Wiki:

**Ontology (IS)** – formal naming & definitions of types, properties, and relations of entities that exist in a particular domain.

**RDFs** – used for describing information on web resources (also used for knowledge management applications) [RDF is similar to entity-relationship or class diagrams]. RDFs make statements about resources in the form of *subject-predicate-object (triples)* expression. (subject=resource; predicate=relationship|trait; object=object)

**OWL** – knowledge (ontologies) representation language. OWL operates on RDFs.

**Protégé** – semantic editor that uses OWL. Also uses semantic reasoner.

**RDF Schema** – language for RDF vocabulary sharing.

3 variants of OWL (different levels of expressiveness): [ordered by increasing expressiveness]

1. **OWL Lite** – for those, who need classification hierarchy & simple constraints (cardinality constraints {0, 1}).
2. **OWL DL** – for those, who need maximum expressiveness, while having computational completeness (guarantee to compute conclusions) & decidability (all computations will finish in finite time). Includes all language constructs, but has some restrictions (i.e. class may be subclass of many classes, a class cannot be an instance of another class)
3. **OWL Full** – for those, who want maximum expressiveness & syntactic freedom (with no computational guarantees). In OWL Full, for example, a class can be treated as a collection of individuals and as an individual on its own at the same time.

**OWL Reasoner** – helps to check ontology as it is being built
Lecture 1:

Course Content (ab)

- Semantic Web Technologies:
  - Intro to Sem Web and Ontologies,
  - RDF, SPARQL
  - OWL,
  - Reasoning,
  - Protégé,
  - Big Data vs Semantic Web.

What is the Semantic Web?

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."


Berners-Lee:
- semantic: machine processable
- web: navigable space
Semantic Web Layer Cake

- each layer builds on top of the previous (lower) layer;
- XML (eXtensible Markup Language): much like HTML, but you can define your own tags;
- RDF: basic data model; RDF Schema: defining RDF vocabularies;
- OWL (Web Ontology Language): a more powerful ontology modelling language;
- Logic layer: application-specific declarative knowledge; rules that enable inferences;
- Proof layer: to allow the explanation of given answers by automated software agents;
- Trust Layer: use digital signatures to authenticate services and users; use encryption for transfer of sensitive data, etc.;

Semantic Web Technologies: RDF and RDF Schema

- RDF: Resource Description Framework;
- Resources, Properties, Statements;
- Example: John Smith created Document 1;

```
<Document 1>  
	<DC:Creator> "John Smith" </DC:Creator>
</Document 1>

<rdf:RDF>
  <rdf:Description xmlns:rdf="http://www.w3.org/RDF/" prefix="RDF"/>
  <rdf:Description xmlns="http://purl.oclc.org/DC/" prefix="DC"/>

  <RDF:DC/>
  <RDF:Description rdf:resource="http://uri-of-Document-1">  
    <DC:Creator> <rdf:RDF xmlns:rdf="http://www.w3.org/RDF/" prefix="RDF"/>
  </DC:Creator>
</RDF:Description>
</RDF:RDF>
```

- RDF Schema: lets users describe resources using their own vocabularies;
Semantic Web Technologies: OWL

- OWL: Web Ontology Language;
- expressivity of RDF and RDF Schema too limited;
- need additional modelling primitives, e.g. SameIndividual(\{e, e\}), EquivalentClasses(\{c, c\}), DisjointClasses(\{c, c\});
- Three levels of OWL:
  - OWL Lite;
  - OWL DL;
  - OWL Full;
- Three OWL-2 profiles:
  - RDF Schema ⊆ OWL Lite ⊆ OWL DL ⊆ OWL Full;
  - tradeoff between expressiveness and computational complexity;
- OWL has already significant uptake in bioinformatics, e.g. the GO ontologies can be downloaded in OWL format;

Semantic Web Stack
Example:

SeaLife Main Components

- Ontologies:
  - background knowledge for browser;

- Text Mining and Concept Mapping:
  - bridging gap between free text on current web and ontology-based mark-up for Semantic Web;
  - developing automated mark-up for free text, based on text-mining and natural language processing techniques;
1 – User load web page
2 – Browser uses concept mapper to identify ontological concepts and highlight them
3 – User selects concept and adds it to the shopping cart
4 – Task Composition Manager determines available (combinations of) services based on the shopping cart
5 – User selects services of interest
6 – Task Composition Manager executes services and displays results in the browser
7 – Browser displays results to the user
What are Ontologies?

“An ontology is an explicit and formal specification of a conceptualisation.”

(by T.R. Gruber, R. Studer)

“... an ontology is a formal explicit description of concepts in a domain of discourse (classes (sometimes called concepts)), properties of each concept describing various features and attributes of the concept (slots (sometimes called roles or properties)), and restrictions on slots (facets (sometimes called role restrictions)). An ontology together with a set of individual instances of classes constitutes a knowledge base. In reality, there is a fine line where the ontology ends and the knowledge base begins. ...”


Reasons for Ontologies

• To share common understanding of the structure of information among people or software agents
• To enable reuse of domain knowledge
• To make domain assumptions explicit
• To separate domain knowledge from the operational knowledge
• To analyze domain knowledge

In practical terms, developing an ontology includes:
• defining classes in the ontology,
• arranging the classes in a taxonomic (subclass–superclass) hierarchy,
• defining slots and describing allowed values for these slots,
• filling in the values for slots for instances.

Lecture 2 (RDF & SPARQL):

- Resource Description Framework;
- making statements (provide information) about resources on the web;
- representation in the form of triplets: [Subject, Predicate, Object]
- uses URIs (Unique Resource Identifiers) to identify resources;
RDF Example

John Smith created Document 1;

![Diagram showing RDF example](image)

```xml
<?xml:namespace ns = "http://www.w3.org/RDF/RDF/" prefix ="RDF" ?>
<?xml:namespace ns = "http://purl.oclc.org/DC/" prefix = "DC" ?>

<RDF:RDF>
  <RDF:Description RDF:HREF = "http://uri-of-Document-1">
    <DC:Creator>John Smith</DC:Creator>
  </RDF:Description>
</RDF:RDF>
```

Namespaces, URIs and Identity

- Each resource must have a globally unique identity -> URIs;
- Triples can be merged if they share resources: triples -> graphs;
- Writing out long URIs can be tedious -> namespaces;
SPARQL overview:

- Protocol:
  - for remote invocation of SPARQL queries from a client to a query processing service;
  - uses WSDL 2.0;

- Query Language:
  - SQL-like;
  - for querying RDF graphs via pattern matching;

---

**Triple Patterns**

- RDF triples -> SPARQL triple patterns;
- Subject predicate object. (Note "." at the end of the triple.)
- <http://www.daml.org/2003/01/periodictable/PeriodicTable#Na> table:name "sodium".

- URIs in single angle brackets <...> property in qname style literal string in double or single quotes

- Variables begin either with ‘$’ or ‘?’;
- Subject predicate object. (Note "." at the end of the triple.)
  - matches the triple above;
  - binds <http ... #Na> to ?element and "sodium" to ?name;
- ?element ?predicate ?object. matches all triples in an RDF graph;
- Combinations of triple patterns form graph patterns;

---

SPARQL query example:

```sparql
PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE { ?element table:name ?name. }
```
Graph Patterns

PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name ?symbol ?number
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE
{
  ?element table:name ?name.
  ?element table:atomicNumber ?number.
}

- multiple triple patterns;
- all patterns must be matched for a RDF triple to be selected;
- no variable can be selected unless it appears in the graph pattern; (Note: this is different from SQL!)

Graph Pattern Shortcut

PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT *
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE
{
  ?element table:name ?name;
  table:symbol ?symbol;
  table:atomicNumber ?number.
}

- '*' used to select all variables from the graph patterns;
- predicate-object lists:
  - share subject over multiple patterns;
  - ':' instead of '.'
Optional Patterns

PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name ?symbol ?number ?color
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE
{
  ?element table:name ?name.
  ?element table:atomicNumber ?number.
  OPTIONAL { ?element table:color ?color. }
}

- in our example, not all elements have a 'color', e.g. ununseptium;
- without using OPTIONAL, an element without the color predicate would not match the pattern;
- with using OPTIONAL, the triple pattern must be matched if the element has the color predicate;

Union

PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?symbol ?number
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
WHERE
{
  {
    ?element table:symbol ?symbol;
    table:atomicNumber ?number;
    table:group table:group 17.
  }
  UNION
  {
    ?element table:symbol ?symbol;
    table:atomicNumber ?number;
    table:group table:group 18.
  }
}

- UNION allows the selection of alternatives (logic-OR);
- any number of unions can be used;
**ORDER BY**

PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name ?number
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE
{
  ?element table:name ?name;
  table:atomicNumber ?number;
  table:group table:group_18.
}
ORDER BY ?number

- ORDER BY is a *solution sequence modifier*;
- orders the results;
- can list one or more variables;
- by default sorts in ascending order;
- use DESC to sort in descending order:
  ORDER BY DESC (?number)

---

**LIMIT and OFFSET**

PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE
{
  ?element table:name ?name;
  table:atomicWeight ?weight.
}
ORDER BY DESC (?weight)
LIMIT 10
OFFSET 10

- LIMIT sets the maximum number of rows returned;
- OFFSET lets us skip a specified number of result rows;
- usually combined with ORDERED BY;
More complex RDF modelling

How can I model that fact that "Frank created that statement about the location of the BaronWay Building"?

- Solutions:
  - Reification;
  - Named Graphs;

Reification

Introduce an auxiliary statement that relates to each part of the original statement:

Not a very elegant solution -> introduction of named graphs.
Named Graph

An explicit identifier is given to a statement or a set of statements.

Richer Predicates

- RDF only offers binary predicates.
- In general predicates may have more than two arguments.
- broker (X, Y, Z).
  
  X is a broker in a home sale between seller Y and purchaser Z.

Solution:
- broker(home-sale, X)
- seller(home-sale, Y)
- purchaser(home-sale, Z)
RDF System Architecture

- Application
  - SPARQL
  - Inference and Query Engine
    - OWLIM
    - RDF Store
      - Parser/Serializer
        - RDF files
          - N-Triples
          - Turtle
          - RDF/XML

RDF Schema

- RDF is general purpose;
- RDF Schema is a semantic extension of RDF to provide a means to describe domain specific vocabularies;
- Allows us to talk about:
  - classes of resources;
  - properties describing resources;
- Does not contain domain specific terms, such as ‘Gene’, ‘Tissue’ or ‘Vehicle’, ‘Bus’;
RDF Schema Classes and Properties

Classes
- rdfs:Resource
- rdfs:Class
- rdfs:Literal
- rdfs:Datatype
- rdf:XMLLiteral
- rdf:Property

Properties
- rdfs:range
- rdfs:domain
- rdf:type
- rdfs:subClassOf
- rdfs:subPropertyOf
- rdfs:label
- rdfs:comment

Note that 'subClassOf' and 'subPropertyOf' provide the mechanism for creating hierarchies of classes and properties;

List of classes and properties on the left is not complete. For more details see: http://www.w3.org/TR/rdf-schema
Lecture 3 (OWL & Protege):

- expressivity of RDF and RDF Schema too limited;
- need additional modelling primitives, e.g. SameIndividual(e₁, e₂), EquivalentClasses(c₁, c₂), DisjointClasses(c₁, c₂);
- Three levels of OWL:
  - OWL Lite;
  - OWL DL;
  - OWL Full;
- RDF Schema ⊆ OWL Lite ⊆ OWL DL ⊆ OWL Full;
- tradeoff between expressiveness and computational complexity;
- RDF(S) still not semantically rich enough for web;
- OWL contains additional semantics for:
  - local scope of properties;
  - disjointness of classes;
  - boolean combination of classes;
  - cardinality restrictions;
  - special characteristics of properties, e.g. unique, transitive and inverse;

---

**OWL Ontology Components**

- Classes (e.g. professor, student);
- Properties (e.g. isTaughtBy, supervises);
- Instances of classes - individuals (e.g. Bill Hill, Yiya Yang);
### OWL Lite Language Constructs

**RDF Schema Features:**
- `Class` (Thing, Nothing)
- `rdfs:subClassOf`
- `rdfs:Property`
- `rdfs:subPropertyOf`
- `rdfs:domain`
- `rdfs:range`
- `Individual`

**Inequality:**
- `equivalentClass`
- `equivalentProperty`
- `sameAs`
- `differentFrom`
- `AllDifferent`
- `distinctMembers`

**Property Characteristics:**
- `ObjectProperty`
- `DatatypeProperty`
- `inverseOf`
- `transitiveProperty`
- `SymmetricProperty`
- `FunctionalProperty`
- `inverseFunctionalProperty`

**Property Restrictions:**
- `Restriction`
- `onProperty`
- `allValuesFrom`
- `someValuesFrom`

**Restricted Cardinality:**
- `minCardinality (only 0 or 1)`
- `maxCardinality (only 0 or 1)`
- `cardinality (only 0 or 1)`

**Datatypes:**
- `xsd:datatype`

**Class Intersection:**
- `intersectionOf`

**Versioning:**
- `versionInfo`
- `preVersion`
- `backwardsCompatibleWith`
- `incompatibleWith`
- `DeprecatedClass`
- `DeprecatedProperty`

**Header Information:**
- `Ontology`
- `imports`

**Annotation Properties:**
- `rdfs:label`
- `rdfs:comment`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`
- `AnnotationProperty`
- `OntologyProperty`

(from [http://www.w3.org/TR/owl-features/](http://www.w3.org/TR/owl-features/))

### Additional OWL DL/Full Language Constructs

**Class Axioms:**
- `oneOf`, `dataRange`
- `disjointWith`
- `equivalentClass` (applied to class expressions)
- `rdfs:subClassOf` (applied to class expressions)

**Boolean Combinations of Class Expressions:**
- `unionOf`
- `complementOf`
- `intersectionOf`

**Arbitrary Cardinality:**
- `minCardinality`
- `maxCardinality`
- `cardinality`

**Filler Information:**
- `hasValue`

(from [http://www.w3.org/TR/owl-features/](http://www.w3.org/TR/owl-features/))
OWL Classes

- **Named Classes**: defined by “user”;

- **Complex Classes**: defined in terms of other classes, e.g.:
  - intersection classes (e.g. Human AND Male);
  - union classes (e.g. Staff OR Student);
  - complement classes (e.g. Staff AND NOT Student);

- **Class hierarchies** may be created by making a class a subclass of another class (using `subClassOf`), e.g. `subClassOf(Car,Vehicle)`;

OWL Properties

- **Object Properties**: relates objects to other objects;
  - e.g. supervises, taughtBy;

- **Data Type Properties**: relates objects to data type values;
  - e.g. age, phone number, title;
  - OWL has no predefined data types; allows use of XML Schema data types;

- **Property domains and ranges** can be specified;

- A property may have an inverse property;
  - e.g. isTaughtBy vs teaches;

- **Property hierarchies** may be created by making a property a subproperty of another property (using `subPropertyOf`), e.g. `subPropertyOf(hasSibling, hasRelative);`
Property Restrictions

- Property restrictions can be used to describe a class of individuals that satisfy certain conditions;

- **owl:Restriction:**
  - **owl:onProperty:** indicates the restricted property;
  - forms of restriction:
    - **owl:allValuesFrom:** universal quantification (\(\forall\));
    - **owl:someValuesFrom:** existential quantification (\(\exists\));
    - **owl:hasValue:** specifies classes based on the existence of particular property value;
    - **owl:cardinality:** a min and a max value can be specified on a numeric interval;

---

Some examples (from http://www.w3.org/TR/owl-guide/#PropertyRestrictions)

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

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

"The first does not require a wine to have a maker. If it does have one or more, they must all be wineries. The second requires that there be at least one maker that is a winery, but there may be makers that are not wineries." (OWL Guide)

<table>
<thead>
<tr>
<th>Relation</th>
<th>Implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>allValuesFrom</td>
<td>For all wines, if they have makers, all the makers are wineries.</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>For all wines, they have at least one maker that is a winery.</td>
</tr>
</tbody>
</table>
Property Characteristics:

- Further description of properties; useful for enhanced reasoning;
- owl:TransitiveProperty
  - $P(x,y)$ and $P(y,z)$ implies $P(x,z)$
  - e.g. "has better grade than", "is taller than";
- owl:SymmetricProperty
  - $P(x,y)$ iff $P(y,x)$
  - e.g. "has same grade as”, "is sibling of”;
- owl:FunctionalProperty
  - $P(x,y)$ and $P(x,z)$ implies $y = z$
  - e.g. "age", "height", "directSupervisor”;
- owl:InverseFunctionalProperty
  - $P(y,x)$ and $P(z,x)$ implies $y = z$
  - e.g. "isTheSocialSecurityNumberFor”;

Data Properties

- Object properties describe relationships between individuals;
- Datatype properties link individuals and XML Schema Datatype values or rdf literal;
- E.g.: size of pizza, number of hours employed, age of person;
Cardinality Restrictions

- To specify a minimum, maximum and exact number of relationships between individuals and datatype values;
- E.g. a full-time student studies at least 3 courses, a motorbike has exactly one engine;

hasValue Restriction

- To describe the relationship to a specific individual;
- E.g.: a database student studies a database course; Albert’s bicycle is owned by Albert;
Lecture 4 (OWL 2):

**OWL 2 Summary**

- Extension of OWL 1: every OWL 1 ontology is a valid OWL 2 ontology;
- New features for increased expressive power;
- Extended Datatypes;
- Metamodelling and Annotations;
- Syntactic shortcuts: e.g. disjoint unions;
- Alternative Syntax;
- Profiles;

**New constructs for Properties**

- Self Restriction;
  - gene regulates itself;
- Property Qualified Cardinality Restrictions;
  - part which has two parts that have gene G expressed;
- Reflexive, Irreflexive and Asymmetric Object Properties;
  - p cannot be a proper part of p (irreflexive);
- Disjoint Properties;
  - parent of vs child of;
- Property Chain Inclusion;
  - anything located in a part is located in the whole;
  - the brother of your parent is your uncle;
- Keys;
  - to uniquely identify individuals of a given class by values of a set of key properties;
Profiles

- Language subsets with certain expressive powers and computational properties:
  - **OWL 2 EL**: based on DL language EL;
  - **OWL 2 QL**: restrictions enable integration with RDBMS;
  - **OWL 2 RL**: for rule-based DBs;

- Users requiring a scalable profile for large but (rather) simple ontologies and good time performance for ontology (TBox/schema) reasoning may want to consider OWL 2 EL.

- Users requiring a profile that can easily interoperate with relational database systems, and where scalable reasoning on large datasets is the most important task may want to consider OWL 2 QL.

- Users requiring a profile that can easily interoperate with rules engines and rule extended DBMSs, and where scalable reasoning on large datasets is the most important task may want to consider OWL 2 RL.

From [http://www.w3.org/TR/owl2-overview/](http://www.w3.org/TR/owl2-overview/)
Standard DL Reasoning Services

- **Consistency checking**: Ensures that an ontology does not contain any contradictory facts. The OWL 2 Direct Semantics provides the formal definition of ontology consistency used by pellet.

- **Concept satisfiability**: Determines whether it’s possible for a class to have any instances. If a class is unsatisfiable, then defining an instance of that class will cause the whole ontology to be inconsistent.

- **Classification**: Computes the subclass relations between every named class to create the complete class hierarchy. The class hierarchy can be used to answer queries such as getting all or only the direct subclasses of a class.

- **Realization**: Finds the most specific classes that an individual belongs to; i.e., realization computes the direct types for each of the individuals. Realization can only be performed after classification since direct types are defined with respect to a class hierarchy. Using the classification hierarchy, it is also possible to get all the types for each individual.

One slide from this lecture that actually is important:

---

### Ontology Based Data Access

Ideal architecture based on a DBMS:

Pre-DBMS architecture:

In many cases, we are back at the pre-DBMS situation:

Diagrams from Diego Calvanese

Paper 1 - The MASTRO system for ontology-based data access
Not relevant...
Paper 2 – Semantic Web technologies for the big data in life sciences
Basic Elements Of OWL

- **Individuals** (instances)
- **Properties** (slots)
- **Classes** (concepts)

Common Mistakes

- Forgetting to make classes disjoint
- The mistaken use of universal rather than existential restrictions
- Open world reasoning
- Confusion about domain and range

Disjoint Classes

In situations where classes should not overlap, they must be **explicitly** made disjoint by the use of **disjoint axioms**
Restrictions

- Restrictions constrain the relationships between individuals.

- Many newcomers to OWL lean towards the use of universal (all values from) restrictions (∀).

- In general the ‘default’ type of restriction that should be used is an existential (some values from) restriction (∃).
Example

Describe a Margherita Pizza using universal restrictions:

Class(MargheritaPizza)
  Pizza
  restriction(hasTopping allValuesFrom(MozzarellaTopping))
  restriction(hasTopping allValuesFrom(TomatoTopping))

Example

Describe a Margherita Pizza using existential restrictions:

Class(MargheritaPizza)
  Pizza
  restriction(hasTopping someValuesFrom(MozzarellaTopping))
  restriction(hasTopping someValuesFrom(TomatoTopping))

Open World Reasoning

OWL uses the Open World Assumption (OWA)

Many OWL neophytes come from using closed world systems, such as databases.

Information that has hasn’t been explicitly added to a knowledge base is assumed to be ‘missing’ information, which could be added sometime in the future.
Two relations: 1. Ice cream has topping; 2. Pizza has topping. => Reasoner would think that ice cream is a part of pizza.

Solution

A Margherita Pizza
(the correct way)

Margherita Pizzas have toppings of Tomato and Mozzarella – moreover, they only have toppings of Tomato and Mozzarella.
Pizza tutorial