



Australian  
National  
University

# Web Ontology Language with Protegé (for v 5.2.0)

COMP3425/8410 Data Mining

# RDF

- Describes relations as triples of subject-predicate-object where *subjects* and *objects* are resources, and the predicate names a *property* of the subject.
- The object might be a *literal* instead of a resource.

# RDF Schema (RDFS)

- Assigns a class structure to RDF objects
- Assigns semantics to RDF triples  
(...just as E-R does for relational data!)
- Used to define vocabularies and ontologies over RDF
- Supports inference of implicit information

## OWL is a much more expressive ontology language

- It is a very expressive data modelling language with a formal semantics (*c.f. UML*).
- Is fundamentally a logical language: a form of Description Logic: a decidable fragment of First-Order Predicate Calculus.
- Is best used with a *reasoner* for logical inference
  - There are many OWL docs out there which do not say what they claim to say and which do not respect the semantics because someone just looked at the syntax and went from there
  - Especially happens with property **owl:sameAs**, used for linked data
- OWL 2 standardised October 2009 (supersedes OWL 2004). Minor update December 2012 (for XML types).
- Spec: [http://www.w3.org/standards/techs/owl#w3c\\_all](http://www.w3.org/standards/techs/owl#w3c_all)
- Comes with several syntaxes: RDF/XML is normative; Functional-Style syntax is used in the specs; OWL/XML is an XML encoding of that; Manchester Syntax is readable (e.g. in Protégé ontology editor), Turtle is most popular
- Is *meant* to be built on RDFS, but they are not quite comfortable co-habiting. Advise to *avoid* OWL2 Full = OWL2 DL + RDF(S).
- Instead use OWL 2 DL or a sublanguage (OWL EL, OWL QL, OWL RL): think of it as a new language that reuses some RDF(S) vocabulary


# OWL preliminaries

- OWL namespace is owl: <<http://www.w3.org/2002/07/owl#>>
- Open World Assumption means
  - Can assert negative things, but cannot infer it from the absence of the positive thing (beware, database modellers!)
  - No unique names assumption: i.e cannot infer that two things are not the same because they have different names (URIs)
- OWL is strongly typed
  - Disjoint domains of individuals, classes, object properties, datatype properties, datatypes, annotation properties.
  - Where the same name is used for different ones, they are different things (punning).

# OWL modelling

- Is all about constructing statements about the relationships between and amongst various classes, properties and individuals .
- There are two predefined classes:
  - **owl:Thing** == the superclass of all classes, contains all individuals
  - **owl:Nothing** == the subclass of all classes, contains no individuals
- Two predefined object properties (similarly)
  - **owl:topObjectProperty**, **owl:bottomObjectProperty**
- And two predefined data properties (similarly)
  - **owl:topDataProperty**, **owl:bottomDataProperty**
- And some annotation properties (only for human readability)
  - e.g. **owl:versionInfo**, **owl:DeprecatedClass**,  
**owl:imports** ...

# Exercise: Start Protege and import an ontology

- Go to Active ontology tab and then Ontology imports Tab
- Direct Import -> + -> ..located on the web <http://www.w3.org/ns/ssn/>
- Notice under Active Ontology tab:
  - Indirectly Imports <http://www.w3.org/ns/sosa/> SOSA
- Click on Entities tab:
  - Click on each sub-tab: Classes, Object Properties, Data properties, Annotation properties, Annotation properties, Datatypes, Individuals to see each predefined type of modelling primitives
  - Note under Entities -> Classes and Entities -> Object Properties tabs: **Thing** and **topObjectProperty** are predefined
- Click under Entities tab:
  - On triangle adjacent Thing to see subclasses; on Feature of Interest itself to see annotations top right and description lower right.
  - On triangle adjacent topObjectProperty to see sub-properties; on has property and hosts to see annotations top right and description lower right

# Make your own ontology

Add an ontology annotation (this is documentation/metadata only)

- Active Ontology **tab** -> Annotations+ -> dcterms:creator <yourname>
- Annotations+ -> dcterms:created -> Type -> xsd:dateTime 2018-06-30T12:00:00Z




# Save your ontology as Turtle

- Active Ontology -> Ontology IRI -> **<give your ontology a URI you like>**
- File -> Save as -> Select format -> Turtle Syntax -> OK ...
- Open your ontology file with a text editor and look at it.
- Check you can read the turtle structure.

# Individuals

- Can be asserted to belong to a class using `rdf:type` as for RDF(S)
- Can be declared as property instances in RDF(S)
- Can be declared to be not related by a named property **`owl:NegativePropertyAssertion`**
- Can be forced to be different using **`owl:differentFrom`** or **`owl:allDifferent`**
- Can be forced to be the same using **`owl:sameAs`**

# Exercise: Make some individuals

- In the Entity -> Classes tab, navigate to *Thing*-> *System*-> *Sensor*
- Go to the Individuals by class tab
- In the Instances pane, noting For: *Sensor*, create a new named individual of class *Sensor* by clicking  and giving it a name e.g. *my1sensor*
- Annotate it in the Annotations pane (use the `rdfs:comment` annotation property)
- And another *Sensor* individual e.g. *my2sensor*
- Notice that you can see their types and annotations under the Entities -> Individuals tab.
- Navigate to *Sensor* in the Class hierarchy pane of the Classes tab and you can see them as Instances in the Description for *Sensor* on the right.
- Save your ontology as Turtle: File->SaveAs and look at it.

# Exercise: Dress up those individuals

- Select one of the individuals in the `Instances` pane of `Individuals` by class tab.
- Make *my1sensor* and *my2sensor* always different using `Different Individuals` in the `Description` pane
- Add a data property assertion for *my1sensor*:
  - In the `Property assertions` pane click `Data Property assertions` +
  - Choose a property and set a value, *result time 23*
  - N.B. this is not the intended use of *has result time*, and it is not clear what the *result time* of a *Sensor* should even be, but it will do for now
- Save your ontology as Turtle: `File->Save` and look at it.

# More on Properties

- Like RDF properties, an OWL *object property* relates two things (*individuals* only on the left).
- An OWL *datatype property* relates an individual to a value (usually) drawn from an XSD datatype. Think of it as a named attribute value of an individual.
- A property might be defined to have a an `rdfs:domain` or an `rdfs:range` which is a class.
- Properties may be declared: `owl:SymmetricProperty`,  
`owl:AsymmetricProperty`, `owl:ReflexiveProperty`,  
`owl:IrreflexiveProperty`, `owl:TransitiveProperty`,  
`owl:FunctionalProperty`, `owl:InverseFunctionalProperty`
- Of the above, only `owl:FunctionalProperty` can be used for datatype properties. The others are for object properties.

# Exercise: Reasoning to check typing

- Choose a reasoner from the menu Reasoner->select one Fact++, Hermit, and Pellet are all usually ok. Pellet is often the best for debugging.
- Start it Reasoner-> Start reasoner
- What happened? Look at results and correct the error
  - Hint: look at `Classes` tab
  - Hint: look at the range of property *result time* on the `Data properties` tab
  - Hint: use the Usage pane
  - Hint: there are many possible fixes; suggest deleting the property of *my1sensor* in the `Property assertions` pane.
- Check again with the reasoner Reasoner->Synchronize reasoner

# Properties may be related to other properties

- A property can be
  - equivalent to another named property: `owl:equivalentProperty`
  - or a subproperty of one: `rdfs:subPropertyOf`
  - or the inverse of one `owl:inverseOf` (swap the two individual positions)
- Using the property constructor, a property can be defined in terms of other properties:
  - `owl:propertyChainAxiom` (composition/concatenation of several properties can be a `rdfs:subPropertyOf` of another one)
- Also, properties may be mutually disjoint
  - `owl:propertyDisjointWith`, `owl:AllDisjointProperties`

# Exercise: Create an object property

- Create a property *owns* as a subproperty of *has property*, by clicking on *has property* under the *Object properties* tab



- Then create a property *is\_owned\_by* at the top (by adding a subproperty to `owl:topObjectProperty`)
- Make them inverses
  - Hint: look for `inverse Of` in the description pane for either one



# Exercise: Object Property Reasoning

- Check with a reasoner: Synchronise the reasoner. Tick `Show Inferences` on bottom right corner.
- Note the locations of *is\_owned\_by* and *owns* in the object property hierarchy pane on the left.
- Select `Inferred` on the tick box in *Object property hierarchy* pane and look again at *is\_owned\_by*
- Note that the `Description` of *is\_owned\_by* now recognises it is a subproperty of *is\_property\_of*. Why?
  - Hint: look at the `Description` of *is\_property\_of*
- Make this inference explicit: assert *is\_owned\_by* to be a subproperty of *is\_property\_of*
- Check with the reasoner: `Reasoner-> Synchronise reasoner`
- Save your ontology as Turtle: `File->Save` and look at it.

# The Real Action is about Defining Classes

- A class description can be simply a class name.
- Usually, give your class a name and define it as `rdfs:subclassOf` or `owl:equivalentClass` to another class that might be named or might be an anonymous complex class description.
- The `rdfs:subclassOf` asserts a class hierarchy arrangement.
- If you use `owl:equivalentClass` or a complex class on the left of `rdfs:subclassOf`, a reasoner can infer a class hierarchy.
- A complex class description is made up of conjunctions or disjunctions or negations of other [complex] class descriptions, respectively
  - `owl:intersectionOf`
  - `owl:unionOf`
  - `owl:disjointUnionOf`
  - `owl:complementOf`
  - `owl:disjointWith`

# Exercise: Look at some classes

- We are going to load the upper ontology DUL and its alignment to SSN
- Turn off the reasoner: Reasoner->Stop Reasoner
- Go to Active ontology **tab** and then Ontology imports Tab
- Direct Import -> + -> ...located on the web  
<http://www.w3.org/ns/ssn/dul>
- Note: this also shows us something about multiple-ontology structure and dependency
- Select from the top menu bar: View -> show all loaded ontologies

# Exercise: Look at Classes

- On the `Entities` -> `Classes` tab Look at the Description of *Entity* – equivalent To (exactly, fully-defined, equivalent class) as the disjoint union of classes *Abstract*, *Event*, *Object*, *Quality* (and *Region* is mentioned too).
- Click on each of classes *Abstract*, *Event*, *Object*, *Quality* and look at Description pane Disjoint With to show the disjoint part of this.
- Mostly, `subClassOf` (primitive, subsumed by) is used.
- Look at the Description of *Sensor* – a subclass of both *System* and *Object*
  - Hint: That means a *Sensor* falls in the intersection of *System* and *Object*. However, since *Objects* are *Systems*, the assertion that *Sensor* is an *Object* is redundant.
- Simplify your workspace by going back to your saved ontology file: `File`-> `Open` <your last saved file> -> `No` And then `.exit` from the older window

# Make a Class

- Create a class that is a subclass of *Sensor*, called *ANUSensor*:

Entities-> Classes ->Sensor ->rightclick Add subclass ->  
*ANUSensor*

- Notice the “inheritance” in the Subclassof (anonymous ancestor) part of the `Description` pane
- Save it as Turtle and look at it.

# Exercise: Validate

- Use <http://mowl-power.cs.man.ac.uk:8080/validator/>
- Check your ontology is in OWL 2.
- Check your ontology is in OWL 2 DL.
- Have a look at OWL-RL, OWL-QL, OWL-EL

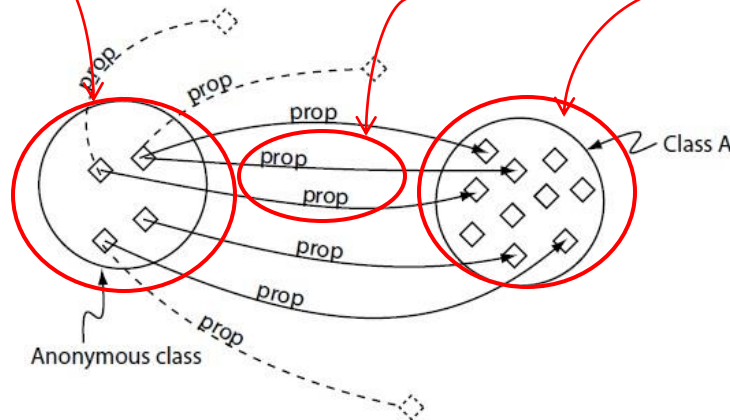
# Complex Class Descriptions: Existential Restrictions

Classes can also be defined as a property restriction:

eg (A Sensor is something that) implements **some** Thing

**owl:someValuesFrom** prop ClassA

the class comprised of individuals for which there is at least one prop-value that is a type of ClassA



Also called  
*existential  
restriction*

Figure A.2: A Schematic Of An Existential Restriction (prop some ClassA)

From Horridge, "A Practical  
Guide.." Copyright  
University of Manchester  
2011

# Complex Class Descriptions: Universal Restrictions

Classes can also be defined as a property restriction:  
e.g. (A Sensor is something that) detects **only** Stimulus

**owl:allValuesFrom** prop ClassA

- the class of individuals for which all the prop-values are a type of ClassA

Also called  
*universal  
restriction*

Beware: also allows  
individuals with no prop-  
values at all.

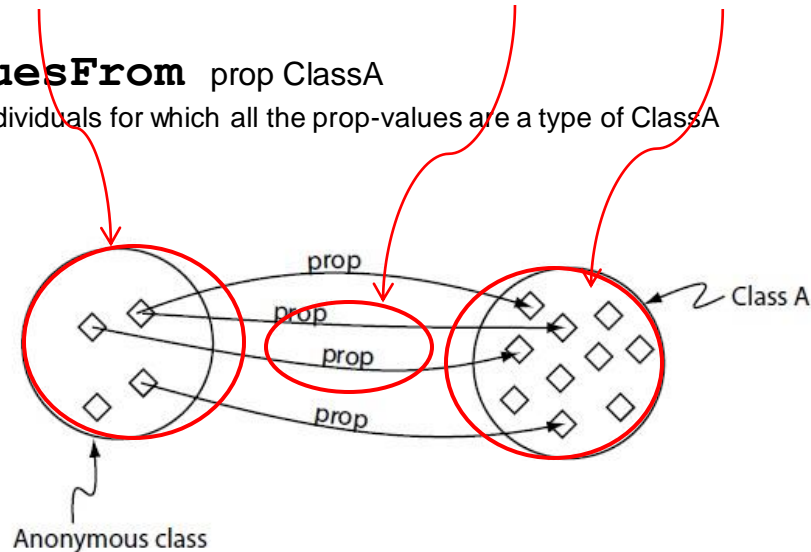


Figure A.3: A Schematic View Of The Universal Restriction, **prop only ClassA**



# Exercise: Restriction

- Define the *ANUSensors* to be *is\_owned\_by* only an *Agent*.
  - Click on the `SubClassOf+` in the `Description` pane and choose `Object Restriction Creator` tab
  - Navigate to *is\_owned\_by* on the left and *foaf:Agent* on the right.
  - Choose `Only(universal)` from the `Restriction type` pulldown.
  - Click `ok`
- Check with the reasoner
- Move those 2 individuals earlier to be *ANUSensors* (lots of ways to do this) e.g. by `Instances +` on `Description` pane for *ANUSensor*
- Check with reasoner
- Save the ontology

# More Complex Class Descriptions

- There are some other property restrictions:

**owl:minqualifiedCardinality**

**owl:maxqualifiedCardinality**

**owl:qualifiedCardinality**  $P\ C$

- the class of individuals for which there are (respectively)  $\geq$ ,  $\leq$ ,  $=$   $P$ -values of type  $C$

**owl:hasValue**  $P\ I$

- The class of individuals for which the only  $P$ -value is the individual  $I$

- A class description can be also be

**owl:oneOf**  $I, J, K, \dots$

- the class comprised exactly of the named individuals

**owl:hasSelf**  $P$

- The class of individuals for which the property is reflexive

# Exercise: Cardinality and the OWA

- Change previous restriction on *ANUSensor* to require it *is\_owned\_by* exactly one *foaf:Agent* (use *object restriction creator* by clicking on the circle edit button for the restriction).
- Go to *Individuals by class* tab and create an individual *anu* which is a *foaf:Agent*
- Create an individual *csiro* which is a type of *foaf:Agent*
- Assert *my1sensor* to be *is\_owned\_by anu* using
  - Property assertions pane -> Object Property assertions +
- Check with reasoner.
- Assert *my1sensor* to be *is\_owned\_by csiro* (as well as *anu*).
- Check with reasoner. What is going on? Why has the cardinality restriction been ok?
  - Hint: look at what the reasoner has inferred about *anu* on the Description pane for Entities->Individuals
- Assert *anu* and *csiro* are different. Check with the reasoner
- Retract that *my1sensor is\_owned\_by csiro*.
- Check with the reasoner. Save the ontology.

# Other bits

- Can declare classes to be mutually disjoint: `owl:AllDisjointClasses`
- Can declare a key for a class (the tuple of named property values uniquely defines the individual class member): `owl:hasKey`
- The datatype properties have many of the same features as object properties, but in some places use a different `owl:` name,
  - `owl:onDatatype` (construct named members), `owl:datatypeComplementOf` (construct members that are not the datatype,)
  - `owl:withRestrictions` (construct a range of values within a datatype)
- There are more keywords too – but we have seen most of them and there is some redundancy due to backwards compatibility

# Reasoning

- A logical inference reasoner can be applied to check for:
  - **GLOBAL CONSISTENCY** (ie satisfiability). Can there ever be a set of individuals over which all the axioms hold true?
    - This is the basic inference– all for all the others you just fiddle the ontology a bit and ask about global consistency
  - **SUBSUMPTION**: Is some class a subclass of some other? Computing all the subclass relations is called CLASSIFYING.
  - **EQUIVALENCE**: Is some class necessarily the same as some other?
  - **DISJOINTNESS**: Can a pair of classes never have a common instance?
  - **CLASS CONSISTENCY**: Can a class ever have an instance? (i.e it is not subsumed by owl:Nothing)
  - **INSTANCE CHECKING**: Does some individual belong to some class?
  - **INSTANCE RETRIEVAL**: What are the individuals belonging to some class?
- SPARQL does none of these!

# Summary

- OWL is a very expressive and machine-interpretable modelling language
- We have covered most of OWL 2 and most of Protege
- Protege is not quite complete for OWL DL (but I don't think you can do anything that is *\*not\** OWL DL)
- Protégé is great for ontology design but not so great for individuals (data).
- OWL distinguishes types: individuals, classes, object properties, data properties, datatypes and the axioms you can write about them. RDF modellers beware! Often you can “pun”...
- Don't ‘think’ UML, think *sets* or *implication*
- Be careful with the OWA: will surprise you with cardinality, negation, only restrictions if you are a relational modeller
- Be careful with the no\_UNA: it will surprise you whoever you are!