

# Ontologies and OWL

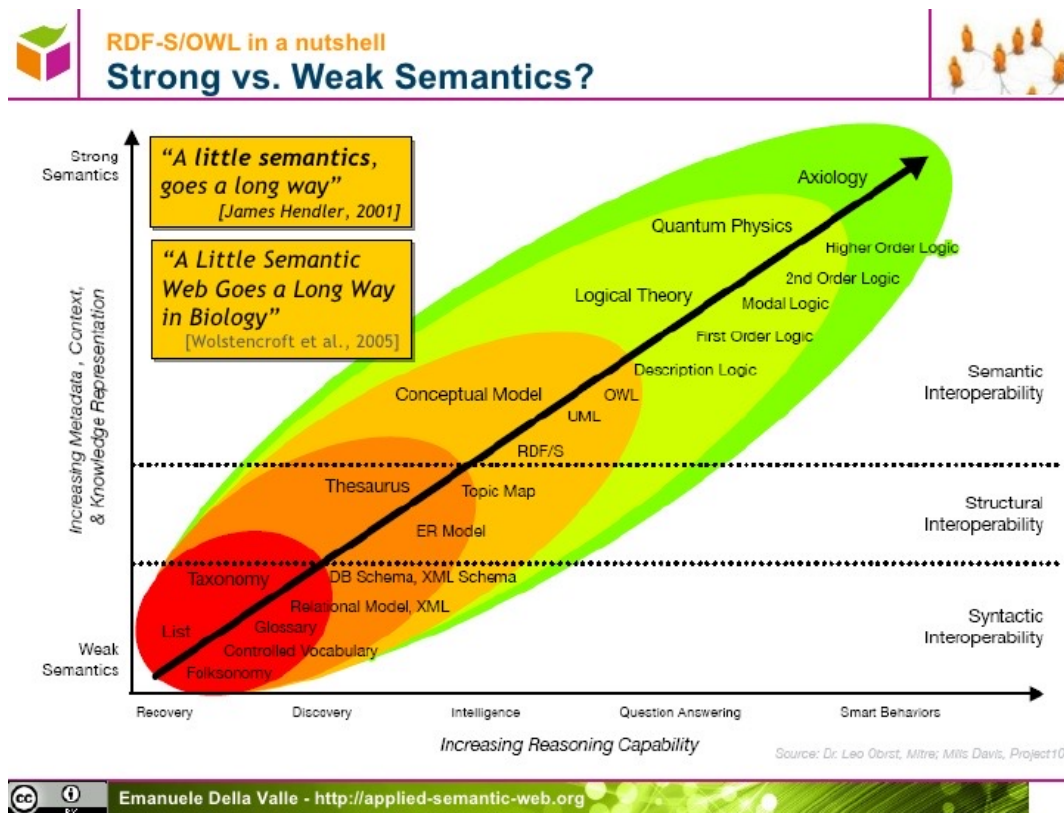
*Dr. Radu Mihailescu*

*Associate Professor*

# Recap

- Sharing knowledge requires embedding semantics with data
- RDF: graph model for knowledge
  - No type, no restrictions on terms
- RDFS: knowledge model extends RDF
  - Allows creating vocabularies (terminologies)
  - lacks means for details: no restriction on range and domain, no cardinality, other properties
- SKOS: builds on RDFS/RDF to create structured vocabularies
  - Taxonomies, thesauri...
  - Have limitations in expressiveness and reasoning
- Need for a language to express strong semantics

# Semantic spectrum



# What is an ontology

- **Metadata**

- Data describing the content and meaning of resources and services
- Requires parties to use the **same terms**

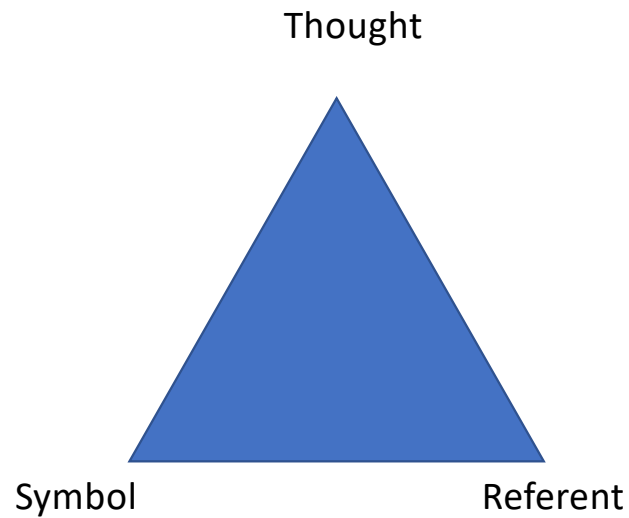
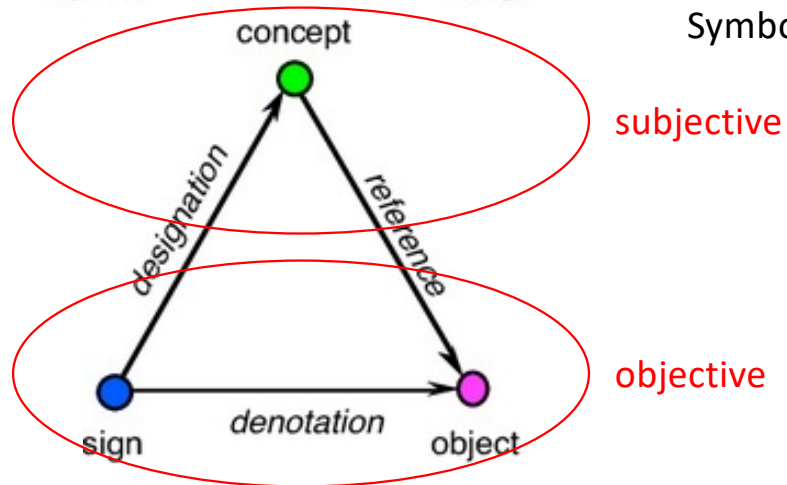
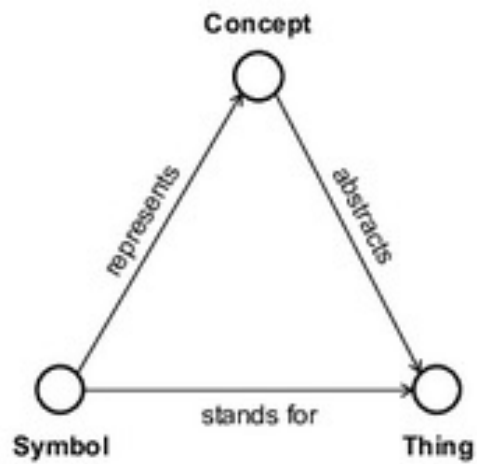
- **Terminologies**

- Shared and common vocabularies
- Used with search engines, software agents, librarians, authors...
- Requires parties to use the **same meaning**

- **Ontologies**

- Shared and common **knowledge of a domain**
- Meant for exchange, search, and discovery

# The meaning triangle



(I would like to drink **camel** milk)

C.a.m.e.l  
جمل



# Ontology

- An *explicit specification of a conceptualization (Gruber 1993)*
- An **ontology** describes the common words, concepts and relationships between concepts used to describe and represent knowledge in a specific domain
- The concept of ontology refers to different forms of knowledge representation, with weak to strong semantics
  - Taxonomy (structured vocabulary) – relational model
  - Thesaurus (words and synonyms) – entity relationship
  - Conceptual models (complex knowledge, concepts, properties, rules) - RDF/RDFS
  - Logical theory (rich complex consistent knowledge, axioms, inference rules) – OWL, Description Logic, First order logic
- An ontology is said to be **well-formed** if it is expressed in well-defined formalism that can be machine interpretable

# Ontology modelling

- Explicit description of a domain, in terms of
  - **Concepts**: classes, sets, types, predicates
  - **Properties** of concepts: attributes (named ***data properties***)
  - **Relationships**: relations between concepts (named ***object properties***)
  - **Constraints** (or axioms): constraints on properties and concepts such as type of values, range of values, cardinality...
  - **Values**: actual values, strings...
  - **Individuals** (instances): actual things, people, concepts...
- When ontology includes also individuals (actual data), it is called a **knowledge base**

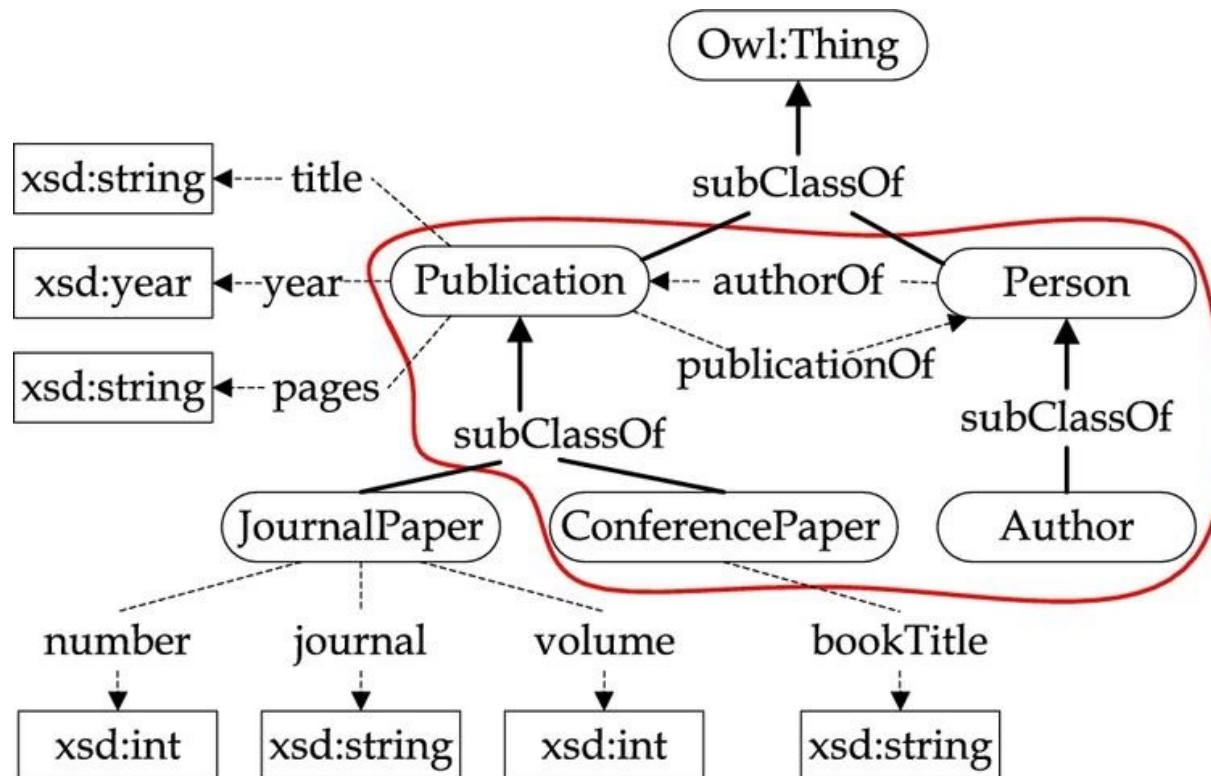
# Relational Schema vs. Ontology

- Relational schema
  - Meant to organise data into databases
  - Relationships are implicit and require human or code to do the interpretation (semantics in the code or in the human mind)
  - Without semantics, no human or program can use the data in a meaningful way
- Ontology
  - Meant to share information along with semantics
  - Relationships defined formally (based on logical constructs)
  - Interpretation is possible by both human and machine



# Example

Simple ontology with informal representation



# Terminology

In semantic computing, we use two conventional terms:

- TBox
  - the ontology (knowledge model), with concepts and relationships...
- ABox
  - the data instances (individuals) as per the ontology

# OWL: Web Ontology Language

A knowledge modelling language whose requirements are

- Extends existing Web standards: XML, RDF, RDFS
- Easy to understand and use: based on familiar KR idioms
- Formally specified - describes the meaning of knowledge precisely
- Having high expressive power
- Provides automated reasoning support

# OWL: what for?

- OWL is primarily concerned with defining terminology that can be used in RDF documents, i.e., classes and properties.
- OWL specifies the terminology in terms of classes and properties and their characteristics
- Individuals (instances) can also be defined in an owl document

# Example of OWL

```
<rdf:RDF
```

It is an RDF document

```
  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/XMLSchema#"
  <owl:Ontology rdf:about="">
```

Metadata about the ontology

```
    <rdfs:comment>An example OWL ontology</rdfs:comment>
```

```
    <owl:imports rdf:resource="http://www.mydomain.org/persons"/>
```

```
    <rdfs:label>University Ontology</rdfs:label>
```

```
  </owl:Ontology>
```

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

Hierarchy of classes

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

```
    <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
```

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

```
  ...
```

```
</rdf:RDF>
```

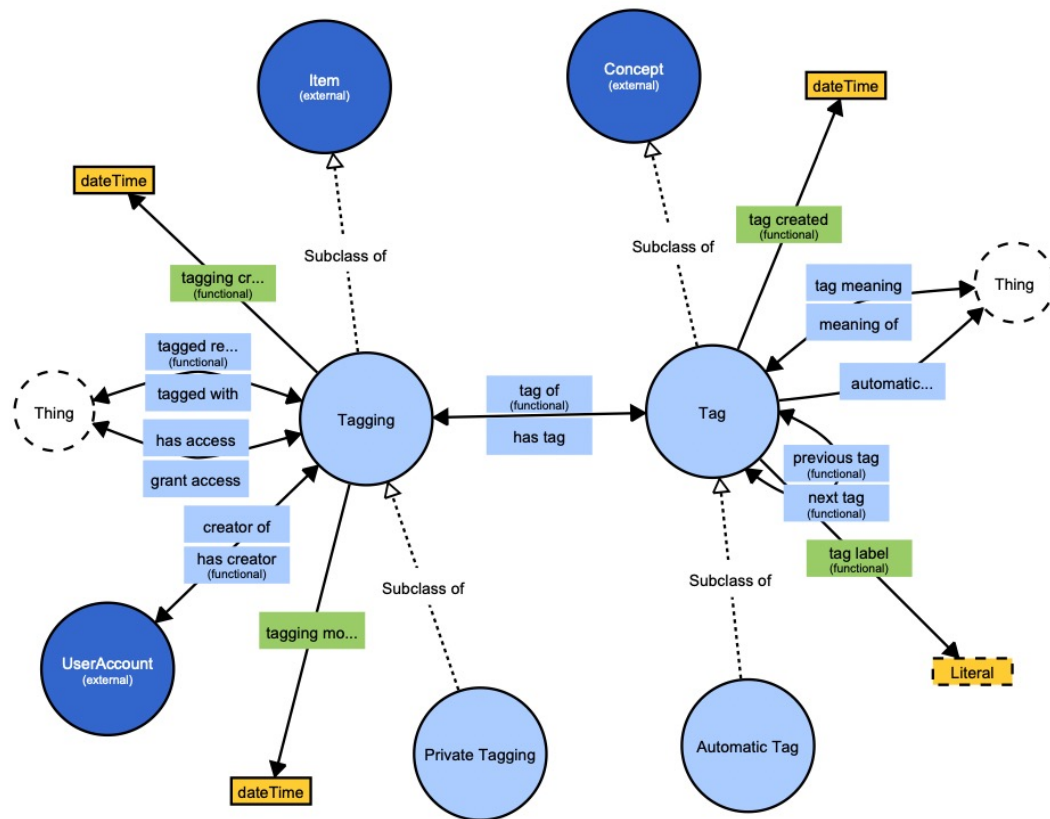
# Another example

```
<!DOCTYPE rdf:RDF [  
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">]>  
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#" Namespaces  
  xmlns:rdf="http://www.w3.org/2002/02/22-rdf-syntax-ns#" metadata  
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">  
<owl:Ontology rdf:about="">  
  <rdfs:label>My Ontology</rdfs:label>  
  <rdfs:comment>An example ontology</rdfs:comment>  
</owl:Ontology>  
<owl:Class rdf:ID="Person" />  
<owl:Class rdf:ID="Man"> Class hierarchy  
  <rdfs:subClassOf rdf:resource="#Person" />  
</owl:Class>  
<owl:ObjectProperty rdf:ID="hasChild" />  
<owl:ObjectProperty rdf:ID="hasDaughter">  
  <rdfs:subPropertyOf rdf:resource="#hasChild" />  
</owl:ObjectProperty> Object property hierarchy
```

```
  <owl:DatatypeProperty rdf:ID="age" /> Data property  
<owl:ObjectProperty rdf:ID="isParentOf">  
  <owl:inverseOf rdf:resource="#isChildOf" Object property  
</owl:ObjectProperty>  
<owl:ObjectProperty rdf:ID="isTallerThan">  
  <rdf:type rdf:resource="&owl;TransitiveProperty" /> Object property  
</owl:ObjectProperty>  
<owl:ObjectProperty rdf:ID="isFriendOf">  
  <rdf:type rdf:resource="&owl;SymmetricProperty" />  
</owl:ObjectProperty> Object property  
<owl:ObjectProperty rdf:ID="hasSSN">  
  <rdf:type rdf:resource="&owl;FunctionalProperty" />  
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />  
</owl:ObjectProperty> Object property with its own characteristics  
</rdf:RDF>
```

# VOWL Representation

Defines a visual language for the user-oriented representation of ontologies.



Graphical primitives used in VOWL

Primitive	Application	Primitive	Application
○	classes	□	datatypes, property labels
( )	properties	.....	special classes/properties
▷▷	property directions	text number symbol	labels, cardinalities

Excerpt of the VOWL color scheme

Name	Color	Application
General	Light Blue	classes, object properties, disjointness
External	Dark Blue	external classes and properties
Deprecated	Grey	deprecated classes and properties
Datatype	Yellow	datatypes, literals
Datatype property	Light Green	datatype properties
Highlighting	Red	circles, rectangles, lines, borders, arrows

# Reasoning with OWL

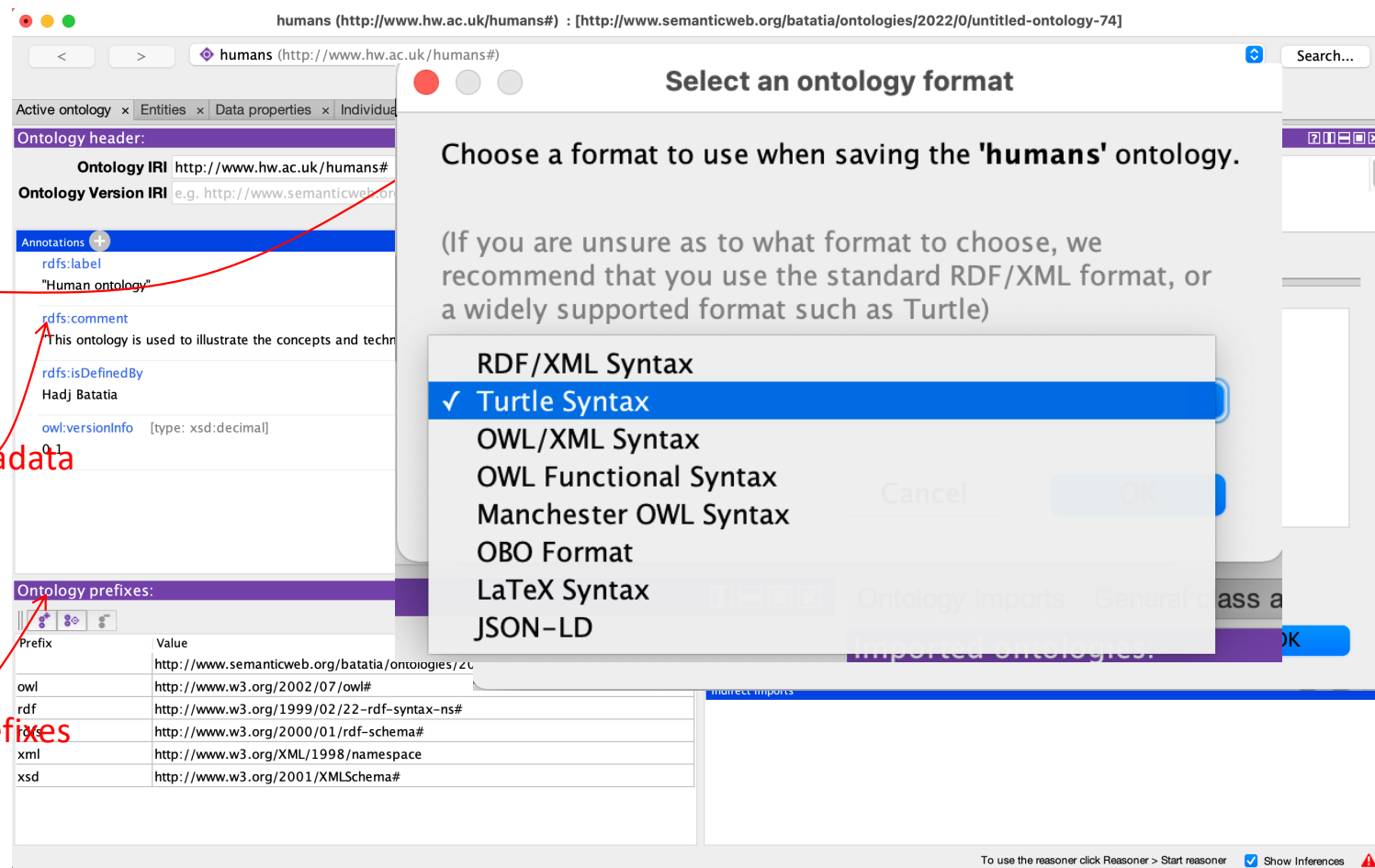
- **Consistency checking**: No contradictions
  - ABox does not contradict TBox (instances compliant with ontology)
- **Concept satisfiability**: possibility of adding individuals from class
  - Can a class have individuals?
- **Classification**: create full inheritance hierarchy to answer questions
  - Does class A **subsumes** class B?
- **Realisation**: compute the direct type (class) of each individual
  - What class does individual X belongs to?
  - Realisation can be done after classification as we need the full inheritance hierarchy



# Protégé

- Protégé is an open source ontology editor using OWL
- It allows creating through a graphical user interface all components of the ontology
- Can be downloaded freely from <https://protege.Stanford.edu>
- It provides a number of views to suit the need of the user at different stages of the ontology engineering process

# 1. Creating an ontology



1. set IRI

2. Add metadata

3. Define prefixes

4. Save ontology

5. Select preferred format

## 2. Create class hierarchy

Use meaningful names

Add annotations to each class

Create a hierarchy

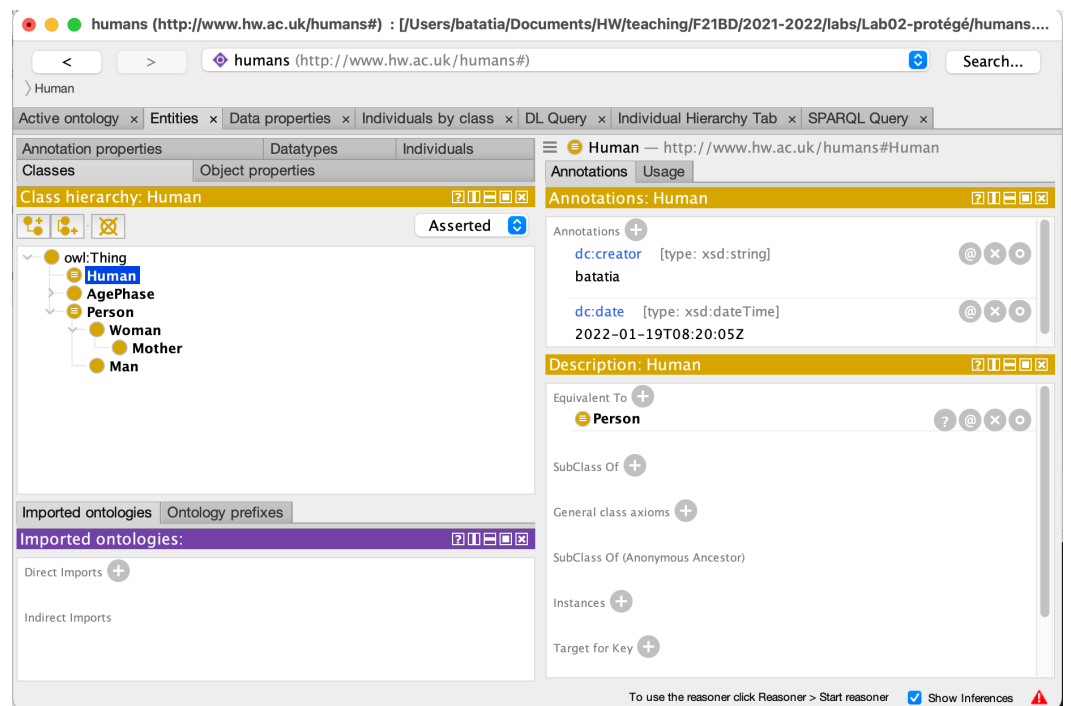
Set any class characteristics,  
example disjoint

The screenshot displays the Protégé ontology editor interface. The main window shows the 'Class hierarchy: Person' view, which is a tree structure starting from 'owl:Thing' and branching into 'AgePhase' (with sub-classes 'Old', 'Adult', 'Teenager', 'Child') and 'Person' (with sub-classes 'Woman' and 'Man'). The 'Person' class is selected, and its 'Annotations' tab is active, showing properties like 'dc:creator' (value: 'batatia'), 'rdfs:comment' (value: 'Any human being without consideration of gender, race or age'), and 'dc:date' (value: '2022-01-19T08:01:01Z'). The 'Description: Woman' tab is also visible, showing 'Equivalent To' and 'SubClass Of' (with 'Person' selected). A 'Woman' dialog box is open in the foreground, showing the 'Class hierarchy' tab with a tree structure where 'Person' is the parent of 'Man' and 'Woman'. The 'Disjoint With' button is highlighted in the main window's class list.

# owl:equivalentClass

- Two classes that refer to the same things
- Written in Turtle format

:Human owl:equivalentClass :Person



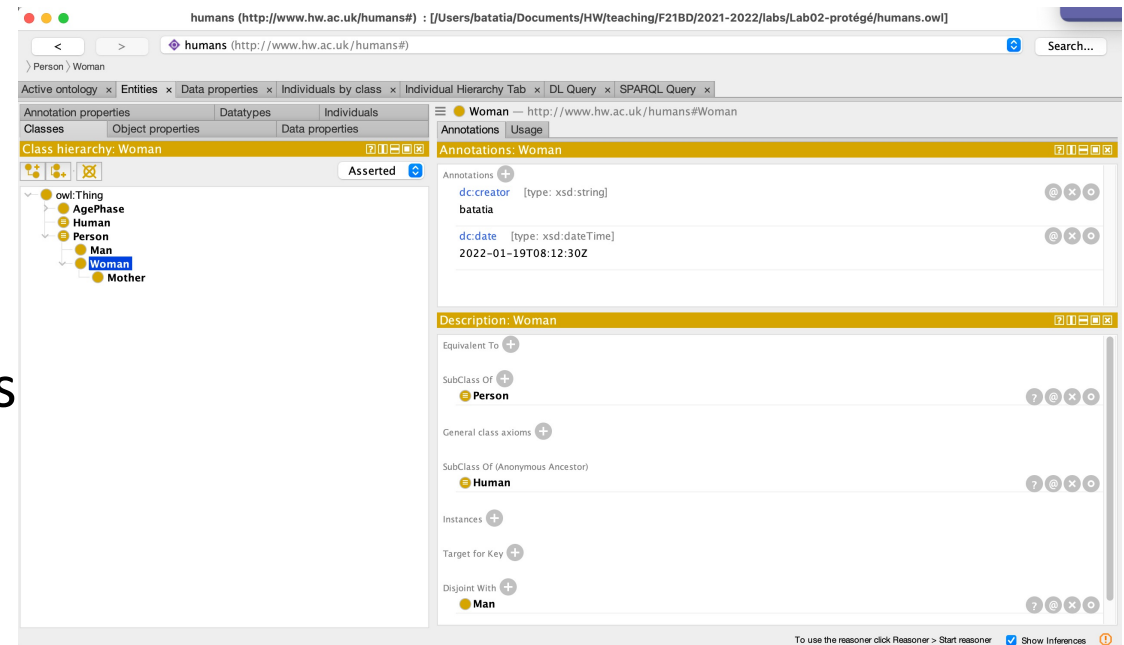
# Disjoint classes

- When individuals cannot belong to two classes, these classes are declared **disjoint**

`:Man owl:disjointWith :Woman ;`

- We can also make one class disjoint from a list of classes, this gives in turtle (a collection):

```
[ rdf:type owl:AllDisjointClasses ;  
  owl:members ( :C1 :C2 :C3 ) ] .
```



### 3. Create object properties (relationships)

- Use verbs to avoid ambiguity with classes
- Add any characteristics

`:marriedTo` a `rdf:Property`.

- Add `:hasHusband`
- Make it inverse of `:hasWife`

The screenshot displays an OWL editor interface with two panels on the left and two on the right.

**Left Panel (Object property hierarchy):**

- Top Panel (have spouse):** Shows a hierarchy starting with `owl:topObjectProperty` and a child `have spouse`. The status is "Asserted".
- Bottom Panel (hasHusband):** Shows a hierarchy starting with `owl:topObjectProperty` and two children, `hasHusband` and `hasWife`. The status is "Asserted".

**Right Panel (Annotations and Description):**

- Annotations: have spouse:** Shows `rdfs:label` with the value `have spouse`.
- Annotations: hasHusband:** Shows `dc:creator` with the value `batatia` (type: `xsd:string`) and `dc:date` with the value `2022-01-19T09:41:37Z` (type: `xsd:dateTime`).
- Description: hasHusband:** Contains a list of characteristics on the left and a list of relationships on the right.
  - Characteristics:** ☐ Functional, ☐ Inverse function, ☐ Transitive, ☐ Symmetric, ☐ Asymmetric, ☐ Reflexive, ☐ Irreflexive.
  - Relationships:** `Equivalent To` (+), `SubProperty Of` (+), `Inverse Of` (+) with `hasWife` selected, `Domains (intersection)` (+), and `Ranges (intersection)` (+).

# Property hierarchy

- Properties can be in a hierarchy
- Example

`:hasWife rdfs:subPropertyOf :hasSpouse .`

The screenshot displays a web-based ontology editor interface. The main panel, titled "Object property hierarchy: hasSpouse", shows a tree structure under "owl:topObjectProperty". The "hasSpouse" property is selected and expanded, revealing its subproperties: "hasWife" and "hasHusband". A toolbar at the top of this panel includes icons for adding, deleting, and asserting properties, along with a status indicator set to "Asserted".

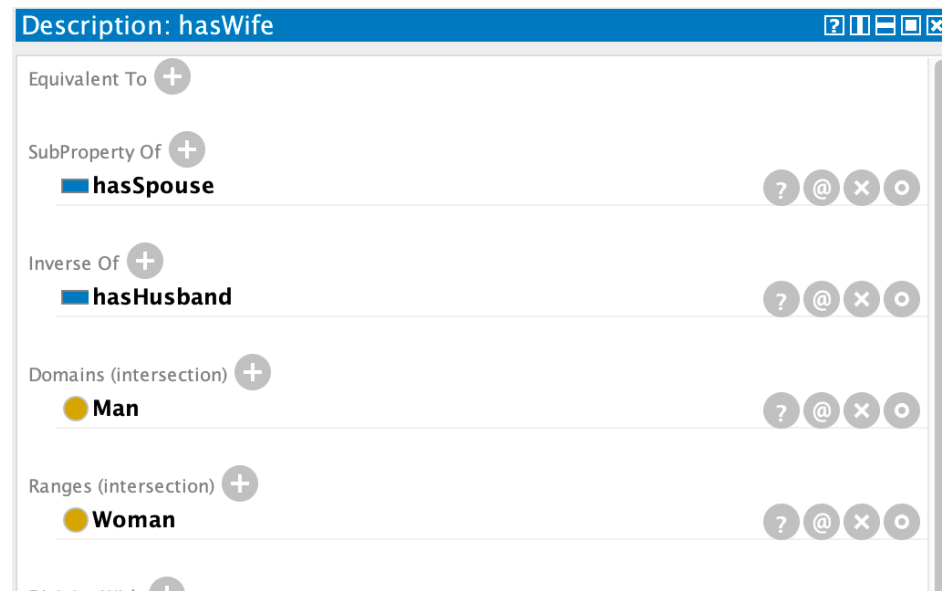
To the right of the main panel are two sidebars. The top sidebar, "Annotations: hasSpouse", lists two annotations: "dc:creator" with the value "batatia" (type: xsd:string) and "dc:date" with the value "2022-01-19T10:00:49Z" (type: xsd:dateTime). The bottom sidebar, "Characteristics: hasSpouse", contains a list of property characteristics with checkboxes: Functional, Inverse function, Transitive, Symmetric, Asymmetric, Reflexive, and Irreflexive. To the right of these checkboxes is a list of logical relationships, each with a plus icon for adding: Equivalent To, SubProperty Of, Inverse Of, Domains (intersection), Ranges (intersection), Disjoint With, and SuperProperty Of (Chain).

# Domain and range

- Restrict the type of subject and object for a property

- Example

```
:hasWife rdf:type owl:ObjectProperty ;  
rdfs:subPropertyOf :hasSpouse ;  
rdf:type owl:AsymmetricProperty ;  
rdfs:domain :Man ;  
rdfs:range :Woman.
```





## 4. Data properties

- These are attributes to describe individuals from classes
- Example

The screenshot displays a web ontology editor interface with three main panels:

- Data property hierarchy: bornOn**: Shows a tree structure where `owl:topDataProperty` is the parent of `bornOn`.
- Annotations: bornOn**: Lists annotations for the `bornOn` property, including `dc:creator` with value `batatia` (type `xsd:string`) and `dc:date` with value `"2022-01-19T10:10:38Z"` (type `xsd:dateTime`).
- Description: bornOn**: Shows the logical description of the property, including the domain `Person` and the range `xsd:dateTime`.

```
### http://www.hw.ac.uk/humans#bornOn
:bornOn rdf:type owl:DatatypeProperty ;
        rdfs:domain :Person ;
        rdfs:range xsd:dateTime ;
        <http://purl.org/dc/elements/1.1/creator> "batatia"^^xsd:string ;
        <http://purl.org/dc/elements/1.1/date> "2022-01-19T10:10:38Z"^^xsd:dateTime .
```

## 5. Add individuals

- Add individuals (instances)
- Define their type (class)

`:Mary rdf:type owl:NamedIndividual :Woman ;`

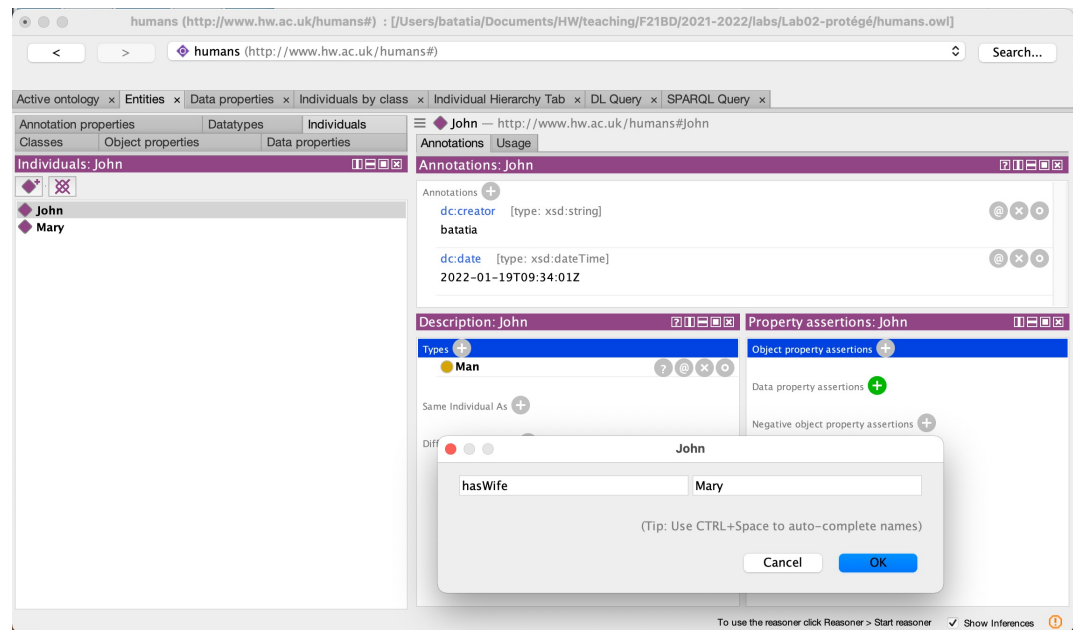
- Assert their properties

`:John :hasWife :Mary;`

- No need to assert that

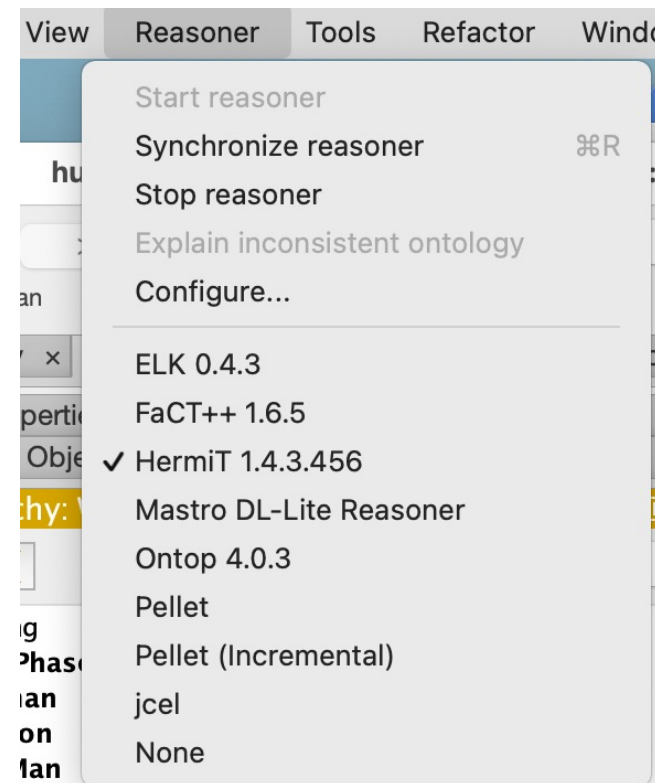
`:Mary :hasHusband :John;`

- It will be discovered by reasoner...



# Protégé: using a reasoner

- A reasoner is an algorithm that applies local inference to discover
  - Inconsistencies (any contradiction)
  - Subsumptions (class inheritance)
  - Realisations (individuals of classes)
- By applying class and property characteristics (inverseOf, symmetric...), find new relationships



# Inferred information in yellow

Property hasWife

**Description: Man**

Equivalent To +

SubClass Of +

- Person
- Human

General class axioms +

SubClass Of (Anonymous Ancestor)

- Human

Instances +

- John

Class Man

**Description: hasWife**

Equivalent To +

SubProperty Of +

- hasSpouse
- inverse (hasSpouse)

Inverse Of +

- hasHusband

**Description: John**

Types +

- Man

Same Individual As +

Different Individuals +

**Property assertions: John**

Object property assertions +

- hasWife Mary
- hasSpouse Mary

Data property assertions +

- bornOn "1777-08-15T03:25:00"

Individual John

**Property assertions: Mary**

Object property assertions +

- hasHusband John
- hasSpouse John

Individual Mary

# Principles when creating an ontology

- There exists **no unique** perfect knowledge model for a domain
  - The model should serve your application
- An ontology cannot be created at once
  - An **iterative** process is required to refine the ontology multiple times to consider various use cases
- Classes and properties should be as close as possible to the real/physical objects of your domain
  - Consider nouns for classes and verbs for properties

# Process of creating an ontology

1. Determine scope and domain of ontology
2. Consider reusing existing ontologies
3. Enumerate important terms in the ontology
4. Define the classes and class hierarchy
5. Define the properties of classes
  - Properties:
6. Relationships
7. Define restrictions on properties
  - Value type (String, Number, Boolean, Enumerated, Instance)
  - Value cardinality
8. Define the instances

# Ontology Development Process

	determine scope		consider reuse		enumerate terms		define classes		define properties		define constraints		create instances
--	--------------------	--	-------------------	--	--------------------	--	-------------------	--	----------------------	--	-----------------------	--	---------------------

*In reality - an iterative process:*

	determine scope		consider reuse		enumerate terms		consider reuse		define classes		enumerate terms		define classes
	define properties		define classes		define properties		define constraints		create instances		define classes		create instances
	consider reuse		define properties		define constraints		create instances						

# Rules for FAIR vocabularies (deployment)

1. Provide a *license* that allows repurposing
2. Determine the *content governance* arrangements
3. Check minimal term definition *completeness*
4. Select a domain and service for the web identifiers, i.e. a *namespace*
5. Design an *identifier schema* and pattern
6. Create a *semantic-standards* based vocabulary – **Interoperability**
7. Add rich *metadata* – **Reusability**
8. *Register* the vocabulary, e.g. with LOV – **Findability**
9. Make the *IRIs resolve* – **Accessibility**
10. Implement a process for *maintaining* the vocabulary



# Semantic-standards compliant vocabulary

1. Identify **terms**
2. Encode term **labels** and synonyms
3. Add textual **definitions**
4. Add notes or **comments** for clarifications
5. Add codes and **symbols**
6. Define the **hierarchy** of terms
7. Encode **relationships**
8. Define **subsets**
9. Define and **document** the whole vocabulary

Next

Advanced



**OWL**