OWL Web Ontology Language

Dipartimento di Informatica, Sistemistica e Comunicazione
Università di Milano-Bicocca

bandini@disco.unimib.it
tel. 02 6448 7835

tel. 02 6448 7879

ale.m@disco.unimib.it
matteo.palmonari@disco.unimib.it
andrea.bonomi@disco.unimib.it
What are ontologies?

Ontology is a study of conceptions of reality and the nature of being.

An ontology is a specification of a conceptualization.

Tom Gruber

Philosophy

Computer Science
How is Tom Gruber?

Tom Gruber is an innovator in technologies that extend human intelligence. Building on early work in computer-mediated learning and artificial intelligence, he focuses on creating environments for collective intelligence.

At Stanford University in the early 1990's, Gruber was a pioneer in the use of the Web for knowledge sharing and collaboration. He established the DARPA Knowledge Sharing Library, a web-based public exchange for ontologies, software, and knowledge bases. Gruber also led the Stanford team that invented and deployed the first Virtual Document applications on the web that generate natural language explanations in response to questions.

From the Short Biography of Tom Gruber - http://tomgruber.org/
An example:
A Concept Hierarchy

Relations between concepts.
Why ontologies in CS?

- Applications:
  - E-commerce, Ubiquitous Computing, Expert Systems, ERP, e-Government, etc
- System Modeling and Design
  (AI and Expert Systems, Object Oriented Programming)
  - Which Entities need to be explicitly represented?
  - Which Objects need to be represented?
- Semantic Web
- Integration and Interoperability
- Semantic Web Services
  - Related to both Semantic Web and Interoperability
- Information Retrieval
- Natural Language Processing
How to represent Ontologies?

• Need for **languages**... *formal, machine readable, (standardized?)* for the ontological representations:
  
  – **Concepts** ... Conceptual entities of a domain (e.g. Horse)
  – **Properties** ... Attributes describing concepts (e.g. hasColor)
  – **Relations** ... Relations among concepts (e.g. Is-A [Animal])
  – **Axioms** ... Constraints and commitments on the ontological representation (e.g. No horse is a human)
  – **Individuals and memberships** (e.g. Furia is a Horse)
How to represent Ontologies?

- **Reasoning is important to**
  - Ensure the quality of an ontology during ontology design
    - it can be used to test whether concepts are non-contradictory and to derive implied relations.
    - test whether the concept definitions in the ontology have the intended consequences or not.
  - When searching Web pages, data and documents annotated with such concepts.
    - E.g. Want to find all resources about Historical Novel
      - Pick up also those about Classic Historical Novel, Mediaeval Historical Novel (i.e. subconcept)
      - Pick up also those about Historical Essay (i.e. similar concept)

Artificial Intelligence and KR has a long tradition!
First Order Logic

Terms

Confucius \( \text{Length}(\alpha_{A14}) \) \( \text{Age}(\text{Confucius}) \)

Atomic formulas

\( \text{NormalTraffic}(A14_{[25,30]}) \) \( \text{OVERLAP}^{T,S}(x,y) \)

Well formed formulas

\( \forall x. \exists y. \text{WITHIN}(y,x) \) \( \forall x. F(x) \rightarrow \text{ATS}(x) \lor \text{NormalTraffic}(x) \)

GREAT EXPRESSIVITY, BUT WHAT ABOUT REASONING?

Many Calculi, but computability problems!

FOL is \textbf{undecidable}!
Semantic Nets

Representation in a Semantic Net

The physical attributes of a person can be represented as in Fig. 9.
E.g. Frame

(frame conducto-sanguineo
  (forma tubular)
  (contiene sangre))

(frame arteria
  (es-un conducto-sanguineo)
  (localizacion (brazo, cabeza, pierna, tronco))
  (sangre rica-en-oxigeno)
  (pared muscular))

(frame vena
  (es-un conducto-sanguineo)
  (pared fibrosa))

(frame aorta
  (es-un arteria)
  (localizacion tronco)
  (diametro 2.5))

(frame arteria-izquierda-X
  (es-un arteria)
  (localizacion brazo)
  (sangre pobre-en-oxigeno)
  (diametro 0.4))
% Sfumatura di località con relazione di contiguità (simmetrica, riflessiva). La vicinanza non è transitiva.

```
vicino(LUOGO, LUOGO) :- luogo(LUOGO).
vicino(LUOGO_2, LUOGO_1) :- vicino(LUOGO_1, LUOGO_2), luogo(LUOGO_1), luogo(LUOGO_2).
    -vicino(L1, L2) :- not vicino(L1, L2), luogo(L1), luogo(L2).
```

% La coincidenza spaziale è una relazione di equivalenza tra luoghi.

```
coincide(L, L) :- luogo(L).
coincide(L1, L2) :- luogo(L1), luogo(L2), coincide(L2, L1).
coincide(L1, L3) :- coincide(L1, L2), coincide(L2, L3), luogo(L1), luogo(L2), luogo(L3).
    -coincide(L1, L2) :- not coincide(L1, L2), luogo(L1), luogo(L2).
```
Ontologies

• Conceptual Modeling
• Representation + Reasoning
• Not FOL: BALANCE between Expressiveness and Computational Acceptable Behavior
• Semantic, machine-readable annotation of data, documents, resources (Semantic Web, SOC, Information retrieval, Ubiquitous Computing, ...)
• OWL = RDF Schema + Formal Semantic

• Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics.

• OWL is a family of three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

Description logics

W3C Recommendation 10 February 2004

http://www.w3.org/TR/owl-features/
Modeling: major representational focus...

- **Concepts** ... Conceptual entities of a domain (e.g. Horse)
- **Properties** ... Attributes describing concepts (e.g. hasColor)
- **Relations** ... Relations among concepts (e.g. Is-A [Animal])
- **Axioms** ... Constraints and commitments on the ontological representation (e.g. No horse is a human)
- **Individuals and memberships** (e.g. Furia is a Horse)
**OWL Family**

**OWL Lite** supports those users primarily needing a classification hierarchy and simple constraint features.

For example, while OWL Lite supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and provide a quick migration path for thesauri and other taxonomies.

**OWL DL** supports those users who want the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems.

OWL DL includes all OWL language constructs with restrictions such as type separation (a class can not also be an individual or property, a property can not also be an individual or class). OWL DL is so named due to its correspondence with description logics [Description Logics], a field of research that has studied a particular decidable fragment of first order logic. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems.

**OWL Full** is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. Another significant difference from OWL DL is that a owl:DatatypeProperty can be marked as an owl:InverseFunctionalProperty. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support every feature of OWL Full.

The choice between OWL Lite and OWL DL depends on the extent to which users require the more expressive restriction constructs provided by OWL DL. Reasoners for OWL Lite will have desirable computational properties. Reasoners for OWL DL, while dealing with a decidable sublanguage, will be subject to higher worst-case complexity. The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modelling facilities of RDF Schema (i.e. defining classes of classes). When using OWL Full as compared to OWL DL, reasoning support is less predictable.
OWL Reasoning example:

- Require: a payment service
- Payment: mastercard payment

Need a reasoner service!
(Kaon2, Pellet, Racer, ...)

- mastercard payment service
- credit card payment service
- payment service

is a (infered)
subclass of
System Architecture

Knowledge Base

Tbox (schema)

- Man $\equiv$ Human $\cap$ Male
- Happy-Father $\equiv$ Man $\cap$ $\exists$ has-child
- Female $\cap$ ...

Abox (data)

- John : Happy-Father
- $\langle$John, Mary$\rangle$ : has-child
- John: $\leq$ 1 has-child

Inference System

Interface
Reasoning Example

- The “GuineaPigSelling” Service is an e-Commerce service that sells Guinea Pig only.
What is a Guinea Pig?

- It is defined in the ontology!

Definition based on the UNSPS Code
## Classification: UNSPSC

- United Nations Standard Products and Services Code®

>15,000 codes!

<table>
<thead>
<tr>
<th>Key</th>
<th>Code</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3024</td>
<td>43223205</td>
<td>Instant messaging platform</td>
</tr>
<tr>
<td>3025</td>
<td>43223206</td>
<td>Wireless internet gateway</td>
</tr>
<tr>
<td>143026</td>
<td>43223207</td>
<td>Video streaming system</td>
</tr>
<tr>
<td>143027</td>
<td>43223208</td>
<td>Mobile or messaging game platform</td>
</tr>
<tr>
<td>143028</td>
<td>43223209</td>
<td>Location based messaging service platforms</td>
</tr>
<tr>
<td>143029</td>
<td>43223210</td>
<td>Micropayment messaging systems</td>
</tr>
<tr>
<td>136028</td>
<td>43223211</td>
<td>Paging controllers</td>
</tr>
<tr>
<td>136029</td>
<td>43223212</td>
<td>Paging terminals, Datacom and network connectivity</td>
</tr>
</tbody>
</table>

**Definition**

This platform is a Telecommunications Network Component that is capable of managing the presence ‘awareness’ of a user. It basically shows if the user is online, offline, away, in a call etc. WAP Gateway (Wireless application protocol); this platform is a Telecommunications Network Component that is capable of sending and receiving data in WAP format to Mobile devices that can access the Internet. This platform is a combination of a Telecommunications Network Component and an IT Network Component capable of sending a video in parts to a user where the user is able to start viewing the video before the complete video data is downloaded. The video data can be retrieved and stored by the subscriber of the service. A combination of Telecommunications Network - and IT Network components that is capable of providing entertainment applications to subscribers (Bundle of HW, SW, Service Provider). A special platform which enables an operator to locate subscribers by mobile internal data and offer a special content for this region (e.g., Events in the city, cinema, etc.), bundle of HW, SW, Service Provider, a special platform that enables subscribers to e-commerce transactions of very low value (Bundle of HW, SW, Service Provider). Provide voice-paging connection between a telephone system and one zone paging/background music system. It also interfaces with a PBX system or with a key system using central office line. Provide commercial paging service within a metropolitan region. The equipment, called a paging terminal, has the ability to answer calls seeking to make the page, decodes or records the message, certifies the paging customer’s validity, encodes the message to the correct format for the customer’s pager, and manages the page through the radio transmission network.

Simple Named Classes

The most basic concepts in a domain should correspond to *classes that are the roots of various taxonomic trees*. Every individual in the OWL world is a member of the class `owl:Thing`. Thus each user-defined class is implicitly a subclass of `owl:Thing`. Domain specific root classes are defined by simply declaring a named class. OWL also defines the empty class, `owl:Nothing`.

```
<owl:Class rdf:ID="ConsumableThing"/>
<owl:Class rdf:ID="Region"/>
<owl:Class rdf:ID="Winery"/>
```

And while the classes exist, they may have no members. For all we know at the moment, these classes might as well have been called `Thing1`, `Thing2`, and `Thing3`.
The fundamental taxonomic constructor for classes is `rdfs:subClassOf`.

It relates a more specific class to a more general class. If X is a subclass of Y, then every instance of X is also an instance of Y. The `rdfs:subClassOf` relation is **transitive** (if X is a subclass of Y and Y a subclass of Z then X is a subclass of Z).

```xml
<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="ConsumableThing"/>
  ...
</owl:Class>
```

We define PotableLiquid (liquids suitable for drinking) to be a subclass of ConsumableThing.

A class definition has two parts: a **name introduction** or reference and a **list of restrictions**. Each of the immediate contained expressions in the class definition further restricts the instances of the defined class. Instances of the class belong to the **intersection** of the restrictions.
A class is simply a name and collection of properties that describe a set of individuals. Individuals are the members of those sets. Classes should correspond to naturally occurring sets of things in a domain of discourse, and individuals should correspond to actual entities that can be grouped into these classes. An individual is minimally introduced by declaring it to be a member of a class.

Subclass vs. instance: It is very easy to confuse the instance-of relationship with the subclass relationship. For example, it may seem arbitrary to choose to make CabernetSauvignonGrape an individual that is an instance of Grape, as opposed to a subclass of Grape. This is not an arbitrary decision. The Grape class denotes the set of all grape varietals, and therefore any subclass of Grape should denote a subset of these varietals. Thus, CabernetSauvignonGrape should be considered an instance of Grape, and not a subclass. It does not describe a subset of Grape varietals, it is a grape varietal.
Properties let us assert general facts about the members of classes and specific facts about individuals. A property is a binary relation.

[1] object properties, relations between instances of two classes (individuals to individuals)
[2] datatype properties, relations between instances of classes (individuals to datatypes)

```xml
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="Wine"/>
  <rdfs:range rdf:resource="WineGrape"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="course">
  <rdfs:domain rdf:resource="Meal" />
  <rdfs:range rdf:resource="MealCourse" />
</owl:ObjectProperty>
```
Properties & Inference

The use of range and domain information in OWL is different from type information in a programming language. Among other things, types are used to check consistency in a programming language. In OWL, a range may be used to infer a type.

We can infer that LindemansBin65Chardonnay is a wine because the domain of madeFromGrape is Wine.
Properties Taxonomy

Property hierarchies may be created by making one or more statements that a property is a subproperty of one or more other properties.

```xml
<owl:Class rdf:ID="WineDescriptor" />

<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="WineDescriptor" />
  ...
</owl:Class>

<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="Wine" />
  <rdfs:range rdf:resource="WineDescriptor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf rdf:resource="hasWineDescriptor" />
  <rdfs:range rdf:resource="WineColor" />
  ...
</owl:ObjectProperty>
```

WineDescriptor properties relate wines to their color and components of their taste, including sweetness, body, and flavor.

hasColor is a subproperty of the hasWineDescriptor property, with its range further restricted to WineColor.
Anything with a `hasColor` property with value `x` also has a `hasWineDescriptor` property with value `x`
At least, at most Restrictions

[OWL Lite] Cardinality is useful to state that a property on a class has both minCardinality 0 and maxCardinality 0 or both minCardinality 1 and maxCardinality 1

```xml
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="PotableLiquid"/>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="madeFromGrape"/>
      <owl:minCardinality rdf:datatype="nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>

  ...
</owl:Class>
```

A wine is made from at least one WineGrape
Properties of Individuals

```
<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="CaliforniaRegion" />
</Region>

&Winery rdf:ID="SantaCruzMountainVineyard" />

<CabernetSauvignon rdf:ID="SantaCruzMountainVineyardCabernetSauvignon">
  &locatedIn rdf:resource="SantaCruzMountainsRegion"/>
  &hasMaker rdf:resource="SantaCruzMountainVineyard" />
</CabernetSauvignon>
```
Properties Characteristics

**inverseOf**
One property may be stated to be the inverse of another property. If the property $P_1$ is stated to be the inverse of the property $P_2$, then if $X$ is related to $Y$ by the $P_2$ property, then $Y$ is related to $X$ by the $P_1$ property.

**TransitiveProperty**
Properties may be stated to be transitive. If a property is transitive, then if the pair $(x,y)$ is an instance of the transitive property $P$, and the pair $(y,z)$ is an instance of $P$, then the pair $(x,z)$ is also an instance of $P$.

OWL Lite (and OWL DL) impose the side condition that transitive properties (and their superproperties) cannot have a maxCardinality 1 restriction. Without this side-condition, OWL Lite and OWL DL would become undecidable languages.

**SymmetricProperty**
Properties may be stated to be symmetric. If a property is symmetric, then if the pair $(x,y)$ is an instance of the symmetric property $P$, then the pair $(y,x)$ is also an instance of $P$.

**FunctionalProperty**
P(x,y) and P(x,z) implies $y = z$
Properties may be stated to have a unique value. If a property is a FunctionalProperty, then it has no more than one value for each individual (it may have no values for an individual). This characteristic has been referred to as having a unique property. FunctionalProperty is shorthand for stating that the property’s minimum cardinality is zero and its maximum cardinality is 1.

**InverseFunctionalProperty**
P(y,x) and P(z,x) implies $y = z$
Properties may be stated to be inverse functional. If a property is inverse functional then the inverse of the property is functional. Thus the inverse of the property has at most one value for each individual. This characteristic has also been referred to as an unambiguous property.
Properties Characteristics

SymmetricProperty

```xml
<owl:ObjectProperty rdf:ID="adjacentRegion">
    <rdf:type rdf:resource="SymmetricProperty" />
    <rdfs:domain rdf:resource="Region" />
    <rdfs:range rdf:resource="Region" />
</owl:ObjectProperty>
```

The MendocinoRegion is adjacent to the SonomaRegion and vice-versa.

The MendocinoRegion is located in the CaliforniaRegion but not vice versa.

TransitiveProperty

```xml
<owl:ObjectProperty rdf:ID="locatedIn">
    <rdf:type rdf:resource="TransitiveProperty" />
    <rdfs:domain rdf:resource="owl:Thing" />
    <rdfs:range rdf:resource="Region" />
</owl:ObjectProperty>
```

Because the SantaCruzMountainsRegion is locatedIn the CaliforniaRegion, then it must also be locatedIn the USRegion, since locatedIn is transitive.
Properties Restrictions

**allValuesFrom**
The restriction `allValuesFrom` is stated on a property with respect to a class. It means that this property on this particular class has a local range restriction associated with it. Thus if an instance of the class is related by the property to a second individual, then the second individual can be inferred to be an instance of the local range restriction class.

```xml
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="PotableLiquid" />
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasMaker" />
      <owl:allValuesFrom rdf:resource="Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

For all wines, if they have makers, all the makers are wineries.

**someValuesFrom**
The restriction `someValuesFrom` is stated on a property with respect to a class. A particular class may have a restriction on a property that at least one value for that property is of a certain type.

```xml
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="PotableLiquid" />
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasMaker" />
      <owl:someValuesFrom rdf:resource="Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

For all wines, they have at least one maker that is a winery.
Equality & Inequality

equivalentClass
Two classes may be stated to be equivalent. Equivalent classes have the same instances. Equality can be used to create synonymous classes.

For example, Car can be stated to be equivalentClass to Automobile. From this a reasoner can deduce that any individual that is an instance of Car is also an instance of Automobile and vice versa.

equivalentProperty
Two properties may be stated to be equivalent. Equivalent properties relate one individual to the same set of other individuals. Equality may be used to create synonymous properties.

For example, hasLeader may be stated to be the equivalentProperty to hasHead. From this a reasoner can deduce that if X is related to Y by the property hasLeader, X is also related to Y by the property hasHead and vice versa. A reasoner can also deduce that hasLeader is a subproperty of hasHead and hasHead is a subProperty of hasLeader.

sameAs
Two individuals may be stated to be the same. These constructs may be used to create a number of different names that refer to the same individual.

For example, the individual Deborah may be stated to be the same individual as DeborahMcGuinness.

differentFrom
An individual may be stated to be different from other individuals.

For example, the individual Frank may be stated to be different from the individuals Deborah and Jim. Thus, if the individuals Frank and Deborah are both values for a property that is stated to be functional (thus the property has at most one value), then there is a contradiction. Explicitly stating that individuals are different can be important in when using languages such as OWL (and RDF) that do not assume that individuals have one and only one name. For example, with no additional information, a reasoner will not deduce that Frank and Deborah refer to distinct individuals.

AllDifferent
A number of individuals may be stated to be mutually distinct in one AllDifferent statement.

For example, Frank, Deborah, and Jim could be stated to be mutually distinct using the AllDifferent construct. Unlike the differentFrom statement above, this would also enforce that Jim and Deborah are distinct (not just that Frank is distinct from Deborah and Frank is distinct from Jim). The AllDifferent construct is particularly useful when there are sets of distinct objects and when modelers are interested in enforcing the unique names assumption within those sets of objects. It is used in conjunction with distinctMembers to state that all members of a list are distinct and pairwise disjoint.
Useful links

Resource Description Framework: http://www.w3.org/RDF/
w3c RDF Data Access Working groups: http://www.w3.org/2001/sw/DataAccess/
RDF Vocabulary Description Language (RDF Schema): http://www.w3.org/TR/rdf-schema/
OWL: http://www.w3.org/TR/OWL-features/