Ontology Languages for the Semantic Web

Sean Bechhofer
School of Computer Science, University of Manchester, UK
http://www.cs.manchester.ac.uk

The Semantic Web

- The Web made possible through established standards
  - TCP/IP for transporting bits down a wire
  - HTTP & HTML for transporting and rendering hyperlinked text
- Applications able to exploit this common infrastructure
  - Result is the WWW as we know it
- 1st generation web mostly handwritten HTML pages
- 2nd generation web often machine generated/active
  - But still intended for direct human processing/interaction
- In next generation web, resources should be more accessible to automated processes
  - Metadata annotations that describe content/function
  - Web of Data.
What’s the Problem?

- Consider a typical web page
- Markup consists of:
  - rendering information (e.g., font size and colour)
  - Hyper-links to related content
- Semantic content is accessible to humans but not (easily) to computers...
- Requires (at least) NL understanding

A Semantic Web

- Make web resources more accessible to automated processes
- Extend existing rendering markup with semantic markup
  - Metadata annotations that describe content/function of web accessible resources
- Use Ontologies to provide vocabulary for annotations
  - New terms can be formed by combining existing ones
  - “Formal specification” is accessible to machines
- A prerequisite is a standard web ontology language
  - Need to agree 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 content. No 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. No 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.

---

Building a Semantic Web

- **Annotation**
  - Associating metadata with resources

- **Integration**
  - Integrating information sources

- **Inference**
  - Reasoning over the information we have.
    - Could be light-weight (taxonomy)
    - Could be heavy-weight (logic-style)

- **Interoperation and Sharing** are key goals
Semantic Web Languages

- A number of languages have been defined that provide basic machinery used to represent the semantic information
  - XML
  - RDF
  - RDF(S)
  - OWL
  - ...

Object Oriented Models

- Many languages use an “object oriented model” with
  - Objects/Instances/Individuals
    - Elements of the domain of discourse
  - Types/Classes/Concepts
    - Sets of objects sharing certain characteristics
  - Relations/Properties/Roles
    - Sets of pairs (tuples) of objects
- Such languages are/can be:
  - Well understood
  - Formally specified
  - (Relatively) easy to use
  - Amenable to machine processing
Structure of an Ontology

Ontologies typically have two distinct components:

- **Names** for important concepts in the domain
  - Elephant is a concept whose members are a kind of animal
  - Herbivore is a concept whose members are exactly those animals who eat only plants or parts of plants
  - AdultElephant is a concept whose members are exactly those elephants whose age is greater than 20 years

- **Background knowledge/constraints** on the domain
  - AdultElephants weigh at least 2,000 kg
  - All Elephants are either AfricanElephants or IndianElephants
  - No individual can be both a Herbivore and a Carnivore

Ontology Languages

- There are a wide variety of languages for “Explicit Specification”
  - Graphical Notations
    - Semantic Networks
    - Topic Maps
  - UML
  - RDF
Ontology Languages

- There are a wide variety of languages for “Explicit Specification”
  - Graphical Notations
    - Semantic Networks
    - Topic Maps
    - UML
    - RDF
  - Logic Based
    - Description Logics
    - Rules
    - First Order Logic
    - Conceptual Graphs

Formal Languages

- The degree of formality of ontology languages varies widely
- Increased formality makes languages more amenable to machine processing (e.g., automated reasoning).
- The formal semantics provides an unambiguous interpretation of the descriptions.
**Why Semantics?**

- What does an expression in an ontology mean?
- The semantics of a language can tell us precisely how to interpret a complex expression.
- Well defined semantics are vital if we are to support machine interpretability
  - They remove ambiguities in the interpretation of the descriptions.

**RDF**

- RDF stands for Resource Description Framework
- It is a W3C Recommendation
  - [http://www.w3.org/RDF](http://www.w3.org/RDF)
- RDF is a graphical formalism (+ concrete syntax)
  - for representing metadata
  - for describing the semantics of information in a machine-accessible way
- Provides a simple data model based on triples.
The RDF Data Model

• Statements are \(<\text{subject}, \text{predicate}, \text{object}>\) triples:
  – \(<\text{Sean}, \text{hasColleague}, \text{Uli}>\)
• Can be represented as a graph:

• Statements describe properties of resources
• A resource is any object that can be pointed to by a URI
• Properties themselves are also resources (URIs)

Linking Statements

• The subject of one statement can be the object of another
• Such collections of statements form a directed, labeled graph

• Note that the object of a triple can also be a “literal” (a string)
RDF Syntax

- RDF has an XML syntax that has a specific meaning:
  - Every Description element describes a resource
  - Every attribute or nested element inside a Description is a property of that Resource
  - We can refer to resources by URIs

```xml
<Description about="some.uri/person/sean_bechhofer">
  <hasColleague resource="some.uri/person/uli_sattler"/>
  <hasName rdf:datatype="&xsd;string">Sean K. Bechhofer</hasName>
</Description>
<Description about="some.uri/person/uli_sattler">
  <o:hasHomePage>http://www.cs.mam.ac.uk/~sattler</o:hasHomePage>
</Description>
<Description about="some.uri/person/carole_goble">
  <o:hasColleague resource="some.uri/person/uli_sattler"/>
</Description>
```

What does RDF give us?

- A mechanism for annotating data and resources.
- Single (simple) data model.
- Syntactic consistency between names (URIs).
- Low level integration of data.
RDF(S): RDF Schema

- RDF gives a formalism for meta data annotation, and a way to write it down in XML, but it does not give any special meaning to vocabulary such as `subClassOf` or `type`
  - Interpretation is an arbitrary binary relation
- RDF Schema extends RDF with a schema vocabulary that allows you to define basic vocabulary terms and the relations between those terms
  - Class, Property
  - type, subClassOf
  - range, domain

RDF(S)

- 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>`
- Semantics gives “extra meaning” to particular RDF predicates and resources
  - specifies how terms should be interpreted
RDF(S) Inference

![Ontology Diagram](image)

1. Person
2. Academic
3. Lecturer

- Person: rdfs:subClassOf Academic
- Person: rdf:type rdfs:Class
- Academic: rdf:type rdfs:Class
- Academic: rdfs:subClassOf Lecturer
- Lecturer: rdf:type rdfs:Class
- Sean: rdf:type Academic
- Sean: rdf:type Lecturer

Ontology Languages, SSSW'08
RDF/RDF(S) “Liberality”

- No distinction between classes and instances (individuals)
  
  \(<\text{Species}, \text{type}, \text{Class}>\>
  
  \(<\text{Lion}, \text{type}, \text{Species}>\>
  
  \(<\text{Leo}, \text{type}, \text{Lion}>\>

- No distinction between language constructors and ontology vocabulary, so constructors can be applied to themselves/each other
  
  \(<\text{type}, \text{range}, \text{Class}>\>
  
  \(<\text{Property}, \text{type}, \text{Class}>\>
  
  \(<\text{type}, \text{subPropertyOf}, \text{subClassOf}>\>

- In order to cope with this, RDF(S) has a particular non-standard model theory.

What does RDF(S) give us?

- Ability to use simple schema/vocabularies when describing our resources.
- Consistent vocabulary use and sharing.
- Basic inference
Problems with RDF(S)

- RDF(S) is too weak to describe resources in sufficient detail
  - No localised range and domain constraints
    - Can’t 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
  - May be possible to reason via FO axiomatisation

OWL

- OWL: Web Ontology Language
- Extends existing Web standards
  - Such as XML, RDF, RDFS
- Is (hopefully) easy to understand and use
  - Based on familiar KR idioms
- Of “adequate” expressive power
- Formally specified
  - Possible to provide automated reasoning support
The OWL Family Tree

Aside: Description Logics

- A family of logic based Knowledge Representation formalisms
  - Descendants of semantic networks and KL-ONE
  - Describe domain in terms of concepts (classes), roles (relationships) and individuals
- Distinguished by:
  - Formal semantics (typically model theoretic)
    - Decidable fragments of FOL
    - Closely related to Propositional Modal & Dynamic Logics
  - Provision of inference services
    - Sound and complete decision procedures for key problems
    - Implemented systems (highly optimised)
DL Semantics

- **Model theoretic** semantics. An interpretation consists of
  - A domain of discourse (a collection of objects)
  - Functions mapping
    - classes to sets of objects
    - properties to sets of pairs of objects
  - Rules describe how to interpret the constructors and tell us when an interpretation is a model.
- In a DL, a class description is thus a characterisation of the individuals that are members of that class.

OWL Layering

- There are three “species” of OWL
  - OWL Full
  - OWL DL
  - OWL Lite
- **Syntactic Layering**
- **Semantic Layering**
  - OWL DL semantics = OWL Full semantics (within DL fragment)
  - OWL Lite semantics = OWL DL semantics (within Lite fragment)
**OWL Full**

- No restriction on use of OWL vocabulary (as long as legal RDF)
  - Classes as instances (and much more)
- RDF style model theory
  - Semantics should correspond with OWL DL for suitably restricted KBs

**OWL DL**

- Use of OWL vocabulary restricted
  - Can't be used to do "nasty things" (i.e., modify OWL)
  - No classes as instances
  - Defined by abstract syntax + mapping to RDF
- Standard DL/FOL model theory (definitive)
  - Corresponds to \( \text{SHOIN(D_n)} \) Description Logic
  - Direct correspondence with (first order) logic
- Benefits from DL research
  - Well defined semantics
  - Formal properties well understood (complexity, decidability)
  - Known reasoning algorithms
  - Implemented (optimised) systems
**OWL Lite**

- Like DL, but fewer constructs
  - No explicit negation or union
  - Restricted cardinality (zero or one)
  - No nominals (oneOf)
- Semantics as per DL
  - Reasoning via standard DL engines (+datatypes)
- In practice, not really used.
  - Possible alternative: “tractable fragments”

**OWL Syntaxes**

- Abstract Syntax
  - Used in the definition of the language and the DL/Lite semantics
- OWL in RDF (the “official” concrete syntax)
  - RDF/XML presentation
- XML Presentation Syntax
  - XML Schema definition
- Other, Human readable syntaxes
  - Manchester OWL Syntax
  - Sydney OWL Syntax
  - Rabbit
OWL (DL) Semantics

- OWL has a number of operators for constructing class expressions.
- These have an associated semantics which is given in terms of a domain:
  - $\Delta$
- And an interpretation function
  - $\text{I(concepts)} \to \wp(\Delta)$
  - $\text{I(properties)} \to \wp(\Delta \times \Delta)$
  - $\text{I(individuals)} \to \Delta$
- $I$ is then extended to concept expressions.

OWL Class Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Example</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>Class: Human</td>
<td>$I(\text{Human})$</td>
</tr>
<tr>
<td></td>
<td>(Human and Male)</td>
<td>$I(\text{Human}) \cap I(\text{Male})$</td>
</tr>
<tr>
<td></td>
<td>(Doctor or Lawyer)</td>
<td>$I(\text{Doctor}) \cup I(\text{Lawyer})$</td>
</tr>
<tr>
<td>not</td>
<td>not(Male)</td>
<td>$\Delta \setminus I(\text{Male})$</td>
</tr>
<tr>
<td>{}</td>
<td>{john, mary}</td>
<td>${I(\text{john}), I(\text{mary})}$</td>
</tr>
</tbody>
</table>
### OWL Class Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Example</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>some</td>
<td>hasChild some Lawyer</td>
<td>${x</td>
</tr>
<tr>
<td>only</td>
<td>hasChild only Doctor</td>
<td>${x</td>
</tr>
<tr>
<td>min</td>
<td>hasChild min 2</td>
<td>${x</td>
</tr>
<tr>
<td>max</td>
<td>hasChild max 2</td>
<td>${x</td>
</tr>
</tbody>
</table>

### OWL Axioms

- **Axioms** allow us to add further statements about arbitrary concept expressions and properties
  - Subclasses, Disjointness, Equivalence, characteristics of properties etc.
- An interpretation $I$ **satisfies** an axiom if the interpretation of the axiom is true.
  - Axioms **constrain** the allowed models
  - They provide the additional “assumptions” about the way in which the domain should be interpreted.
- $I$ **satisfies** or **is a model** of an ontology (or knowledge base) if the interpretation satisfies all the axioms in the knowledge base.
## OWL Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Example</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubClassOf</td>
<td>Class: Human&lt;br&gt;SubClassOf: Animal</td>
<td>$I(\text{Human}) \subseteq I(\text{Animal})$</td>
</tr>
<tr>
<td>EquivalentTo</td>
<td>Class: Man&lt;br&gt;EquivalentTo: (Human and Male)</td>
<td>$I(\text{Man}) = I(\text{Human}) \cap I(\text{Male})$</td>
</tr>
<tr>
<td>Disjoint</td>
<td>Disjoint: Animal, Plant</td>
<td>$I(\text{Animal}) \cap I(\text{Plant}) = \emptyset$</td>
</tr>
</tbody>
</table>

## OWL Individual Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Example</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual</td>
<td>Individual: Sean&lt;br&gt;Types: Human</td>
<td>$I(\text{Sean}) \in I(\text{Human})$</td>
</tr>
<tr>
<td>Individual</td>
<td>Individual: Sean&lt;br&gt;Facts: worksWith Ian</td>
<td>$I(\text{Sean}), I(\text{Ian}) \in I(\text{worksWith})$</td>
</tr>
<tr>
<td>DifferentIndividuals</td>
<td>Individual: Sean&lt;br&gt;DifferentFrom: Ian</td>
<td>$I(\text{Sean}) \neq I(\text{Ian})$</td>
</tr>
<tr>
<td>SameIndividuals</td>
<td>Individual: GeorgeWBush&lt;br&gt;SameAs: PresidentBush</td>
<td>$I(\text{GeorgeWBush}) = I(\text{PresidentBush})$</td>
</tr>
</tbody>
</table>
### OWL Property Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Example</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubPropertyOf</td>
<td>ObjectProperty: hasMother SubpropertyOf: hasParent</td>
<td>(I(\text{hasMother}) \subseteq I(\text{hasParent}))</td>
</tr>
<tr>
<td>Domain</td>
<td>ObjectProperty: owns Domain: Person</td>
<td>(\forall x. (x, y) \in I(\text{owns}) \Rightarrow x \in I(\text{Person}))</td>
</tr>
<tr>
<td>Range</td>
<td>ObjectProperty: employs Range: Person</td>
<td>(\forall x. (x, y) \in I(\text{employs}) \Rightarrow y \in I(\text{Person}))</td>
</tr>
<tr>
<td>Transitive</td>
<td>ObjectProperty: hasPart Characteristics: Transitive</td>
<td>(\forall x, y, z. ((x, y) \in I(\text{hasPart}) \land (y, z) \in I(\text{hasPart})) \Rightarrow (x, z) \in I(\text{hasPart}))</td>
</tr>
</tbody>
</table>

### Consequences

- An ontology (collection of axioms) places constraints on the models that are allowed.
- Consequences may be derived as a result of those constraints.
- \(C\) subsumes \(D\) w.r.t. an ontology \(O\) iff for every model \(I\) of \(O\), \(I(D) \subseteq I(C)\)  
- \(C\) is equivalent to \(D\) w.r.t. an ontology \(O\) iff for every model \(I\) of \(O\), \(I(C) = I(D)\)  
- \(C\) is satisfiable w.r.t. \(O\) iff there exists some model \(I\) of \(O\) s.t. \(I(C) \neq \emptyset\)  
- An ontology \(O\) is consistent iff there exists some model \(I\) of \(O\).
Reasoning

• A reasoner makes use of the information asserted in the ontology.

• Based on the semantics described, a reasoner can help us to discover inferences that are a consequence of the knowledge that we’ve presented that we weren’t aware of beforehand.

• Is this new knowledge?
  – What’s actually in the ontology?

Reasoning

• Subsumption reasoning
  – Allows us to infer when one class is a subclass of another
  – B is a subclass of A if it is necessarily the case that (in all models), all instances of B must be instances of A.
  – This can be either due to an explicit assertion, or through some inference process based on an intensional definition.
  – Can then build concept hierarchies representing the taxonomy.
  – This is classification of classes.

• Satisfiability reasoning
  – Tells us when a concept is unsatisfiable
    • i.e. when there is no model in which the interpretation of the class is non-empty.
  – Allows us to check whether our model is consistent.
Instance Reasoning

- Instance Retrieval
  - What are the instances of a particular class $C$?
  - Need not be a named class
- Instantiation
  - What are the classes that $x$ is an instance of?

Why Reasoning?

- Reasoning can be used as a design support tool
  - Check logical consistency of classes
  - Compute implicit class hierarchy
- May be less important in small local ontologies
  - Can still be useful tool for design and maintenance
  - Much more important with larger ontologies/multiple authors
- Valuable tool for integrating and sharing ontologies
  - Use definitions/axioms to establish inter-ontology relationships
  - Check for consistency and (unexpected) implied relationships
  - Already shown to be useful technique for DB schema integration
Example

Class: pet_owner
EquivalentTo:
  person
  and has_pet some animal

• A *pet_owner* is a *person* that has some *pet* that is an *animal*.
• This is a *equivalent* class, thus any *person* who has a *pet* that is an *animal* will be a *pet_owner*.
• If we know someone is a *pet_owner*, then we know that there must be some *animal* that is their *pet*: we may not know the name of this particular *animal* though.

Example

Class: giraffe
SubClassOf:
  animal,
  eats only leaf

• A *giraffe* is an *animal* that only eats *leaves*.
• This is a *partial* definition, thus every *giraffe* must have these characteristics, however there may be *animals* that eat only *leaves* that are not *giraffes*. 
Necessary and Sufficient Conditions

- Classes can be described in terms of necessary and sufficient conditions.
  - This differs from some frame-based languages where we only have necessary conditions.
- **Necessary** conditions
  - Must hold if an object is to be an instance of the class
- **Sufficient** conditions
  - Those properties an object must have in order to be recognised as a member of the class.
  - Allows us to perform automated classification.

Example

- An **animal_lover** is a **person** that has **at least 3 pets**.
- All of these **pets** must be **distinct** individuals.
- Any **person** with 5 **pets** will be inferred to be an instance of this class.
Example

- A newspaper is either a broadsheet or a tabloid.
- By default there is no mutual exclusion.
- If we know something is a newspaper we can infer that it must be either a broadsheet or a tabloid, but we may not know for sure which one it actually is (cf Open World).

Example

- Mick is an individual and an instance of the class male.
- He is related to individuals Daily_Mirror and Q123_ABC via the properties reads and drives.
Common Misconceptions

- Disjointness of primitives
- Interpreting domain and range
- And and Or
- Quantification
- Closed and Open Worlds

Disjointness

- By default, primitive classes are not disjoint.
- Unless we explicitly say so, the description \(\text{(Animal and Vegetable)}\) is not inconsistent.
- Similarly with individuals -- the so-called Unique Name Assumption (often present in DL languages) does not hold, and individuals are not considered to be distinct unless explicitly asserted to be so.
Domain and Range

- OWL allows us to specify the domain and range of properties.
- Note that this is not interpreted as a constraint.
- Rather, the domain and range assertions allow us to make inferences about individuals.
- Consider the following:
  - ObjectProperty: employs
    Domain: Company
    Range: Person
    Individual: IBM
    Facts: employs Jim
- If we haven’t said anything else about IBM or Jim, this is not an error. However, we can now infer that IBM is a Company and Jim is a Person.

And/Or and Quantification

- The logical connectives And and Or often cause confusion
  - Tea or Coffee?
  - Milk and Sugar?
- Quantification can also be contrary to our intuition.
  - Universal quantification over an empty set is true.
  - Sean is a member of hasChild only Martian
  - Existential quantification may imply the existence of an individual that we don’t know the name of.
Closed and Open Worlds

- The standard semantics of OWL makes an Open World Assumption (OWA).
  - We cannot assume that all information is known about all the individuals in a domain.
  - Facilitates reasoning about the intensional definitions of classes.
  - Sometimes strange side effects
- Closed World Assumption (CWA)
  - Named individuals are the only individuals in the domain
- Negation as failure.
  - If we can’t deduce that \( x \) is an \( A \), then we know it must be \( \neg A \).
  - Facilitate reasoning about a particular state of affairs.

OWL isn’t everything

- OWL is not intended to be the answer to all our problems.
- For some applications, less formal vocabularies may be more appropriate
- For some applications, more expressiveness may be needed.
Lightweight Vocabularies

- For many applications, lightweight representations are more appropriate.
- Thesauri, classification schemes, taxonomies and other controlled vocabularies
  - Many of these already exist and are in use in cultural heritage, library sciences, medicine etc.
  - Often have some taxonomic structure, but with a less precise semantics.

Concept Schemes

- A concept scheme is a set of concepts, potentially including statements about relationships between those concepts
  - Broader Terms
  - Narrower Terms
  - Related Terms
  - Synonyms, usage information etc.

- Concept schemes aren’t formal ontologies in the way that OWL ontologies are formal ontologies.
SKOS: Simple Knowledge Organisation System

- SKOS aims to provide an RDF vocabulary for the representation of such schemes.
- W3C Semantic Web Deployment Group currently working towards a Recommendation for SKOS
- Focus on Retrieval Scenarios
  A. Single controlled vocabulary used to index and then retrieve objects
  B. Different controlled vocabularies used to index and retrieve objects
    - Mappings then required between the vocabularies
    - Initial use cases/requirements focus on these tasks
    - Not worrying about activities like Natural Language translation
SKOS

- Semantic Web Deployment Working Group
  http://www.w3.org/2006/07/SWD/
- SKOS Reference:
  http://www.w3.org/TR/skos-reference/
- SKOS Primer
  http://www.w3.org/TR/skos-primer/
- Last Call Planned 1st July

Rules: RIF

- W3C Group Rules Interchange Format WG
  http://www.w3.org/2005/rules/
- Identification of Use Cases and Requirements
- Drafts identifying Basic Logic Dialect (BLD)
  - Horn rules plus equality
  - FOL Semantics
- Extensions defined using Framework for Logic Dialects (FLD)
- Somewhat slow moving process
  - Large and disparate group, e.g. production rules, business rules, FOL.
A number of domains require expressivity that is not in the current OWL specification

- Driven by User Requirements and technical advances
- OWLED series of workshops

Much of this functionality can be added in a principled way that preserves the desirable properties of OWL (DL).

The proposed extended language is now known as OWL 2

http://www.w3.org/2007/OWL

- Syntactic Sugar
  - DisjointUnion
  - Negated Property assertions
- Richer Datatypes
- Complex Role Axioms
  - Role inclusion
- Metamodelling and Annotations
  - Punning
- Tractable Fragments
  - Language fragments with desirable computational complexity
Many applications (for example medicine) have requirements to specify interactions between roles:
- A fracture located in part of the Femur is a fracture of the Femur.
- We cannot express such general patterns in OWL.
- Algorithms have been developed to support sound and complete reasoning in a DL extended with complex role inclusions.

OWL DL has strict rules about separation of namespaces.
- A URI cannot be typed as both a class and individual in the same ontology.
- OWL 2 allows punning, where a URI can be used in multiple roles.
  - However, the use of the URI as an individual has no bearing on the use of the URI as a class.
  - Requires explicit context telling us the role that a URI is playing.
OWL 2: Fragments

- **EL++**
  - Medical Ontologies
  - SNOMED/GALEN

- **DL Lite**
  - Tailored for handling large numbers of facts
  - Efficient Querying

- **DLP**
  - Subset of OWL DL and Horn Logic
  - OWL semantics

- **Horn-SHIQ**
  - Similar to DLP

- **RDF Schema**
  - RDFS ontologies that are valid OWL 2
Query and Retrieval

- In standard DLs, reasoning is split into:
  - T-Box: reasoning about classes
  - A-Box: reasoning about instances
- T-Box reasoning is well understood, at least for languages like SHIQ (~OWL Lite)
  - e.g. subsumption & satisfiability testing
- Full A-Box reasoning is much more challenging
  - E.g. instance retrieval & instantiation

Query Languages

- SPARQL Query Language for RDF.
  - http://www.w3.org/TR/rdf-sparql-query/
- SPARQL Protocol
  - http://www.w3.org/TR/rdf-sparql-protocol/
- W3C Recommendations as of 15th January
Tools

• Editors
  - Protégé OWL, SWOOP, ICOM, TopQuadrant Composer, OntoTrack, NeOn…
  - Tend to present the user with “frame-like” interfaces, but allow richer expressions
  - Offer the possibility of using reasoners.

• Reasoners
  - DL style reasoners based on tableaux algorithms
    • Racer, FaCT++, Pellet
  - Based on rules or F-logic
    • F-OWL, E-Wallet…..

• APIs and Frameworks
  - Jena, WonderWeb OWL-API, Protégé OWL API, OWLIM

Take Home

• Representation languages are needed to allow us to describe our annotations and semantic information
• RDF, RDF Schema, OWL, SKOS etc. all provide mechanisms for this, with greater or lesser degrees of formality
• Formal Semantics seen as important when unambiguous interpretations of expressions are required
  - Facilitates use of reasoning
• Tools, both research and commercial, now available to support these languages
Acknowledgements

- "If I have seen further, it is by standing on the shoulders of giants"
- Many thanks to all the people who I “borrowed” material from, in particular
  - Ian Horrocks, Frank van Harmelen, Alan Rector, Nick Drummond, Matthew Horridge, Uli Sattler, Bijan Parsia
- and thanks to all those that they borrowed material from!
  - Too many to mention...

Thank you!