of the RDF processing of the RDF/XML document, if any, accessible at u into an RDF graph. The imports closure of a collection of RDF graphs is the smallest import-closed collection of RDF graphs containing the graphs.

OWL DL RDFS-Compatible Semantics

Section 5.4 of the OWL Semantics and Abstract Syntax document:

> Definition: Let T be the mapping from the abstract syntax to RDF graphs from Section 4.1. Let O be a collection of OWL DL ontologies and axioms and facts in abstract syntax form. O is said to be imports closed iff for any URI, u, in an imports directive in any ontology in O the RDF parsing of the document accessible on the Web at u results in T(K), where K is the ontology in O with name u.

Specification in OWL 1.1

OWL 1.1 Syntax

Section 3 of the OWL 1.1 Structural Specification and Functional Syntax:

> The structure of OWL 1.1 ontologies is shown in Figure 1. Each ontology is uniquely identified with an ontology URI. This URI need not be equal to the physical location of the ontology file. For example, a file for an ontology with a URI http://www.my.domain.com/example need not be physically stored in that location. A specification of a mechanism for physically locating an ontology from its ontology URI is not in scope of this specification.
> Each ontology contains a possibly empty set of import declarations. An ontology O directly imports an ontology O' if O contains an import declaration whose value is the ontology URI of O'. The relation imports is defined as a transitive closure of the relation directly imports. The axiom closure of an ontology O is the smallest set containing all the axioms of O and of all ontologies that O imports. Intuitively, an import declaration specification states that, when reasoning with an ontology O, one should consider not only the axioms of O, but the entire axiom closure of O.

Summary

Summary of OWL 1.0 Design

In the OWL DL semantics the ontology retrievable at URI u is supposed to have name u. It is thus irrelevant whether imports is by location or by ontology name. For OWL Full, it is not required that the ontology at u have name u, but it is clear that imports is by location.
The end result is that in OWL 1.0, imports works like *entire-document XML inclusion.*

Summary of OWL 1.1 Design

Imports is by name, as in the OWL DL semantics, but without the intent that names and locations correspond.
The end result is that in OWL 1.1, imports (if it works at all) by searching for the ontology with the matchine *name*, not location.

XML Inclusions
: XML inclusions is compatible with an imports-by-location mechanism, but XML inclusions provide much more power. XML inclusions do not appear to be appropriate for any non-XML OWL syntax, as they are too tied to XML. Full XML inclusions might be appropriate for an XML-based OWL syntax, provided that imports-by-location is the imports paradigm that is wanted.

Imports: Options

- Ontology names: optional or required
- Ontology with name u at location u?
- Imports by ontology name
- Imports by ontology location

Proposal: Loosen the name-location tie, so that ontologies can be handled like W3C publications. An ontology has a base name (a URI, as now) plus version information (as allowed now). The permanent location of an ontology is at its base name concatenated with its version. The current version of an ontology is also linked to from its base name.

Proposal: Imports is by location. If an importing ontology cares about version information then it uses the permanent location of an ontology. If an importing ontology wants to use the current version, then it uses the base name. Tools can use ontology caches if they want, with the usual aging and caveats.

Proposal: Ontologies need not have names, but such ontologies are not importable.

Rich Annotations

Peter F. Patel-Schneider, Bell Labs Research

Basic ideas

1. Allow arbitrary syntax (including annotations) as annotations.
2. Annotations are split into different "spaces".
3. Tools place annotation information into the spaces, and reason in these spaces independently.

4. Some spaces signal the use of extensions. HOW?
5. Use of spaces marked "mustUnderstand" require that tools understand the space's extension.

Syntax

ontology := 'Ontology' '(' *ontologyURI* { *importDeclaration* }
{ *annotationSpaceDeclaration* } { *annotation* } { *axiom* } ')'
annotationSpaceDeclaration := 'AnnotationSpace' '(' *spaceURI* [*status*] ')'
status := 'mayIgnore' | 'mustUnderstand'
annotation := 'Annotation' '(' [*spaceURI*] *importDeclaration* ')'
annotation := 'Annotation' '(' [*spaceURI*] *annotationSpaceDeclaration* ')'
annotation := 'Annotation' '(' [*spaceURI*] *annotation* ')'
annotation := 'Annotation' '(' [*spaceURI*] *axiom* ')'

A special symbol (SELF) in an annotation refers to the axiom or entity being annotated.

Least controversial
: Annotations can be any ontology content (imports, annotations, axioms)

More controversial
: There are multiple annotation spaces.

Most controversial
: Some annotation spaces affect the meaning of regular ontology constructs.

Abbreviations

- Label (annotations constant):
  Annotation (annotations DataPropertyAssertion (rdfs:label SELF constant))
- Comment (annotations constant):
  Annotation (annotations DataPropertyAssertion (rdfs:comment SELF constant))
- Annotation (annotations annotationURI constant):
  Annotation (annotations DataPropertyAssertion (annotationURI SELF constant))
- Annotation (annotations annotationURI XXX(URI)):
  Annotation (annotations ObjectPropertyAssertion (annotationURI SELF URI))

Semantics

- NONE

- Annotations have no semantics *in the ontology*.

Rich Annotations: Pragmatics

- Tools must:
  - reject an ontology if they don't implement a 'mustUnderstand' annotation space.
- Implementing an annotation space involves:
  1. extracting annotations for that space into a separate KBs,
  2. reasoning in that KB, and
  3. using the results of this reasoning to modify behaviour.
- Tools are encouraged to do steps 1 and 2 for all ontology spaces.

Example

Probabilistic extension to OWL 1.1 (like in Pronto):

- Axioms can have probabilities associated with them via mustUnderstand annotations.
- These probabilities affect reasoning (as per some externally-specified spec).
- Ignoring the annotations effects makes you unsound.

Encoding Syntax in facts:

- Encode SWRL rules into OWL (RDF?) triples.
- Annotate (some of) the triple "facts" with a (mustUnderstand?) annotation.
- Ignoring the annotations effects doesn't make you unsound.