OWL: Web Ontology Langauge

- OWL = RDF Schema + new constructs for better expressiveness
- OWL documents became a formal W3C recommendation on February 10th of 2004 (also known as OWL 1)
- OWL 2 became a formal W3C standard On October 27th of 2009
- they provide **additional primitives** for **heavyweight** ontologies

OWL: Web Ontology Langauge

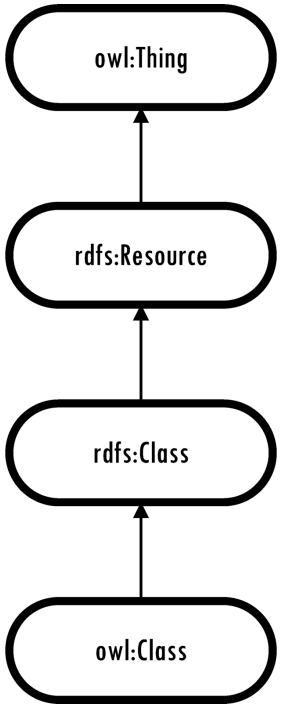
- like RDFS, OWL can be viewed as a collection of terms we can use to define classes and properties for a specific application domain
- these predefined OWL terms all have the following URI as their leading string (applicable to both OWL 1 and OWL 2),

`http://www.w3.org/2002/07/owl#`

- and by convention, this URI prefix string is associated with namespace prefix `owl:`, and is typically used in RDF/XML documents with the prefix `owl`
- most of the language constructs in OWL (1/2) are intuitive, yet some of them do need some explanation

OWL: defining classes

- OWL's view of classes
- owl:Class should be used for defining classes




```
<rdf:Description rdf:about="http://www.example.com/camera#Camera">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
</rdf:Description>
```

or, in short-form,

```
<owl:Class rdf:about="http://www.example.com/camera#Camera">
</owl:Class>
```

OWL: defining classes

- and more examples:

```
<?xml version="1.0"?>
<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:myCamera="http://www.example.com/camera#"
         xml:base="http://www.example.com/camera#">

  <owl:Class rdf:about="http://www.example.com/camera#Camera">
  </owl:Class>

  <owl:Class rdf:about="http://www.example.com/camera#Lens">
  </owl:Class>
  ...
  <owl:Class rdf:about="http://www.example.com/camera#Digital">
     <rdfs:subClassOf rdf:resource="#Camera"/>
  </owl:Class>

  <owl:Class rdf:about="http://www.liyangyu.com/camera#Film">
     <rdfs:subClassOf rdf:resource="#Camera"/>
  </owl:Class>
```

OWL: defining classes

- defining classes by localizing global properties

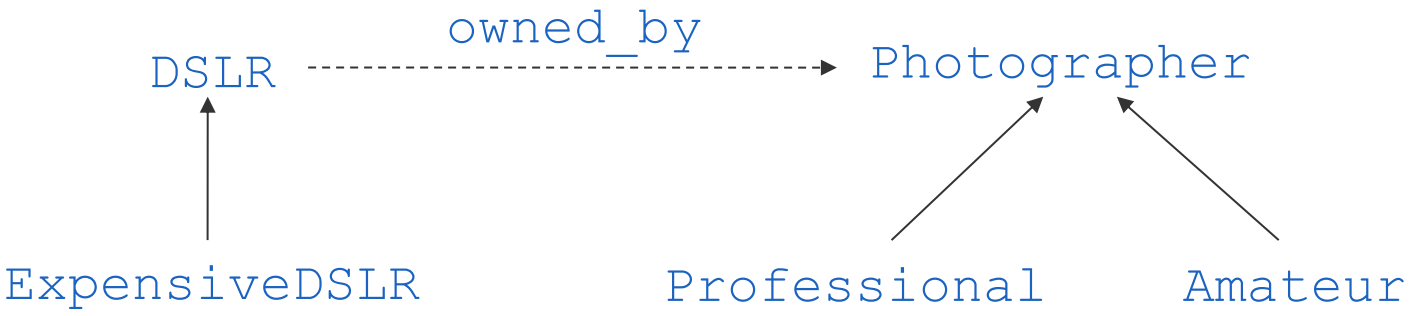
```
<rdf:Property
     rdf:about="http://www.liyangyu.com/camera#owned_by">
  <rdfs:domain rdf:resource="#DSLR"/>
  <rdfs:range rdf:resource="#Photographer"/>
</rdf:Property>
```

- `rdfs:range` imposes a global restriction on `owned_by` property, i.e., the `rdfs:range` value applies to `Photographer` class and all sub-classes of `Photographer` class

what if we want to express the following fact: DSLR, especially an expensive one, is normally used by professional photographers?

OWL: defining classes

our solution:




```
<owl:Class rdf:about="http://www.example.com/camera#Professional">
   <rdfs:subClassOf rdf:resource="#Photographer"/>
</owl:Class>

<owl:Class rdf:about="http://www.example.com/camera#Amateur">
   <rdfs:subClassOf rdf:resource="#Photographer"/>
</owl:Class>

<owl:Class rdf:about="http://www.example.com/camera#ExpensiveDSLR">
   <rdfs:subClassOf rdf:resource="#DSLR"/>
</owl:Class>
```

OWL: defining classes

- since `owned_by` has `DSLR` as its `rdfs:domain` and `Photographer` as its `rdfs:value`, and given the fact that `ExpensiveDSLR` is a sub-class of `DSLR`, `Professional` and `Amateur` are both sub-classes of `Photographer`, these new sub-classes all inherit the `owned_by` property
- so, both of the following are correct (which is not what we wanted):

```
ExpensiveDSLR owned_by Professional
ExpensiveDSLR owned_by Amateur
```

- we need to modify the definition of `ExpensiveDSLR` to make sure it can be owned *only* by Professional photographers?

OWL: defining classes

here is the new definition:

```
<owl:Class rdf:about="http://www.example.com/camera#ExpensiveDSLR">
   <rdfs:subClassOf rdf:resource="#DSLR"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#owned_by"/>
         <owl:allValuesFrom rdf:resource="#Professional"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

OWL: defining classes

here is the new definition:

```
<owl:Class rdf:about="http://www.example.com/camera#ExpensiveDSLR">
   <rdfs:subClassOf rdf:resource="#DSLR"/>
    <rdfs:subClassOf>
       <owl:Restriction>
          <owl:onProperty rdf:resource="#owned_by"/>
          <owl:allValuesFrom rdf:resource="#Professional"/>
       </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

`ExpensiveDSLR` is an intersection of two different classes

OWL: defining classes

here is the new definition:

```
<owl:Class rdf:about="http://www.example.com/camera#ExpensiveDSLR">
   <rdfs:subClassOf rdf:resource="#DSLR"/>
    <rdfs:subClassOf>
       <owl:Restriction>
          <owl:onProperty rdf:resource="#owned_by"/>
          <owl:allValuesFrom rdf:resource="#Professional"/>
       </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- `ExpensiveDSLR` is an intersection of two different classes
- `owl:Restriction` is an OWL 1 term used to describe an ***anonymous*** class, which is defined by adding some restriction on some property
- furthermore, all the instances of this anonymous class must satisfy this restriction, hence the term `owl:Restriction`

OWL: defining classes

here is the new definition:

```
<owl:Class rdf:about="http://www.example.com/camera#ExpensiveDSLR">
   <rdfs:subClassOf rdf:resource="#DSLR"/>
    <rdfs:subClassOf>
       <owl:Restriction>
          <owl:onProperty rdf:resource="#owned_by"/>
          <owl:allValuesFrom rdf:resource="#Professional"/>
       </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- the restriction itself has two parts
- the first part is about to which property this restriction is applied to, and this is specified by using `owl:onProperty` property
- the second part is about the property constraint itself, or, exactly what is the constraint
- `owl:allValuesFrom`: when this property is used, the value of the restricted property must all come from the specified class or data range

OWL: defining classes

here is the new definition:

```
<owl:Class rdf:about="http://www.example.com/camera#ExpensiveDSLR">
   <rdfs:subClassOf rdf:resource="#DSLR"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#owned_by"/>
         <owl:allValuesFrom rdf:resource="#Professional"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

we can now read this definition like this:

> *Here is a definition of class* `ExpensiveDSLR`*, it is a sub-class of* `DSLR`*, and a sub-class of an anonymous class which has a property* `owned_by` *and all values for this property must be instances of* `Professional`**.**

OWL: defining classes

what if we allow both amateur and professional photographers to own expensive DSLRs, however, we still require that at least one of the owners must be a **`Professional`**?

```
<owl:Class rdf:about="http://www.example.com/camera#ExpensiveDSLR">
   <rdfs:subClassOf rdf:resource="#DSLR"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#owned_by"/>
         <owl:someValuesFrom rdf:resource="#Professional"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

A class called `ExpensiveDSLR` *is defined. It is a sub-class of* `DSLR`*, and it has a property called* `owned_by`*. Furthermore, at least one value of* `owned_by` *property is an instance of* `Professional`*.*

OWL: defining classes

- another way to define class by adding restrictions on properties is to constrain the *cardinality* of a property based on the class on which it is intended to use
- class **`Digital`** represents a digital camera, and property `effectivePixel` represents the picture resolution of a given digital camera
- it would be useful to indicate that there can be only one `effectivePixel` value for any given digital camera
- also think about the case where one person can have only one SSN

OWL: defining classes

```
<owl:Class rdf:about="http://www.example.com/camera#Digital">
  <rdfs:subClassOf rdf:resource="#Camera"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#effectivePixel"/>
      <owl:cardinality rdf:datatype=
        "http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
        1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

- Can also have min/max cardinalities to express a range: `owl:minCardinality`, `owl:maxCardinality`

OWL: defining classes

- OWL also gives us the ability to construct classes by using set operators: `owl:intersectionOf`, `owl:unionOf`, `owl:complementOf`

- we can also construct classes by using Enumeration, Equivalent and Disjoint: `owl:oneOf`, `owl:equivalentClass`, `owl:disjointWith`

OWL: defining properties

- using RDFS, we have the following terms to use: `rdfs:domain`, `rdfs:range` and `rdfs:subPropertyOf`
- using these RDFS terms, the general procedure is to define the property first and then use it to connect two things together: connect one resource to another resource, or connect one resource to a typed or un-typed value

```
<rdf:Property rdf:about="http://www.example.com/camera#owned_by">
  <rdfs:domain rdf:resource="#DSLR"/>
  <rdfs:range rdf:resource="#Photographer"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.example.com/camera#model">
  <rdfs:domain rdf:resource="#Camera"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>

<rdfs:Datatype rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
```

OWL: defining properties

In the world of OWL 1/2, two different classes are used to implement these two different connections:

- `owl:ObjectProperty` is used to connect a resource to another resource
- `owl:DatatypePropery` is used to connect a resource to a `rdfs:Literal` (un-typed) or an XML schema built-in datatype (typed) value
- in addition, `owl:ObjectProperty` and `owl:DatatypeProperty` are both sub-classes of `rdf:Property`

```
<owl:ObjectProperty rdf:about="http://www.example.com/camera#owned_by">
  <rdfs:domain rdf:resource="#DSLR"/>
  <rdfs:range rdf:resource="#Photographer"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="http://www.example.com/camera#model">
  <rdfs:domain rdf:resource="#Camera"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<rdfs:Datatype rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
```

OWL: defining properties

OWL 1/2 provides much richer features when it comes to property definitions:

- property can be symmetric
- property can be transitive
- property can be functional
- property can be inverse functional
- property can be the inverse of another property

OWL: defining properties

symmetric property example:

```
<owl:ObjectProperty
    rdf:about="http://www.example.com/camera#friend_with">
  <rdf:type rdf:resource=
      "http://www.w3.org/2002/07/owl#SymmetricProperty"/>
  <rdfs:domain rdf:resource="#Photographer"/>
  <rdfs:range rdf:resource="#Photographer"/>
</owl:ObjectProperty>
```

of course, here is the short form:

```
<owl:SymmetricProperty
    rdf:about="http://www.example.com/camera#friend_with">
  <rdfs:domain rdf:resource="#Photographer"/>
  <rdfs:range rdf:resource="#Photographer"/>
</owl:SymmetricProperty>
```

OWL: defining properties

transitive property: if a resource **R1** is connected to resource **R2** by property **P**, and resource **R2** is connected to resource **R3** by the same property, then resource **R1** is also connected to resource **R3** by property **P**

```
<owl:ObjectProperty
     rdf:about="http://www.example.com/camera#betterQPRatio">
  <rdf:type rdf:resource=
     "http://www.w3.org/2002/07/owl#TransitiveProperty"/>
  <rdfs:domain rdf:resource="#Camera"/>
  <rdfs:range rdf:resource="#Camera"/>
</owl:ObjectProperty>
```

OWL: defining properties

functional property:

- describes the situation where for any given instance there is at most one value for that property
- a ***many-to-one*** relation: there is at most one unique `rdfs:range` value for each `rdfs:domain` instance
- example: each person has only one birthday, each camera has only one manufacturer...

```
<owl:ObjectProperty
  rdf:about="http://www.example.com/camera#manufactured_by">
  <rdf:type rdf:resource=
     "http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Camera"/>
  <rdfs:range rdf:resource="#Manufacturer"/>
</owl:ObjectProperty>
```

OWL: defining properties

functional property:

```
<myCamera:DSLR
          rdf:about="http://www.example.com/camera#Nikon_D300">
  <myCamera:manufactured_by
           rdf:resource="http://dbpedia.org/resource/Nikon"/>
</myCamera:DSLR>

<DSLR rdf:about="http://www.example.com/camera#Nikon_D300"
      xmlns="http://www.example.com/camera#">
  <manufactured_by rdf:resource=
                   "http://www.freebase.com/view/en/nikon"/>
</DSLR>
```

your application will claim this (reasoner can infer this):

```
<http://dbpedia.org/resource/Nikon> owl:sameAs
<http://www.freebase.com/view/en/nikon>.
```

OWL: defining properties

inverse functional property:

- for a given `rdfs:range` value, the value of the `rdfs:domain` property must be unique
- recall functional property: for a given `rdfs:domain` value, there a unique `rdfs:range` value
- example: email address, driver's license...

camera review example: let us assume the reviewers themselves are often photographers and let us assign a unique reviewer ID to each photographer - if two photographers have the same reviewerID, these two photographers should be the same person.

```
<owl:DatatypeProperty
    rdf:about="http://www.example.com/camera#reviewerID">
  <rdf:type rdf:resource=
   http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  <rdfs:domain rdf:resource="#Photographer"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

OWL: defining properties

- an even stronger statement about photographers and their reviewer IDs: not only one reviewer ID is used to identify just one photographer, but each photographer has also only one reviewer ID
- we need to define `reviewerID` property as *both* functional and inverse functional property

```
<owl:DatatypeProperty
    rdf:about="http://www.example.com/camera#reviewerID">
  <rdf:type rdf:resource=
       "http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdf:type rdf:resource=
   "http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  <rdfs:domain rdf:resource="#Photographer"/>
  <rdfs:range
      rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<rdfs:Datatype
      rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
```

OWL: defining properties

understand the difference between functional and inverse functional property:

- birthday is a functional property, it cannot be inverse functional property
- e-mail is an inverse functional property; it cannot be a functional property
- SSN, passport number (ID-like numbers) are often modeled as functional properties and inverse functional properties at the same time

OWL: a summary so far

- in RDFS, you can subclass existing classes… that is all
- in OWL, you can construct classes from existing ones:
  - ✓ through intersection, union, complement
  - ✓ enumerate its members
  - ✓ in OWL, you can define equivalent classes, or two classes without any common individuals
- in OWL, you can define classes by restricting the property values on another class
  - ✓ `allValuesFrom`, `someValuesFrom`
  - ✓ etc.

examples:

- `Carnivore` class represents those animals who eat only meat – all the values of it's `eat` property must come from `Meat` class
- `Driver` class represents those who at least drive a car – some of the values of its `drive` property should take the value of `Car` class

OWL: a summary so far

- OWL allows us to characterize the behavior of properties: symmetric, transitive, functional, inverse functional, …
  - ✓ if two resources have the same driver's license number, these two individuals are same
  - ✓ if individual A is friend with individual B, then B is friend with A
  - ✓ if A costs more than B, B costs more than C, then A costs more then C
  - ✓ and more …
- OWL also separates data and object properties; datatype property means that its range are typed literals

OWL: a summary so far

the Semantic Web is about coding meanings by using RDF statements and shared ontology terms and adding these meanings back to the current Web.