5.2: Standardizing an Ontology Language

This section and the next one are intentionally kept short, as listing language features isn’t the most interesting of content, those lists exist also online in the standard\(^1\), and are not meant to be memorized but to be consulted as the need arises. This section and the next one, instead, focus on the gist of it and provide some contextual information.

**Historical Notes**

Before OWL, there were a plethora of ontology languages, such as the obo format (directed acyclic graphs) initiated by the Gene Ontology Consortium, KL-ONE, and F-logic (frames, and older versions of the Protégé ODE). Unsurprisingly, this caused ontology interoperation problems even at the syntactic level and hampered development and use of ontology tools, hence, its uptake. To solve those issues, researchers set out to standardize a logic language. At the time that people sat at the standardization table (first around 2001) of the World Wide Web Consortium (W3C), there were several logics that had a considerable influence on the final product, most notably the SHOE, DAML-ONT, OIL, and DAML+OIL languages, and, more generally, the fruits of 20 years of research on languages and prototyping of automated reasoners by the DL community.

Following good engineering practices, a document of requirements and objectives was devised to specify what such an ontology language for the Semantic Web should meet. It is useful to list them here, so that you can decide yourself how well OWL meets them, as well as any contender languages, or if you would want to design one of your own. It specified the following design goals: Shareable; Change over time; Interoperability; Inconsistency detection; Balancing expressivity and complexity; Ease of use; Compatible with existing standards; and Internationalization. There were also requirements on the features of the language. They were: Ontologies are objects on the Web; they have their own meta-data, versioning, etc.; Ontologies are extendable; They contain classes, properties, data-types, range/domain, individuals; they must be able to express equality for classes and for individuals; Classes as instances; Cardinality
constraints; and XML syntax\(^2\).

First, as standardization is typically consensus-based, there was, on the one hand, a language called OWL full with RDF-based semantics (graphs) and two DL-based species, being OWL lite and OWL DL. But what makes OWL a Semantic Web language compared to the regular DL languages that were introduced in the previous chapter? This is the second step. There are the following main differences:

- OWL uses URI references as names; e.g., http://www.mysite.co.za/UniOnto.owl#Student is the URI of the class Student in the ontology with file name UniOnto.owl file that is online at http://www.mysite.co.za;
- It gathers information into ontologies stored as documents written in RDF/XML including things like owl:imports to import one ontology into another; e.g. another ontology, HigherEd.owl can import the UniOnto.owl so that the class http://www.mysite.co.za/UniOnto.owl#Student becomes, in a way, part of HigherEd.owl (though it still exists independently as well);
- It adds RDF data types and XML schema data types for the ranges of data properties (attributes), so one can use, e.g., string and integer in a similar way as you are familiar with in UML class diagrams and databases.
- A different terminology: a DL concept is now called a class, a DL role is called an object property, and data property is added for attributes (i.e., the relation that relates a class to a data type).

With this information, we can start going from a paper-based representation to its computational version. The first steps are illustrated in the next example, which will be extended in later chapters.

Example \(\PageIndex{(1)}\):

The African Wildlife Ontology (AWO) is a basic tutorial ontology based on the examples in the “A Semantic Web Primer” book [AvH03]. The fist step is to represent that in OWL, using your favorite ontology development environment (ODE). An OWL version of it, AfricanWildlifeOntology0.owl, has 10 classes and 3 object properties concerning animals such as Lion, Giraffe, Plant, and object properties eats and is-part-of, and has annotations that give an idea of what should be modeled (else: see 4.3.1 pages 119-133 in [AvH03]). Upon running the reasoner, it will classify, among others, that Carnivore is a subclass of Animal (i.e, \(\text{AWO}\models\texttt{Carnivore}\sqsubseteq\texttt{Animal}\)).

This is not really exciting, and the tutorial ontology is not of a particularly good quality. First, we extend it by having loaded the ontology and adding knowledge to it: among others, proper parthood, a few more plant parts and animals, such as Impala, Warthog, and RockDassie. This version of the AWO is named AfricanWildlifeOntology1.owl. With this additional knowledge, warthogs are classified as omnivores, lions as carnivores, giraffes as herbivores, and so on.

Another aspect is purely engineering practice: if the intention is to put the ontology online, it should be named properly, i.e., the URI has to be set so that its contents can be identified appropriately on the Semantic Web; that is, do not simply use the default URI generated by the tool (e.g., http://www.semanticweb.org/ontologie...7204526617.owl), but specify an appropriate one where the ontology will be published, like http://www.meteck.org/files/ontologies/myAWO.owl.
The OWL 1 Family of Languages

Purely for legacy purposes, I include here the first three 'species' of OWL (version 1). This because 1) there are still plenty of ontologies around that are represented in either of these languages and you may have to deal with them, and 2) OWL DL seems to be a fairly good 'sweet spot' in the expressivity/complexity trade-off, meaning: still being able to represent the domain fairly well, or at least sufficiently to the extent that the increased expressiveness of OWL 2 DL doesn’t weigh up against the increased slowness of completing the reasoning tasks. The key points to remember are:

• OWL Lite and OWL DL are DL-based, with model-theoretic semantics.
• OWL DL is more expressive than OWL Lite, but there’s not a lot of gain computationally, so OWL Lite was hardly used specifically.
• OWL full never really took off either.
• There are multiple syntaxes for the serialization: functional-style syntax (optional) and RDF/XML (required for tool interoperability).

OWL Lite

OWL Lite has a classification hierarchy and (relative to OWL DL) simple constraints. While OWL Lite has strong syntactic restrictions, it has only limited semantics restrictions compared to OWL DL\(^3\). OWL Lite corresponds to the DL \(\mathcal{SHIF(D)}\). Putting the DL symbols to the names of the features, we have:

• Named classes \((A)\)
• Named properties \((P)\)
• Individuals \((C(o))\)
• Property values \((P(o, a))\)
• Intersection \((C\sqcap D)\)
• Union \((C\sqcup D)\)
• Negation \((\neg C)\)
• Existential value restrictions \((\exists P.C)\)
• Universal value restrictions \((\forall P.C)\)
• Unqualified \((0/1)\) number restrictions \((\geq nP, \leq nP, = nP), 0 \leq n \leq 1)\)

OWL DL

OWL DL had, at the time, ‘maximal’ expressiveness while maintaining tractability, and has, as the name suggestion, an underlying DL. It has all the features of OWL-lite, and, in addition: Negation, Disjunction, (unqualified) Full cardinality, Enumerated classes, and hasValue. OWL DL corresponds to the DL \(\mathcal{SHOIN(D)}\). It has the following features:

• All OWL Lite features
• Arbitrary number restrictions \((\geq nP, \leq nP, = nP))\), with \((0 \leq n)\)
• Property value \(\exists P.\{ o\}\)
• Enumeration \(\{ o_1, ..., o_n\}\)

---

**OWL Full**

OWL Full, has a very high expressiveness (losing tractability) and all syntactic freedom of RDF (self-modifying). OWL full has meta-classes and one can modify the language. Note that OWL Full is not a Description Logic.

As mentioned earlier, OWL and DLs are tightly related, in particular OWL Lite and OWL DL. They have, just like their base DLs, a model theoretic semantics. Table 4.1.1 shows a few examples of some OWL syntax and its DL counterpart notation. There is also the not-for-human-consumption RDF/XML serialization.

<table>
<thead>
<tr>
<th>OWL Construct</th>
<th>DL Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>(C_1 \sqcap ... \sqcap C_n)</td>
<td>\texttt{Human} \sqcap \texttt{Male}</td>
</tr>
<tr>
<td>unionOf</td>
<td>(C_1 \sqcup ... \sqcup C_n)</td>
<td>\texttt{Doctor} \sqcup \texttt{Lawyer}</td>
</tr>
<tr>
<td>complementOf</td>
<td>(\neg C)</td>
<td>(\neg \texttt{Male})</td>
</tr>
<tr>
<td>oneOf</td>
<td>({ o_1, ..., o_n})</td>
<td>({giselle, juan})</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>(\forall P. C)</td>
<td>(\forall \texttt{hasChild.Doctor})</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>(\exists P. C)</td>
<td>(\exists \texttt{hasChild.Lawyer})</td>
</tr>
<tr>
<td>value</td>
<td>(\exists P.{ o})</td>
<td>(\exists \texttt{citizenOf.{RSA}})</td>
</tr>
<tr>
<td>minCardinality</td>
<td>(\geq nP)</td>
<td>(\geq 2 \texttt{hasChild})</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>(\leq nP)</td>
<td>(\leq 6 \texttt{enrolledIn})</td>
</tr>
</tbody>
</table>

*Table 4.1.1: Some examples of OWL’s construct, the same in DL notation, and an example.*

---

**Footnotes**

1. start with a general non-technical overview in the OWL primer [http://www.w3.org/TR/owl2-primer/](http://www.w3.org/TR/owl2-primer/).

2. The “making of an ontology language” article [HPSvH03] gives a longer general historical view and it also summarizes OWL with its three species (OWL lite, OWL-DL, and OWL full). The details of the standard are freely available at [http://www.w3.org/TR/owl-ref/](http://www.w3.org/TR/owl-ref/).

3. More specifically regarding the latter, if you really want to know: negation can be encoded using disjointness and with negation and conjunction, you can encode disjunction. Take, for instance:
This is equivalent to

\[
\text{DisjointClasses(not} B
\text{)}
\]

\[
\text{DisjointClasses(not} C
\text{)}
\]

\[
\text{Class(notB and notC complete notB and notC)}
\]

\[
\text{DisjointClasses(notB and notC or} C
\text{)}
\]

\[
\text{Class(C complete notB and notC)}
\]