Web Ontology Language (OWL)

Athens 2011

Mikel Egaña Aranguren
3205 Facultad de Informática
Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo
28660 Boadilla del Monte
Spain

http://www.oeg-upm.net
megana@fi.upm.es
http://mikeleganaaranguren.com
Learning dynamic

Research and fill the wiki
http://delicias.dia.fi.upm.es/athens/index.php/OWL

Idea sharing

Hands-on
Estimated timetable

Today:

Introduction to OWL (15 min.)
Research OWL entities (5 min.)
Idea sharing OWL entities (5 min.)
Research OWL axioms (30 min.)
Idea sharing OWL axioms (15 min.)
Hands-on (40 min.)
Hands-on idea-sharing (10 min.)

Tomorrow:

OWL reasoning
Introduction to OWL
OWL is a Knowledge Representation language proposed by the W3C as a standard to codify ontologies in a prospective Semantic Web.
OWL is based in Description Logics

We can represent a knowledge domain computationally in an OWL ontology, in order to:

Apply automated reasoning: infer “new” knowledge, queries, consistency, classify entities against the ontology, …

Integrate knowledge from different resources
Everything about OWL 2:
http://www.w3.org/standards/techs/owl

Document overview:
http://www.w3.org/TR/2009/REC-owl2-overview-20091027/

Primer:
http://www.w3.org/TR/2009/REC-owl2-primer-20091027/

Manchester OWL + Protégé tutorial (Copied some examples :-)):
http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/
“OWL 1”: OWL lite, OWL DL, OWL Full

OWL 2 profiles
For computers: RDF/XML, OWL/XML, …

RDF/XML:

```xml
<owl:Class rdf:about="#arm">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#part_of"/>
      <owl:someValuesFrom rdf:resource="#body"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

For humans: Manchester OWL Syntax, functional, …

Manchester OWL Syntax: arm subClassOf art_of some body

http://www.co-ode.org/resources/reference/manchester_syntax/
Ontology editors:

Protégé: http://protege.stanford.edu/
TopBraid composer:
NeOn toolkit: http://neon-toolkit.org

APIs:

OWL API: http://owlapi.sourceforge.net/

Reasoners:

Pellet: http://clarkparsia.com/pellet/
HermiT: http://hermit-reasoner.com/
FaCT++: http://code.google.com/p/factplusplus/
Racer: http://www.racer-systems.com/
OWL semantics
An OWL ontology comprises:

**Entities**: the named elements from the knowledge domain, created by the ontology creator. Entities are identified using URIs (To work in a web setting)

**Axioms**: axioms relate the entities to each other using the OWL logic vocabulary

An OWL ontology can import other ontologies (`owl:import`): the entities of the imported ontology can be referenced by axioms on our ontology
OWL is “Axiom-centric”

Entities only “exist” as part of axioms, and therefore the only way of creating an entity in an ontology is by adding an axiom that refers to it. We cannot create the class $A$, but we can state that $A$ subClassOf owl:Thing
There are three types of entities in an OWL ontology:

Individuals

Properties

Classes
Individuals: the objects of the knowledge domain

- Italy
- England
- USA
- Gemma
- Matthew
- Fluffy
- Fido

(Tutorial Manchester)
Properties: they can be used to link individuals in binary relations

(Tutorial Manchester)
Classes: sets of individuals with common characteristics
Classes
Classes: Sets of individuals
Classes can be subclasses of other classes: all the instances of the subclass are also instances of the superclass (But no the other way around)
Classes are equivalent if the extent of their sets is exactly the same: all the instances of A are also instances of B and the other way around.
A taxonomy can be built combining different class-subclass axioms
In order to define the qualities that the individuals of a class must hold to be members of that class, *restrictions* on the number and type of binary relations are used.

Thus, the restrictions define the conditions that must be fulfilled to be a member of a given class.
For example, we can state (In our ontology!) that in order to be human something must eat plants.

Eating plants is a *necessary condition* to be human: all the humans eat plants, but there are other organisms that also eat plants that are not humans.

We can also define a *necessary and sufficient* condition: producing language is a unique quality of humans: if we find an individual (Organism) capable of producing language we can infer that is human, since no other organism does it.
Conditions are anonymous classes: the named class we are defining with such conditions can be a subclass (Necessary) or equivalent class (Necessary and sufficient) to the anonymous class.
The class *Humano* is a subclass (N) of the anonymous class comprised of the individuals that have at least one *come* binary relation with an individual of the class *Planta*.
The class **Humano** is equivalent (N+S) to the anonymous class comprised of the individuals that have at least one relation with the property **produce** with an individual of the class **Lenguaje**.
The classes with necessary and sufficient conditions are *defined* classes, and they are exploited for automated reasoning.

The classes with only necessary conditions are *primitive* classes.
Existential restrictions

owl:someValuesFrom: the anonymous class comprised of the individuals that, amongst other things, have at least one relation to an individual of a given class with a given property: humano subClassOf come some Planta

(Tutorial Manchester)
Universal restriction

owl:allValuesFrom: the anonymous class comprised of the individuals that, if having a relation with a given property, must be to an individual of a concrete class or none: humano subClassOf come only Organismo
hasValue

the anonymous class comprised of the individuals that have a relation to a concrete individual `humano subClassOf come value este_tomate`

(Tutorial Manchester)
Cardinal restrictions:

Min: humano subClassOf come min 1

Max: humano subClassOf come max 5

Exactly: humano subClassOf come exactly 3
QCR (Qualified Cardinality Constraint):

Min: humano subClassOf come min 1 Planta

Max: humano subClassOf come max 5 Planta

Exactly: humano subClassOf come exactly 3 Planta
We can state that a class is different to other class (They don't have any individual in common) using disjointFrom: `humano disjointFrom planta`.

We can state that two classes are the same (They have the same extent of individuals) using equivalentTo: `humano equivalentTo persona`.
Booleans

Not: humano subClassOf not (come some electrodomestico)

And (Intersection):  
man equivalentTo human and male

Or (Union):  
human equivalentTo woman or man
In a class hierarchy, the subclass “inherits” the conditions of the superclass: it can have further conditions but not a condition that conflicts with the conditions of the superclass.
Conditions can be very complex, combining different OWL elements
Properties
Object Properties

```
Matthew hasSister Gemma
```

DataType Properties

```
Matthew hasAge "25"^^xsd:integer
```

Annotation Properties*

```
JetEngine dc:creator "Matthew Horridge"
```
Object Properties
Property hierarchy:

Sub/SuperProperties

\( p \text{ SubPropertyOf } q \)
If \( A p B \), \( A q B \)
But if \( D q F \), not \( D p F \)

Equivalent Properties

Disjoint Properties
null
Functional

Inverse functional

Transitive

OWL semantics

(Tutorial Manchester)
OWL semantics

Symmetric

Antisymmetric*

Reflexive

Irreflexive*
Inverse properties

Matthew \text{hasParent} Jean

Matthew \text{hasChild} Jean

(Tutorial Manchester)
Domain and Range:

Usually classes or class unions

But any anonymous expression class can be used

They are not constraints, they are axioms
Data Type Properties
OWL semantics
Equivalent / sub-super / disjoint

Only Functional (No transitive, inverse functional, … )

Domain: ~ Object Properties

Range:

- Built-in datatypes

Data range expression
Annotation Properties
Add non-semantic annotations in natural language to entities, axioms or the ontology

rdfs:label, rdfs:comment, …

Dublin Core ([http://dublincore.org/](http://dublincore.org/))

Custom annotation properties

Language (en, es, ...) and type (xsd:string, …)
Individuals
An individual can be a member of one or more anonymous or named classes (Types)

An individual can be the same as other individual (SameAs)

An individual can be different from another individual (DifferentFrom)

Individuals can be related in binary relations (Object Properties):

my_wheel part_of my_car
my_wheel not part_of your_car

Individuals can be related with data (Data Type properties):

my_car has_power "90"^^xsd:positiveInteger
my_car not has_power "90"^^xsd:positiveInteger
OWL semantics
Some extra constructs
OWL oneOf
Role chains

- topObjectProperty
  - tiene_padre
  - tiene_primo
  - tiene_sobrino

Description: tiene_primo
- Domains (intersection)
- Ranges (intersection)
- Equivalent object properties
- Super properties
- Inverse properties
- Disjoint properties
- Property chains

- tiene_padre o tiene_sobrino SubPropertyOf tiene_primo
OWL Self

- Class hierarchy:
  - Thing
  - Familia
  - FamiliaMikel

- Annotations:
  - Familia

- Description:
  - Familia
  - Equivalent classes

- Superclasses:
  - hace_negocios_con some Self
OWL keys

http://www.w3.org/TR/2009/REC-owl2-primer-20091027/#Keys

~ “datatype inverse functional”

numero_seguridad_social “7”^^xsd:integer
numero_seguridad_social “8”^^xsd:integer
numero_seguridad_social “7”^^xsd:integer