# ADVANCED KNOWLEDGE ENGINEERING

**Instructor:**
KHUAT Thanh Tung
University of Technology Sydney

# Subject's Outline

- **Topic 1:** An Overview of Knowledge Engineering
- **Topic 2:** An Overview of Knowledge-based Systems
- **Topic 3:** Knowledge Acquisition
- **Topic 4:** Knowledge Representation and Reasoning

**Mid-term assessment**

- **Topic 5:** Ontology
- **Topic 6:** Knowledge Graphs
- **Topic 7:** Expert Systems
- **Topic 8:** Uncertain Reasoning
- **Topic 9:** Hybrid Knowledge-based Systems
- **Topic 10:** Automated AI Planning

**Group projects for the advanced topics**

# ONTOLOGY

# Objectives of this topic

By the end of this topic, you will be able to:

- ✓ understand basic knowledge of Ontology
- ✓ explain how Ontology represent data
- ✓ understand how to query data from Ontology
- ✓ know how to apply Ontology to practical projects
- ✓ know how to use OWL language
- ✓ understand basic knowledge of Semantic Web

# The Origin of Ontology

- The term ontology derives from Greek words “ontologos”
  - “onto” means “being”
  - “logos” means “science”
- So, **ONTOLOGY** is the **SCIENCE OR STUDY OF BEING**
- Ontology aggregates to the study of anything and everything
  - It is a branch of metaphysics , the study of first principles or the root of things
- For everything, it is a part of being
- For anything, it is under the aspect of its being, of what is involved in its existing.

# The Definitions of Ontology

- In information management and knowledge sharing place, ontology can be defined as follows:
  - An **ontology** is a set of **concepts** - such as **things, events,** and **relations** that are specified in some way in order to create an agreed-upon vocabulary for exchanging information. (Tom Gruber, an AI specialist at Stanford University.)
  - An **ontology** is a **vocabulary of concepts** and **relations** rich enough to enable us to express knowledge and intention without **semantic ambiguity.**
  - **Ontology** describes **domain knowledge** and provides an agreed upon understanding of a **domain.**
  - **Ontologies:** are collections of statements written in a language such as RDF that **define the relations between concepts** and **specify logical rules for reasoning** about them.

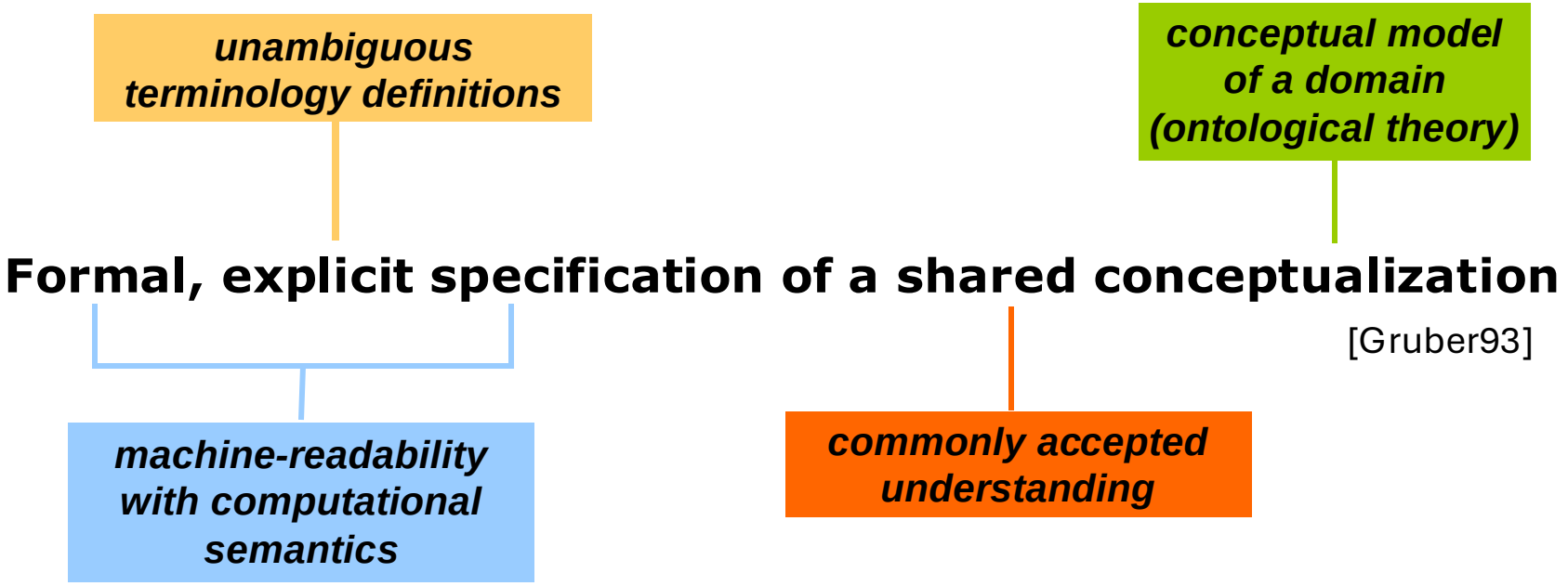
# The Definitions of Ontology

- A more formal definition is:

  "**An ontology is a formal, explicit specification of a shared conceptualization, used to help programs and humans share knowledge**" (Tom Gruber)

  - "**explicit**" means that "the type of concepts used and the constraints on their use are explicitly defined".
  - "**formal**" refers to the fact that "it should be machine readable".
  - "**shared**" refers to the fact that the knowledge represented in an ontology are agreed upon and accepted by a group".
  - "**conceptualization**" refers to an abstract model that consists the relevant concepts and the relationships that exists in a certain situation.
- The conceptualization consists of:
  - the identified concepts (**objects, events, beliefs,** etc)
    - E.g. **Concepts:** *disease, symptoms, treatments*
  - the conceptual relationships that are assumed to exist and to be relevant.
    - E.g. **Relationships:** "disease causes symptoms", "therapy treats disease"

# The Definitions of Ontology



# The World without Ontology = Ambiguity

- Example 1:

**Cook?**

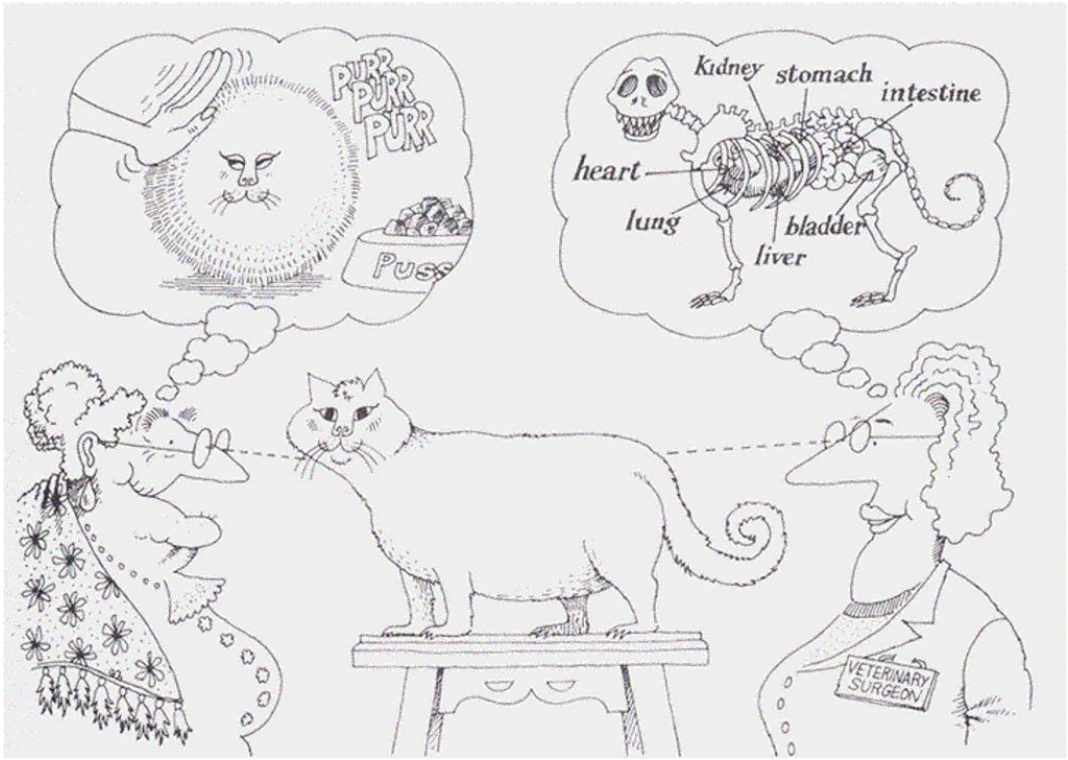
You mean:

- chef
- information about how to cook something,
- or simply a place, person, business or some other entity with "cook" in its name.

- The problem is that the word "**cook**" has no meaning, or semantic content, to the computer.

# The World without Ontology = Ambiguity

- Example 2:




**Ambiguity for humans** **Cat**

The Vet and Grandma associate different view for the concept cat

# Why using the Ontology

- The reason for ontologies becoming so important is that currently we lack standards (shared knowledge) which are rich in semantics and represented in machine understandable form. (*Ying Ding, Ontoweb*)
- Ontologies have been proposed to solve the problems that arise from using different terminology to refer to the same concept or using the same term to refer to different concepts. (*Howard Beck* and *Helena Sofia Pinto*)

# Why using the Ontology

- **Inability to use the abundant information resources on the web**
  - The WEB has tremendous collection of useful information however getting information from the web is difficult.
  - Search engines are restricted to simple keyword-based techniques. Interpretation of information contained in web documents is left to the human user.
- **Difficulty in Information Integration**
  - The integration of data from various sources is a challenging task because of synonyms and homonyms.
- **Problem in Knowledge Management**
  - Multi-actor scenario involved in distributed information production and management.
  - "People as well as machines can't share knowledge if they do not speak a common language" (*T. Davenport*)

Ontologies provide the required **conceptualizations** and **knowledge representation** to meet these challenges

# Benefits of Ontology

- To facilitate communications among people and organizations
  - aid to human communication and shared understanding by specifying meaning
- To facilitate communications among systems without semantic ambiguity, i.e., to achieve inter-operability.
- To provide foundations to build other ontologies (reuse)
- To save time and effort in building similar knowledge systems (sharing)
- To make domain assumptions explicit
  - Ontological analysis
    - clarifies the structure of knowledge
    - allow domain knowledge to be explicitly defined and described.

# Structure of Ontology

- Most ontologies are structured as taxonomies or hierarchies.
- Basic ontology has two classes of elements: the **entities** and the **relationships** between them.
  - Organized according to axioms or rules that control *how the world will be defined*.

# Ontology Example

## Concept

conceptual entity of the domain

## Attribute

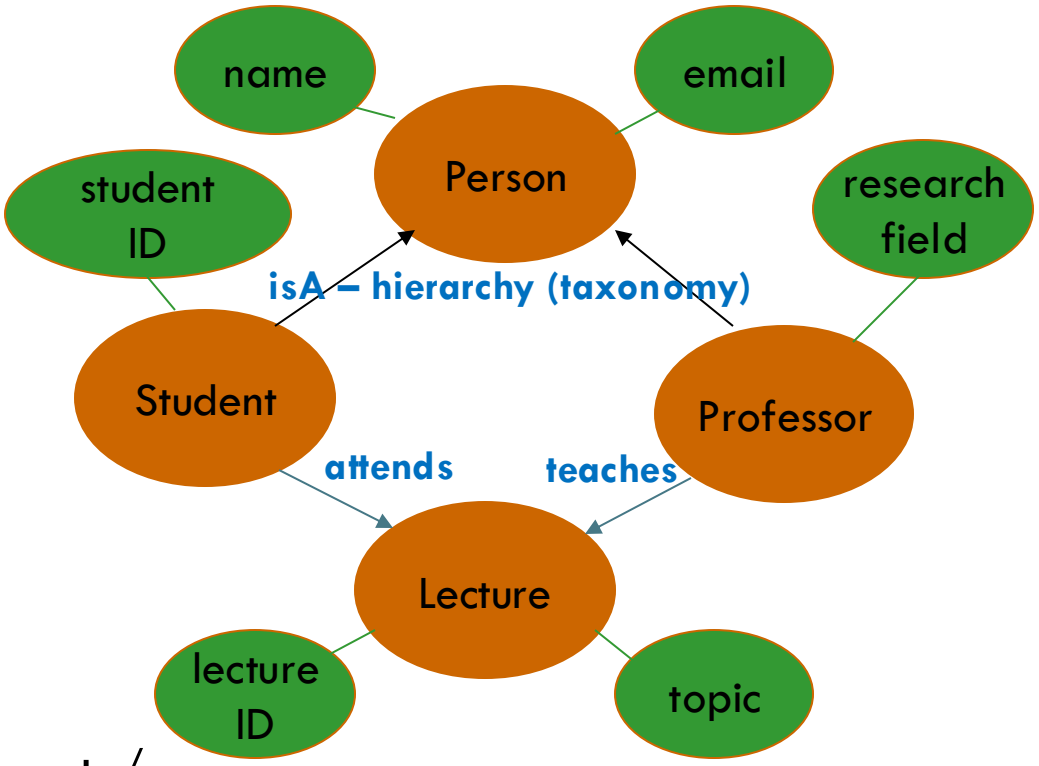
property of a concept

## Relation

relationship between concepts or properties

## Axiom (Tiên đề)

coherent description between Concepts/ Properties / Relations via **logical expressions**




teaches(Professor, Lecture) ⇒ Lecture.topic ∈ Professor.researchField

# An example of Biomedical Ontologies

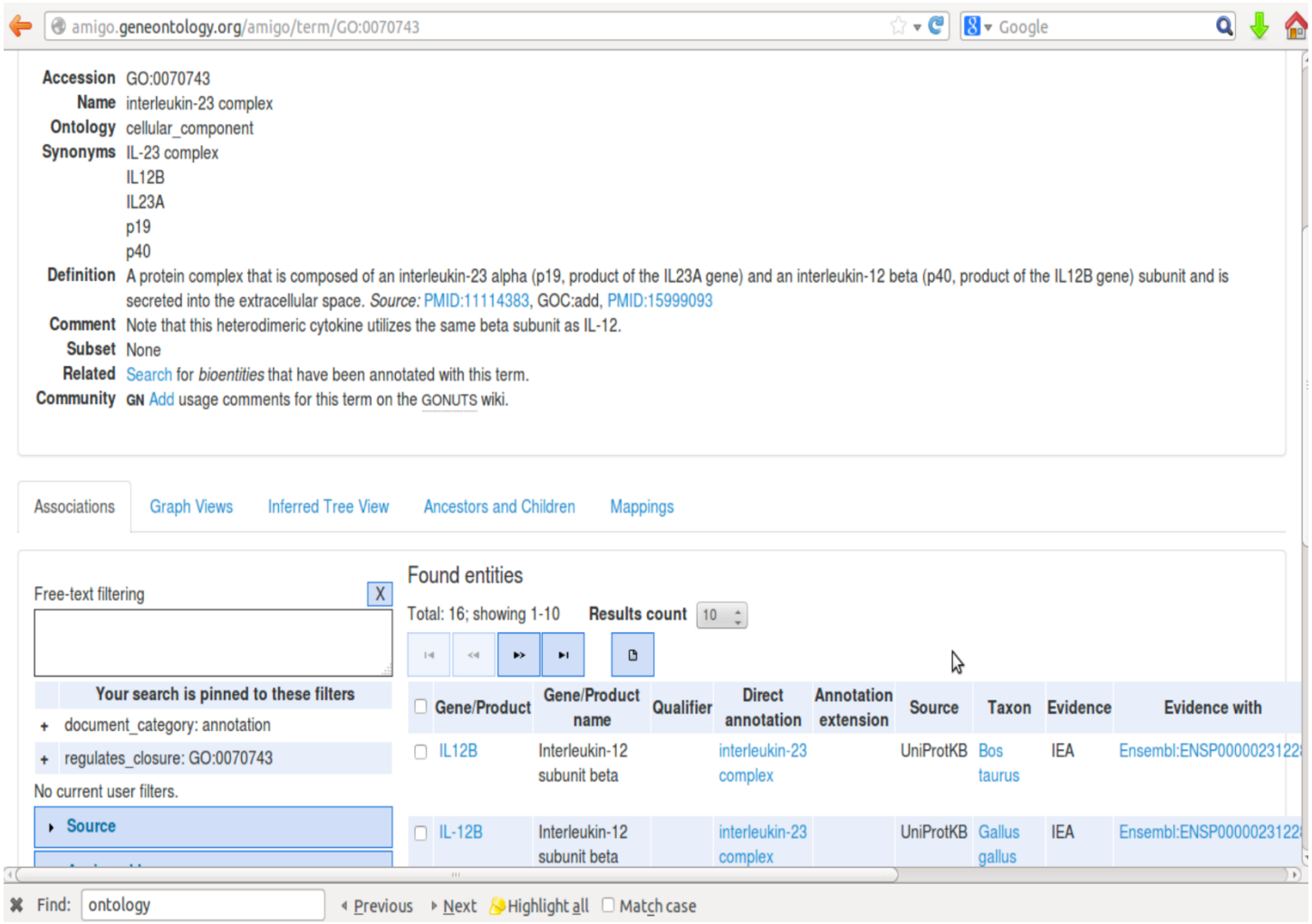
- GO--Gene Ontology
  - https://geneontology.org/docs/ontology-documentation/
- OBO--Open Biomedical Ontology.
  - umbrella web address for well-structured controlled vocabularies for shared use across different biological and medical domains.
  - http://obo.sourceforge.net/
- FMA--Foundational Model of Anatomy
  - http://sig.biostr.washington.edu/projects/fm/AboutFM.html

# Applications of Ontologies

- Gene Ontology (GO)
  - GO, is a major bioinformatics initiative to unify the representation of gene and gene product attributes across all species.
  - GO is part of a larger classification effort, the Open Biomedical Ontologies (OBO).
  - **What can scientists do with GO?**
    - Access gene product functional information
    - Find how much of a proteome is involved in a process/ function/component in the cell.
    - Map GO terms and incorporate manual annotations into own databases.
    - Provide a link between biological knowledge and gene expression profiles or proteomics data.

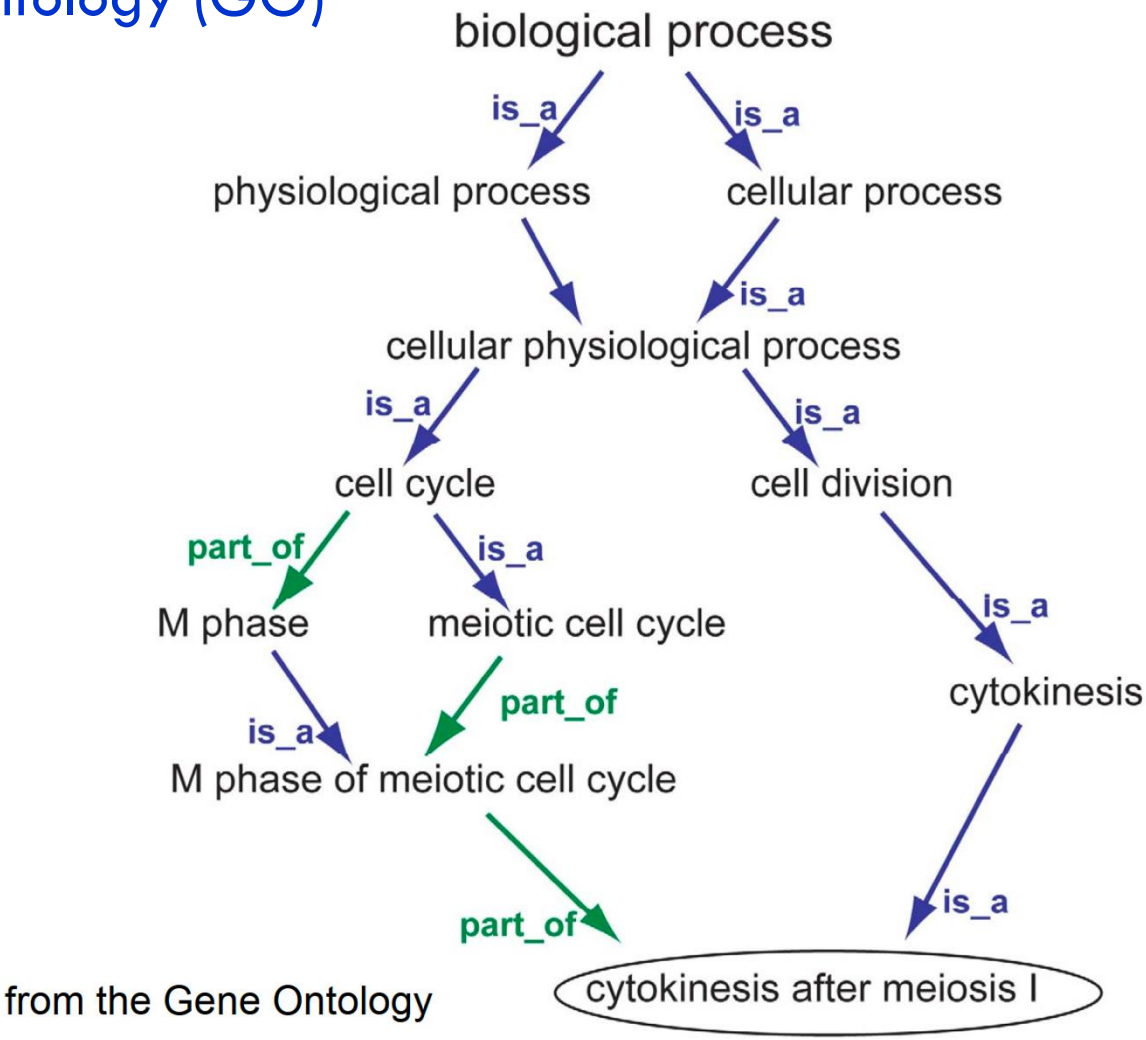
# Applications of Ontologies

- Gene Ontology (GO)



# Applications of Ontologies

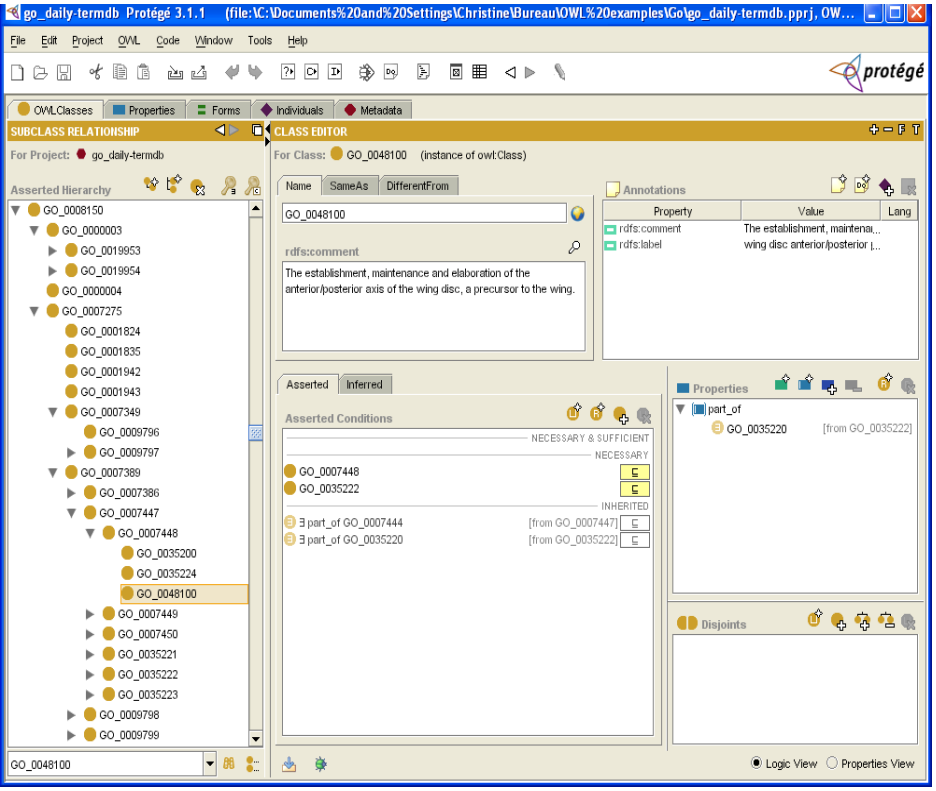
- Gene Ontology (GO)

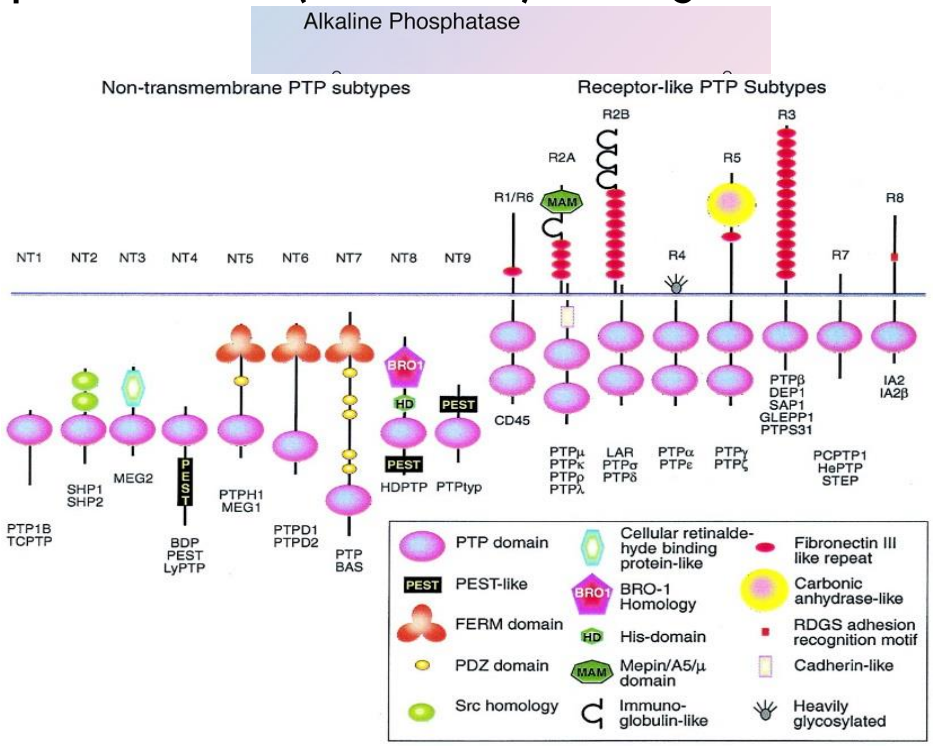



from the Gene Ontology

# Applications of Ontologies

- e-Science, e.g., Bioinformatics.
  - Open Biomedical Ontologies Consortium (GO, MGED)
  - Used for "in silico" investigations relating theory and data
    - E.g., relating data on phosphatases to (model of) biological knowledge

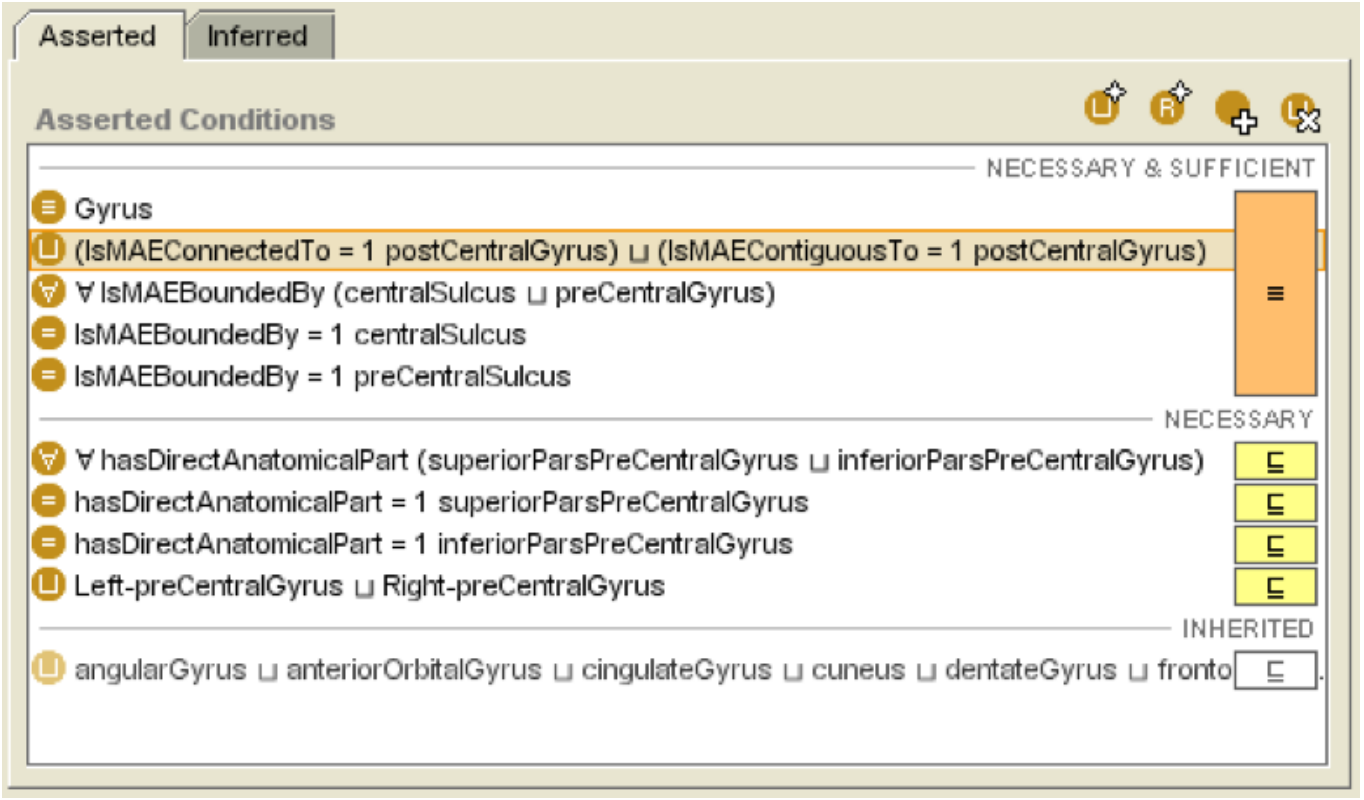



Schematic representation of PTP family of enzymes.

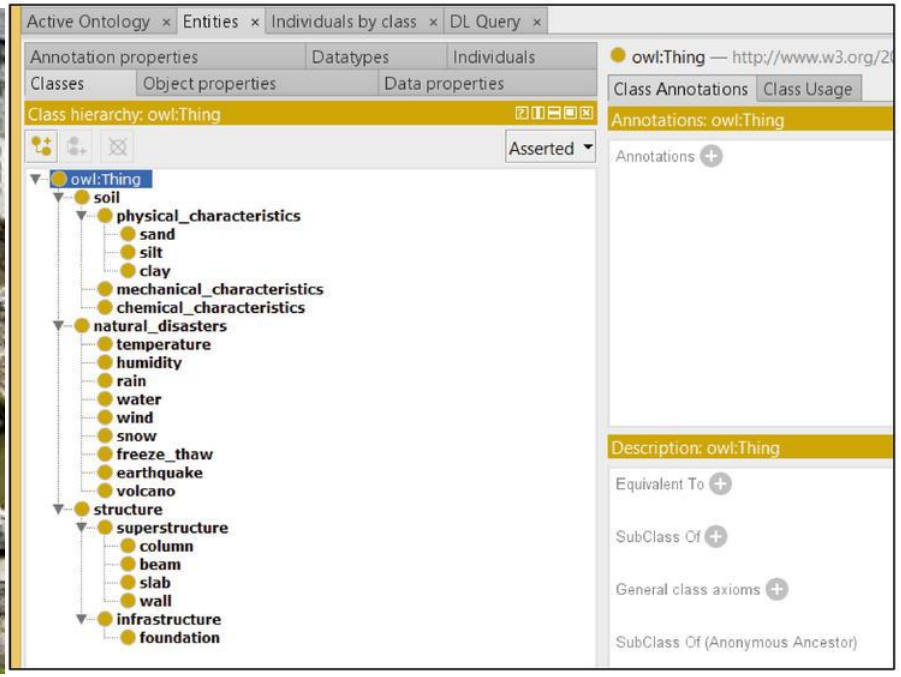
# Applications of Ontologies

- Medicine
  - Building/maintaining terminologies such as Snomed, NCI & Galen



# Applications of Ontologies

- Organising complex and semi-structured information
  - UN-FAO, NASA, Ordnance Survey, General Motors, Lockheed Martin, …



# Applications of Ontologies

- Military/Government
  - DARPA, NSA, NIST, SAIC, MoD, Department of Homeland Security, …
- The Semantic Web and so-called Semantic Grid


# Applications of Ontologies

- System Engineering
  - Specification
    - The shared understanding can assist the process of identifying requirements and defining a specification for an IT system
  - Reliability
    - Informal ontologies can improve the reliability of software systems by serving as a basis for manual checking of the design against the specification.
    - Using formal ontologies enables the use of semi-automated consistency checking of the software system with respect to the declarative specification
  - Reusability
    - To be effective, ontologies must also support reusability so that we can import and export modules among different software systems.
    - Ontologies provide an easy to reuse library of class objects for modeling problems and domains.

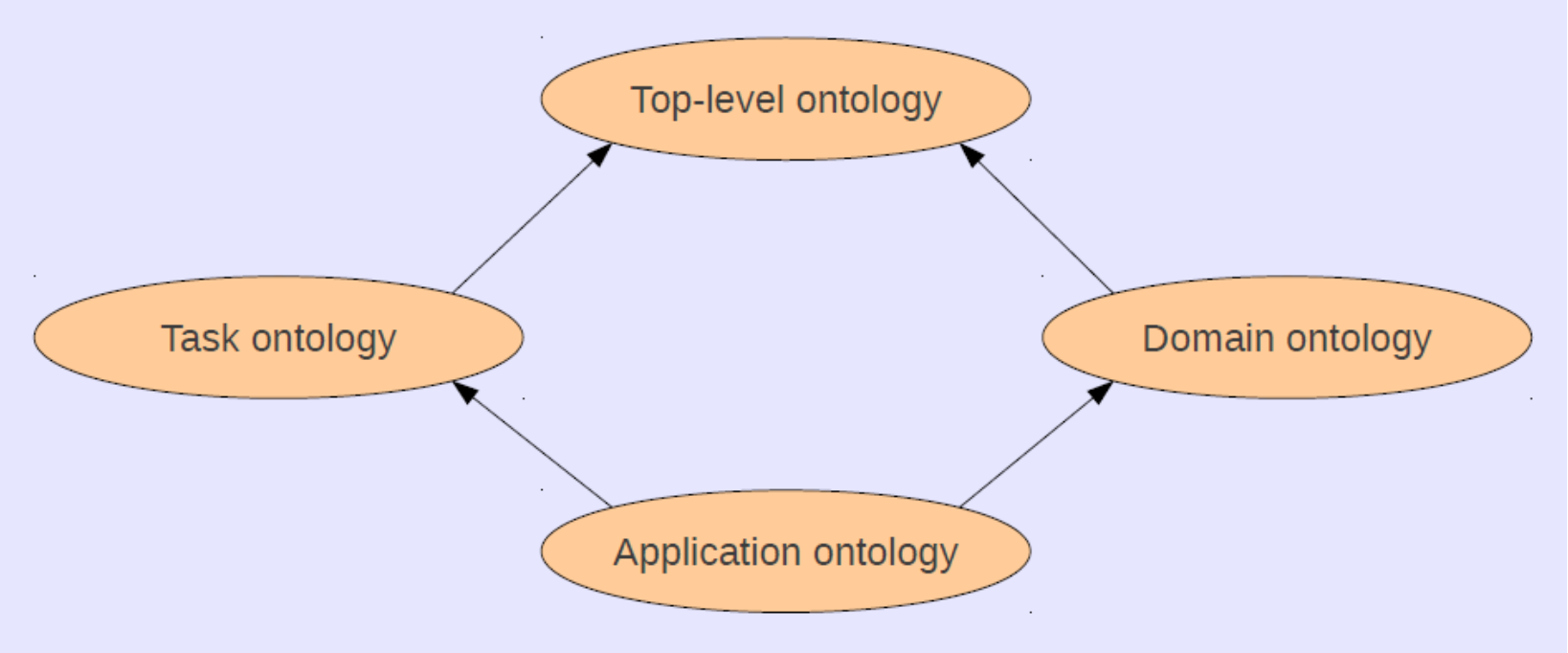
# Application Areas of Ontologies

- Information Retrieval
  - As a tool for intelligent search through inference mechanism instead of keyword matching.
  - Easy retrievability of information without using complicated Boolean logic.
  - Cross Language Information Retrieval.
  - Improve recall by query expansion through the synonymy relations.
  - Improve precision through Word Sense Disambiguation (identification of the relevant meaning of a word in a given context among all its possible meanings).
- Digital Libraries
  - Building dynamical catalogues from machine readable meta-data
  - Automatic indexing and annotation of web pages or documents with meaning.
  - To give context-based organisation (semantic clustering) of information resources.
  - Site organization and navigational support.

# Application Areas of Ontologies

- Information Integration
  - Seamless integration of information from different websites and databases.
- Knowledge Engineering and Management
  - As a knowledge management tools for selective semantic access (meaning oriented access)
  - Guided discovery of knowledge.
- Natural Language Processing
  - Better machine translation
  - Queries using natural language.

# Types of Ontologies




Classification of ontologies according to the object of conceptualization

# Types of Ontologies

- Top level ontology
  - this type of ontology describes very general concepts or common sense knowledge such as space, time, event, action,...
  - These concepts are independent of a problem or a particular area.
- Domain ontology
  - this ontology governs a set of vocabularies and concepts describing an application domain or the target world. It characterizes the knowledge of the area where the task is performed. Most existing ontologies are domain ontologies.
- Task ontology
  - this type of ontology is used to conceptualize specific tasks in systems. It governs a set of vocabularies and concepts describing a structure of performing the tasks domain-independent.
- Application ontology
  - this ontology is the most specific. The concepts in the application ontology are very domain specific and particular application. In other words, the concepts often correspond to roles played by domain entities while performing a certain activity

# Complexity of Ontologies

- **Depending on the wide range of tasks to which the ontologies are put, ontologies can vary in their complexity.**
- Ontologies range from simple taxonomies to highly tangled networks including constraints associated with concepts and relations
- Light-weight Ontology
  - Concepts
  - 'is-a' hierarchy among concepts
  - relations between concepts
- Middle-weight Ontology
  - Some formal notations
  - but only a modest amount of logic and reasoning
- Heavy-weight Ontology
  - cardinality constraints
  - taxonomy of relations
  - Axioms (restrictions)

# RDF and RDFS

- RDF stands for: Resource Description Framework.
  - is a W3C standard, which provides tool to describe Web resources.
  - provides interoperability between applications that exchange machine-understandable information.
  - But does not give any special meaning to vocabulary such as subClassOf or type.
- *RDF example*:
  - Bob Johnson is the creator of the resource http://www.w3.org/Home/Lassila

```
<rdf:RDF>
 <rdf:Description about=
    "http://www.w3.org/Home/Lassila">
  <s:Creator> Bob Johnson </s:Creator>
 </rdf:Description>
</rdf:RDF>
```

# RDF and RDFS

- RDFS extends RDF with "schema vocabulary"
  - Allow us to define basic vocabulary terms and the relationships among these terms:
  - Class, type, subClassOf
  - Property, subPropertyOf, range, domain
  - It gives extra meaning to particular RDF predicates and resources
  - This 'extra meaning', or semantics, specifies how a term should be interpreted.

# OWL

- 10 February 2004 the World Wide Web Consortium announced final approval of two key *Semantic Web* technologies, the revised Resource Description Framework (RDF) and the Web Ontology Language (OWL)
- Example: **OWL Web Ontology Language**
  http://www.w3.org/TR/owl-features/

# OWL Example

- There are two types of animals, Male and Female.

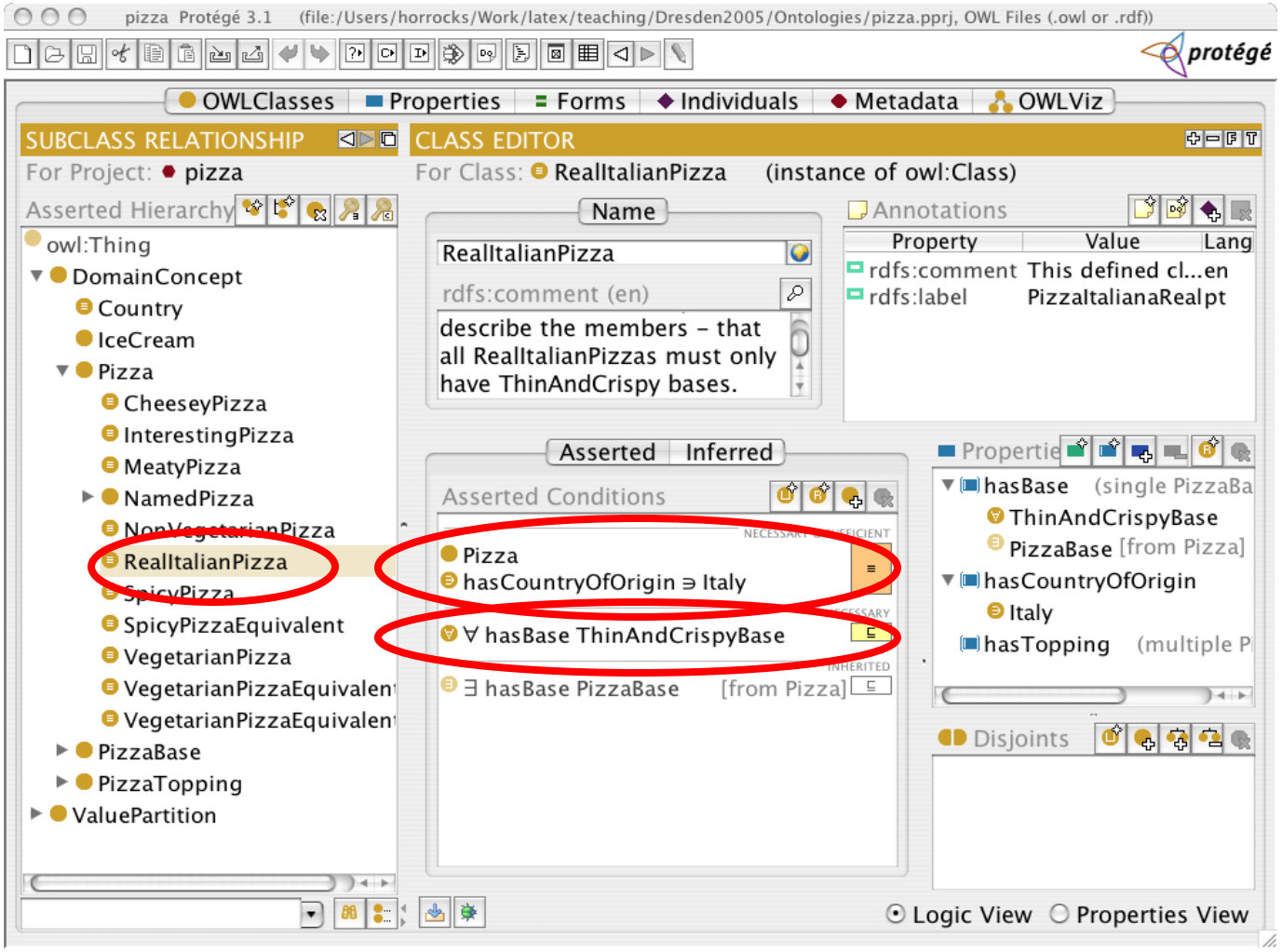
```
<rdfs:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</rdfs:Class>
```

- The subClassOf element asserts that its subject - Male - is a subclass of another object -- the resource identified by #Animal

```
<rdfs:Class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <owl:disjointWith rdf:resource="#Male"/>
</rdfs:Class>
```

- Some animals are Female, too, but nothing can be both Male and Female (in this ontology) because these two classes are disjoint (using the disjointWith tag).

# Ontology Example using Protégé



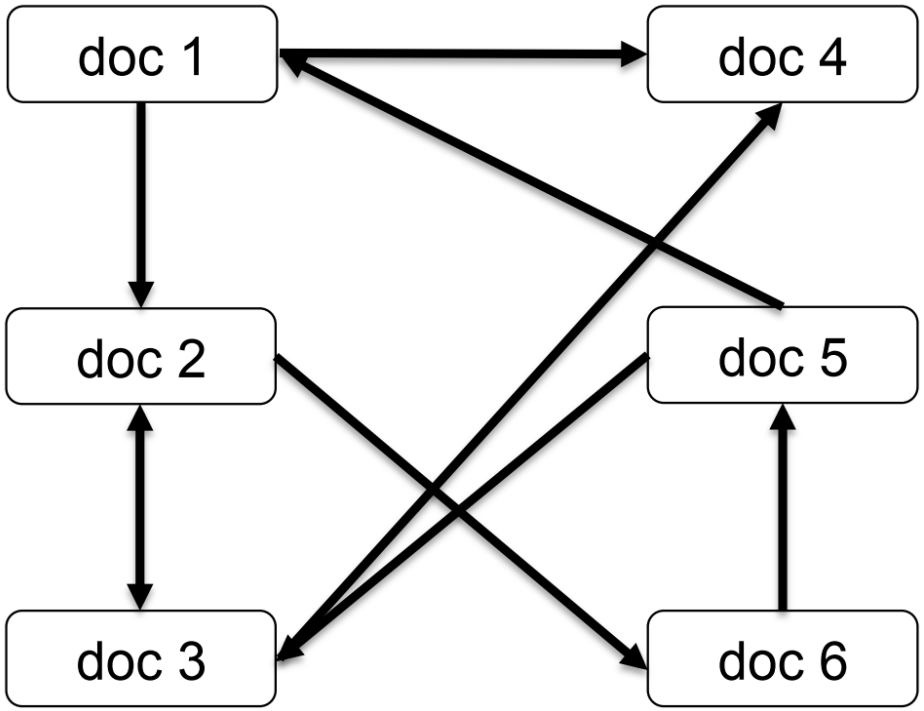
# Tools to Work with Ontology

- Protégé
  - http://protege.stanford.edu/
- WC3 Web 2.0 tools
  - OWL Web Ontology Language (http://www.w3.org/TR/owl-features/)
  - RDF Resource Description Framework (http://www.w3.org/RDF/)

# An Introduction to Web Ontology Language (OWL)

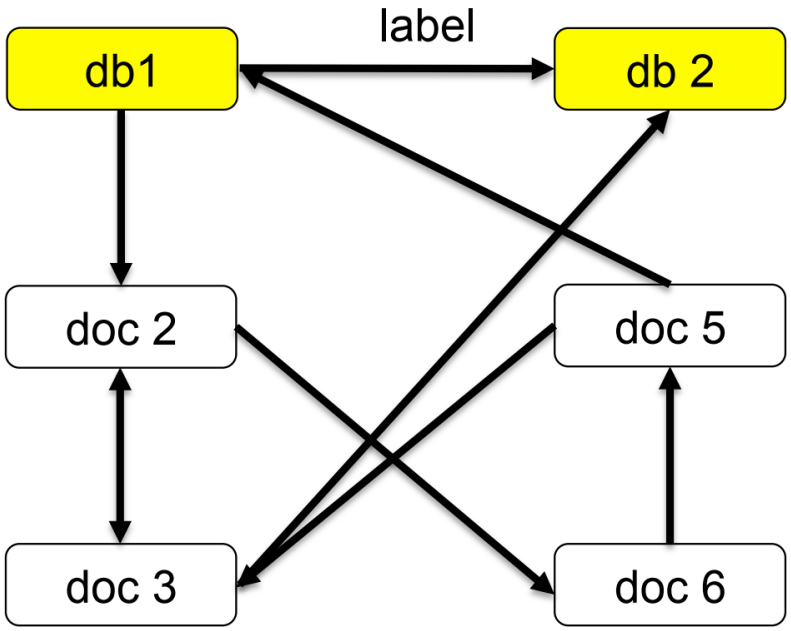
# Today's Web

- Web of documents – processed by humans
- Currently, users search for data on the Web asking questions like "*which documents contain these words or phrases*"



# Semantic Web

- Web of things – processed by machines
- Search in not based on word matching but on related items and relationships (semantic contents)
- Resources should be more accessible to automated processes.
  - To be achieved through semantic markup
  - Metadata annotations that describe content/function



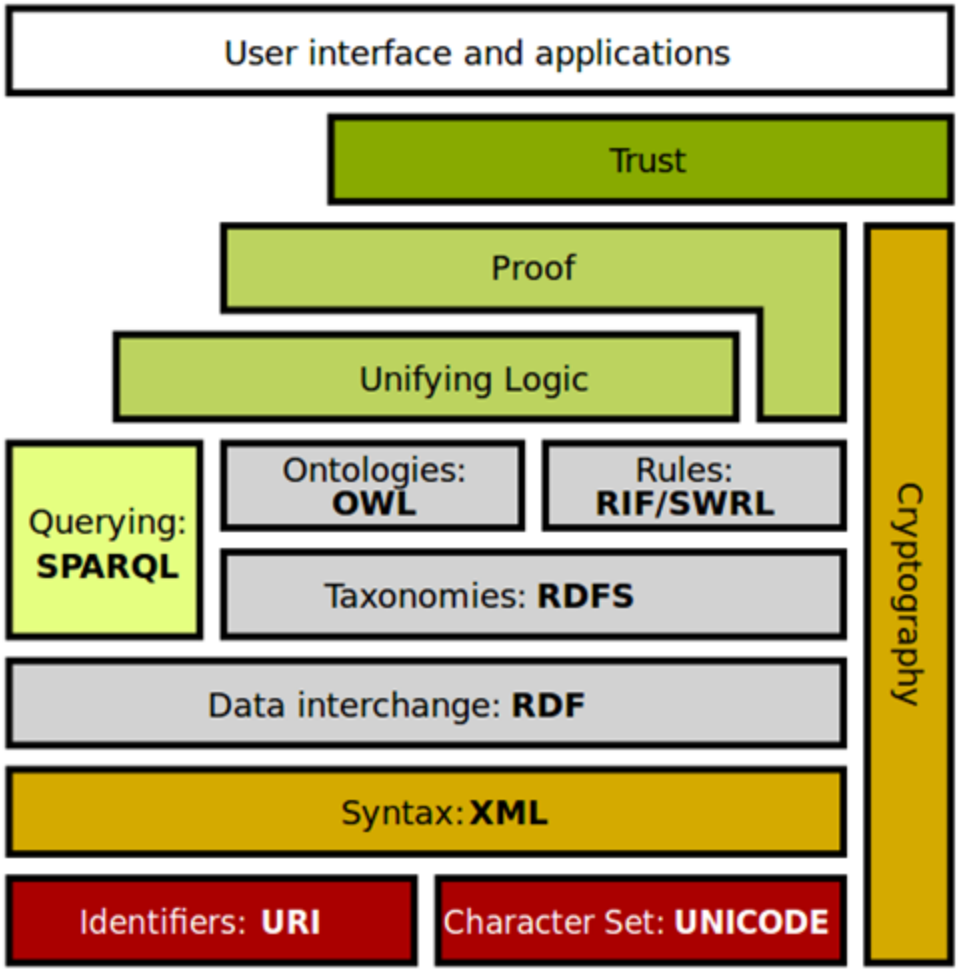
# Semantic Web – First step

- Make web resources more accessible to automated processes.
- Extend existing rendering markup (information content, hyperlinks) with semantic markup
  - Metadata annotations that describe content/function of web accessible resources
- Using ontologies to provide vocabularies of annotations
  - New terms can be formed by combining existing ones
  - 'Formal specifications' is accessible to machines
- A prerequisite is a standard web ontology language
  - Need to agree a common syntax before we can share semantics.
  - Syntactic web based on standards such as HTTP and HTML

# Technologies for the Semantic Web

- Metadata
  - Resources are marked-up with descriptions of their contents. Not good unless everyone speaks the same language.
- Terminologies
  - Provide shared and common vocabularies of a domain, so search engines, agents, authors, and users can communicate. Not good unless everyone means the same thing.
- Ontologies
  - Provide a shared and common understanding of a domain that can be communicated across people and applications and will play a major role in supporting information exchange and discovery.

# Stack Architecture for Semantic Web




OWL: Web Ontology Language

RDF: Resource Description Framework

RDFS: RDF Schema

SPARQL: query language

URI: Uniform Resource Identifier

# Semantic Web Technologies

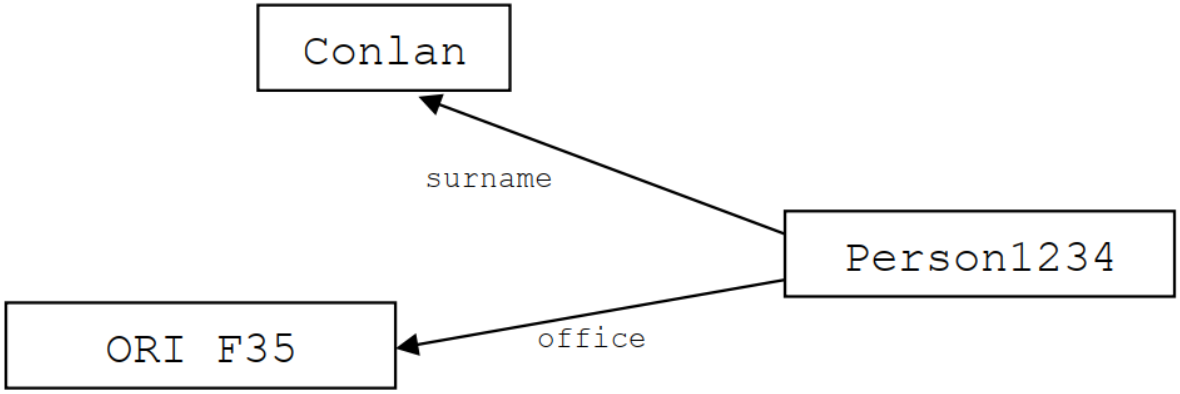
- Set of technologies and frameworks that enable such integration (the Web of Data) possible.
  - Semantic annotation and retrieval: RDF, RDFS
  - Storing the Semantic Web: Repositories
  - Querying the Semantic Web: SPARQL
  - Reasoning on the Semantic Web: OWL, reasoning tools

# Knowledge Representation in Semantic Web

- There are a number of options.
  - As objects, using the well-accepted techniques of object-oriented analysis and design to capture a model.
  - As clauses, going back to the early days of AI and Lisp.
  - As XML, using the industry-standard structured mark-up language
  - As graphs, making use of the things we know about graph theory
  - As some combination of these
- We are looking for: extensibility, ease of use, ease of querying

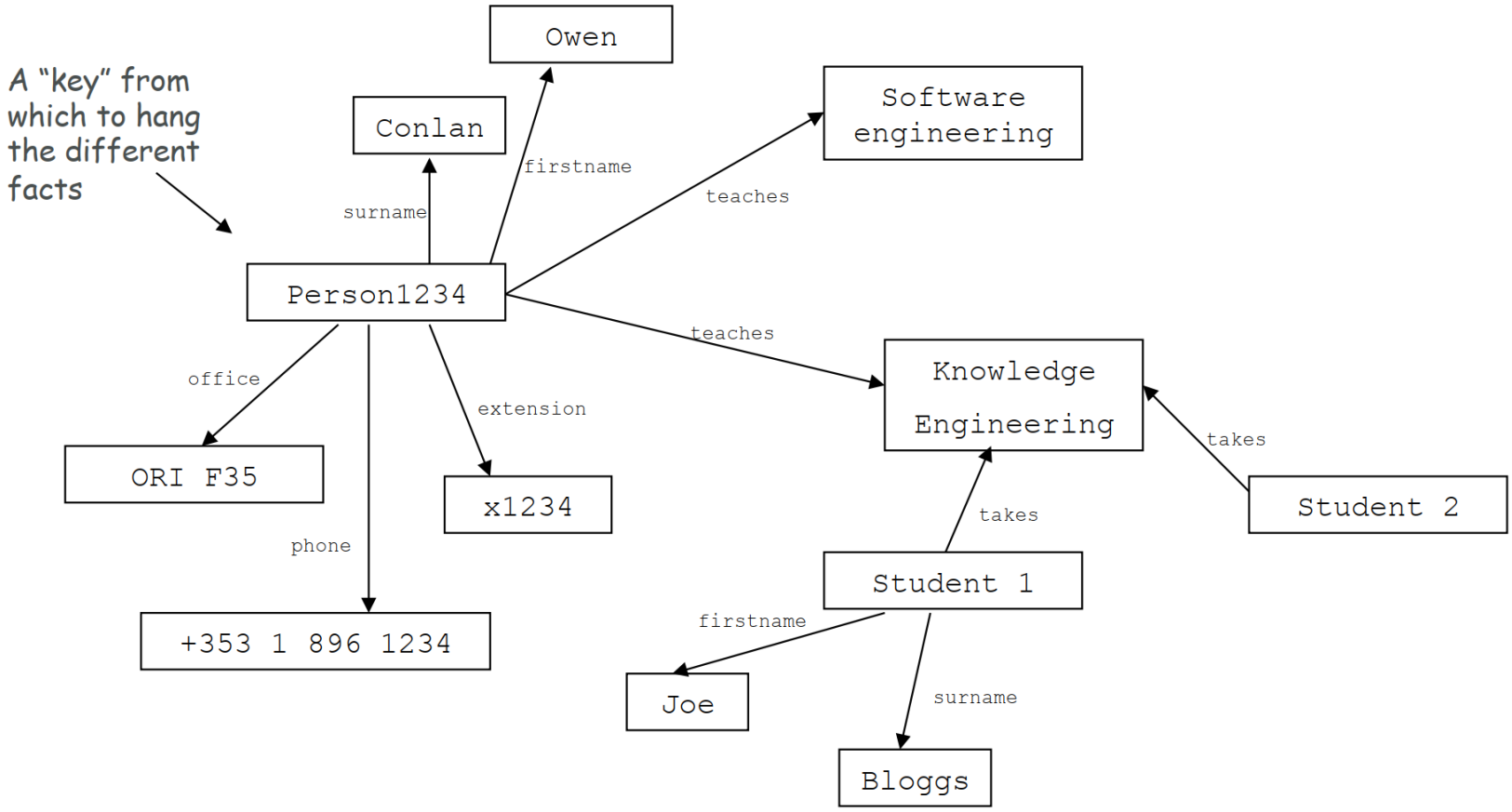
# Graphs

- We can use the nodes of a graph for facts and the arcs as (binary) relationships between them.
  - Arcs are typically called predicates or relationships in this view.
  - The set of arcs intersecting a node tells us the information we know about that fact or entity.



# Graphs as Knowledge

- How do we use graphs to represent knowledge?



# Graphs as Knowledge

## Things to note

- Scaling – the same graph can represent a load of different knowledge simultaneously
- Agreement – need to know what the various predicates "mean"
- Structure – you need to know what nodes are related by a predicate
- Plurality – the same relationship may appear several times
- Symmetry – the same predicates can be used for common information, despite minor changes
- Asymmetry – relationships are inherently directed, which sometimes makes things awkward

…and this can get very tricky

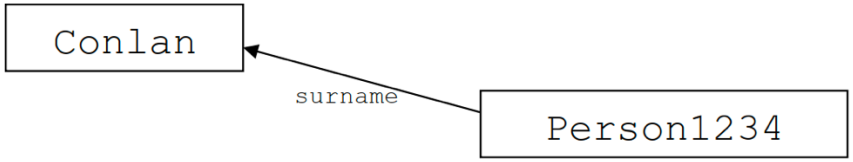
…and this can be difficult to keep straight

For example both lecturers and students have names

So a knowledge (context) graph is inherently directed

# Two Ways to View a Graph

- As nodes and arcs]
  - Nodes store facts, edges store relationships between them



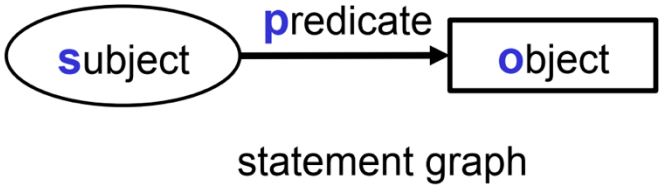

- As triples
  - A three-place relationship of “subject, predicate, object”
  - The node and edge structure is induced by the triples – each triple defines an edge, the different subjects/objects are the population of nodes, one node per individual string

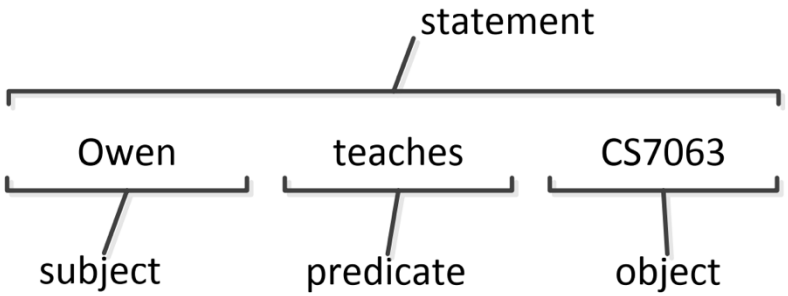
# Resource Description Framework (RDF)

- RDF is a W3C recommendation that enables encoding, exchange and reuse of structured metadata.
- Resource: anything we want to talk about
- RDF is graphical formalism for expressing data models about "something" using statements expressed as triples.
- RDF Triples: a labelled connection between two resources, a labelled arc in a graph.
- An RDF model is an unordered collection of statements, each with a subject, predicate and object.
- RDF describes the semantics of information in a machine-accessible way.

# RDF Triples

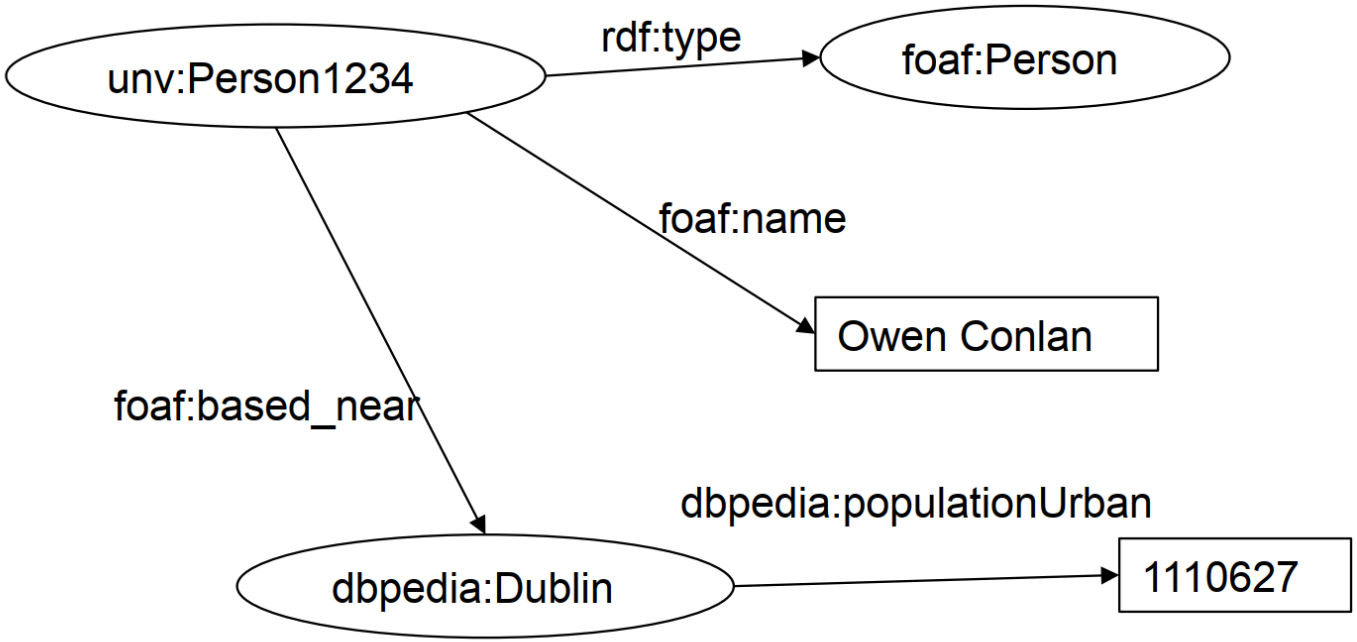
- Graph representation of a triple.




- This can be read as
  - s has a property p with a value o (left to right)
  - o is the value of p for s (right to left)
  - The p of s is o (as directed relationship)

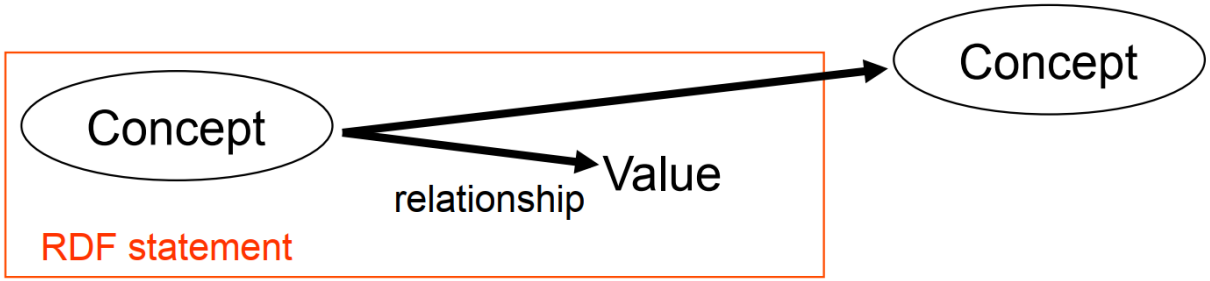
# Example RDF Triples as Graphs



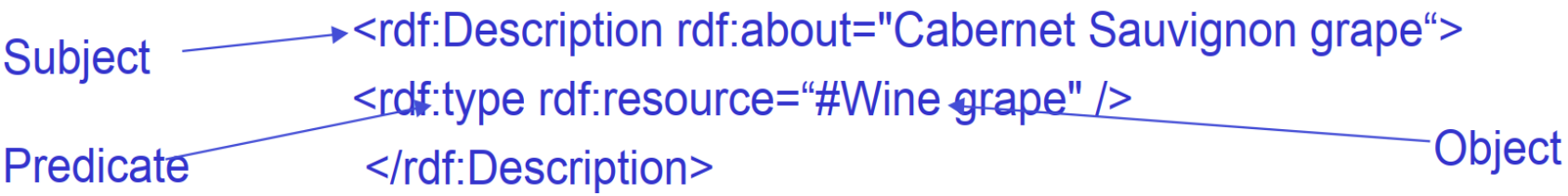

- unv:Person1234 = http://www.scss.tcd.ie/owen.conlan
- dbpedia:Dublin = http://dbpedia.org/resource/Dublin

# Example RDF Triples as XML

- Triples of assertions can be expressed using XML tags.




- E.g. "Cabernet Sauvignon grape", "is a type of" ,"Wine grape"




- Each resource can be assigned a different Universal Resource Identifier (URI).
  - Thus different meanings for the same term can be assigned different URIs

# URI (Uniform Resource Identifier)

- URI is used for identifying (naming) resources on the Web.
- URLs (Uniform Resource Locators) are a particular type of URI, the resources can be accessed on the Web
- URIs unlike URLs are not limited to identifying thinks that have network locations
- In RDF, URIs often have fragment identifiers to point at specific parts of a document:
  - http://www.somedomain.com/some/path/to/file#fragmentID
- URIs are unambiguous, Web provides a global namespace
- Different URI schemas – http, mailto, ftp, urn …

# XML to RDF

- Modify XML to an RDF document

XML

```
<?xml version="1.0"?>
<River id="Shannon"
        xmlns="http://www.scss.tcd.ie/rivers">
    <length>360 kilometers</length>
    <startingLocation>Cuilcagh Mountain, County Cavan</startingLocation>
    <endingLocation>Limerick</endingLocation>
</River>
```

RDF

```
<?xml version="1.0"?>
<rdf:Description rdf:about=" http://www.scss.tcd.ie/rivers/Shannon"
                 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                 xmlns="http://www.scss.tcd.ie/rivers#">
    <rdf:type rdf:resource="http://live.dbpedia.org/ontology/River"/>
    <length>360 kilometers</length>
    <startingLocation>Cuilcagh Mountain, County Cavan</startingLocation>
    <endingLocation>Limerick</endingLocation>
</rdf:Description>
```

# RDF: XML-Based Syntax Elements

| | |
|---|---|
| rdf:RDF | root element of RDF documents, where a number of descriptions are defined |
| rdf:Description | element contains the description of the resource |
| rdf:type | instance of |
| rdf:Bag | an unordered container of resources |
| rdf:Seq | an ordered container of resources |
| rdf:Alt | Defines a set of alternative resources |

# RDF: XML-Based Syntax Attributes

| | |
|---|---|
| rdf:ID | indicating a new resource |
| rdf:about | referencing an existing resource |
| rdf:resource | allows property elements to be defined as resources |

# A cluster of facts

- Given a common subject we can build a cluster of facts using nested predicate elements

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://www.dsg.cs.tcd.ie/xml/demo.html#">
  <rdf:Description about="http://www.dsg.cs.tcd.ie/">
    <s:about>Distributed Systems Group</s:about>
    <s:author>S. Punter</s:author>
    <s:phone>+353 1 123 4567</s:phone>
  </rdf:Description>
</rdf:RDF>
```

Each of these gives rise to a triple with the same subject (inherited from the containing `Description` element)

As long as we agree what the predicates mean, we can use whichever we want

# RDF Classes & Properties

## Classes

- rdf:XMLLiteral
- rdf:Property
- rdf:Alt
- rdf:Bag
- rdf:Seq
- rdf:List
- rdf:nil
- rdf:Statement

## Properties

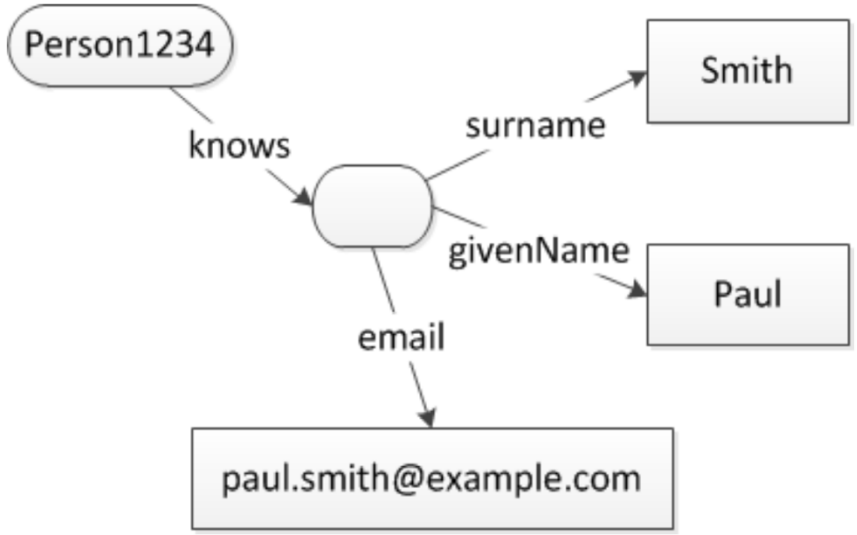
- rdf:type
- rdf:first
- rdf:rest
- rdf:value
- rdf:subject
- rdf:predicate
- rdf:object

# Identify the Resource

- URI references may be either absolute or relative
- When a (relative) URI reference consists of just a fragment identifier, it refers to the document that appears
- An element rdf:Description has
  - rdf:about attribute – references an existing resource
  - rdf:ID attribute – indicating a new resource
    - *The value of rdf:ID is a "relative" URI*
  - Without a name creating an anonymous resource

# Blank nodes

- RDF doesn't require every resource in a statement to be identified with a URI.
- Blank nodes are graph nodes that represent a resource for which we would like to make assertions, but have no way to address with a proper URI
- These resources are not visible outside – they are anonymous
- From a logic point of view, blank nodes represent an "existential" statement



# Literals

- Literals are objects that are not URIs but actual content
- There are two kinds of literals
  - plain (untyped) – have a lexical form and optionally a language tag: "wellcome"@en
  - Typed – is formed by pairing a string with a particular datatype (e.g. from XML Schema)
    "27"^^http://www.w3.org/2001/XMLSchema#integer
  - RDF has no built-in set of datatypes. RDF uses externally datatypes that are identifies by a URI

```
<rdf:Description rdf:about="http://…/isbn/51409X">
   <page_number rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">543
   </page_number>
   <price rdf:datatype="http://www.w3.org/2001/XMLSchema#float">6.99</price>
</rdf:Description>
```

# RDF Serialization Formats

- There is a variety of data interchange formats
  - RDF/XML – the original (W3C Recommendation) and most frequently used serialization format
  - N-Triples - simple notation, easy-to-parse, line-based format that is not as compact as Turtle
  - N3 – similar to N-Triples, additional structures to reduce repetition
  - Turtle - a compact, human-friendly format
  - RDFa - a way of annotating XHTML web pages with RDF data
  - Json – a JSON based serialisation

# Common Vocabularies

- Commonly used vocabulary namespaces in RDF
  - RDF: http://www.w3.org/1999/02/22-rdf-syntax-ns#
  - Dublin Core: http://purl.org/dc/elements/1.1/
  - SKOS: http://www.w3.org/2004/02/skos/core#
  - FOAF: http://xmlns.com/foaf/0.1/

# Structuring the knowledge - Limitations

- RDF provides a way of building graphs from triples, but doesn't constrain the graph too much
  - Nothing stops an application from giving a place a surname, for example, although this is probably nonsense
- The problem is that RDF is an untyped mechanism for building graphs
  - No knowledge of which triples are "allowed", or what "thing" must be the subject/object of an arc
- This is a problem in two distinct ways
  - In interpretation – different people may interpret the predicates subtly differently and use them between values you can't handle.
  - In scaling – hard for an application to get it right

# RDF Schemas (RDFS)

- Officially: "RDF Vocabulary Description Language"
- RDF is domain independent – there are no assumptions about a particular domain, concepts,…
- When compared to XML Schema, RDFS defines the vocabulary used in RDF data models, where the XML Schema constrains the structure of XML documents.
- RDFS extends RDF with "schema vocabulary", e.g.:
  - Class, Property
  - type, subClassOf, subPropertyOf
  - range, domain

# RDFS Classes

| | |
|---|---|
| rdfs:Resource | the class of all resources |
| rdfs:Class | the class of all classes |
| rdfs:Literal | the class of all literals (strings) |
| rdfs:Property | the class of all properties |
| rdfs:Datatype | the class of datatypes |

# RDFS Properties

| | |
|---|---|
| rdfs:subClassOf | Relates class to one of its superclasses, declare hierarchies of classes, is transitive by definition |
| rdfs:subPropertyOf | relates a property to one of its superproperties, is transitive by definition |
| rdfs:domain | declares the class of the subject in a triple |
| rdfs:range | declares the class or datatype of the object in a triple |

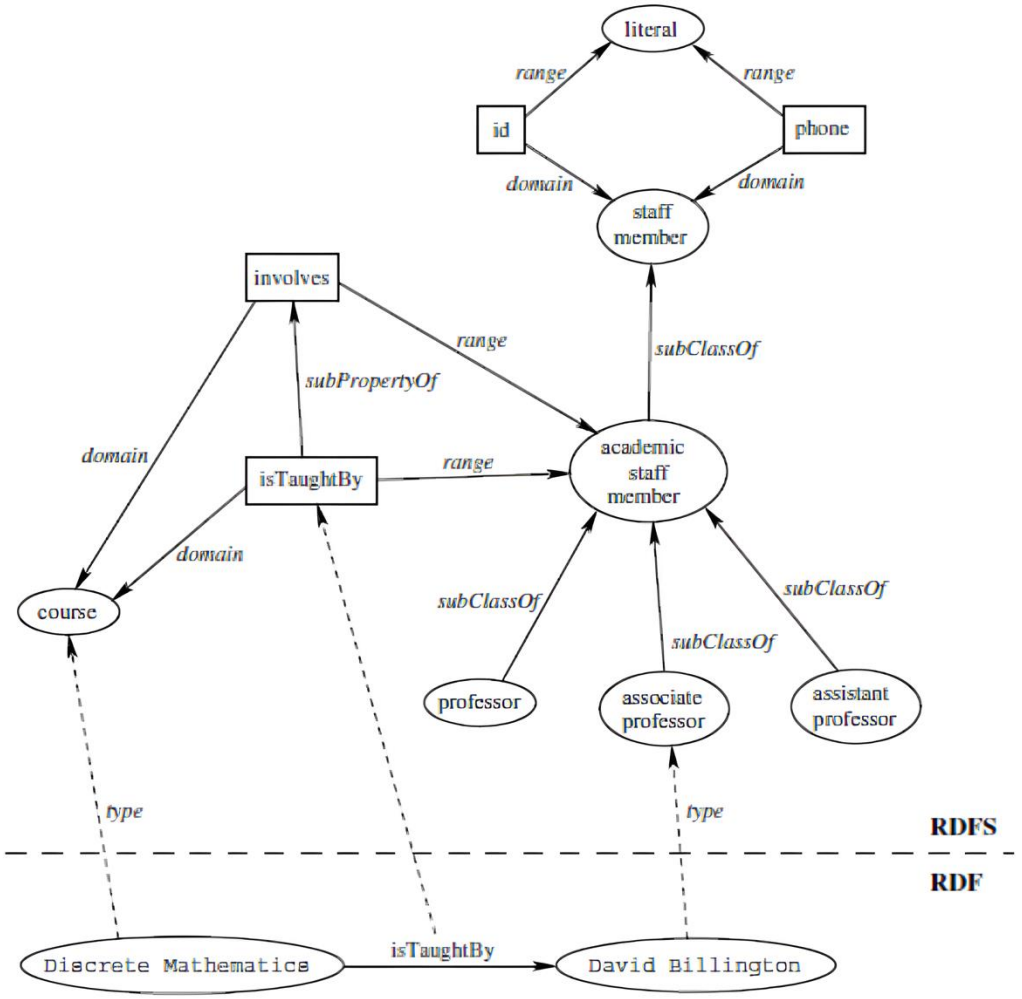
| | |
|---|---|
| rdfs:comment | typically provides a longer text description of the resource |
| rdfs:label | associates the resource with a human-friendly name |
| rdfs:isDefinedBy | relates a resource to the place where its definition, typically an RDF schema |
| rdfs:seeAlso | relates a resource to another resource that explains it |

# RDFS Example

- These terms are the RDF Schema building blocks (constructors) used to create vocabularies:

```
<Person,type,Class>
<hasColleague,type,Property>
<Professor,subClassOf,Person>
<Carole,type,Professor>
<hasColleague,range,Person>
<hasColleague,domain,Person>
```

# RDFS Example – Graph model



# RDF/RDFS "Liberality"

- No distinction between classes and instances (individuals)

  <Species, type, Class>
  <Lion, type, Species>
  <Leo, type, Lion>

- Properties can themselves have properties

  <hasDaughter, subPropertyOf, hasChild>
  <hasDaughter, type, familyProperty>

- No distinction between language constructors and ontology vocabulary, so constructors can be applied to themselves/each other

  <type, range, Class>
  <Property, type, Class>
  <type, subPropertyOf, subClassOf>

# Problems with RDF(S)

- RDF(S) is too weak to describe resources in sufficient detail.
  - No localized range and domain constraints.
    - Cannot say that the range of **hasChild** is **Person** when applied to Persons and Elephant when applied to Elephants.
  - No existence/cardinality constraints
    - Can't say that all *instances* of Person have a mother that is also a Person, or that Persons have exactly 2 parents
  - No **transitive, inverse** or **symmetrical** properties
    - Can't say that isPartOf is a *transitive* property, that hasPart is the inverse of isPartOf or that touches is symmetrical.

- Difficult to provide **reasoning support**
  - No "native" reasoners for non-standard semantics.

# Solutions

- **Extend** RDF(S) with a language that has the following desirable features identified for Web Ontology Language.
  - **Extends** existing Web standards
    - Such as XML, RDF, RDFS
  - **Easy** to understand and use
    - Should be based on familiar Knowledge Reasoning idioms
  - Of "**adequate**" expressive power
  - **Formally** specified - describes the meaning of knowledge precisely
    - Possible to provide automated reasoning support
- That language is **OWL**

# OWL Language

- Three species of OWL.
  - **OWL full** is union of OWL syntax and RDF
  - **OWL DL** restricted to FOL fragment power
  - OWL Lite is "easier to implement" subset of OWL DL
- OWL DL Benefits from many years of Description Logics (DL) research
  - Well defined semantics
  - Formal properties well understood (complexity, decidability)
  - Known reasoning algorithm
  - Implemented systems (highly optimised)

# Example of OWL Document

```
<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/XMLSchema#">
    <owl:Ontology rdf:about="">
          <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>
    <owl:Class rdf:ID="associateProfessor">
          <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
     </owl:Class>
     …
</rdf:RDF>
```

# Example: Defining terms and a subclass relationship

Define the term “Room”

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

Define term “Restroom” and state that a Restroom is a type of Room

```
 <owl:Class rdf:ID="Restroom">
    <rdfs:subClassOf rdf:resource="#Room"/>
</owl:Class>
```

Note: owl:Thing is a predefined OWL Class and is the root of all classes. Similarly, owl:Nothing is the empty class

# Defining Classes

- OWL provides several other mechanisms for defining classes.
  - **equivalentClass** allows you to state that two classes are synonymous
  - **disjointWith** allows you to state that an instance of this class cannot be an instance of another
    - E.g., Man and Woman could be stated as *disjoint* classes

Boolean combinations

- **unionOf** allows you specify that a class contains things that are from more than one class
  - E.g. Restroom could be defined as a union of MensRoom and LadiesRoom
- **intersectionOf** allows you to specify that a class contains things that are both in one and the other)
- **complementOf** allows you specify that a class contains things that are not other things.
  - E.g. Children are not SeniorCitizens

# Example: equivalentClass and unionOf

```
<owl:Class rdf:ID="AtomicPlaceInBuilding">
   <rdfs:subClassOf>
     <owl:Class rdf:about="#AtomicPlace"/>
   </rdfs:subClassOf>
   <owl:equivalentClass>
     <owl:Class>
       <owl:unionOf rdf:parseType="Collection">
         <owl:Class rdf:about="#Room"/>
         <owl:Class rdf:about="#Hallway"/>
         <owl:Class rdf:about="#Stairway"/>
         <owl:Class rdf:about="#OtherPlaceInBuilding"/>
       </owl:unionOf>
     </owl:Class>
   </owl:equivalentClass>
 </owl:Class>
```

# Example disjointWith

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

# Defining Properties

In RDF Schema the rdf:Property is used both to

- Relate one Resource to another Resource
  - *For example, a "accessRestrictedToGender" property can relate a Restroom to a Gender*
- Relate a resource to a rdfs:Literal or datatype
  - *For example, a "latitude" property relates a Room to a xsd:string type*

OWL provides different statements for two cases

- **owl:ObjectProperty** is used to relate a resource to another

```
<owl:ObjectProperty rdf:ID="accessRestrictedToGender">
   <rdfs:range rdf:resource="#Gender"/>
   <rdfs:domain rdf:resource="#Restroom"/>
```

What range of values

`what classes is this property associated with

- **owl:DatatypeProperty** is used to relate a resource to a rdfs:Literal or XML schema data type

```
<owl:DatatypeProperty rdf:ID="latitude">
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   <rdfs:domain rdf:resource="#Place"/>
```

# Characterising Properties

- OWL allows use of three of the RDFS statements.
  - **<rdfs:range>** used to indicate the possible value types for a property
  - **<rdfs:domain>** use to associate a property with a class.
  - <**rdfs:subPropertyOf**> use this to specialize a property.

- Example:

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

# Characterising Properties

owl:equivalentProperty – to define equivalence of properties

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

owl:inverseOf - to relate inverse properties

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

# Restricting Properties

Associate the property (defined elsewhere) with the Class by using combination of <rdfs:subClassOf> <owl:onProperty> and defining local restrictions on that property using <owl:Restriction>

- owl:allValuesFrom - all values of the property must come from a specific class
- owl:someValuesFrom - at least one value of the property must come from a specific class
- owl:hasValue - states a specific value that the property specified

- owl:minCardinality - it has at least (individuals or data values)
- owl:maxCardinality – it has at most (individuals or data values)
- owl:cardinality – it has a specific number of (individuals or data values)

# Example - Restricting Properties

```
<owl:Class rdf:ID="Restroom">
    <rdfs:subClassOf rdf:resource="#Room" />
    < owl:subClassOf>
     <owl:Restriction owl:cardinality="1">
       <owl:onProperty>
         <owl:ObjectProperty rdf:about="#accessRestrictedToGender" />
       </owl:onProperty>
       <owl:allValuesFrom>
         <owl:Class rdf:about="#Gender" />
       </owl:allValuesFrom>
     </owl:Restriction>
    </owl:subClassOf>
</owl:class>
```

# Instances

- Instances of classes are declared as in RDF

```
<rdf:Description rdf:ID="949352">
       <rdf:type rdf:resource="#academicStaffMember"/>
</rdf:Description>

or equivalently

<academicStaffMember rdf:ID="949352"/>
```

# Summary Example

```
<?xml version "1.0"?>
<Room rdf:ID="LargeConferenceRoom">
        <address rdf:resource="G.02"/>
        <spatiallySubsumedBy rdf:resource="O'Reilly Institute"/>
        <adjacentRoom rdf:resource="SmallConferenceRoom" />
        <coordinates rdf:resource=44,55 />
</Room>
```

Given the preceding definitions it can be inferred automatically:

1. The O'Reilly Institute *spatially subsumes* the Large Conf Room (since spatiallySubsumedBy is an inverse property)
2. The Small Conference Room is *adjacent to* Large Conference Room (since adjacentRoom is symmetric)
3. Only the Large Conference Room can be found at coordinates 44,55 (since coordinates is inverse functional)
4. The Large Conference Room has only one address (since address is functional)

# SPARQL: RDF Query Language

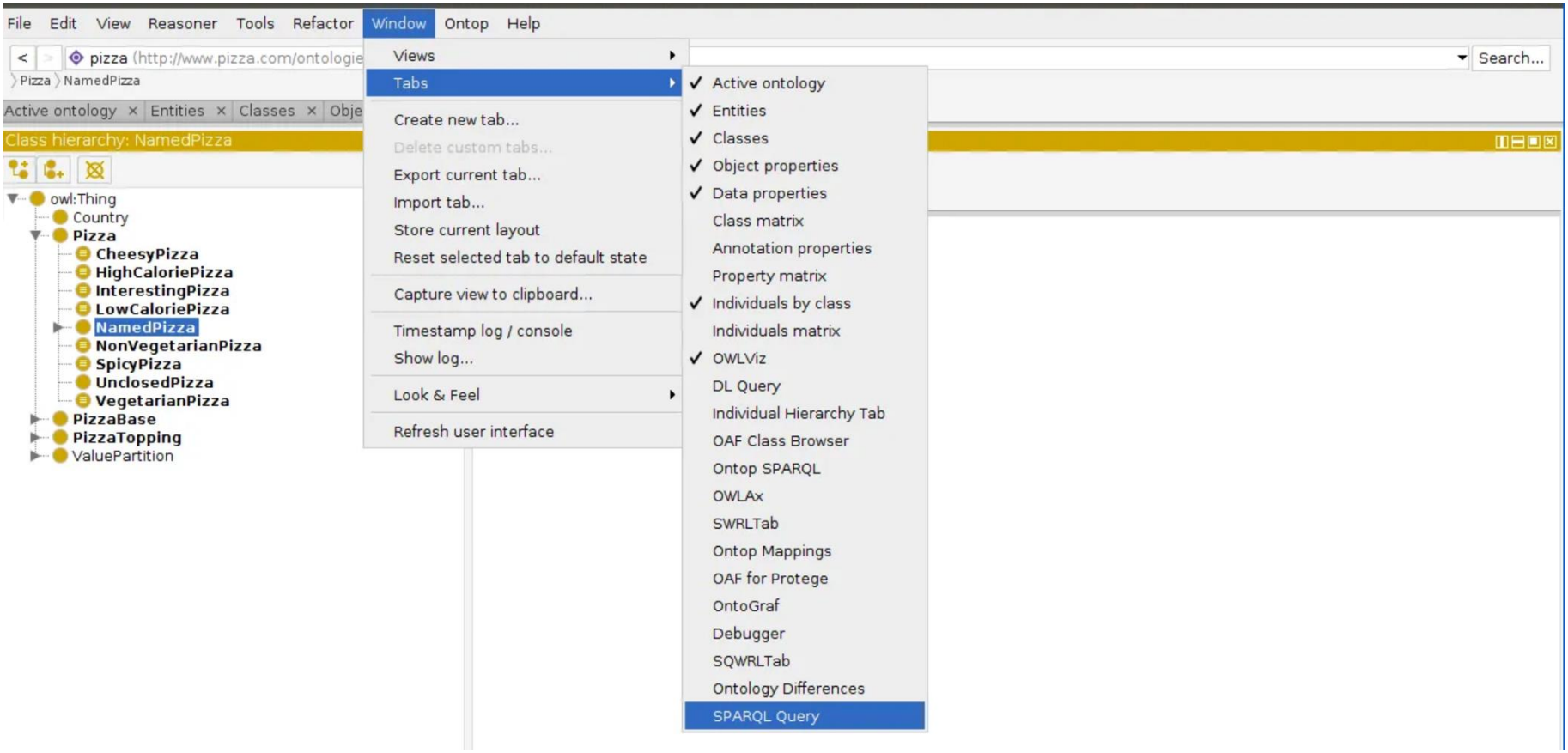
- SPARQL is a semantic query language for databases - able to retrieve and manipulate data stored in RDF format.
- SPARQL queries are represented in triple format

# Uses of SPARQL

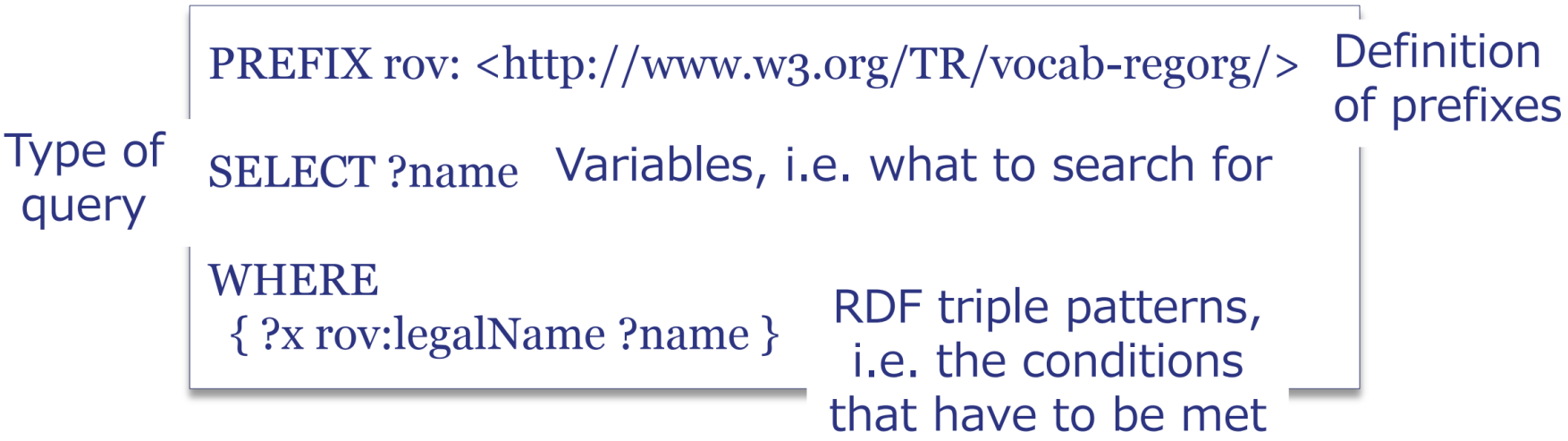
- Pull values from structured and semi structured data represented in RDF.
- Explore RDF by querying unknown relationships
- Perform complex joins off desperate RDF repositories in a single query.
- Transform RDF data form one vocabulary to another.
- Develop higher level cross platform applications.

# Running SPARQL Queries in Protégé

- Open SPARQL tab:



# Structrure of a SPARQL Query




- **WHERE:** clause specifies which data to pull out
  - Triple (s p o) patterns
- **SELECT:** picks what data to display

# Basic Types of SPARQL Queries

- **SELECT**
  Return a table of all X, Y, etc. satisfying the following conditions ...
- **CONSTRUCT**
  Find all X, Y, etc. satisfying the following conditions ... and substitute them into the following template in order to generate (possibly new) RDF statements, creating a new graph.
- **DESCRIBE**
  Find all statements in the dataset that provide information about the following resource(s) ... (identified by name or description)
- **ASK**
  Are there any X, Y, etc. satisfying the following conditions ...

# Example: SELECT

- Return the name of an organization with particular URI

Sample data

```
comp:A rov:haslegalName “Niké” .
comp:A org:hasRegisteredSite site:1234 .

Comp:B rov:haslegalName “BARCO” .

site:1234 locn:fullAddress “Dahliastraat 24, 2160 Wommelgem .
```

Query

```
PREFIX comp: < http://example/org/org/>
PREFIX org: < http://www.w3.org/TR/vocab-regorg/ >
PREFIX site: <http://example.org/site/>
PREFIX rov: <http://www.w3.org/TR/vocab-regorg/>

SELECT ?name

WHERE
  { ?x org:hasRegisteredSite site:1234 .
    ?x rov:haslegalName ?name  .}
```

Result

| name |
| --- |
| “Niké” |

# Example: SELECT

- Return the name and address of organizations

Sample data

```
comp:A rov:haslegalName “Niké” .
comp:A org:hasRegisteredSite site:1234 .
Comp:B rov:haslegalName “BARCO” .
site:1234 locn:fullAddress “Dahliastraat 24, 2160 Wommelgem” .
```

Query

```
PREFIX org:<http://www.w3.org/TR/vocab-regorg/ >
PREFIX locn:<http://www.w3.org/ns/locn#>
PREFIX rov:<http://www.w3.org/TR/vocab-regorg/>

SELECT ?name ?address
WHERE
{ ?x org:hasRegisteredSite ?site.
?x rov:haslegalName ?name .
?site locn:fullAddress ?address . }
```

Result

| name | address |
|---|---|
| “Niké” | “Dahliastraat 24, 2160 Wommelgem” |

# Example: CONSTRUCT

- Create a new graph with another label for name

Sample data

```
comp:A rov:haslegalName “Niké” .
comp:A org:hasRegisteredSite site:1234 .
comp:B rov:haslegalName “BARCO” .
site:1234 locn:fullAddress “Dahliastraat 24, 2160 Wommelgem” .
```

Query

```
PREFIX comp:< http://example/org/org/>
PREFIX org:<http://www.w3.org/TR/vocab-regorg/>
PREFIX rdfs:<http://www.w3.org/TR/rdf-schema/>

CONSTRUCT {?comp rdfs:label ?name}

WHERE
{ ?comp org:haslegalName ?name. }
```

Resulting graph

```
@prefix comp: <http://example/org/> .
@prefix rdfs: <http://www.w3.org/TR/rdf-schema/>

comp:a rdfs:label “Niké" .
comp:b rdfs:label “BARCO" .
```

# Example: DESCRIBE

- Return all triples of organizations registered at a particular site

Sample data

```
comp:A rov:haslegalName “Niké” .
comp:A org:hasRegisteredSite site:1234 .
comp:B rov:haslegalName “BARCO” .
site:1234 locn:fullAddress “Dahliastraat 24, 2160 Wommelgem” .
```

Query

```
PREFIX comp: <http://example/org/>
PREFIX site: <http://example/site>
PREFIX org: < http://www.w3.org/TR/vocab-regorg/

DESCRIBE  ?organization

WHERE
{?organization org:hasRegisteredSite site:1234}
```

Result

```
@prefix comp: <http://example/org/> .
@prefix org: <http://www.w3.org/TR/vocab-regorg/> .

comp:A has:legalName “Niké” .
comp:A org:hasRegisteredSite site:1234 .
```

# Example: ASK

- Are there any organizations having "1234" as their registered site?

Sample data

```
comp:A rov:haslegalName "Niké" .
comp:A org:hasRegisteredSite site:1234 .

comp:B rov:haslegalName "BARCO" .

site:1234 locn:fullAddress "Dahliastraat 24, 2160 Wommelgem" .
```

Query

```
PREFIX org: < http://www.w3.org/TR/vocab-regorg/

ASK

WHERE
{?organisation org:hasRegisteredSite site:1234}
```

Result

```
TRUE
```

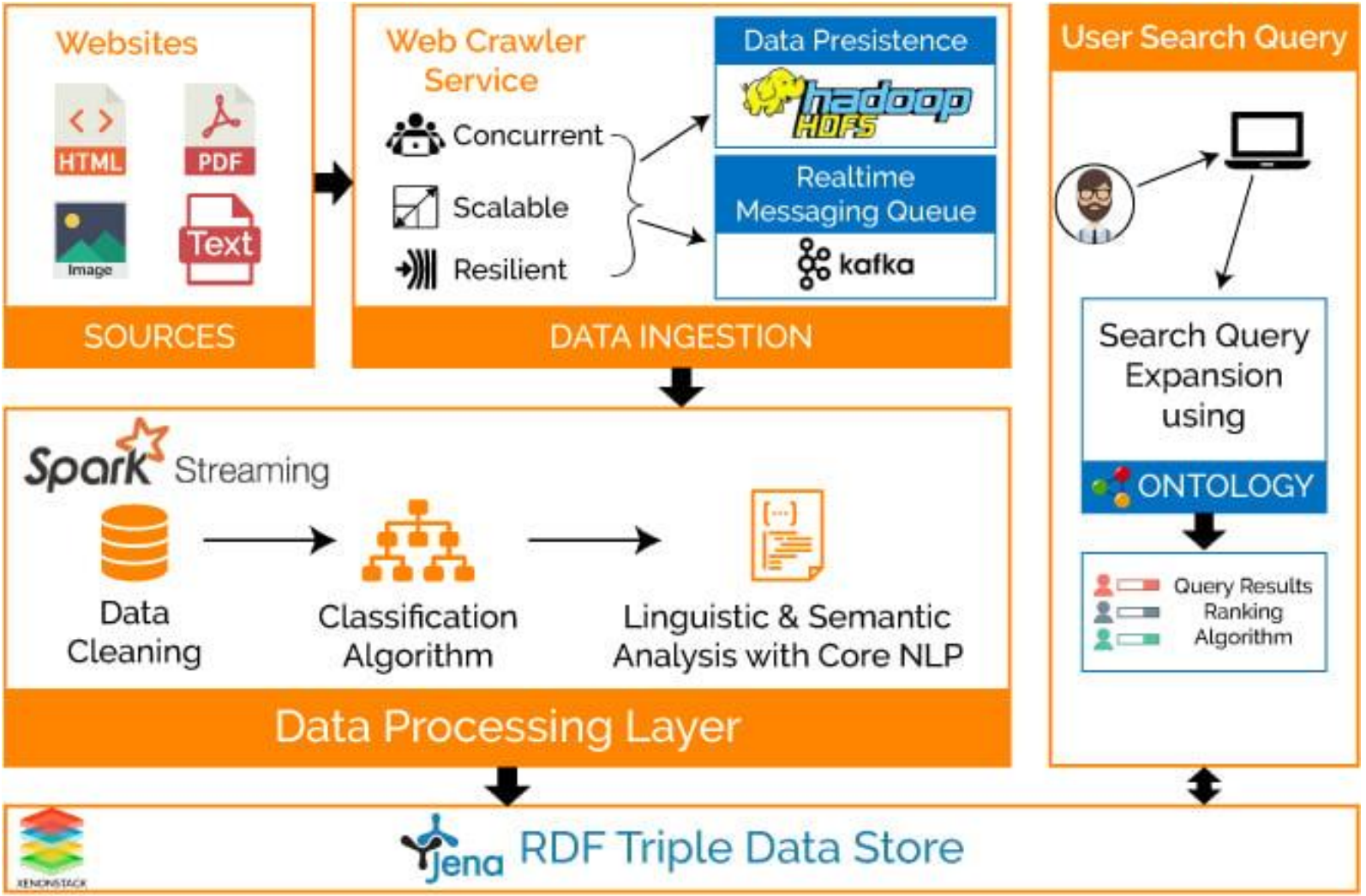
# Keyword-based Search vs Semantic Search

- Keyword-based search
  - The query is initially processed for text cleaning and preprocessing and then based on the words used in the query the searching is done on the documents.
  - The documents are returned based on the greatest number of matches of the query words with documents.
- Semantic search
  - take care of the frequency of the words, syntactic structure of the natural language and other linguistic elements.
  - the system understands the exact requirement of the search query.

# Keyword-based Search vs Semantic Search

- Keyword-based search
  - The query is initially processed for text cleaning and preprocessing and then based on the words used in the query the searching is done on the documents.
  - The documents are returned based on the greatest number of matches of the query words with documents.
- Semantic search
  - take care of the frequency of the words, syntactic structure of the natural language and other linguistic elements.
  - the system understands the exact requirement of the search query.

# Real-Time Semantic Search Engine Architecture



# Real-Time Semantic Search Engine Architecture

- The steps to build a Semantic Search Engine
  - Crawl the documents (DOC, PDF, XML, HTML etc) about the topics.
  - Convert the unstructured text present in various formats to structured RDF form.
  - Build Ontology for the specific domains.
  - Store the data of Ontology and RDFs in the suitable repositories, e.g., in Apache Jena RDF triple store
  - Use SPARQL query language on the data.
- An example
  - https://www.xenonstack.com/blog/semantic-search-engine