Knowledge Representation and Reasoning

University "Politehnica" of Bucharest
Department of Computer Science

Fall 2012

Adina Magda Florea

Lecture 9

KR for the Semantic Web

Lecture outline

- The Semantic Web
- RDF
- OWL
- Correspondences
- OWL Example
- Conditions
- OWL Dialects

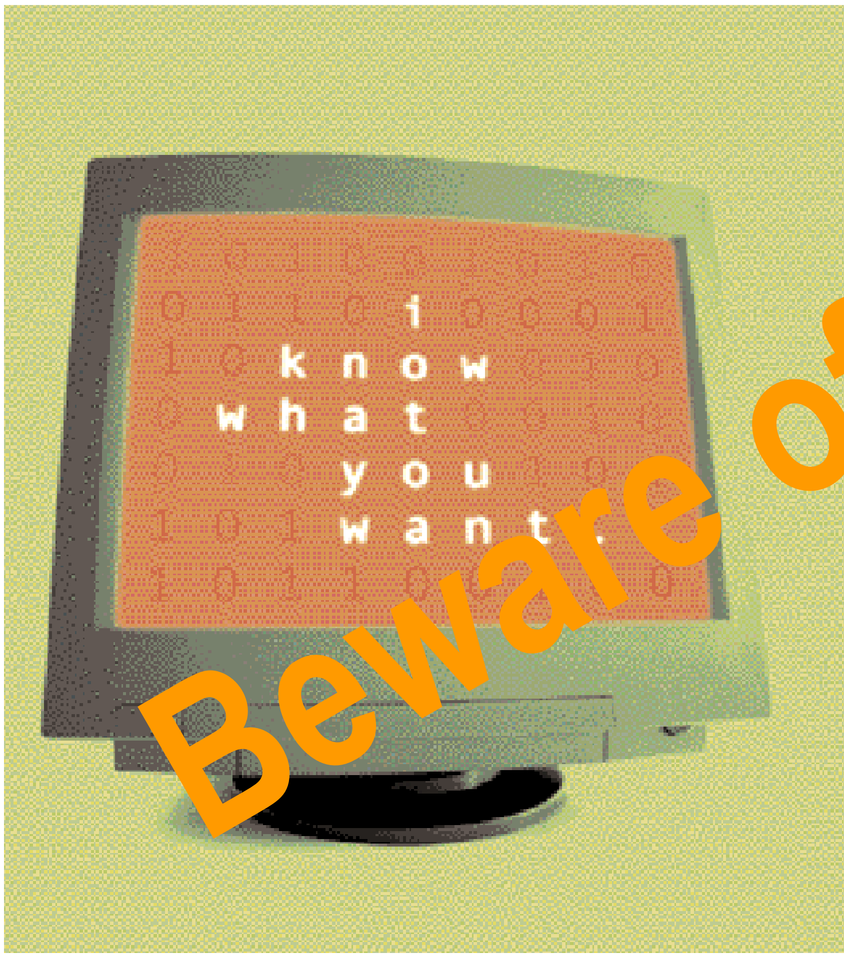
1. The Semantic Web

- Web was “invented” by Tim Berners-Lee (amongst others), a physicist working at CERN

“... a goal of the Web was that, if the interaction between person and hypertext could be so intuitive that the machine-readable information space gave an accurate representation of the state of people's thoughts, interactions, and work patterns, then machine analysis could become a very powerful management tool, seeing patterns in our work and facilitating our working together through the typical problems which beset the management of large organizations.”

- TBL’s original vision of the Web was much more ambitious than the reality of the existing (syntactic) Web
- TBL (and others) have since been working towards realising this vision, which has become known as the Semantic Web
  - E.g., article in May 2001 issue of Scientific American…

Scientific American, May 2001



Beware of the Hype

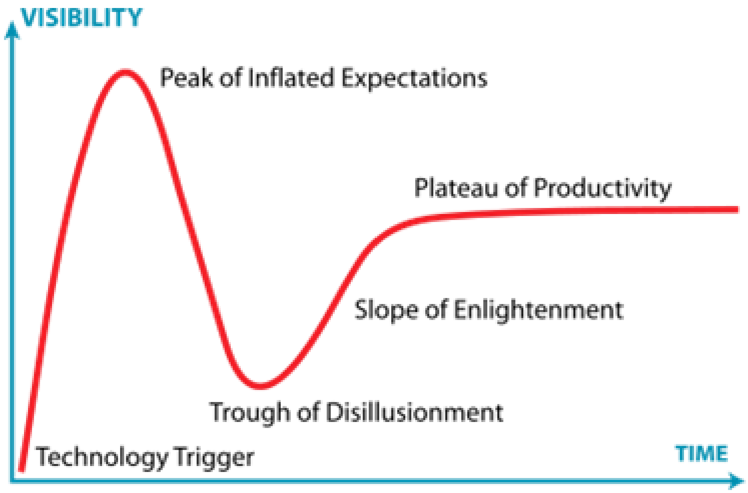
THE SEMANTIC WEB

A new form of Web content that is meaningful to computers will unleash a revolution of new abilities

by
TIM BERNERS-LEE,
JAMES HENDLER and
ORA LASSILA

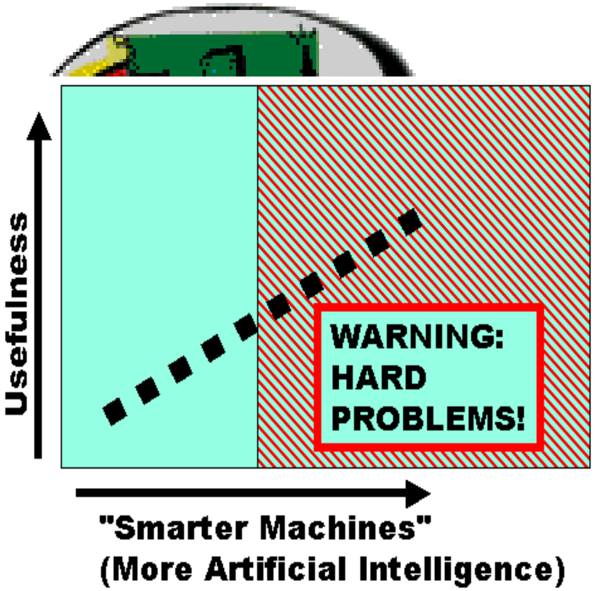
Beware of the Hype

- A **hype cycle** is a graphic representation of the maturity, adoption and business application of a specific technology.
- Since 1995, Gartner has used hype cycles to characterize the over-enthusiasm or "hype" and subsequent disappointment that typically happens with the introduction of new technologies



Beware of the Hype

- Hype seems to suggest that Semantic Web means: **"semantics + web = AI"**
  - "A new form of Web content that is meaningful to computers will unleash a revolution of new abilities"
- More realistic to think of it as meaning: **"semantics + web + AI = more useful web"**
  - Realising the complete "vision" is too hard for now (probably)
  - But we can make a start by adding **semantic annotation** to web resources



Images from Christine Thompson and David Booth

Today: the Syntactic Web

- A hypermedia, a digital library
  - A library of documents called (web pages) interconnected by a hypermedia of links
- A database, an application platform
  - A common portal to applications accessible through web pages, and presenting their results as web pages
- A platform for multimedia
  - e.g., BBC Radio anywhere in the world
- A naming scheme
  - Unique identity for those documents

A place where computers do the presentation (easy) and people do the linking and interpreting (hard).

Impossible (?) using the Syntactic Web

- Complex queries involving background knowledge
  - Find information about “animals that use sonar but are not either bats or dolphins”
- Locating information in data repositories
  - Travel enquiries
  - Prices of goods and services
  - Results of human genome experiments
- Finding and using web services
  - Visualise surface interactions between two proteins
- Delegating complex tasks to web agents
  - Book me a holiday next weekend somewhere warm, not too far away, and where they speak French or English

What is the Problem?

- **Make web resources more accessible to automated processes**
- Extend existing rendering markup with semantic markup
  - Metadata annotations that describe content/function of web accessible resources
- Use Ontologies to provide vocabulary for annotations
  - Formal specification is accessible to machines

Ontology in Philosophy

- **Ontology = a philosophical discipline** - a branch of philosophy that deals with the nature and the organisation of reality

- *Science of Being* (Aristotle, Metaphysics, IV, 1)

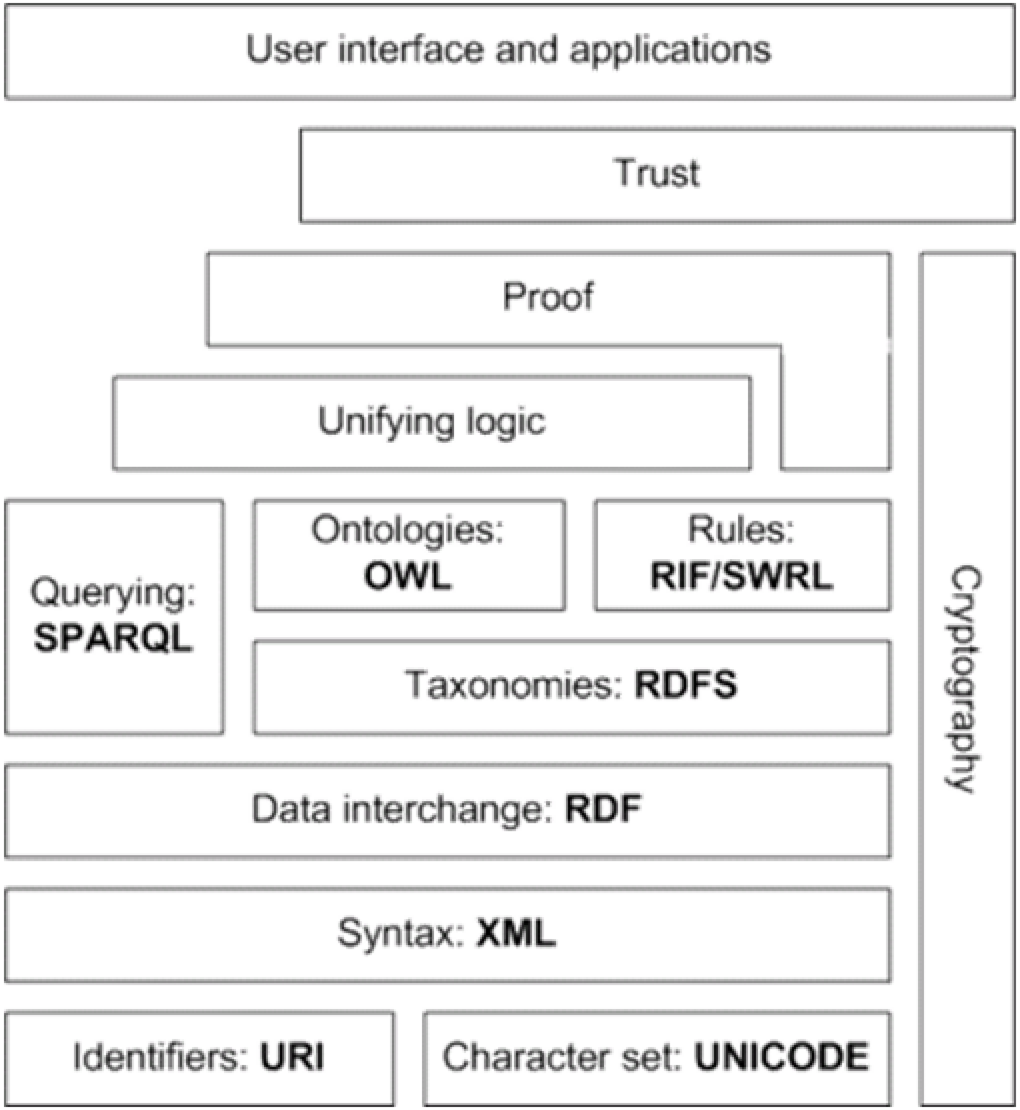
  "the science of being *qua* being"

- Tries to answer the questions:
  - *What characterizes being?*
  - *Eventually, what is being?*

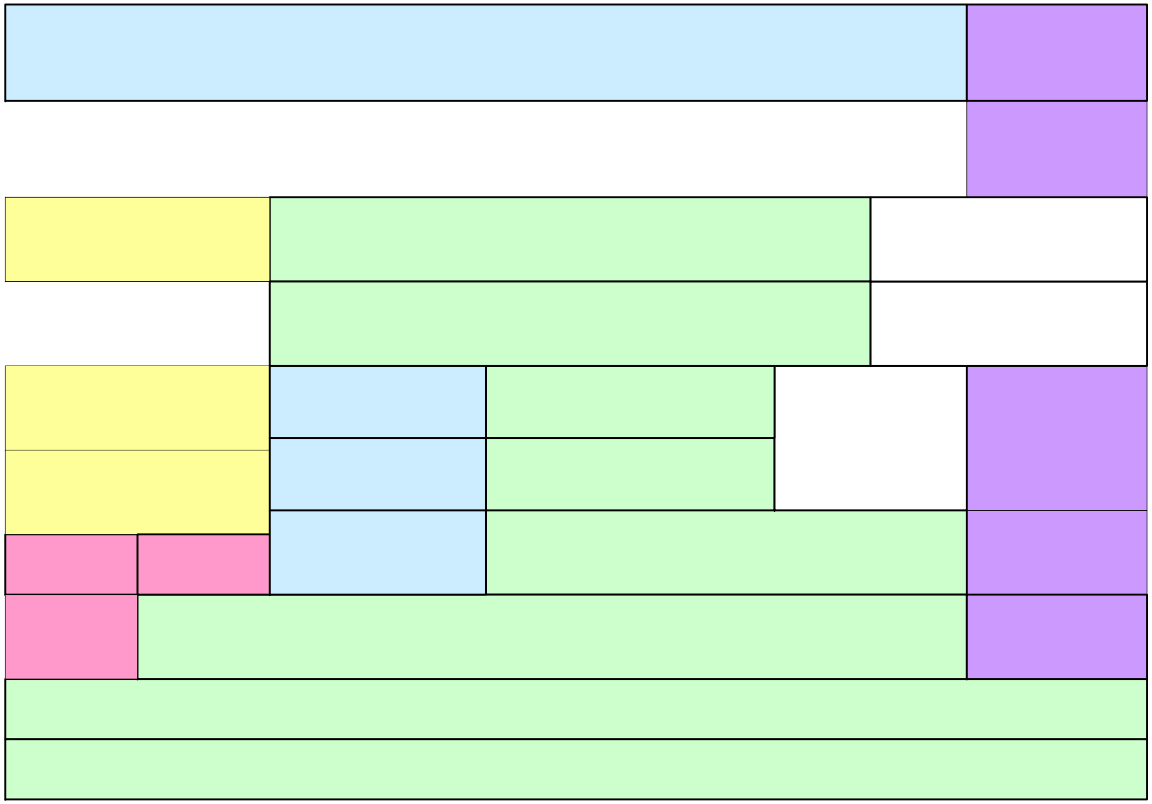
Ontology in Computer Science

- A specification of a **conceptualization** or a set of knowledge terms for a particular domain, including
  - The vocabulary: concepts and relations
  - The semantic interconnections: relationships among concepts and relations
  - Some rules of inference
- An ontology describes **a formal specification of a certain domain**:
  - Shared understanding of a domain of interest
  - Formal and machine manipulable model of a domain of interest

The Semantic Web Stack



Parenthesis – The Web Services Stack



Schema from Service-Oriented Computing: Semantics, Processes, Agents – Munindar P. Singh and Michael N. Huhns, Wiley, 2005

2. RDF

- Provides a basis for knowledge representation
- Based on KR ideas (frames) but uses the Web to enhance interoperability

- XML
  - Gives a document tree
  - Doesn’t identify the content represented by a document, where *content* means
    - Concepts the document is about
    - Relationships among them
  - Enables multiple representations for the same content

RDF

- RDF captures descriptions of resources
- A resource is an “addressable” object
  - Of which a description can be given
  - Which is identified via a URI (Uniform Resource Identifier)
- A literal is something simpler
  - A value, e.g., string or integer
  - Cannot be given a description

RDF

- RDF is based on a simple grammar
- An RDF document is just a set of statements or triples
- Each statement consists of
  - *Subject:* a resource
  - *Object:* a resource or a literal
  - *Predicate:* a resource
- RDF uses:
  - XML serialization
  - Standard XML namespace syntax
  - Namespaces are defined by the RDF standard
    - Typically abbreviated **rdf** and **rdfs**
- Comes with RDFS - a meta-vocabulary

RDF

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
   <rdf:Description rdf:about="http://www.wiley.com/SOC">
      <dc:title>Service-Oriented Computing</dc:title>
      <dc:creator>Munindar</dc:creator>
      <dc:creator>Michael</dc:creator>
      <dc:publisher>Wiley</dc:publisher>
   </rdf:Description>
</rdf:RDF>
```

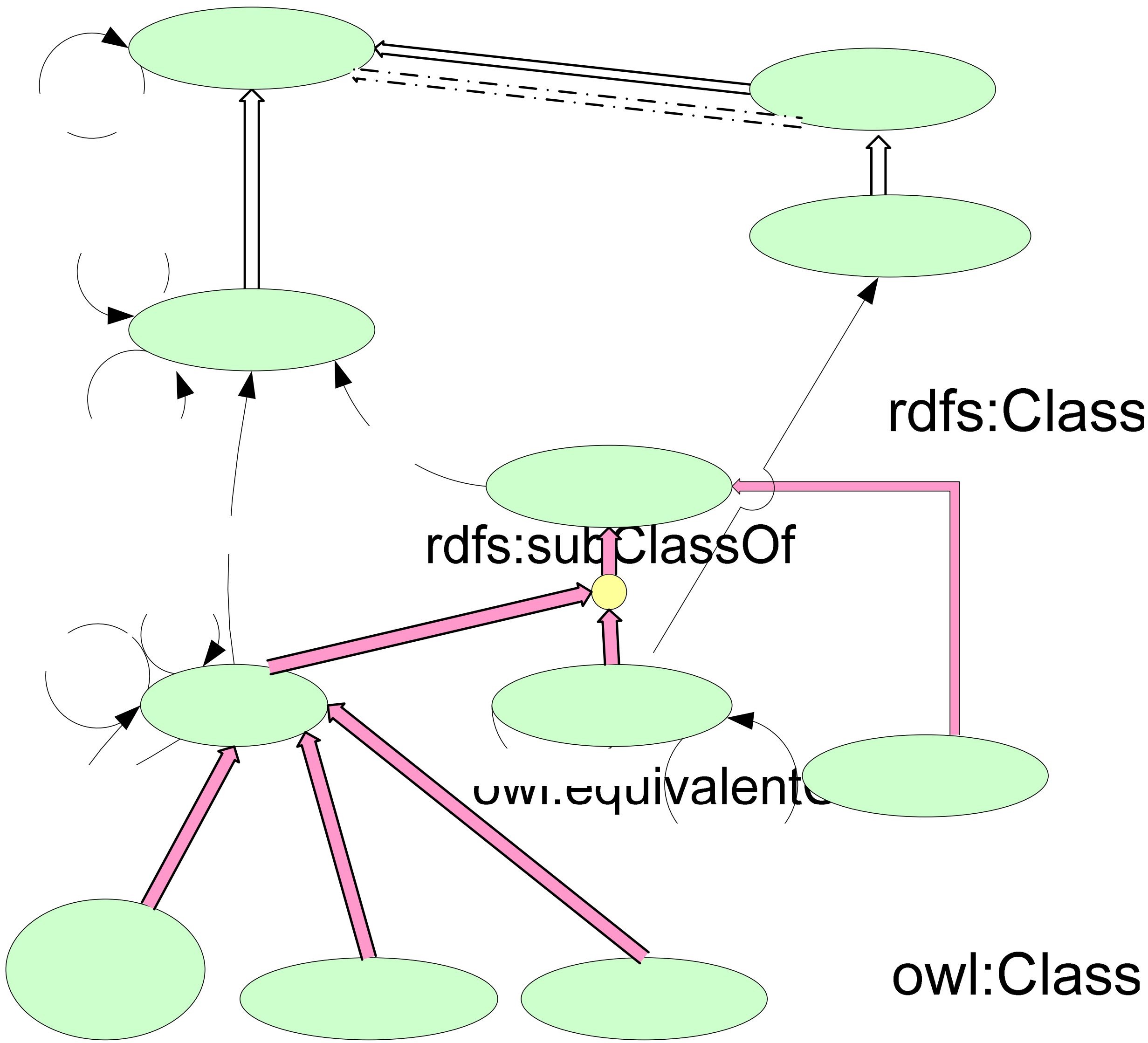
RDF Schema

- Analogous to an object-oriented type system built on top of RDF.
- RDFS defines:
  - rdfs:Class, rdfs:subClassOf
  - rdfs:Resource, rdfs:Literal
  - rdfs:Property, rdfs:subPropertyOf
  - rdfs:range, rdfs:domain
  - rdfs:label, rdfs:comment, rdfs:seeAlso
- OWL - greatly enhances the above

3. OWL

- OWL standardizes additional constructs to be able to capture more meaning
  - Builds on RDF, by *limiting* it
  - Gives formal semantics to new terms
- Based on description logic
- DL Concepts = OWL Classes
- DL individuals = OWL Individuals
- DL Roles = OWL Properties

OWL Entities and Relationships



Picture from Service-Oriented Computing: Semantics, Processes, Agents - Munindar Singh and Michael Huhns, Wiley 2005

3.1 OWL – Classes

- OWL Classes correspond to concepts in DL
- **owl:Class** – defined as a subclass of **rdfs:Class**
- All OWL classes are members of **owl:Class**

Owl have some predefined classes:

- Predefined class **owl:Thing** – top of class hierarchy ($\top$)
- Predefined class **owl:Nothing** –no instances, bottom of hierarchy, a subclass of any other class ($\bot$)

Classes

- Simple examples:

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

- **rdf:ID** defines the name of the class
- Region may be referred as
  - rdf:resource="#Region"
- <**rdf:about**="#Winery"/> may be used to extend the class "Winery"

Subclasses

- Class definitions

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

- A class may have superclasses

```
<owl:Class rdf:ID="Mammals">
  <rdfs:subClassOf  rdf:resource="#Animals"/>
  <rdfs:subClassOf  rdf:resource="#Vertebrate"/>
</owl:Class>
```

Subclasses

- Subclasses/Superclasses define a subsumtion relation $\sqsubseteq$

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

- DL equivalent

  Pasta $\sqsubseteq$ ConsumableThing $\qquad \forall x\ Pasta(x) \rightarrow ConsumableThing(x)$

- Use

```
<owl:Class rdf:about="Pasta">
  <rdfs:subClassOf rdf:resource="#EdibleThing"/>
  ...
</owl:Class>
```

  Pasta $\sqsubseteq$ EdibleThing

3.2 Individuals

- Describe members of a class
- Declare an individual named CentralCoastRegion as a member of class Region

```
<Region rdf:ID="CentralCoastRegion"/>
```

- CentralCoastRegion: Region
- This is equivalent to

```
<owl:Thing rdf:ID="CentralCoastRegion">
    <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

- rdf:type is an RDF property which links an individual to the class to which belongs

Individuals

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

```
<owl:Class rdf:ID="CabernetSauvignon">
  <rdfs:subClassOf  rdf:resource="#Winery"/>
</owl:Class>
```

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

```
<Winery rdf:ID="SantaCruzVineyard"/>
```

```
<CabernetSauvignon
    rdf:ID="SantaCruzVineyardCabernetSauvignon">
        <locatedIn rdf:resource="#SantaCruzRegion"/>
        <hasMaker rdf:resource="#SantaCruzVineyard"/>
</CabernetSauvignon>
```

3.3 Properties

- 2 types of properties:
  - **Object properties (a)**
    - instances of **owl:ObjectProperty**
    - relate instances of 2 classes
    - **domain** + **range** = instances of **owl:Class** ; are **owl:Thing** (unless otherwise specified)
  - **Data type properties (b)**
    - instances of **owl:DatatypeProperty**
    - relate an instance of a class with an instance of a data type
    - **domain** is the same ; **range** = an instance of **rdfs:DataType** and is an **owl:DataRange**

pred(x,y) – x: inst class
y: inst class

pred(x,y) – x: inst class
y: inst data type

(a) Object properties

- A sequence of OWL elements are (implicitly) linked by conjunctions
- Examples of object properties

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

$\exists$ madeFromGrape.T $\sqsubseteq$ Wine

T $\sqsubseteq$ $\forall$madeFromGrape.WineGrape

$\exists x\ madeFromGrape(y,x) \rightarrow Wine(y)$

$\forall x\ madeFromGrape(y,x) \rightarrow WineGrape(x)$

Properties and sub-properties

- rdfs:subPropertyOf
- rdfs:domain
- rdfs:range
- rdfs:equivalentProperty
- rdfs:inverseOf – only for object properties

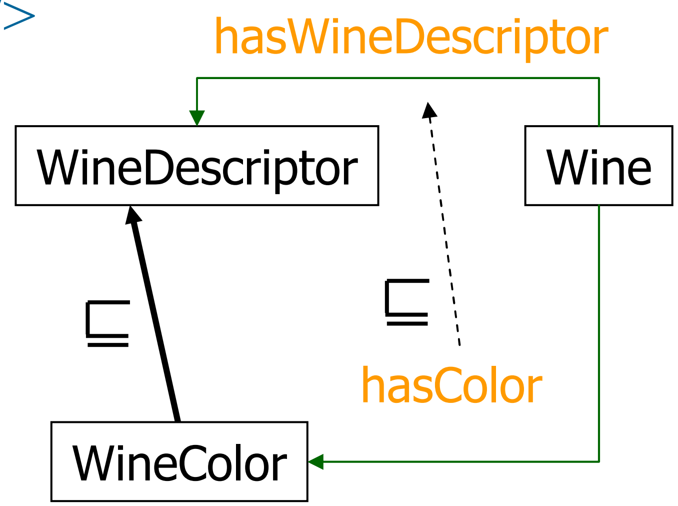
```
<owl:ObjectProperty rdf:ID="livesIn">
   <rdfs:domain rdf:resource="#Animal"/>
   <rdfs:range rdf:resource="#Location"/>
   <rdfs:subPropertyOf rdf:resource="#hasHabitat"/>
   <rdfs:equivalentProperty rdf:resource="#hasHome"/>
</owl:ObjectProperty>
```

Another example of object properties

```
<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:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineColor"/>
</owl:ObjectProperty>
```



(b) Data type properties

- Represent relations between class instances and data types XML Schema
- All OWL engines must support at least the data types:
  - xsd:integer si xsd:string

- Example

```
<owl:DatatypeProperty rdf:ID="yearValue">
  <rdfs:domain rdf:resource="#VintageYear"/>
  <rdfs:range rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

- yearValue binds owl:Thing to positive integer values

More on properties

- **R is Transitive if and only if**

  xRy and yRz imply xRz

- **R is Symmetric if and only if**

  xRy iff yRx

- **R is Functional if and only if**

  xRy and xRz implies y = z

- **R_1 and R_2 are Inverse Properties if and only if**

  xR_1y iff yR_2x

Examples taken from the Wine Ontology

- http://www.w3.org/TR/owl-guide/wine.rdf
- http://oaei.ontologymatching.org/tests/102/onto.html

3.4 Class constructors

- How can we build a class?

(a) By specifying a class name

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

(b) By specifying a class name + descendancy

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

(c) By using logical operators: owl:IntersectionOf ( ), ⊓ owl:unionOf (⊔), owl:complementOf (¬)

or enumeration owl:oneOf (list all individuals)

Used generally with the data type rdf:parseType='Collection'

(d) Impose restrictions on properties = powerful mechanism

Class constructors

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer
complementOf	$\neg C$	$\neg$Male
oneOf	$\{x_1 \ldots x_n\}$	{john, mary}
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer
maxCardinality	$\leqslant n P$	$\leqslant 1$hasChild
minCardinality	$\geqslant n P$	$\geqslant 2$hasChild

Restrictions

- Allows building classes based on restrictions applied to properties (d)
- The objects that satisfy the restriction on the property make an anonymous class
- owl:Restriction – subclass of owl:Class
- A restriction may be of 2 types
  - owl:ObjectRestriction – applied to an Object Property
  - owl:DataRestriction – applied to a Data type Property
- The property on which the restriction applies is specified by owl:onProperty

Restrictions

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
         1
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
...
</owl:Class>
```

$Wine \sqsubseteq \geq 1\ madeFromGrape$

- The blue part defines an anonymous class comprising all objects which have property **madeFromGrape**
- The definition of class **Wine** says that the individuals which are **Wine** are also members of this anonymous class
- Every **Wine** individual must participate in at least one **madeFromGrape** relation

Combining logical operators

```
<owl:Class rdf:ID="WhiteBurgundy">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Burgundy"/>
    <owl:Class rdf:about="#WhiteWine"/>
</owl:intersectionOf>
</owl:Class>
```

```
<owl:Class rdf:ID="WhiteWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor"/>
      <owl:hasValue rdf:resource="#White"/>
   </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Combining logical operators

```
<owl:Class rdf:ID="Fruit">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#SweetFruit"/>
    <owl:Class rdf:about="#NonSweetFruit"/>
  </owl:unionOf>
</owl:Class>
```

Different from:

$\text{Fruit} \equiv \text{SweetFruit} \sqcup \text{NonSweetFruit}$

```
<owl:Class rdf:ID="Fruit">
   <rdfs:subClassOf rdf:resource="#SweetFruit"/>
  <rdfs:subClassOf rdf:resource="#NonSweetFruit"/>
</owl:Class>
```

$\text{Fruit} \sqsubseteq \text{SweetFruit} \sqcap \text{NonSweetFruit}$

Combining logical operators

```
<owl:Class rdf:ID="SweetRedFruit">
 <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#SweetFruit"/>
    <owl:Class rdf:about="#RedFruit"/>
  </owl:unionOf>
</owl:Class>
```

SweetRedFruit $\equiv$ SweetFruit $\sqcap$ RedFruit

Different from:

```
<owl:Class rdf:ID="SweetRedFruit">
   <rdfs:subClassOf rdf:resource="#SweetFruit"/>
  <rdfs:subClassOf rdf:resource="#RedFruit"/>
</owl:Class>
```

SweetRedFruit $\sqsubseteq$ SweetFruit $\sqcap$ RedFruit

Combining logical operators

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

<owl:Class rdf:ID="NonConsumableThing">
   <owl:complementOf rdf:resource="#ConsumableThing"/>
</owl:Class>
```

```
<owl:Class rdf:ID="NonFrenchWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine"/>
    <owl:Class>
      <owl:complementOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#locatedIn"/>
            <owl:hasValue rdf:resource="#FrenchRegion"/>
        </owl:Restriction>
      </owl:complementOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

Enumeration

```
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor"/>
  <owl:oneOf rdf:parseType="Collection">
      <owl:WineColor rdf:about="#White"/>
      <owl:WineColor rdf:about="#Rose"/>
      <owl:WineColor rdf:about="#Red"/>
  </owl:oneOf>
</owl:Class>
```

Restrictions

```
<owl:Class rdf:ID="USACompany">
    <rdfs:subClassOf rdf:resource="#Company"/>
     <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#locatedIn"/>
            <owl:hasValue rdf:resource="#USA"/>
        </owl:Restriction>
     </rdfs:subClassOf>
</owl:Class>
```

$USACompany \sqsubseteq Company \sqcap locatedIn:USA$

Restrictions

```
<owl:Class rdf:ID="EuropeanCompany">
   <rdfs:subClassOf rdf:resource="#Company"/>
    <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#locatedIn"/>
         <owl:someValuesFrom rdf:resource="#EuropeanCountry"/>
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

$$\text{EuropeanCompany} \sqsubseteq \text{Company} \sqcap \exists\ \text{locatedIn.EuropeanCountry}$$

Restrictions

```
<owl:Class>
 <owl:intersectionOf   rdf:parseType="Collection">
    <owl:Class   rdf:about="#Person"/>
    <owl:Restriction>
         <owl:onProperty   rdf:resource="#hasChild"/>
            <owl:allValuesFrom>
                   <owl:unionOf   rdf:parseType="Collection">
                       <owl:Class rdf:about="#Doctor"/>
                       <owl:Restriction>
                              <owl:onProperty  rdf:resource="#hasChild"/>
                              <owl:someValuesFrom  rdf:resource="#Doctor"/>
                       </owl:Restriction>
                   </owl:unionOf>
             </owl:allValuesFrom>
    </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

$\text{Person} \sqcap \forall \text{hasChild.}(\text{Doctor} \sqcup \exists \text{hasChild.Doctor})$

3.5 Axioms

Classes

Individuals

Properties

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq \neg${peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leqslant 1 P$	$\top \sqsubseteq \leqslant$1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1 P^-$	$\top \sqsubseteq \leqslant$1hasSSN$^-$

- $\mathcal{I}$ **satisfies** $C_1 \sqsubseteq C_2$ iff $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$; satisfies $P_1 \sqsubseteq P_2$ iff $P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
- $\mathcal{I}$ satisfies ontology $\mathcal{O}$ (is a **model** of $\mathcal{O}$) iff satisfies every axiom in $\mathcal{O}$

4. Correspondences

- OWL
- Manchester syntax
- DL

Syntactic correspondences

Constructors

OWL	Manchester	DL
intersectionOf	and	$\sqcap$
unionOf	or	$\sqcup$
complementOf	not	$\neg$
subClassOf		$\sqsubseteq$
equivalentClass		$\equiv$

Semantic correspondences

Constructors

OWL	**Manchester**	**DL - sem**
intersectionOf	and	
unionOf	or	
complementOf	not	
subClassOf		
equivalentClass		$C^I = D^I$ for any interpretation

Syntactic correspondences

Restrictions

OWL	Manchester	DL
someValuesFrom	some	$\exists$
allValuesFrom	only	$\forall$
hasValue	value	$:$
minCardinality	min	$\leq$
cardinality	exactly	$=$
maxCardinality	max	$\geq$

Semantic correspondences

Restrictions

Manchester	**DL sem**
some	
only	
value	
min	
exactly	
max	

Fem ≡ Pers and GenFem

Barb ≡ Pers and not GenFem

Mama ≡ Fem and areCopil some Pers

Tata ≡ Barb and areCopil some Pers

Parinte ≡ Tata or Mama

Bunica ≡ Mama and areCopil some Parinte

MamaCuMultiCopii ≡ Mama and areCopil min 3 Pers

MamaFaraFiica ≡ Mama and areCopil only (not Fem)

5. OWL Example

- Consider an academic setting where students take courses and courses are offered by departments. Further, assume that each course is offered by exactly one department. CS is a department, a student must take at least one course, and a full-time student must take between three and five courses.

```
<owl:Class rdf:ID="Student"/>
<owl:Class rdf:ID="Course"/>
<owl:Class rdf:ID="Department"/>
<Department rdf:ID="CS"/>
```

Object Properties: **takes**, **offers**, **offeredBy**

Other classes: **CSCourse**, **FullTimeStudent**, **CS FullTimeStudent**

takes, offers, offeredBy

```
<owl:ObjectProperty rdf:ID="takes">
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#Course"/>
</owl:ObjectProperty>

<owl:InverseFunctionalProperty rdf:ID="offers">
  <rdfs:domain rdf:resource="#Department"/>
  <rdfs:range rdf:resource="#Course"/>
</owl:InverseFunctionalProperty>

<owl:ObjectProperty rdf:ID="offeredBy">
  <owl:inverseOf rdf:resource="#offers"/>
</owl:ObjectProperty>
```

Student

We have captured all constraints except that a student must take at least 1 course

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

FullTimeStudent

```
<owl:Class rdf:ID="FullTimeStudent">
   <owl:IntersectionOf  rdf:parseType="Collection">
      <rdfs:Class rdf:about="#Student"/>
      <owl:Restriction>
          <owl:onProperty rdf:resource="#takes"/>
          <owl:minCardinality
            rdf:datatype="&xsd;nonNegativeInteger">
               3
          </owl:minCardinality>
          <owl:maxCardinality
            rdf:datatype="&xsd;nonNegativeInteger">
               5
          </owl:maxCardinality>
       </owl:Restriction>
   </owl:IntersectionOf>
</owl:Class>
```

CSCourse

```
<owl:Class rdf:ID="CSCourse">
   <owl:IntersectionOf  rdf:parseType="Collection">
     <rdfs:Class rdf:about="#Course"/>
     <owl:Restriction>
         <owl:onProperty rdf:resource="#offeredBy"/>
         <owl:hasValue rdf:resource="#CS"/>
      </owl:Restriction>
  </owl:IntersectionOf>
</owl:Class>
```

CSFullTimeStudent

```
<owl:Class rdf:ID="CSFullTimeStudent">
  <owl:IntersectionOf  rdf:parseType="Collection">
    <rdfs:Class rdf:about="#FullTimeStudent"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#takes"/>
      <owl:allValuesFrom rdf:resource="#CSCourse"/>
    </owl:Restriction>
  </owl:IntersectionOf>
</owl:Class>
```

6. Necessary conditions

- Necessary conditions define the conditions that an individual has to fulfill in order to be an instance of a concept

Mother subClassOf Fem and hasChild some Pers

- if maria is an instance of Mother then it is also an instance of Fem and has at least one child
- if ioana is an instance of Fem and has at least one child ioana is not recognized as an instance of Mother
- Partially defined Class (concept)

Necessary and sufficient conditions

- Necessary and suficient conditions define the conditions that, if an individual fulfills, then the individual is an instance of a concept

Mother ≡ Fem and hasChild some Pers

- In this case if ioana is an instance of Fem and has at least one child then ioana **is** recognized as an instance of Mother
- Totally defined Class (concept)

OWA

- Open World Assumption
- If something is not known this does not mean it is false

  Cal ⊓ ∃areCalaret.Femeie

  Cal ⊓ ∃areCalaret.Barbat

- May have also Copil as areCalaret

- Closure axiom (to "close" the world)

  Cal ⊓∀areCalaret.(Femeie⊔Barbat)

7. OWL Dialects

- **OWL DL** - the core dialect, includes DL primitives; not necessarily (but often practically) tractable
- **OWL Lite** - adds restrictions to OWL DL to make it tractable (card 0 or 1, no disjunction);
- **OWL Full** - lifts restrictions to allow other interpretations; extremely general; potentially intractable (undecidable); included just for fancy expressiveness needs
  - e.g., in OWL Full a class may be treated as a collection of individuals and as an individual in the same time

Credits

- Some slides are based on the book

Service-Oriented Computing: Semantics, Processes, Agents
Munindar P. Singh and Michael N. Huhns, Wiley, 2005