Web Ontology Language: OWL

Grigoris Antoniou[1] and Frank van Harmelen[2]

[1] Department of Computer Science, University of Crete, ga@csd.uoc.gr
[2] Department of AI, Vrije Universiteit Amsterdam,
Frank.van.Harmelen@cs.vu.nl

1 Motivation and Overview

The expressivity of RDF and RDF Schema that was described in [12] is deliberately very limited: RDF is (roughly) limited to binary ground predicates, and RDF Schema is (again roughly) limited to a subclass hierarchy and a property hierarchy, with domain and range definitions of these properties.

However, the Web Ontology Working Group of W3C[3] identified a number of characteristic use-cases for Ontologies on the Web which would require much more expressiveness than RDF and RDF Schema.

A number of research groups in both America and Europe had already identified the need for a more powerful ontology modelling language. This lead to a joint initiative to define a richer language, called DAML+OIL[4] (the name is the join of the names of the American proposal DAML-ONT[5], and the European language OIL[6]).

DAML+OIL in turn was taken as the starting point for the W3C Web Ontology Working Group in defining OWL, the language that is aimed to be the standardised and broadly accepted ontology language of the Semantic Web.

In this chapter, we first describe the motivation for OWL in terms of its requirements, and the resulting non-trivial relation with RDF Schema. We then describe the various language elements of OWL in some detail.

Requirements for ontology languages

Ontology languages allow users to write explicit, formal conceptualizations of domains models. The main requirements are:

[3] http://www.w3.org/2001/sw/WebOnt/
[4] http://www.daml.org/2001/03/daml+oil-index.html
[5] http://www.daml.org/2000/10/daml-ont.html
[6] http://www.ontoknowledge.org/oil/

1. a well-defined syntax
2. a well-defined semantics
3. efficient reasoning support
4. sufficient expressive power
5. convenience of expression.

The importance of a *well-defined syntax* is clear, and known from the area of programming languages; it is a necessary condition for *machine-processing* of information. All the languages we have presented so far have a well-defined syntax. DAML+OIL and OWL build upon RDF and RDFS and have the same kind of syntax.

Of course it is questionable whether the XML-based RDF syntax is very user-friendly, there are alternatives better suitable for humans (for example, see the OIL syntax). However this drawback is not very significant, because ultimately users will be developing their ontologies using authoring tools, or more generally *ontology development tools*, instead of writing them directly in DAML+OIL or OWL.

Formal semantics describes precisely the meaning of knowledge. "Precisely" here means that the semantics does not refer to subjective intuitions, nor is it open to different interpretations by different persons (or machines). The importance of formal semantics is well-established in the domain of mathematical logic, among others.

One use of formal semantics is to allow humans to reason about the knowledge. For ontological knowledge we may reason about:

- *Class membership:* If x is an instance of a class C, and C is a subclass of D, then we can infer that x is an instance of D.
- *Equivalence of classes:* If class A is equivalent to class B, and class B equivalent to class C, then A is equivalent to C, too.
- *Consistency:* Suppose we have declared x to be an instance of the class A. Further suppose that
  - A is a subclass of $B \cap C$
  - A is a subclass of D
  - B and D are disjoint

  Then we have an inconsistency because A should be empty, but has the instance x. This is an indication of an error in the ontology.
- *Classification:* If we have declared that certain property-value pairs are sufficient condition for membership of a class A, then if an individual x satisfies such conditions, we can conclude that x must be an instance of A.

Semantics is a prerequisite for *reasoning support*: Derivations such as the above can be made mechanically, instead of being made by hand. Reasoning support is important because it allows one to

- check the consistency of the ontology and the knowledge;

- check for unintended relationships between classes.
- automatically classify instances in classes

Automated reasoning support allows one to check many more cases than what can be done manually. Checks like the above are valuable for

- *designing* large ontologies, where multiple authors are involved;
- *integrating and sharing* ontologies from various sources.

Formal semantics and reasoning support is usually provided by mapping an ontology language to a known logical formalism, and by using automated reasoners that already exist for those formalisms. We will see that OWL is (partially) mapped on a description logic, and makes use of existing reasoners such as FaCT and RACER.

Description logics are a subset of predicate logic for which efficient reasoning support is possible. See [13] for more detail.

Limitations of the expressive power of RDF Schema

RDF and RDFS allow the representation of *some* ontological knowledge. The main modelling primitives of RDF/RDFS concern the organization of vocabularies in typed hierarchies: subclass and subproperty relationships, domain and range restrictions, and instances of classes. However a number of other features are missing. Here we list a few:

- *Local scope of properties:* `rdfs:range` defines the range of a property, say `eats`, for all classes. Thus in RDF Schema we cannot declare range restrictions that apply to some classes only. For example, we cannot say that cows eat only plants, while other animals may eat meat, too.
- *Disjointness of classes:* Sometimes we wish to say that classes are disjoint. For example, `male` and `female` are disjoint. But in RDF Schema we can only state subclass relationships, e.g. `female` is a subclass of `person`.
- *Boolean combinations of classes:* Sometimes we wish to build new classes by combining other classes using union, intersection and complement. For example, we may wish to define the class `person` to be the disjoint union of the classes `male` and `female`. RDF Schema does not allow such definitions.
- *Cardinality restrictions:* Sometimes we wish to place restrictions on how many distinct values a property may or must take. For example, we would like to say that a person has exactly two parents, and that a course is taught by at least one lecturer. Again such restrictions are impossible to express in RDF Schema.
- *Special characteristics of properties:* Sometimes it is useful to say that a property is *transitive* (like "greater than"), *unique* (like "is mother of"), or the *inverse* of another property (like "eats" and "is eaten by").

So we need an ontology language that is richer than RDF Schema, a language that offers these features and more. In designing such a language one should be aware of the *tradeoff between expressive power and efficient reasoning support*. Generally speaking, the richer the language is, the more inefficient the reasoning support becomes, often crossing the border of non-computability. Thus we need a compromise, a language that can be supported by reasonably efficient reasoners, while being sufficiently expressive to express large classes of ontologies and knowledge.

Compatibility of OWL with RDF/RDFS

Ideally, OWL would be an extension of RDF Schema, in the sense that OWL would use the RDF meaning of classes and properties (`rdfs:Class`, `rdfs:subClassOf`, etc), and would add language primitives to support the richer expressiveness identified above.

Unfortunately, the desire to simply extend RDF Schema clashes with the trade-off between expressive power and efficient reasoning mentioned before. RDF Schema has some very powerful modelling primitives, such as the `rdfs:Class` (the class of all classes) and `rdf:Property` (the class of all properties). These primitives are very expressive, and will lead to uncontrollable computational properties if the logic is extended with the expressive primitives identified above.

Three species of OWL

All this as lead to a set of requirements that may seem incompatible: efficient reasoning support and convenience of expression for a language as powerful as a combination of RDF Schema with a full logic.

Indeed, these requirements have prompted W3C's Web Ontology Working Group to define OWL as three different sublanguages, each of which is geared towards fulfilling different aspects of these incompatible full set of requirements:

- *OWL Full:* The entire language is called OWL Full, and uses all the OWL languages primitives (which we will discuss later in this chapter). It also allows to combine these primitives in arbitrary ways with RDF and RDF Schema. This includes the possibility (also present in RDF) to change the meaning of the pre-defined (RDF or OWL) primitives, by applying the language primitives to each other. For example, in OWL Full we could impose a cardinality constraint on the class of all classes, essentially limiting the number of classes that can be described in any ontology.
  The advantage of OWL Full is that it is fully upward compatible with RDF, both syntactically and semantically: any legal RDF document is also a legal OWL Full document, and any valid RDF/RDF Schema conclusion is also a valid OWL Full conclusion.

The disadvantage of OWL Full is the language has become so powerful as to be undecidable, dashing any hope of complete (let alone efficient) reasoning support.

- *OWL DL:* In order to regain computational efficiency, OWL DL (short for: Description Logic) is a sublanguage of OWL Full which restricts the way in which the constructors from OWL and RDF can be used. We will give details later, but roughly this amounts to disallowing application of OWL's constructor's to each other, and thus ensuring that the language corresponds to a well studied description logic.
  The advantage of this is that it permits efficient reasoning support.
  The disadvantage is that we loose full compatibility with RDF: an RDF document will in general have to be extended in some ways and restricted in others before it is a legal OWL DL document. Conversely, every legal OWL DL document is still a legal RDF document.
- *OWL Lite:* An ever further restriction limits OWL DL to a subset of the language constructors. For example, OWL Lite excludes enumerated classes, disjointness statements and arbitrary cardinality (among others).
  The advantage of this is a language that is both easier to grasp (for users) and easier to implement (for tool builders).
  The disadvantage is of course a restricted expressivity.

Ontology developers adopting OWL should consider which sublanguage best suits their needs. The choice between OWL Lite and OWL DL depends on the extent to which users require the more-expressive constructs provided by OWL DL and OWL Full. The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modeling facilities of RDF Schema (e.g. defining classes of classes, or attaching properties to classes). When using OWL Full as compared to OWL DL, reasoning support is less predictable since complete OWL Full implementations will be impossible.

There are strict notions of upward compatibility between these three sublanguages:

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

OWL still uses RDF and RDF Schema to a large extent:

- all varieties of OWL use RDF for their syntax
- instances are declared as in RDF, using RDF descriptions and typing information
- OWL constructors like `owl:Class`, `owl:DatatypeProperty` and `owl:ObjectProperty` are all specialisations of their RDF counterparts. Figure 1 shows the subclass relationships between some modelling primitives of OWL and RDF/RDFS.

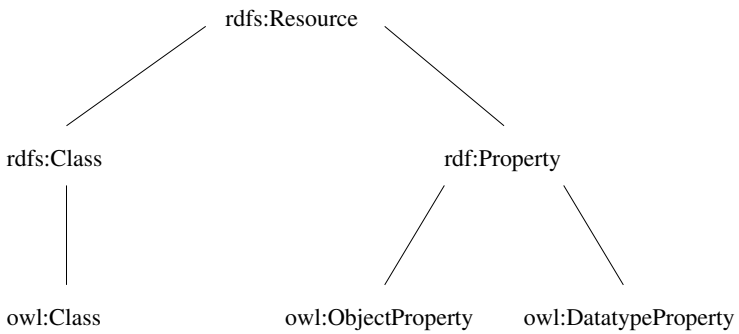

Fig. 1. Subclass relationships between OWL and RDF/RDFS

The original hope in the design of OWL was that there would be a downward compatibility with corresponding re-use of software across the various layers. However, the advantage of full downward compatibility for OWL (that any OWL aware processor will also provide correct interpretations of any RDF Schema document) is only achieved for OWL Full, at the cost of computational intractability.

Chapter overview

Section 2 presents OWL in some detail. Because OWL is such a new language, only very limited examples of its use have been published. Section 3 therefore illustrates the language by giving a few examples.

2 The OWL Language

Syntax

OWL builds on RDF and RDF Schema, and uses RDF's XML syntax. Since this is the primary syntax for OWL, we will use it here, but it will soon become clear that RDF/XML does not provide a very readable syntax. Because of this, other syntactic forms for OWL have also been defined:

- an XML-based syntax which does not follow the RDF conventions. This makes this syntax already significantly easier to read by humans.
- an abstract syntax which is used in the language specification document. This syntax is much more compact and readable then either the XML syntax or the RDF/XML syntax
- a graphical syntax based on the conventions of the UML language (Universal Modelling Language). Since UML is widely used, this will be an easy way for people to get familiar with OWL.

Header

OWL documents are usually called *OWL ontologies*, and are RDF documents. So the root element of a OWL ontology is an `rdf:RDF` element which also specifies a number of namespaces. For example:

```
<rdf:RDF
    xmlns:owl ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd ="http://www.w3.org/2001/XLMSchema#">
```

An OWL ontology may start with a collection of assertions for housekeeping purposes. These assertions are grouped under an `owl:Ontology` element which contains comments, version control and inclusion of other ontologies. For example:

```
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:priorVersion
        rdf:resource="http://www.mydomain.org/uni-ns-old"/>
  <owl:imports rdf:resource="http://www.mydomain.org/persons"/>
  <rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>
```

The only of these assertions which has any consequences for the logical meaning of the ontology is `owl:imports`: this lists other ontologies whose content is assumed to be part of the current document. ontology. Notice that while namespaces are used for disambiguation purposes, imported ontologies provide definitions that can be used. Usually there will be an import element for each used namespace, but it is possible to import additional ontologies, for example ontologies that provide definitions without introducing any new names.

Also note that `owl:imports` is a transitive property: if ontology A imports ontology B, and ontology B imports ontology C, then ontology A also imports ontology C.

Class elements

Classes are defined using a `owl:Class` element[7]. For example, we can define a class `associateProfessor` as follows:

```
<owl:Class rdf:ID="associateProfessor">
   <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
</owl:Class>
```

[7] `owl:Class` is a subclass of `rdfs:Class`.

We can also say that this class is disjoint from the `professor` and `assistantProfessor` classes using `owl:disjointWith` elements. These elements can be included in the definition above, or can be added by referring to the id using `rdf:about`. This mechanism is inherited from RDF.

```
<owl:Class rdf:about="associateProfessor">
   <owl:disjointWith rdf:resource="#professor"/>
   <owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>
```

Equivalence of classes can be defined using a `owl:equivalentClass` element:

```
<owl:Class rdf:ID="faculty">
   <owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>
```

Finally, there are two predefined classes, `owl:Thing` and `owl:Nothing`. The former is the most general class which contains everything (everything is a thing), the latter is the empty class. Thus every class is a subclass of `owl:Thing` and a superclass of `owl:Nothing`.

Property elements

In OWL there are two kinds of properties:

- *Object properties* which relate objects to other objects.
  Examples are `isTaughtBy`, `supervises` etc.
- *Datatype properties* which relate objects to datatype values.
  Examples are `phone`, `title`, `age` etc. OWL does not have any predefined data types, nor does it provide special definition facilities. Instead it allows one to use XML Schema data types, thus making use of the layered architecture the Semantic Web

Here is an example of a datatype property.

```
<owl:DatatypeProperty rdf:ID="age">
   <rdfs:range rdf:resource="http://www.w3.org/2001/XLMSchema
   #nonNegativeInteger"/>
</owl:DatatypeProperty>
```

User-defined data types will usually be collected in an XML schema, and then used in an OWL ontology.

Here is an example of an object property:

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
   <owl:domain rdf:resource="#course"/>
   <owl:range rdf:resource="#academicStaffMember"/>
   <rdfs:subPropertyOf rdf:resource="#involves"/>
</owl:ObjectProperty>
```

More than one domain and range may be declared. In this case the intersection of the domains, respectively ranges, is taken.

OWL allows us to relate "inverse properties". A typical example is `isTaughtBy` and `teaches`.

```
<owl:ObjectProperty rdf:ID="teaches">
   <rdfs:range rdf:resource="#course"/>
   <rdfs:domain rdf:resource="#academicStaffMember"/>
   <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```

Actually domain and range can be inherited from the inverse property (interchange domain with range).

Equivalence of properties can be defined using a `owl:equivalentProperty` element.

```
<owl:ObjectProperty rdf:ID="lecturesIn">
   <owl:equivalentProperty rdf:resource="#teaches"/>
</owl:ObjectProperty>
```

Property restrictions

With `rdfs:subClassOf` we can specify a class C to be subclass of another class C'; then every instance of C is also an instance of C'.

Now suppose we wish to declare, instead, that the class C satisfies certain conditions, that is, all instances of C satisfy the conditions. Obviously it is equivalent to saying that C is subclass of a class C', where C' collects all objects that satisfy the conditions. That is exactly how it is done in OWL, as we will show. Note that, in general, C' can remain anonymous, as we will explain below.

The following element requires first year courses to be taught by professors only (according to a questionable view, older and more senior academics are better at teaching).

```
<owl:Class rdf:about="#firstYearCourse">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#isTaughtBy"/>
         <owl:allValuesFrom rdf:resource="#Professor"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

`owl:allValuesFrom` is used to specify the class of possible values the property specified by `owl:onProperty` can take (in other words: all values of the property must come from this class). In our example, only professors are allowed as values of the property `isTaughtBy`.

We can declare that mathematics courses are taught by David Billington as follows:

```
<owl:Class rdf:about="#mathCourse">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#isTaughtBy"/>
         <owl:hasValue rdf:resource="#949352"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

`owl:hasValue` states a specific value that the property, specified by `owl:onProperty` must have.

And we can declare that all academic staff members must teach at least one undergraduate course as follows:

```
<owl:Class rdf:about="#academicStaffMember">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#teaches"/>
         <owl:someValuesFrom rdf:resource="#undergraduateCourse"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Let us compare `owl:allValuesFrom` and `owl:someValuesFrom`. The example using the former requires *every* person who teaches an instance of the class, a first year subject, to be a professor. In terms of logic we have a *universal quantification*.

The example using the latter requires that *there exists* an undergraduate course that is taught by an instance of the class, an academic staff member. It is still possible that the same academic teaches postgraduate courses, in addition. In terms of logic we have an *existential quantification*.

In general, a `owl:Restriction` element contains a `owl:onProperty` element, and one or more restriction declarations. One type of restriction declarations are those that define restrictions on the kinds of values the property can take:
`owl:allValuesFrom`, `owl:hasValue` and `owl:someValuesFrom`. Another type are *cardinality restrictions*. For example, we can require every course to be taught by at least someone.

```
<owl:Class rdf:about="#course">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#isTaughtBy"/>
   <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
     1
   </owl:minCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Notice that we had to specify that the literal "1" is to be interpreted as a nonNegativeInteger (instead of, say, a string), and that we used the `xsd` namespace declaration made in the header element to refer to the XML Schema document.

Or we might specify that, for practical reasons, a department must have at least ten and at most thirty members.

```
<owl:Class rdf:about="#department">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#hasMember"/>
   <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
    10
   </owl:minCardinality>
   <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
    30
   </owl:maxCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

It is possible to specify a precise number. For example, a PhD student must have exactly two supervisors. This can be achieved by using the same number in
`owl:minCardinality` and `owl:maxCardinality`. For convenience, OWL offers also `owl:cardinality`.

We conclude by noting that `owl:Restriction` defines an anonymous class which has no id, is not defined by `owl:Class` and has only a local scope: it can only be used in the one place where the restriction appears. When we talk about classes please bare in mind the twofold meaning: classes that are defined by `owl:Class` with an id, and local anonymous classes as collections of objects that satisfy certain restriction conditions, or as combinations of other classes, as we will see shortly. The latter are sometimes called *class expressions*.

Special properties

Some properties of property elements can be defined directly:

- `owl:TransitiveProperty` defines a transitive property, such as "has better grade than", "is taller than", "is ancestor of" etc.
- `owl:SymmetricProperty` defines a symmetric property, such as "has same grade as", "is sibling of", etc.
- `owl:FunctionalProperty` defines a property that has at most one unique value for each object, such as "age", "height", "directSupervisor" etc.
- `owl:InverseFunctionalProperty` defines a property for which two different objects cannot have the same value, for example the property "isTheSocialSecurityNumberfor" (a social security number is assigned to one person only).

An example of the syntactic form of the above is:

```
<owl:ObjectProperty rdf:ID="hasSameGradeAs">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
  <rdfs:domain rdf:resource="#student" />
  <rdfs:range rdf:resource="#student" />
</owl:ObjectProperty>
```

Boolean combinations

It is possible to talk about Boolean combinations (union, intersection, complement) of classes (be it defined by `owl:Class` or by class expressions). For example, we can say that courses and staff members are disjoint as follows:

```
<owl:Class rdf:about="#course">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:complementOf rdf:resource="#staffMember"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

This says that every course is an instance of the complement of staff members, that is, no course is a staff member. Note that this statement could also have been expressed using `owl:disjointWith`.

The union of classes is built using `owl:unionOf`.

```
<owl:Class rdf:ID="peopleAtUni">
   <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#staffMember"/>
      <owl:Class rdf:about="#student"/>
   </owl:unionOf>
</owl:Class>
```

The `rdf:parseType` attribute is a shorthand for an explicit syntax for building list with `<rdf:first>` and `<rdf:rest>` tags. Such lists are required because the built-in containers of RDF have a serious limitation: there is no way to close them, i.e., to say "these are all the members of the container". This is because, while one graph may describe some of the members, there is no way to exclude the possibility that there is another graph somewhere that describes additional members. The list syntax provides exactly this facility, but is very verbose, which motivates the `rdf:parseType` shorthand notation.

Note that this does not say that the new class is a subclass of the union, but rather that the new class is *equal* to the union. In other words, we have stated an *equivalence of classes*. Also, we did not specify that the two classes must be disjoint: it is possible that a staff member is also a student.

Intersection is stated with `owl:intersectionOf`.

```
<owl:Class rdf:ID="facultyInCS">
   <owl:intersectionOf rdf:parseType="owl:collection">
      <owl:Class rdf:about="#faculty"/>
      <Restriction>
         <owl:onProperty rdf:resource="#belongsTo"/>
         <owl:hasValue rdf:resource="#CSDepartment"/>
      </Restriction>
   </owl:intersectionOf>
</owl:Class>
```

Note that we have built the intersection of two classes, one of which was defined anonymously: the class of all objects belonging to the CS department. This class is intersected with `faculty` to give us the faculty in the CS department.

Further we note that Boolean combinations can be nested arbitrarily. The following example defines administrative staff to be those staff members that are neither faculty nor technical support staff.

```
<owl:Class rdf:ID="adminStaff">
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#staffMember"/>
      <owl:complementOf>
         <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#faculty"/>
            <owl:Class rdf:about="#techSupportStaff"/>
         </owl:unionOf>
      </owl:complementOf>
   </owl:intersectionOf>
</owl:Class>
```

Enumerations

An enumeration is a `owl:oneOf` element, and is used to define a class by listing all its elements.

```
<owl:oneOf rdf:parseType="Collection">
   <owl:Thing rdf:about="#Monday"/>
   <owl:Thing rdf:about="#Tuesday"/>
   <owl:Thing rdf:about="#Wednesday"/>
   <owl:Thing rdf:about="#Thursday"/>
   <owl:Thing rdf:about="#Friday"/>
   <owl:Thing rdf:about="#Saturday"/>
   <owl:Thing rdf:about="#Sunday"/>
</owl:oneOf>
```

Instances

Instances of classes are declared as in RDF. For example:

```
<rdf:Description rdf:ID="949352">
   <rdf:type rdf:resource="#academicStaffMember"/>
</rdf:Description>
```

or equivalently:

```
<academicStaffMember rdf:ID="949352"/>
```

We can also provide further details, such as:

```
<academicStaffMember rdf:ID="949352">
   <uni:age rdf:datatype="&xsd;integer">39<uni:age>
</academicStaffMember>
```

Unlike typical database systems, OWL does not adopt the *unique names assumption*, thus: just because two instances have a different name (or: ID), that does not imply that they are indeed different individuals. For example, if we state that each course is taught by at most one one staff member:

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>
```

and we subsequently state that a given course is taught by two staff members:

```
<course rdf:about="CIT1111">
  <isTaughtBy rdf:resource="949318">
  <isTaughtBy rdf:resource="949352">
</course>
```

this does *not* cause an OWL reasoner to flag an error. After all, the system could validly infer that the resources `"949318"` and `"949352"` are apparently equal. To ensure that different individuals are indeed recognised as such, we must explicitly assert their inequality:

```
<lecturer rdf:about="949318">
  <owl:differentFrom rdf:resource="949352">
</lecturer>
```

Because such inequality statements occur frequently, and the required number of such statements would explode if we wanted to state the inequality of a large number of individuals, OWL provides a shorthand notation to assert the pairwise inequality of all individuals in a given list:

```
<owl:allDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <lecturer rdf:about="949318">
    <lecturer rdf:about="949352">
    <lecturer rdf:about="949111">
  </owl:distinctMembers>
</owl:allDifferent>
```

Note that `owl:distinctMembers` can only be used in combination with `owl:AllDifferent`.

Datatypes

Although XML Schema provides a mechanism to construct user-defined datatypes (e.g. the datatype of `adultAge` as all integers greater than 18, or the datatype of all strings starting with a number), such derived datatypes cannot be used in OWL. In fact, not even all of of the many the built-in XML Schema datatypes can be used in OWL. The OWL reference document lists all the XML Schema datatypes that can be used, but these include the most frequently used types such as string, integer, boolean, time and date.

Versioning information

We have already seen the *owl:priorVersion* statement as part of the header information to indicate earlier versions of the current ontology. This information has not formal model-theoretic semantics but can be exploited by humans readers and programs alike for the purposes of ontology management.

Besides *owl:priorVersion*, OWL has three more statements to indicate further informal versioning information. None of these carry any formal meaning.

- An `owl:versionInfo`statement generally contains a string giving information about the current version, for example RCS/CVS keywords.
- An `owl:backwardCompatibleWith` statement contains a reference to another ontology. This identifies the specified ontology as a prior version of the containing ontology, and further indicates that it is backward compatible with it. In particular, this indicates that all identifiers from the previous version have the same intended interpretations in the new version. Thus, it is a hint to document authors that they can safely change their documents to commit to the new version (by simply updating namespace declarations and `owl:imports` statements to refer to the URL of the new version).
- An `owl:incompatibleWith` on the other hand indicates that the containing ontology is a later version of the referenced ontology, but is not backward compatible with it. Essentially, this is for use by ontology authors who want to be explicit that documents cannot upgrade to use the new version without checking whether changes are required.

Layering of OWL

Now that we have discussed all the language constructors of OWL, we can completely specify which features of the language can be used in which sublanguage (OWL Full, DL and Lite):

OWL Full

In OWL Full, all the language constructors can be used in any combination as long as the result is legal RDF.

OWL DL

In order to exploit the formal underpinnings and computational tractability of Description Logics, the following constraints must be obeyed in an OWL DL ontology:

- *Vocabulary Partitioning:* any resource is allowed to be only either a class, a datatype, a datatype properties, an object properties, an individuals, a data value or part of the built-in vocabulary, and not more than one of these. This means that, for example, a class cannot be at the same time an individual, or that a property cannot have values some values from a datatype and some values from a class (this would make it both a datatype property and an object property).
- *Explicit typing:* not only must all resources be partitioned (as prescribed in the previous constraint), but this partitioning must be stated explicitly. For example, if an ontology contains the following:

  ```
  <owl:Class rdf:ID="C1">
    <rdfs:subClassOf rdf:about="#C2" />
  </owl:Class>
  ```

  this already entails that `C2` is a class (by virtue of the range specification of rdfs:subClassOf). Nevertheless, an OWL DL ontology must *explicitly* state this information:

  ```
  <owl:Class rdf:ID="C2"/>
  ```

- *Property Separation:* By virtue of the first constraint, the set of object properties and datatype properties are disjoint. This implies that inverse properties, and functional, inverse functional and symmetric characteristics can never be specified for datatype properties.
- *No transitive cardinality restrictions:* no cardinality restrictions may be placed on transitive properties (or their subproperties, which are of course also transitive, by implication).
- *Restricted anonymous classes:* anonymous classes are only allowed in the domain and range of `owl:equivalentClass` and `owl:disjointWith`, and in the range (not the domain) of `rdfs:subClassOf`.

OWL Lite

An OWL ontology must be an OWL DL ontology, and must further satisfy the following constraints:

- the constructors `owl:oneOf`, `owl:disjointWith`, `owl:unionOf`, `owl:complementOf` and `owl:hasValue` are not allowed

- cardinality statements (both minimal, maximal and exact cardinality) can only be made on the values 0 or 1, and no longer on arbitrary non-negative integers.
- `owl:equivalentClass` statements can no longer be made between anonymous classes, but only between class identifiers.

3 Examples

3.1 An African Wildlife Ontology

This example shows an ontology that describes part of the African wildlife. Figure 2 shows the basic classes and their subclass relationships.

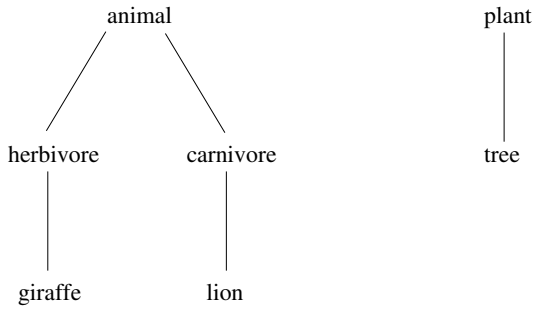

Fig. 2. Classes and subclasses of the African wildlife ontology

Note that the subclass information is only part of the information included in the ontology. The entire graph is much bigger. Figure 3 shows the graphical representation of the statement that branches are parts of trees.

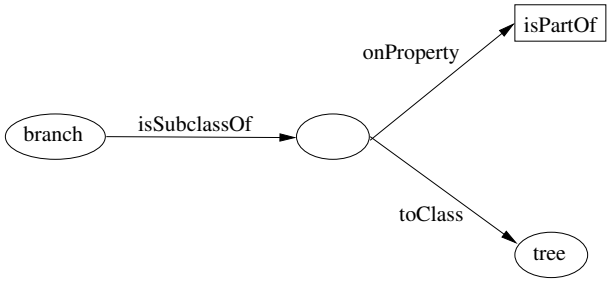

Fig. 3. Branches are parts of trees

Below we show the ontology, with comments written using `rdfs:comment`.

```
<rdf:RDF
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl ="http://www.w3.org/2002/07/owl#"
         xmlns="http://www.mydomain.org/african">

<owl:Ontology rdf:about="">
   <owl:VersionInfo>
      My example version 1.2, 17 October 2002
   </owl:VersionInfo>
</owl:Ontology>

<owl:Class rdf:ID="animal">
   <rdfs:comment>Animals form a class</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="plant">
   <rdfs:comment>
   Plants form a class disjoint from animals
   </rdfs:comment>
   <owl:disjointWith="#animal"/>
</owl:Class>

<owl:Class rdf:ID="tree">
   <rdfs:comment>Trees are a type of plants</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#plant"/>
</owl:Class>

<owl:Class rdf:ID="branch">
   <rdfs:comment>Branches are parts of trees </rdfs:comment>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#is-part-of"/>
         <ow:allValuesFrom rdf:resource="#tree"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="leaf">
   <rdfs:comment>Leaves are parts of branches</rdfs:comment>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#is-part-of"/>
         <owl:allValuesFrom rdf:resource="#branch"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="herbivore">
```

```
   <rdfs:comment>
   Herbivores are exactly those animals that eat only plants,
      or parts of plants
   </rdfs:comment>
   <owl:intersectionOf rdf:parsetype="Collection">
      <owl:Class rdf:about="#animal"/>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#eats"/>
         <owl:allValuesFrom>
            <owl:unionOf rdf:parsetype="Collection">
               <owl:Class rdf:about="#plant"/>
               <owl:Restriction>
                  <owl:onProperty rdf:resource="#is-part-of"/>
                  <owl:allValuesFrom rdf:resource="#plant"/>
               </owl:Restriction>
            </owl:unionOf>
         </owl:allValuesFrom>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="carnivore">
   <rdfs:comment>Carnivores are exactly those animals
      that eat also animals</rdfs:comment>
   <owl:intersectionOf rdf:parsetype="Collection">
      <owl:Class rdf:about="#animal"/>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#eats"/>
         <owl:someValuesFrom rdf:resource="#animal"/>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="giraffe">
   <rdfs:comment>Giraffes are herbivores, and they
      eat only leaves</rdfs:comment>
   <rdfs:subClassOf rdf:type="#herbivore"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#eats"/>
         <owl:allValuesFrom rdf:resource="#leaf"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="lion">
   <rdfs:comment>Lions are animals that eat
      only herbivores</rdfs:comment>
   <rdfs:subClassOf rdf:type="#carnivore"/>
```

```
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#eats"/>
            <owl:allValuesFrom rdf:resource="#herbivore"/>
         </owl:Restriction>
      </rdfs:subClassOf>
   </owl:Class>

   <owl:Class rdf:ID="tasty-plant">
      <rdfs:comment>Tasty plants are plants that are eaten
         both by herbivores and carnivores</rdfs:comment>
      <rdfs:subClassOf rdf:resource="#plant"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#eaten-by"/>
            <owl:someValuesFrom>
                  <owl:Class rdf:about="#herbivore"/>
            </owl:someValuesFrom>
         </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#eaten-by"/>
            <owl:someValuesFrom>
                  <owl:Class rdf:about="#carnivore"/>
            </owl:someValuesFrom>
         </owl:Restriction>
      </rdfs:subClassOf>
   </owl:Class>

   <owl:TransitiveProperty rdf:ID="is-part-of"/>

   <owl:ObjectProperty rdf:ID="eats">
      <rdfs:domain rdf:resource="#animal"/>
   </owl:ObjectProperty>

   <owl:ObjectProperty rdf:ID="eaten-by">
      <owl:inverseOf rdf:resource="#eats"/>
   </owl:ObjectProperty>

   </rdf:RDF>
```

3.2 A printer ontology

The classes and subclass relationships in this example are shown in Figure 4

```
   <rdf:RDF
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

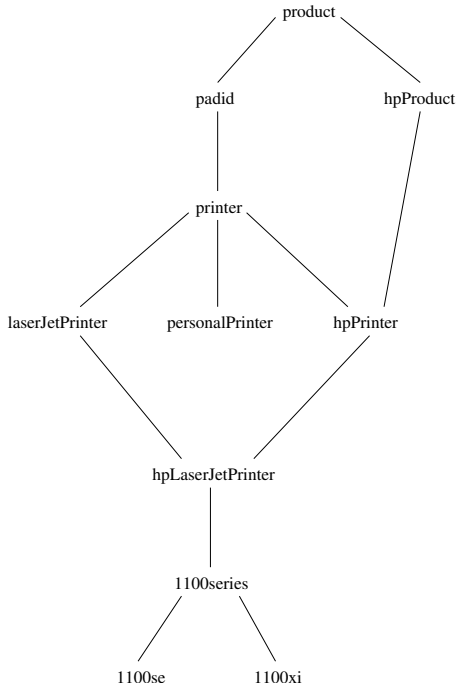

Fig. 4. Classes and subclasses of the printer ontology

```
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
          xmlns:owl ="http://www.w3.org/2002/07/owl#"
          xmlns:xsd="http://www.w3.org/2001/XLMSchema#"
          xmlns="http://www.mydomain.org/printer#">

<owl:Ontology rdf:about="">
   <owl:VersionInfo>
   My example version 1.2, 17 October 2002
   </owl:VersionInfo>

</owl:Ontology>

<owl:Class rdf:ID="product">
   <rdfs:comment>Products form a class</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="padid">
   <rdfs:comment>Printing and digital imaging devices
      form a subclass of products</rdfs:comment>
   <rdfs:label>Device</rdfs:label>
   <rdfs:subClassOf rdf:resource="#product"/>
</owl:Class>
```

```
<owl:Class rdf:ID="hpProduct">
   <rdfs:comment>HP products are exactly those products
      that are manufactured by Hewlett Packard</rdfs:comment>
   <owl:intersectionOf>
      <owl:Class rdf:about="#product"/>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#manufactured-by"/>
         <owl:hasValue>
            <xsd:string rdf:value="Hewlett Packard"/>
         </owl:hasValue>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="printer">
   <rdfs:comment>Printers are printing and
      digital imaging devices</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#padid"/>
</owl:Class>

<owl:Class rdf:ID="personalPrinter">
   <rdfs:comment>Printers for personal use form
      a subclass of printers</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#printer"/>
</owl:Class>

<owl:Class rdf:ID="hpPrinter">
   <rdfs:comment>HP printers are HP products and printers
   </rdfs:comment>
   <rdfs:subClassOf rdf:resource="#printer"/>
   <rdfs:subClassOf rdf:resource="#hpProduct"/>
</owl:Class>

<owl:Class rdf:ID="laserJetPrinter">
   <rdfs:comment>Laser Jet printers are exactly those printers
      that use laser jet printing technology</rdfs:comment>
   <owl:intersectionOf >
      <owl:Class rdf:about="#printer"/>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#printingTechnology"/>
         <owl:hasValue><xsd:string rdf:value="laser jet"/></owl:hasValue>
      </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="hpLaserJetPrinter">
   <rdfs:comment>HP laser jet printers are HP products
      and laser jet printers</rdfs:comment>
```

```
   <rdfs:subClassOf rdf:resource="#laserJetPrinter"/>
   <rdfs:subClassOf rdf:resource="#hpPrinter"/>
</owl:Class>

<owl:Class rdf:ID="1100series">
   <rdfs:comment>1100series printers are HP laser jet printers with 8ppm
      printing speed and 600dpi printing resolution</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#hpLaserJetPrinter"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#printingSpeed"/>
         <owl:hasValue><xsd:string rdf:value="8ppm"/></owl:hasValue>
      </owl:Restriction>
   </rdfs:subClassOf>
   <owl:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#printingResolution"/>
         <owl:hasValue><xsd:string rdf:value="600dpi"/></owl:hasValue>
      </owl:Restriction>
   </owl:subClassOf>
</owl:Class>

<owl:Class rdf:ID="1100se">
   <rdfs:comment>1100se printers belong to the 1100 series
      and cost $450</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#1100series"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#price"/>
         <owl:hasValue>
            <xsd:integer rdf:value="450"/>
         </owl:hasValue>
      </owl:Restriction>
   </owl:subClassOf>
</owl:Class>

<owl:Class rdf:ID="1100xi">
   <rdfs:comment>1100xi printers belong to the 1100 series
      and cost $350</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#1100series"/>
   <owl:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#price"/>
         <owl:hasValue>
            <xsd:integer rdf:value="350"/>
         </owl:hasValue>
      </owl:Restriction>
   </owl:subClassOf>
</owl:Class>
```

```
<owl:DatatypeProperty rdf:ID="manufactured-by">
   <rdfs:domain rdf:resource="#product"/>
   <rdfs:range
      rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="price">
   <rdfs:domain rdf:resource="#product"/>
   <rdfs:range
      rdf:resource="&xsd;nonNegativeInteger"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="printingTechnology">
   <rdfs:domain rdf:resource="#printer"/>
   <rdfs:range
      rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="printingResolution">
   <rdfs:domain rdf:resource="#printer"/>
   <rdfs:range
      rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="printingSpeed">
   <rdfs:domain rdf:resource="#printer"/>
   <rdfs:range
      rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

</rdf:RDF>
```

This ontology demonstrates that siblings in a hierarchy tree need not be disjoint. For example, a personal printer may be a HP printer or a LaserJet printer, though the three classes involved are subclasses of the class of all printers.

4 Summary

- OWL is the proposed standard for Web ontologies. It allows us to describe the semantics of knowledge in a machine-accessible way.
- OWL builds upon RDF and RDF Schema: (XML-based) RDF syntax is used; instances are defined using RDF descriptions; and most RDFS modelling primitives are used.

- Formal semantics and reasoning support is provided through the mapping of OWL on logics. Predicate logic and description logics have been used for this purpose.

While OWL is sufficiently rich to be used in practice, extensions are in the making. They will provide further logical features, including rules.

References

The key references for OWL (at the date of writing, April 2003):

1. D. McGuinness and F van Harmelen (eds) *OWL Web Ontology Language Overview*
   `http://www.w3.org/TR/2003/WD-owl-features-20030331/`
2. M. Dean, G. Schreiber (eds), F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, L. Stein, *OWL Web Ontology Language Reference*
   `http://www.w3.org/TR/2003/WD-owl-ref-20030331/`
3. M. Smith, C. Welty, D. McGuinness, *OWL Web Ontology Language Guide*
   `http://www.w3.org/TR/2003/WD-owl-guide-20030331/`
4. P. Patel-Schneider, P. Hayes, I. Horrocks, *OWL Web Ontology Language Semantics and Abstract Syntax*
   `http://www.w3.org/TR/2003/WD-owl-semantics-20030331/`
5. J. Hefflin, *Web Ontology Language (OWL) Use Cases and Requirements*
   `http://www.w3.org/TR/2003/WD-webont-req-20030331/`

Further interesting articles related to DAML+OIL and OIL include:

6. J. Broekstra et al. Enabling knowledge representation on the Web by Extending RDF Schema. In *Proc. 10th World Wide Web Conference (WWW'10)*, 2001.
7. D. Fensel et al. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems* 16,2 (2001).
8. D. McGuinness. Ontologies come of age. In D. Fensel et al. (eds): *The Semantic Web: Why, What, and How*. MIT Press 2002.
9. P. Patel-Schneider, I. Horrocks, F. van Harmelen, Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web, *Proceedings of AAAI'02*.

There is a number of interesting Web sites. A key site is:

10. On OWL: `http://www.w3.org/2001/sw/WebOnt/`
11. On its precursor DAML+OIL: `http://www.daml.org` Interesting subpages include:
    a) `http://www.daml.org/language`
    b) `http://www.daml.org/ontologies`

c) `http://www.daml.org/tools`

The two most relevant chapters from this Handbook are

12. B. McBride, *The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS*, in: *The Handbook on Ontologies in Information Systems*, S. Staab, R. Studer (eds.), Springer Verlag, 2003.
13. F. Baader, I. Horrocks and U. Sattler, *Description Logics*, in: *The Handbook on Ontologies in Information Systems*, S. Staab, R. Studer (eds.), Springer Verlag, 2003.