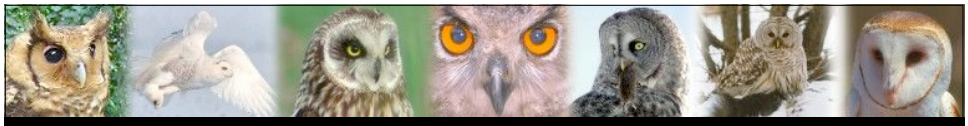


Introduction to OWL

Marco Ronchetti
Università di Trento
Italy

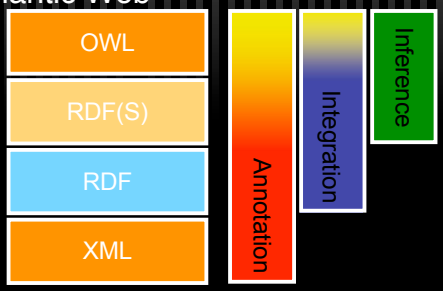
Languages

Work on Semantic Web has concentrated on the definition of a collection or "stack" of languages.

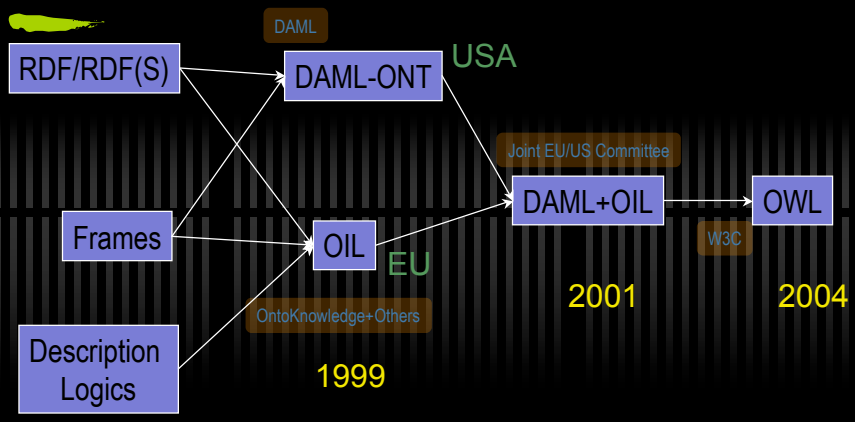
- ✓ These languages are then used to support the representation and use of metadata.

Basic machinery to represent the extra semantic information needed for the Semantic Web

- ✓ XML
- ✓ RDF
- ✓ RDF(S)
- ✓ OWL
- ✓ …



OWL: Web Ontology Language



A Printer Ontology – HP Products

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

Key Words

Ontology
Class
Restriction
Property

Ontology

A heavily overloaded term
with several different meanings
in different disciplines:

- ✓ Philosophy
- ✓ Linguistics
- ✓ Computer Science)

Ontology - Philosophy

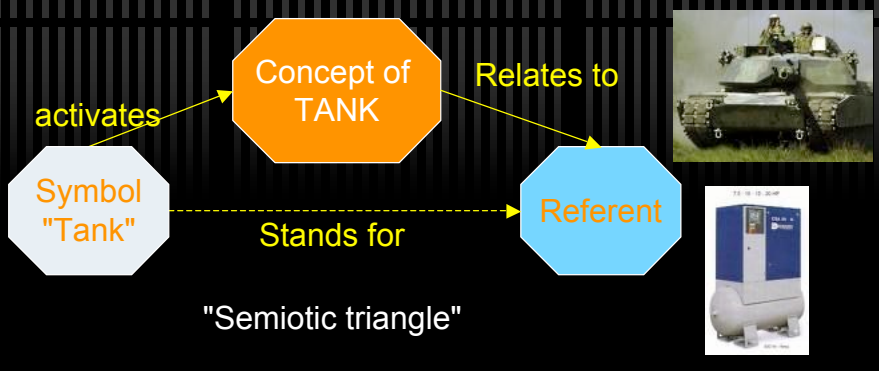
Ontology deals with the nature and organisation of reality (Aristotele)

Tries to answer the questions:

- What characterizes being?
- What is being?

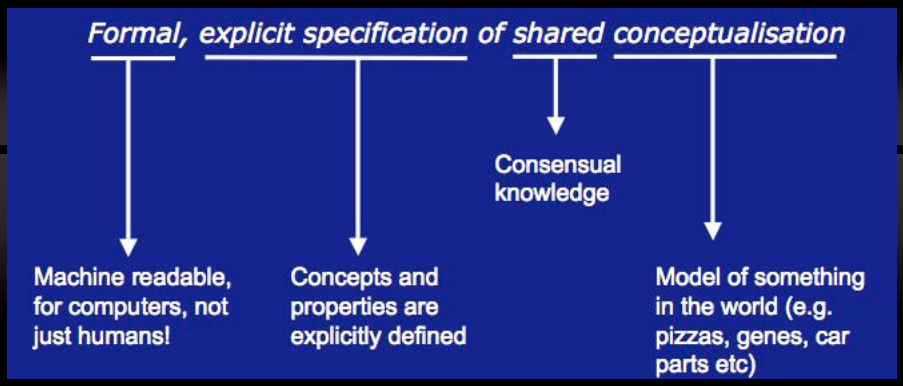
Ontology - Linguistics

a concept, is the mediator that relates the symbol to its object



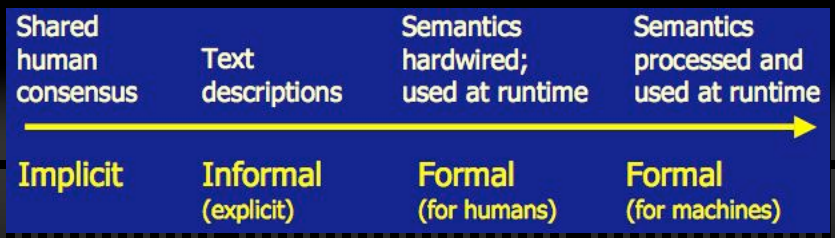
"Semiotic triangle"

Ontology - Computer Science



Rudi Studer(98)

A Semantic continuum



Further to the right:

- Less ambiguity
- Better inter-operation
- More robust
- More difficult

Structure of an Ontology

Ontologies typically have two distinct components:

Names for important concepts in the domain

- ✓ Elephant is a concept whose members are a kind of animal
- ✓ Herbivore is a concept whose members are exactly those animals who eat only plants or parts of plants
- ✓ Carnivore is a concept whose members are exactly those animals who eat other animals
- ✓ Adult_Elephant is a concept whose members are exactly those elephants whose age is greater than 20 years

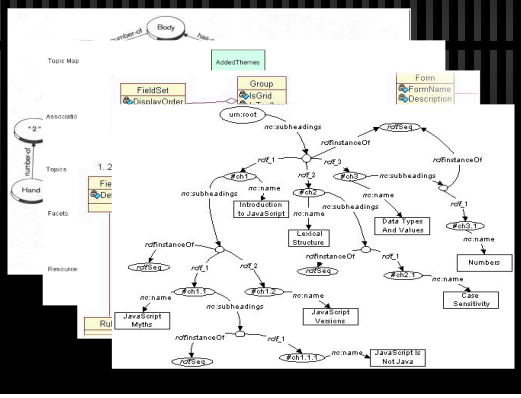
Background knowledge/constraints on the domain

- ✓ Adult_Elephants weight at least 2,000 kg
- ✓ All Elephants are either African_Elephants or Indian_Elephants
- ✓ No individual can be both a Herbivore and a Carnivore

Ontology Languages

There are a wide variety of languages for "Explicit Specification"

- ✓ Graphical Notations
  - ✓ Semantic Networks
  - ✓ Topic Maps
  - ✓ UML
  - ✓ RDF



Ontology Languages

There are a wide variety of languages for "Explicit Specification"

- ✓ Graphical Notations
  - ✓ Semantic Networks
  - ✓ Topic Maps
  - ✓ UML
  - ✓ RDF
- ✓ Logic Based
  - ✓ Description Logics
  - ✓ Rules
  - ✓ First Order Logic
  - ✓ Conceptual Graphs

Every gardener likes the sun.
(Ax) gardener(x) => likes(x,Sun)
You can fool some of the people all of the time.
(Ex)(At) (person(x) ^ time(t)) => can-fool(x,t)
You can fool all of the people some of the time.
(Ax)(Et) (person(x) ^ time(t) => can-fool(x,t)
All purple mushrooms are poisonous.
(Ax) (mushroom(x) ^ purple(x)) => poisonous(x)
No purple mushroom is poisonous.
~(Ex) purple(x) ^ mushroom(x) ^ poisonous(x)
(Ax) (mushroom(x) ^ purple(x)) => ~poisonous(x)
There are exactly two purple mushrooms.
(Ex)(Ey) mushroom(x) ^ purple(x) ^ mushroom(y) ^ purple(y) ^ ~(x=y) ^(Az)
(mushroom(z) ^ purple(z)) => ((x=z) v (y=z))
Clinton is not tall.
~tall(Clinton)

Requirements for Ontology Languages

Ontology languages allow users to write explicit, formal conceptualizations of domain models

The main requirements are:

- ✓ a well-defined syntax
- ✓ efficient reasoning support
- ✓ a formal semantics
- ✓ sufficient expressive power
- ✓ convenience of expression

Tradeoff between Expressive Power and Efficient Reasoning Support

The richer the language is, the more inefficient the reasoning support becomes

Sometimes it crosses the border of *noncomputability*

We need a compromise:

- ✓ A language supported by reasonably efficient reasoners
- ✓ A language that can express large classes of ontologies and knowledge.

Reasoning About Knowledge in Ontology Languages

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 is equivalent to class C, then A is equivalent to C, too

Reasoning About Knowledge in Ontology Languages (2)

Consistency
- ✓ X instance of classes A and B, but A and B are disjoint
- ✓ This is an indication of an error in the ontology

Classification
- ✓ Certain property-value pairs are a sufficient condition for membership in a class A; if an individual x satisfies such conditions, we can conclude that x must be an instance of A

Uses for Reasoning

Reasoning support is important for
- ✓ checking the consistency of the ontology and the knowledge
- ✓ checking for unintended relationships between classes
- ✓ automatically classifying instances in classes

Checks like the preceding ones are valuable for
- ✓ designing large ontologies, where multiple authors are involved
- ✓ integrating and sharing ontologies from various sources

Reasoning Support for OWL

Semantics is a prerequisite for reasoning support

Formal semantics and reasoning support are usually provided by

- ✓ mapping an ontology language to a known logical formalism
- ✓ using automated reasoners that already exist for those formalisms

OWL is (partially) mapped on a description logic, and makes use of reasoners such as Pellet and RACER

Description logics are a subset of predicate logic for which efficient reasoning support is possible

Aside: Description Logics

A family of logic based Knowledge Representation formalisms

- ✓ Descendants of semantic networks and KL-ONE
- ✓ Describe domain in terms of concepts (classes), roles (relationships) and individuals

Distinguished by:

- ✓ Formal semantics (typically model theoretic)
  - ✓ Decidable fragments of FOL
- ✓ Provision of inference services
  - ✓ Sound and complete decision procedures for key problems
  - ✓ Implemented systems (highly optimised)

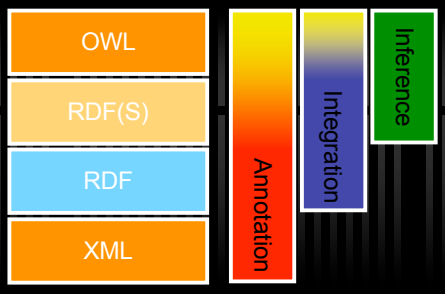
DL Semantics

Model theoretic semantics. An interpretation consists of

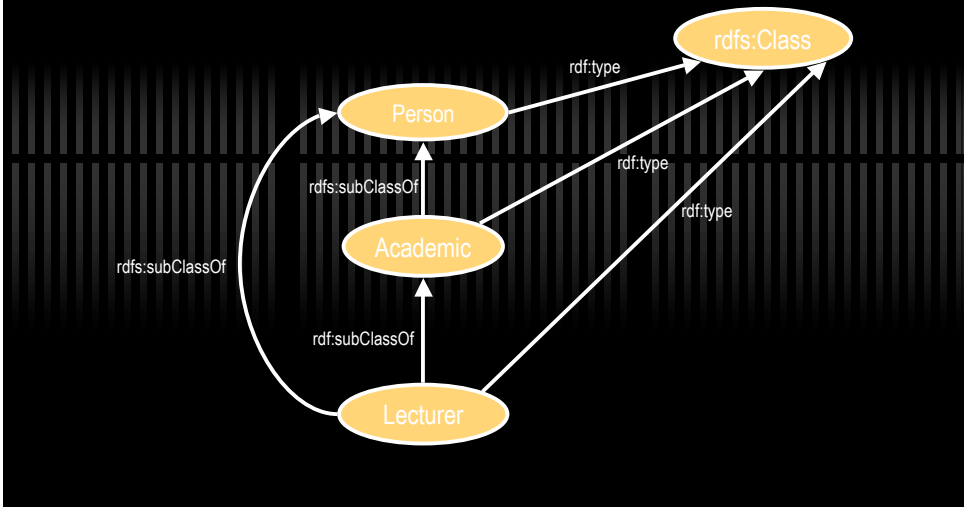
- A domain of discourse (a collection of objects)
- Functions mapping
  - classes to sets of objects
  - properties to sets of pairs of objects
- Rules describe how to interpret the constructors and tell us when an interpretation is a model.

In a DL, a class description is thus a characterisation of the individuals that are members of that class.

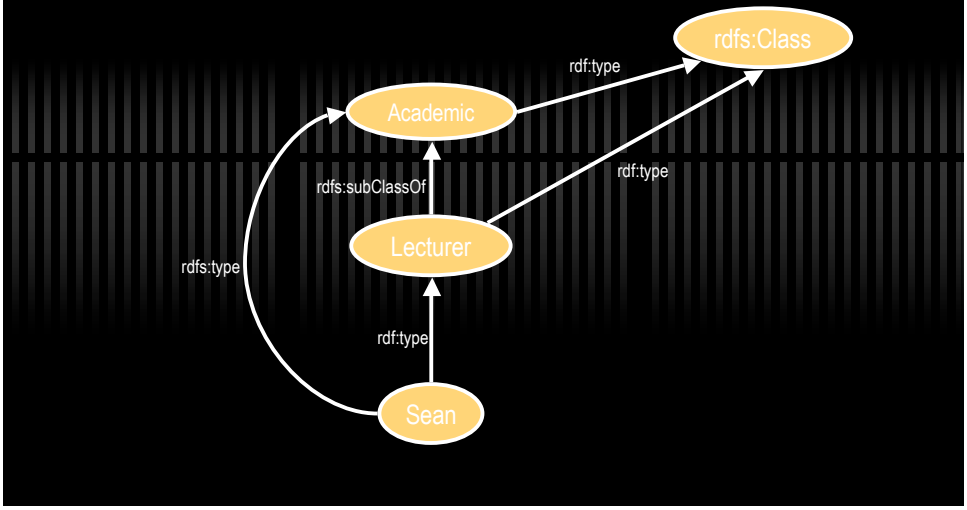
Was'nt RDF(S) enough?



RDF(S) Inference



RDF(S) Inference



Limitations of the Expressive Power of RDF Schema

Local scope of properties

- **rdfs:range** defines the range of a property (e.g. eats) for all classes
- In RDF Schema we cannot declare range restrictions that apply to some classes only
- E.g. we cannot say that cows eat only plants, while other animals may eat meat, too

Limitations of the Expressive Power of RDF Schema (2)

Disjointness of classes

- Sometimes we wish to say that classes are disjoint (e.g. **male** and **female**)

Boolean combinations of classes

- Sometimes we wish to build new classes by combining other classes using union, intersection, and complement
- E.g. **person** is the disjoint union of the classes **male** and **female**

Limitations of the Expressive Power of RDF Schema (3)

Cardinality restrictions

- ✓ E.g. a person has exactly two parents, a course is taught by at least one lecturer

Special characteristics of properties

- ✓ Transitive property (like “greater than”)
- ✓ Unique property (like “is mother of”)
- ✓ A property is the inverse of another property (like “eats” and “is eaten by”)

Combining OWL with RDF Schema

Ideally, OWL would extend RDF Schema

- ✓ Consistent with the layered architecture of the Semantic Web

But simply extending RDF Schema would work against obtaining expressive power and efficient reasoning

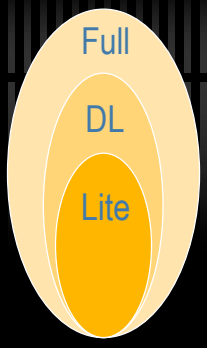
- ✓ Combining RDF Schema with logic leads to uncontrollable computational properties

Three Species of OWL

W3C'sWeb Ontology Working Group defined OWL as three different sublanguages:

- ✓ OWL Full
- ✓ OWL DL
- ✓ OWL Lite

Each sublanguage geared toward fulfilling different aspects of requirements



OWL Full – (FOP)

It uses all the OWL languages primitives

It allows the combination of these primitives in arbitrary ways with RDF and RDF Schema

OWL Full is fully upward-compatible with RDF, both syntactically and semantically

OWL Full is so powerful that it is undecidable

- ✓ No complete (or efficient) reasoning support

OWL DL

OWL DL (Description Logic) is a sublanguage of OWL Full that restricts application of the constructs from OWL and RDF

- ✓ Application of OWL's constructs' to each other is disallowed
- ✓ Therefore it corresponds to a well studied description logic

OWL DL permits efficient reasoning support

But we lose full compatibility with RDF:

- ✓ Not every RDF document is a legal OWL DL document.
- ✓ Every legal OWL DL document is a legal RDF document.

OWL Lite

An even further restriction limits OWL DL to a subset of the language constructs

- ✓ E.g., OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality.

The advantage of this is a language that is easier to

- ✓ grasp, for users
- ✓ implement, for tool builders

The disadvantage is restricted expressivity

practically not used

owl:Thing owl:Nothing

The class owl:Thing is the class that represents the set containing all individuals. All classes are subclasses of owl:Thing.

owl:Thing is part of the OWL Vocabulary, which is defined by the ontology located at http://www.w3.org/2002/07/ owl/¥#

owl:Nothing is the empty class

Disjoint classes

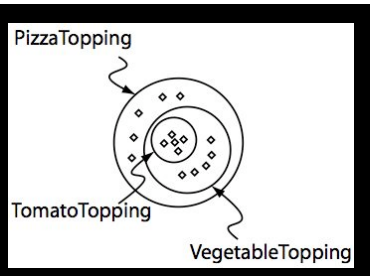
OWL Classes are assumed to "overlap".

We cannot assume that an individual is not a member of a particular class simply because it has not been asserted to be a member of that class.

In order to "separate" a group of classes we must make them disjoint from one another.

This ensures that an individual which has been asserted to be a member of one of the classes in the group cannot be a member of any other classes in that group

Subclasses



If TomatoTopping
- is a subclass of VegetableTopping which
- is a subclass of PizzaTopping

then all individuals that
- are members of the class TomatoTopping
- are also members of the class VegetableTopping and members of the class PizzaTopping

Multiple inheritance

If Car is subclass of ExpensiveItems
and Car is subclass of RoadVehicles

this does not imply a relation between ExpensiveItems and RoadVehicles

Closed World Assumption

OWL currently adopts the open-world assumption:

- ✓ A statement cannot be assumed true on the basis of a failure to prove it
- ✓ On the huge and only partially knowable WWW, this is a correct assumption

Closed-world assumption: a statement is true when its negation cannot be proved

- ✓ tied to the notion of defaults, leads to nonmonotonic behaviour

Unique Names Assumption

Typical database applications assume that individuals with different names are indeed different individuals

OWL follows the usual logical paradigm where this is not the case

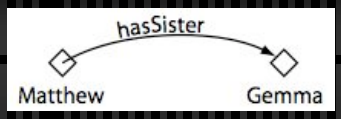
- ✓ Plausible on the WWW

One may want to indicate portions of the ontology for which the assumption does or does not hold

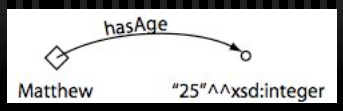
Properties

OWL Properties represent relationships between two individuals.

Object properties link an individual to an individual.



Datatype properties link an individual to an XML Schema Datatype value or an rdf literal.



OWL has also Annotation properties

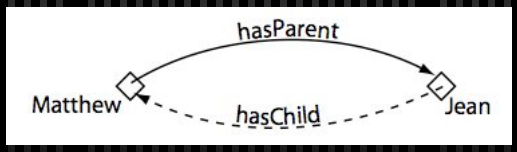
Properties

There is no strict naming convention for properties, but it is suggested that property names start with a lower case letter, have no spaces and have the remaining words capitalised.

It is also reccomended that properties are prefixed with the word has or is

e.g. hasPart, isPartOf,
hasManufacturer, isProducerOf.

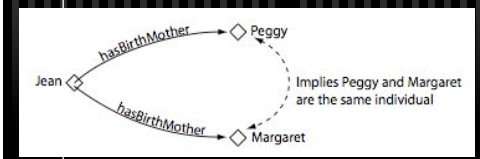
Inverse Properties

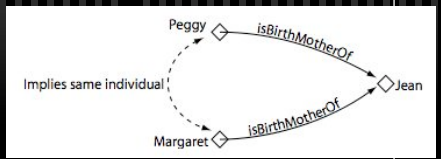


Functional - Inverse Functional

If a property is functional (single valued property, feature), for a given individual, there can be at most one individual that is related to the individual via the property.

(A p B), (A p C) ⇒ B≡C

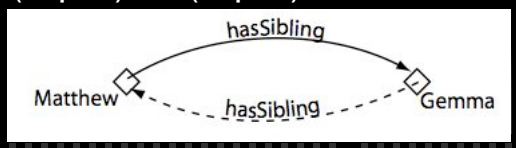




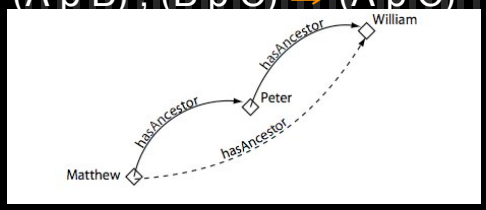
If a property is inverse functional then it means that the inverse property is functional.

Symmetric - Transitive

Symmetric (A p B) ⇒ (B p A)



Transitive (A p B) , (B p C) ⇒ (A p C)



if a property is transitive then it cannot be functional.

Properties: Domain and Range

Properties may have a domain and a range specified. Properties link individuals from the domain to individuals from the range.

It is important to realise that in OWL domains and ranges should NOT be viewed as constraints to be checked. They are used as axioms in reasoning.

Properties: Domain and Range

For example if the property hasTopping has the domain set as Pizza and we then applied the hasTopping property to IceCream, this would generally not result in an error.

It would be used to infer that the class IceCream must be a subclass of Pizza!

An error will only be generated (by a reasoner) if Pizza is disjoint to IceCream

.

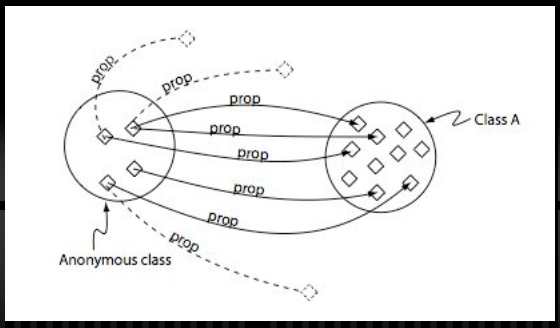
Property restrictions

- ✓ Quantifier Restrictions (∃, ∀)
- ✓ Cardinality Restrictions (e.g. >3)
- ✓ hasValue Restrictions (e.g. oneOf …)

Existential Restriction

∃ Prop ClassA

(someValueFrom)



Every individual must have at least one prop releshionship with a member of class A

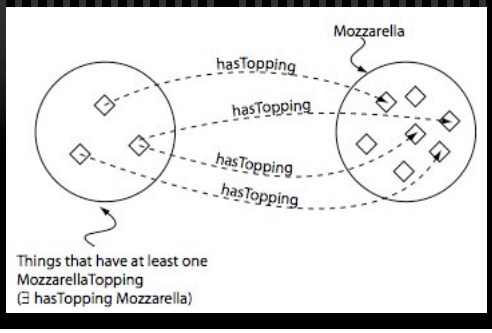
(Necessary condition)

(Open World Assumption)

Existential restriction

∃ hasTopping Mozzarella

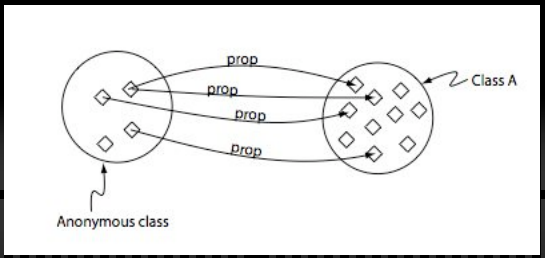
describes the (anonymous) class of individuals that have at least one topping that is Mozzarella



Universal Restriction

∀ Prop ClassA

(allVeluesFrom)



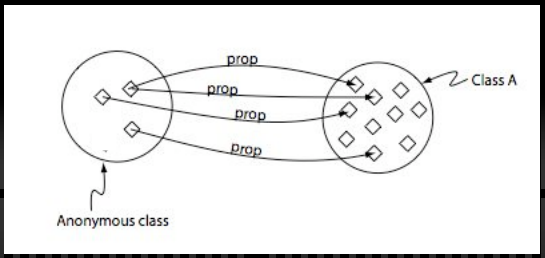
If relation for prop exists, it must be with an element of ClassA

(but there might be elements that do not have prop!)

Combining Universal & Existential

∀ Prop ClassA

∃ Prop ClassA



Every individual has prop with at least one element in class A, and no individual has prop with elements not belonging to class A

Necessary and sufficient condition

Definition

Closure Axiom

∀ Prop (ClassA ∪ ClassB)

∃ Prop ClassA

∃ Prop ClassB

Every individual has prop with at least one element in either class A or Class B, and with no other classes.

Tools

Editors

- ✓ Protégé OWL, SWOOP, ICOM, TopQuadrant Composer, OntoTrack, NeOn…
- ✓ Offer the possibility of using reasoners.

Reasoners

- ✓ DL style reasoners based on tableaux algorithms
  - ✓ Racer, FaCT++, Pellet
- ✓ Based on rules or F-logic
  - ✓ F-OWL, E-Wallet…..

APIs and Frameworks

- ✓ Jena, WonderWeb OWL-API, Protégé OWL API, OWLIM

Protégé

- ✓ Is a knowledge modelling environment
- ✓ Is free, open source software
- ✓ Is developed by Stanford Medical Informatics
- ✓ Has a large user community
- ✓ Supports development of plugins to allow backend / interface extensions (e.g. reasoners)
- ✓ supports OWL

Conclusions: now you should know…

- Why are we using ontologies and reasoners?
- What is a class/property/individual ?
- What is the open world assumption? What are its consequences?
- What is the (not) unique name assumption? What are the consequences?
- What is a universal restriction? And an existential restriction?
- What other property restrictions are there?
- What is a functional property?
- Why do we have 3 different OWLs?

Credits

these slides are a compliation form the following sources (thanks!):

A Practical Introduction to Ontologies & OWL
by Duncan Hall & Nick Drummonds

A Semantic Web Primer
by Grigoris Antoniou & Frank van Harmelen

Ontology Languages for the Semantic Web
by Sean Bechhofer

A Practical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools
by Matthew Horridge et al.

Appendix 1 : OWL Syntax

(from Grigoris Antoniou
Frank van Harmelen)

OWL Syntactic Varieties

OWL builds on RDF and uses RDF's XML-based syntax

Other syntactic forms for OWL have also been defined:

- ✓ An alternative, more readable XML-based syntax
- ✓ An abstract syntax, that is much more compact and readable than the XML languages
- ✓ A graphic syntax based on the conventions of UML

OWL XML/RDF Syntax: Header

```
<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 using **owl:Ontology** element

owl:Ontology

```
<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>
```

owl:imports is a transitive property

Classes

Classes are defined using **owl:Class**

- ✓ **owl:Class** is a subclass of **rdfs:Class**

Disjointness is defined using **owl:disjointWith**

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

Classes (2)

owl:equivalentClass defines equivalence of classes

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

owl:Thing is the most general class, which contains everything

owl:Nothing is the empty class

Properties

In OWL there are two kinds of properties

- ✓ **Object properties, which relate objects to other objects**
  - ✓ **E.g. is-TaughtBy, supervises**
- ✓ **Data type properties, which relate objects to datatype values**
  - ✓ **E.g. phone, title, age, etc.**

Datatype Properties

OWL makes use of XML Schema data types, using the layered architecture of the SW

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

Object Properties

User-defined data types

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

Inverse Properties

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

Equivalent Properties

owl:equivalentProperty

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

Property Restrictions

In OWL we can declare that the class C satisfies certain conditions
- ✓ All instances of C satisfy the conditions

This is equivalent to saying that C is subclass of a class C', where C' collects all objects that satisfy the conditions
- ✓ C' can remain anonymous

Property Restrictions (2)

A (restriction) class is achieved through an **owl:Restriction** element

This element contains an **owl:onProperty** element and one or more restriction declarations

One type defines cardinality restrictions (at least one, at most 3,…)

Property Restrictions (3)

The other type defines restrictions on the kinds of values the property may take

- **owl:allValuesFrom** specifies universal quantification
- **owl:hasValue** specifies a specific value
- **owl:someValuesFrom** specifies existential quantification

owl:allValuesFrom

```
<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:hasValue

```
<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:someValuesFrom

```
<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>
```

Cardinality Restrictions

We can specify minimum and maximum number using **owl:minCardinality** and **owl:maxCardinality**

It is possible to specify a precise number by using the same minimum and maximum number

For convenience, OWL offers also **owl:cardinality**

Cardinality Restrictions (2)

```
<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>
```

Special Properties

owl:TransitiveProperty (transitive property)
- E.g. “has better grade than”, “is ancestor of”

owl:SymmetricProperty (symmetry)
- E.g. “has same grade as”, “is sibling of”

owl:FunctionalProperty defines a property that has at most one value for each object
- E.g. “age”, “height”, “directSupervisor”

owl:InverseFunctionalProperty defines a property for which two different objects cannot have the same value

Special Properties (2)

```
<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

We can combine classes using Boolean operations (union, intersection, complement)

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

Boolean Combinations (2)

```
<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 new class is not a subclass of the union, but rather equal to the union

- ✓ We have stated an equivalence of classes

Boolean Combinations (3)

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

Nesting of Boolean Operators

```
<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 with owl:oneOf

```
<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>
```

Declaring Instances

Instances of classes are declared as in RDF:

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

No Unique-Names Assumption

OWL does not adopt the unique-names assumption of database systems

- ✓ If two instances have a different name or ID does not imply that they are different individuals

Suppose we state that each course is taught by at most one staff member, and that a given course is taught by two staff members

- ✓ An OWL reasoner does not flag an error
- ✓ Instead it infers that the two resources are equal

Distinct Objects

To ensure that different individuals are indeed recognized as such, we must explicitly assert their inequality:

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

Distinct Objects (2)

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>
```

Data Types in OWL

XML Schema provides a mechanism to construct user-defined data types

- ✓ E.g., the data type of **adultAge** includes all integers greater than 18

Such derived data types cannot be used in OWL

- ✓ The OWL reference document lists all the XML Schema data types that can be used
- ✓ These include the most frequently used types such as **string**, **integer**, **Boolean**, **time**, and **date**.

Versioning Information

owl:priorVersion indicates earlier versions of the current ontology

- ✓ No formal meaning, can be exploited for ontology management

owl:versionInfo generally contains a string giving information about the current version, e.g. keywords

Versioning Information (2)

owl:backwardCompatibleWith contains a reference to another ontology

- ✓ All identifiers from the previous version have the same intended interpretations in the new version
- ✓ Thus documents can be safely changed to commit to the new version

owl:incompatibleWith indicates that the containing ontology is a later version of the referenced ontology but is not backward compatible with it

Combination of Features

In different OWL languages there are different sets of restrictions regarding the application of features

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

Restriction of Features in OWL DL

Vocabulary partitioning

- ✓ Any resource is allowed to be only a class, a data type, a data type property, an object property, an individual, a data value, or part of the built-in vocabulary, and not more than one of these

Explicit typing

- ✓ The partitioning of all resources must be stated explicitly (e.g. a class must be declared if used in conjunction with **rdfs:subClassOf**)

Restriction of Features in OWL DL (2)

Property Separation

- ✓ The set of object properties and data type properties are disjoint
- ✓ Therefore the following can never be specified for data type properties:
  - **owl:inverseOf**
  - **owl:FunctionalProperty**
  - **owl:InverseFunctionalProperty**
  - **owl:SymmetricProperty**

Restriction of Features in OWL DL (3)

No transitive cardinality restrictions

- ✓ No cardinality restrictions may be placed on transitive properties

Restricted anonymous classes: Anonymous classes are only allowed to occur as:

- ✓ the domain and range of either **owl:equivalentClass** or **owl:disjointWith**
- ✓ the range (but not the domain) of **rdfs:subClassOf**

Restriction of Features in OWL Lite

Restrictions of OWL DL and more

owl:oneOf, **owl:disjointWith**, **owl:unionOf**, **owl:complementOf** and **owl:hasValue** are not allowed

Cardinality statements (minimal, maximal, and exact cardinality) can only be made on the values 0 or 1

owl:equivalentClass statements can no longer be made between anonymous classes but only between class identifiers

Inheritance in Class Hierarchies

Range restriction: **Courses must be taught by academic staff members only**

Michael Maher is a professor

He inherits the ability to teach from the class of academic staff members

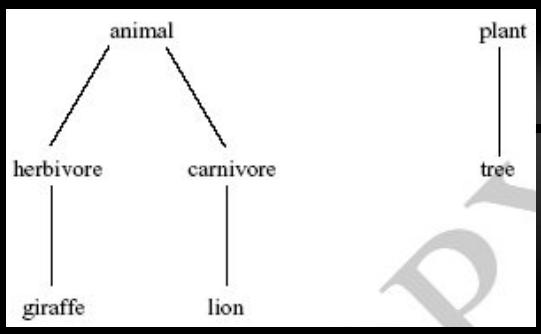
This is done in RDF Schema by fixing the semantics of "is a subclass of"

- ✓ It is not up to an application (RDF processing software) to interpret "is a subclass of

Appendix 2 : OWL Examples

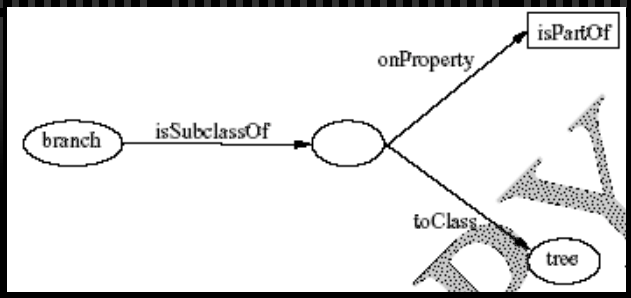
(from Grigoris Antoniou
Frank van Harmelen)

An African Wildlife Ontology – Class Hierarchy



An African Wildlife Ontology – Schematic Representation

Branches are parts of trees



An African Wildlife Ontology – Properties

```
<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>
```

An African Wildlife Ontology – Plants and Trees

```
<owl:Class rdf:ID="plant">
    <rdfs:comment>Plants are disjoint from animals.
    </rdfs:comment>
    <owl:disjointWith="#animal"/>
</owl:Class>

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

An African Wildlife Ontology – Branches

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

An African Wildlife Ontology – Leaves

```
<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>
```

An African Wildlife Ontology – Carnivores

```
<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>
```

An African Wildlife Ontology – Herbivores

```
<owl:Class rdf:ID="herbivore">
        <rdfs:comment>
                Herbivores are exactly those animals
                that eat only plants or parts of plants.
        </rdfs:comment>
        <rdfs:comment>
                Try it out! See book for code.
        <rdfs:comment>
</owl:Class>
```

An African Wildlife Ontology – Giraffes

```
<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>
```

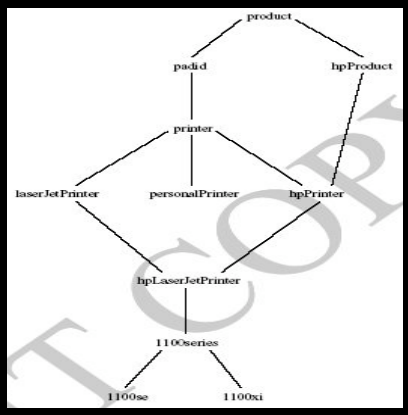
An African Wildlife Ontology – Lions

```
<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>
```

An African Wildlife Ontology – Tasty Plants

```
owl:Class rdf:ID="tasty-plant">
    <rdfs:comment>Plants eaten both by herbivores
    and carnivores </rdfs:comment>
    <rdfs:comment>
        Try it out! See book for code.
    <rdfs:comment>
</owl:Class>
```

A Printer Ontology – Class Hierarchy



A Printer Ontology – Products and Devices

```
<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>
```

A Printer Ontology – HP Products

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

A Printer Ontology – Printers and Personal Printers

```
<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>
```

A Printer Ontology – HP LaserJet 1100se Printers

```
<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>
      </rdfs:subClassOf>
</owl:Class>
```

A Printer Ontology – Properties

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

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

Summary

OWL is the proposed standard for Web ontologies

OWL builds upon RDF and RDF Schema:

- ✓ (XML-based) RDF syntax is used
- ✓ Instances are defined using RDF descriptions
- ✓ Most RDFS modeling primitives are used

Summary (2)

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