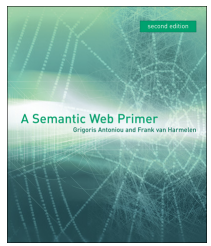
SemanticWeb - Ontologies in the Semantic Web

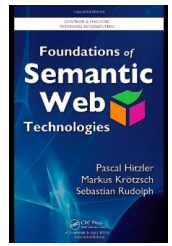
GEIST Research Group
http://geist.agh.edu.pl

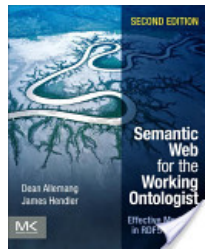
AGH University of Science and Technology, POLAND

Using slides according to license from:

- P. Hitzler – "Knowledge Representation for the Semantic Web" *course based on*
- P. Hitzler, M. Krötzsch, S. Rudolph – Foundations of Semantic Web Technologies
- e-Lite: 01LHVIU - Semantic Web: Technologies, Tools, Applications

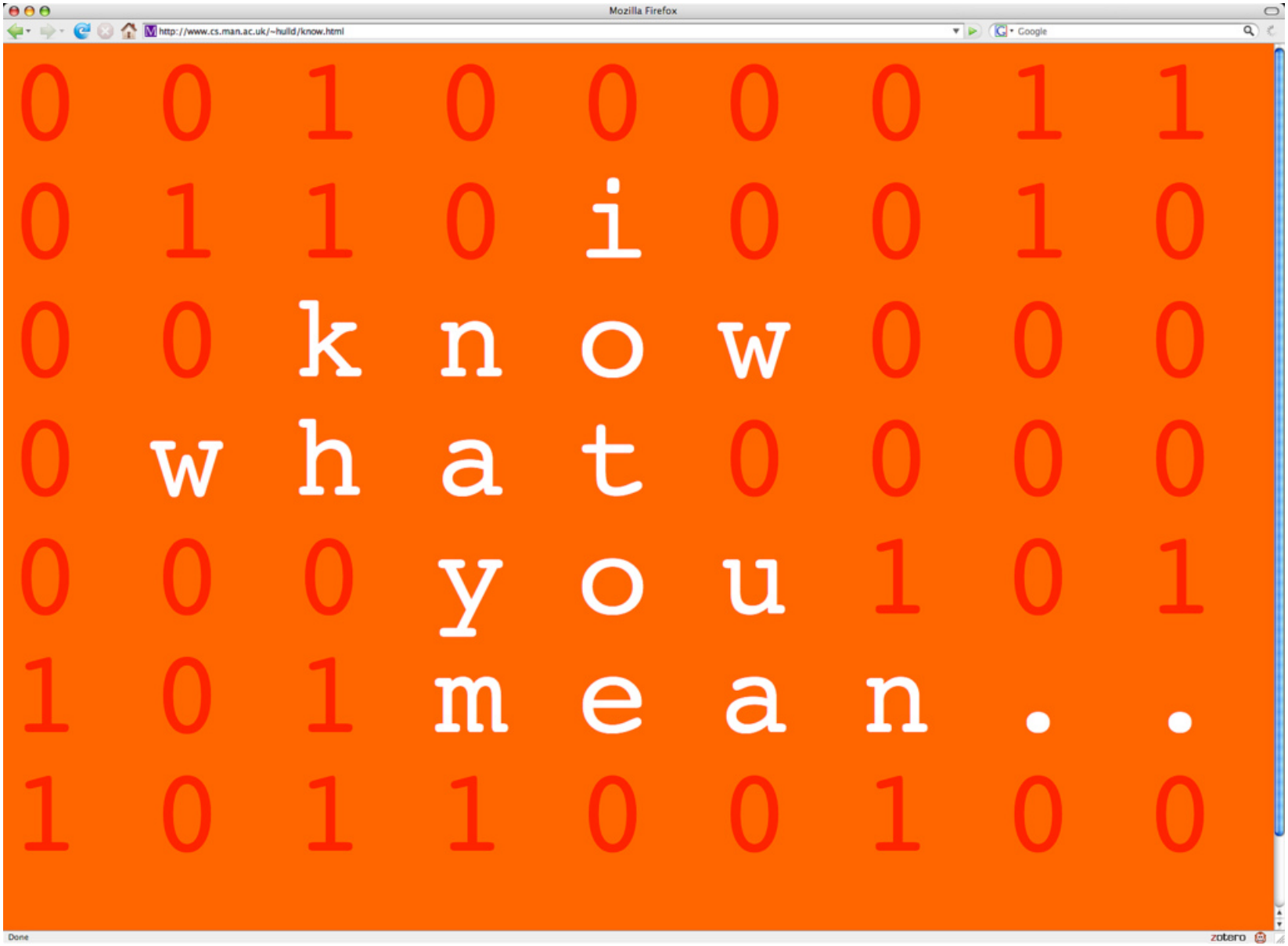






Outline

Outline



Need for knowledge representation

- Semantically rich descriptions need "understanding" the meaning of a resource and the domain related to the resource
  - Disambiguation of terms
  - Shared agreement on meanings
  - Description of the domain, with concepts and relations among concepts

Ontologies in Computer Science

An ontology

"Explicit specification of a shared conceptualization" – *T. Gruber*

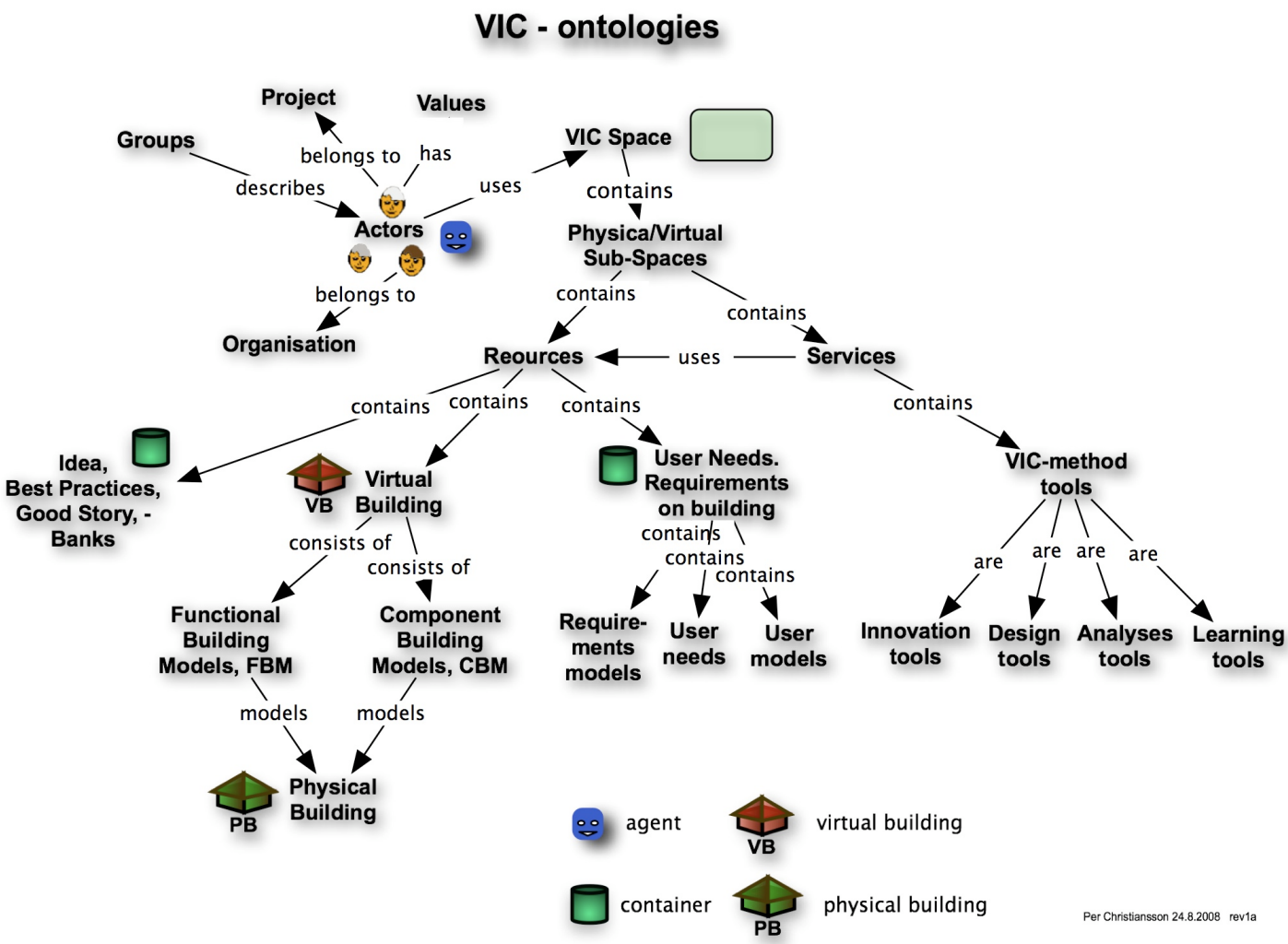
Elements of an ontology (engineering artefact)

- vocabulary + explicit specification of its intended meaning
- classification of objects
- constraints and relations

Objective of ontologies

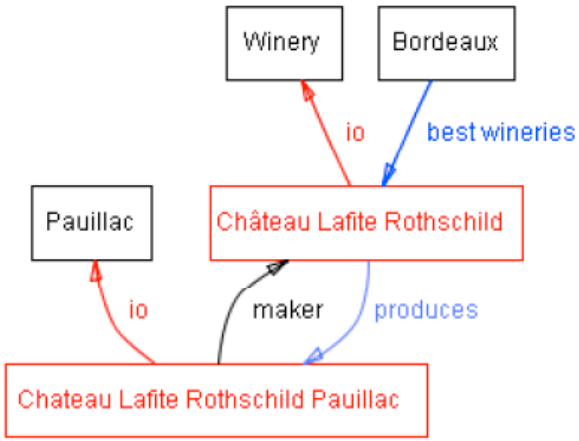
- Capture a shared understanding of a domain of interest
- Provide a formal and machine manipulable model of the domain

Ontology example



Virtual Innovation in Construction: https://arkitema.onconfluence.com/display/vicspace/Public+Space

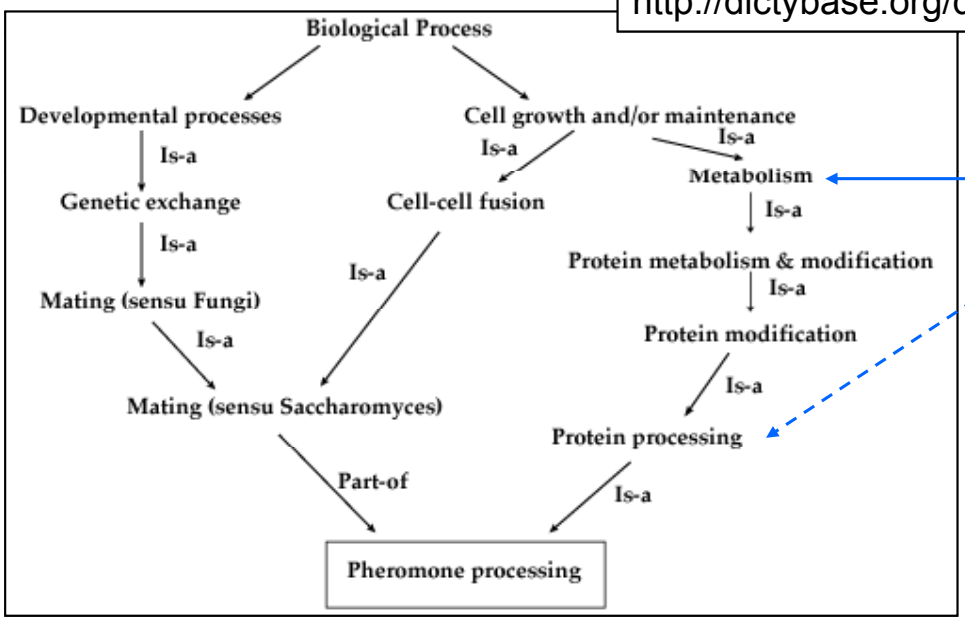
Ontology structure



- Concepts
- Relationships
  - Is-a
  - Other
- Instances

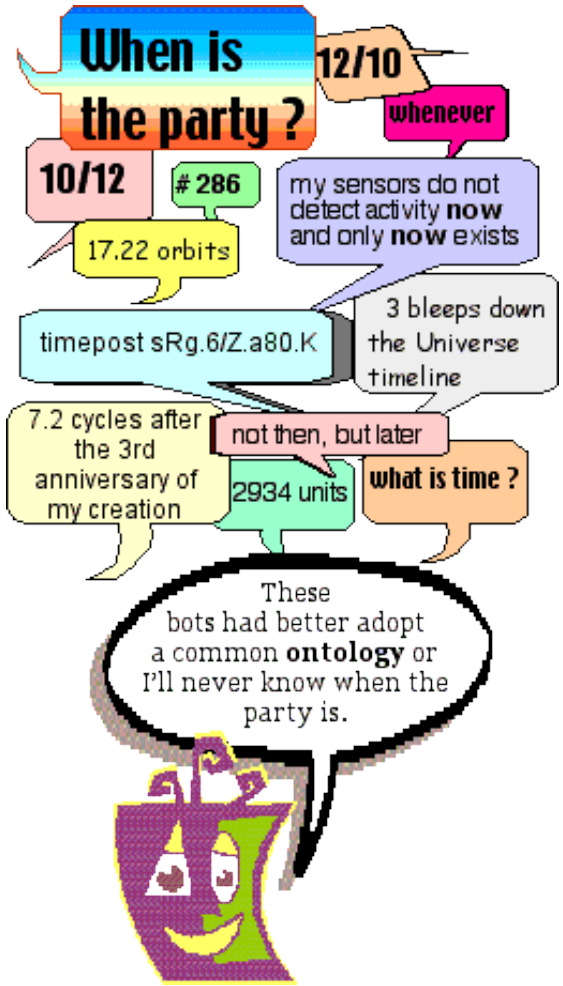
Semantically rich descriptions to support search

http://dictybase.org/db/html/help/GO.html



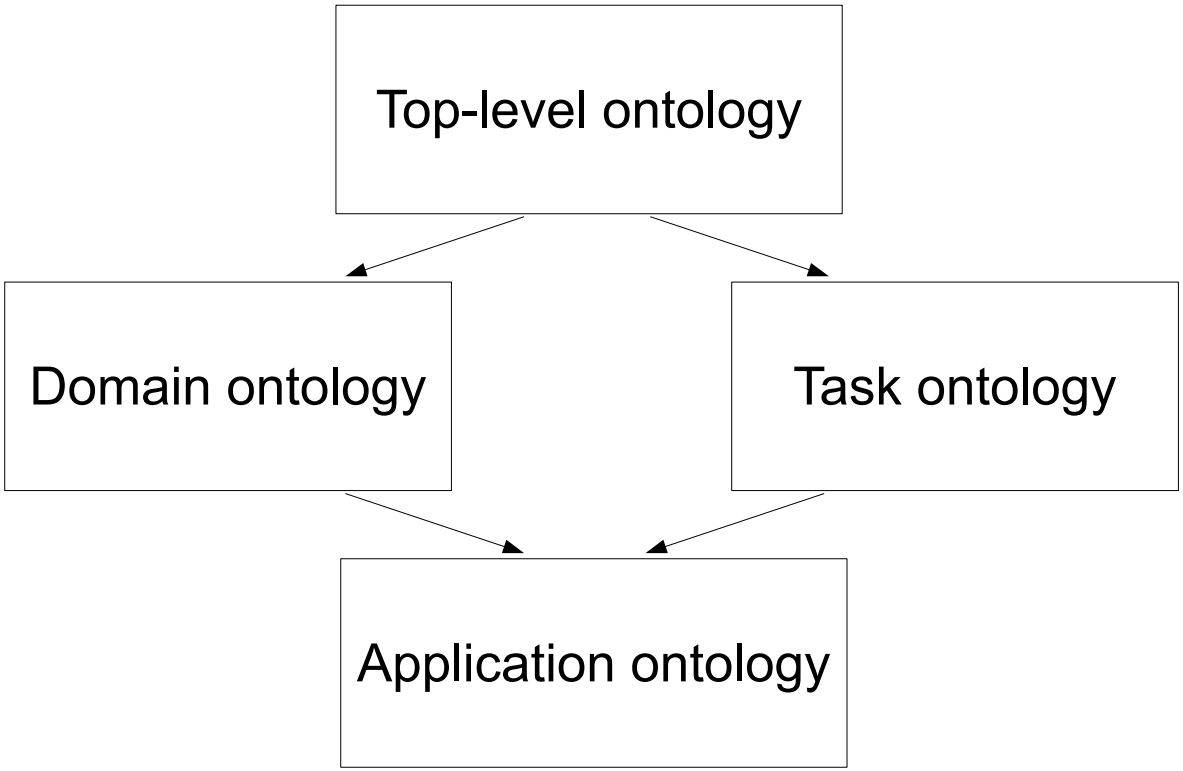
Topic = {metabolism, ...}

Upper level semantic tasks?

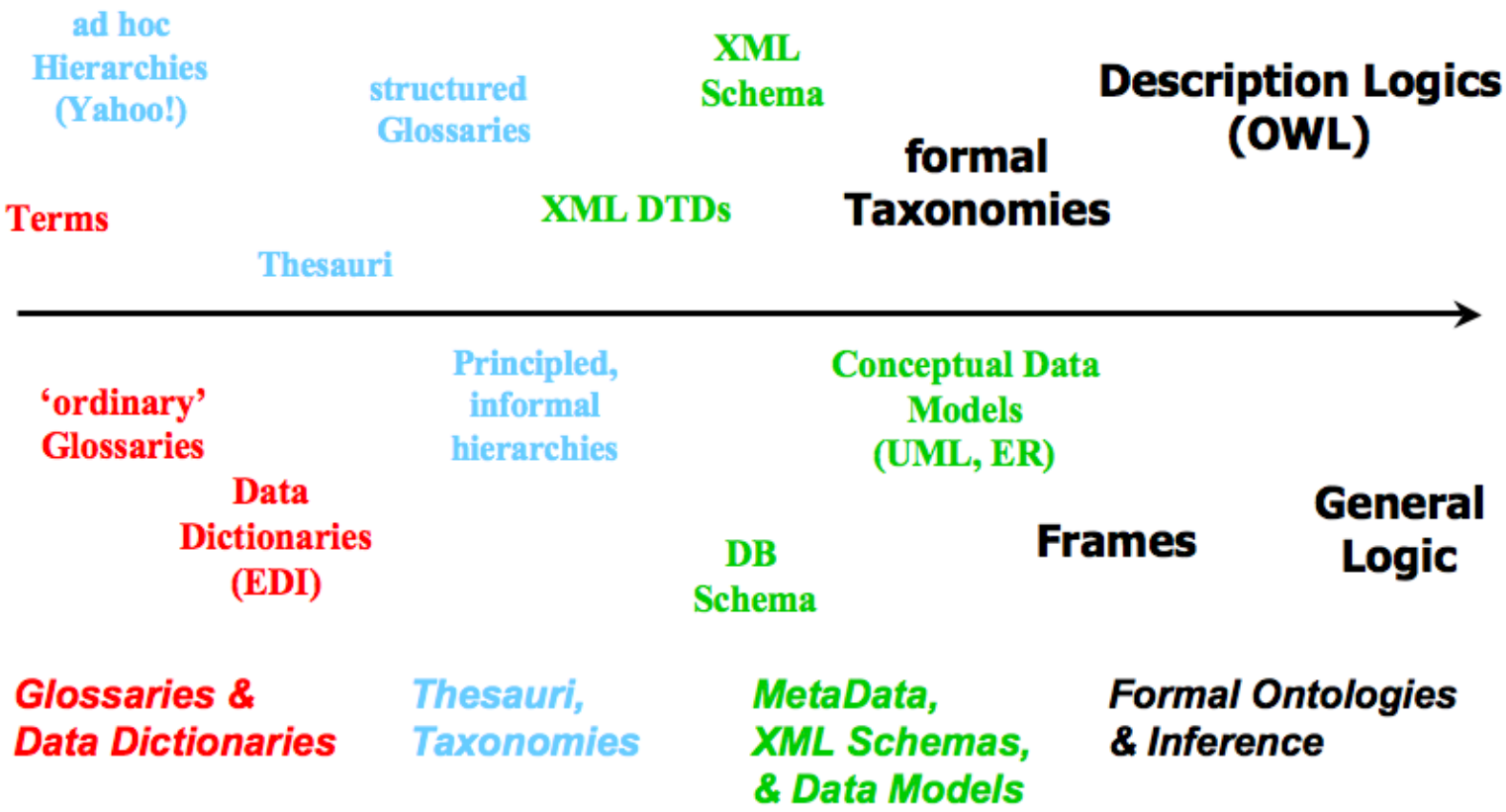


http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/Ontologies

Sorts of ontologies



Formality levels



Outline

Ontology languages for the Semantic Web

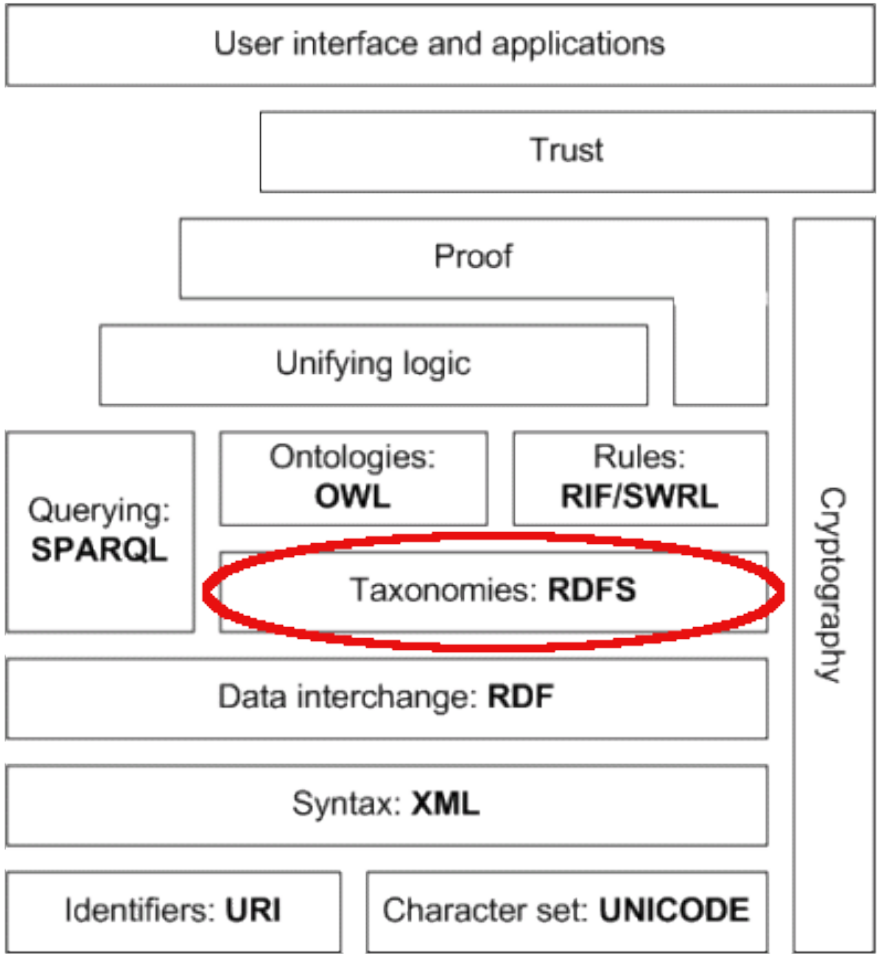
RDF Schema

- simple taxonomies, no complex relations
- RDF/XML syntax
- no syntax restrictions
- non-standard semantics

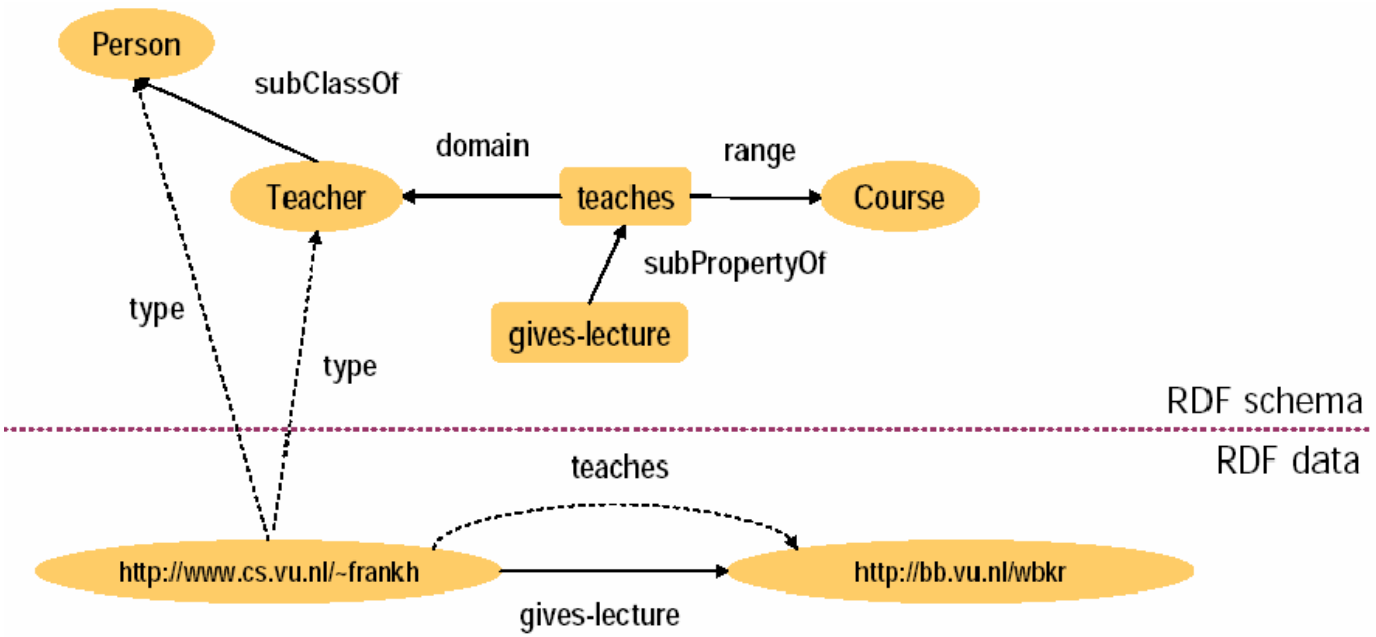
OWL – Web Ontology Language

- complex object properties, datatypes, properties axioms
- 3 sublanguages of OWL(1): OWL Lite, OWL DL, OWL Full
- OWL 2 Profiles: OWL EL, OWL QL, OWL RL
- **formal foundations (Description Logics)**

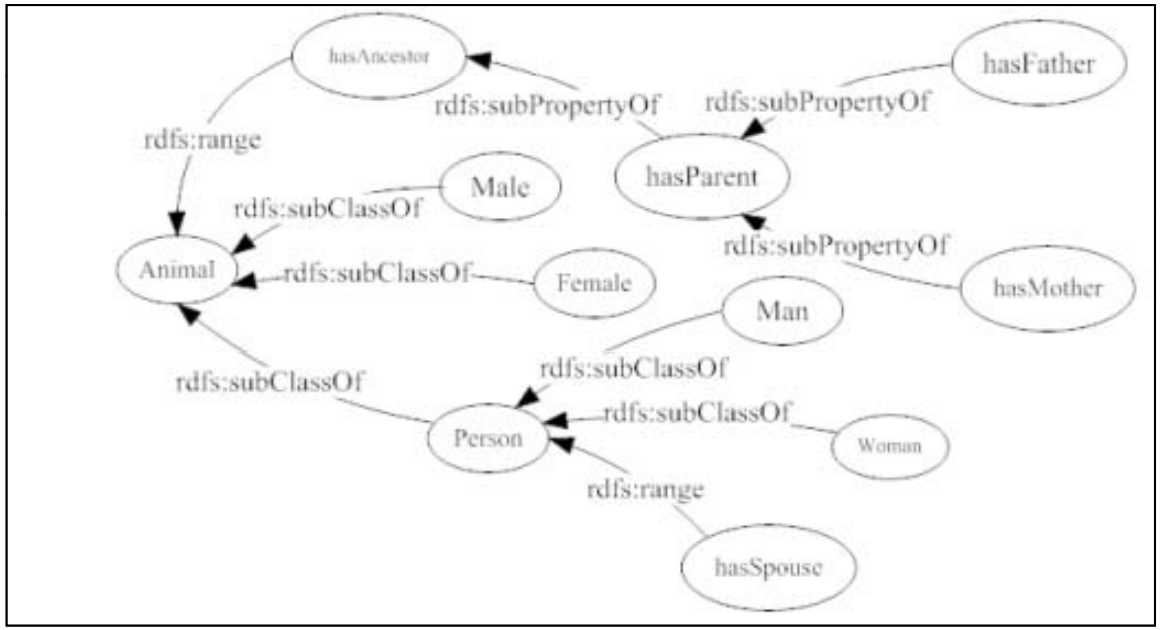
RDFS in the Semantic Web stack



RDF / RDF Schema



RDF Schema example



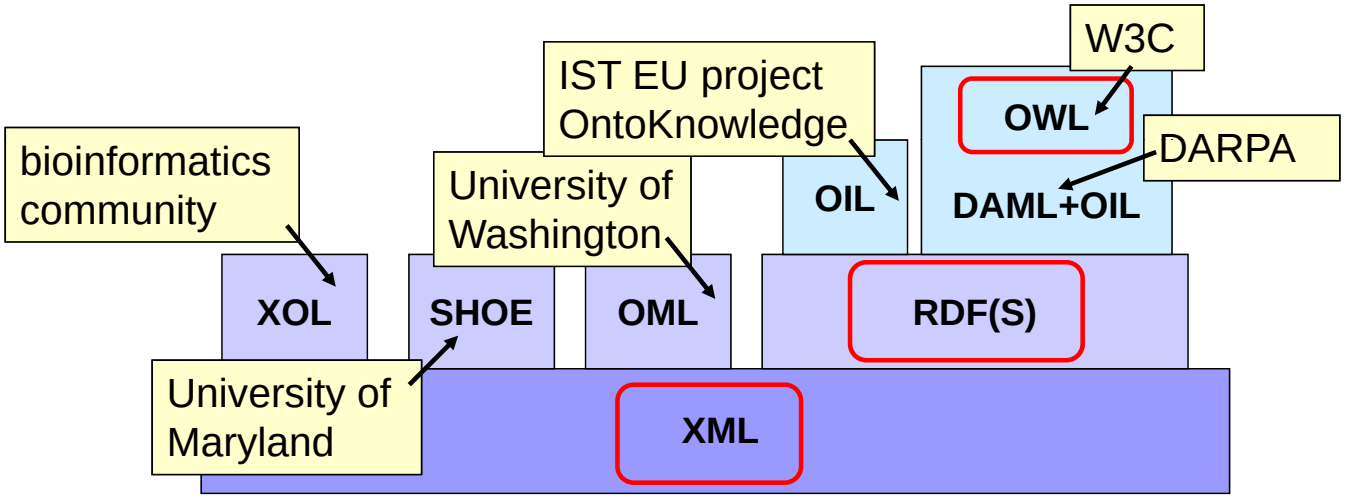
RDFS problems

- RDFS is too “weak” to describe resources with a suitable level of details
  - range and domain cannot be localized (e.g. the range of hasChild is a person when applied to a person, elephant when applied to an elephant)
  - no constraints on existence or cardinality (e.g. all instances of persons have one and only one mother which is a person, and have exactly two parents)
  - it is not possible to define transitive, inverse or symmetrical statements (e.g. part of is a transitive property, hasPart is the inverse of isPartOf, touches is symmetrical)
- Reasoning is not well supported
  - Non standard semantics, no native reasoner exists

Requirements for an ontology language

- Extend existing Web standards
  - XML, RDF, RDFS, ...
- Easy to understand and to use
  - based on well known knowledge representation (KR) languages
- Formally specified
- Adequate expressive power
- Automatic support for reasoning

“History” of Web languages



Web Ontology Language (OWL)

- Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things
- Computational logic-based language such that knowledge expressed in OWL can be reasoned with by computer programs either to verify the consistency of that knowledge or to make implicit knowledge explicit
- OWL documents, known as ontologies, can be published in the World Wide Web and may refer to or be referred from other OWL ontologies
- OWL is not a programming language
  - it is declarative, i.e. it describes a state of affairs in a logical way
- OWL: W3C Recommendation, Feb 10th 2004
- OWL 2: Revised W3C Recommendation, October 27th 2009

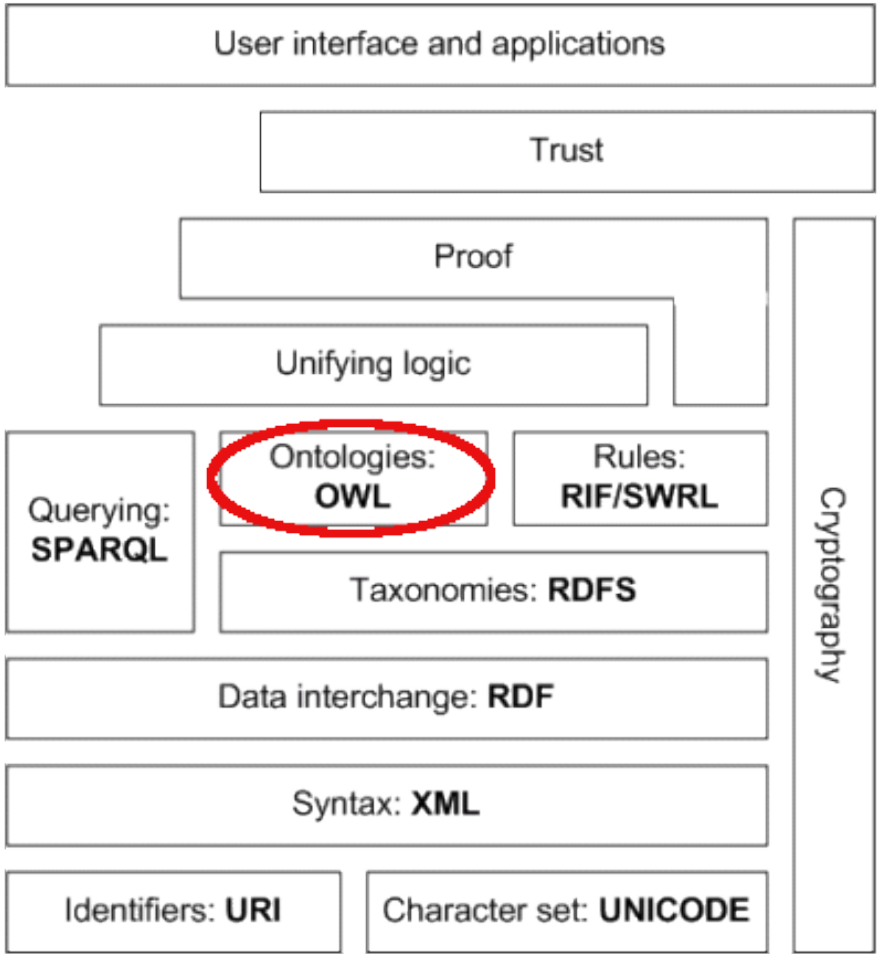
Web Ontology Language (OWL)

- More expressive than RDFS
  - Identity equivalence/difference
  - sameAs, differentFrom, equivalentClass/Property
- More expressive class definitions
  - Class intersection, union, complement, disjointness
  - Cardinality restrictions
- More expressive property definitions
  - Object/Datatype properties
  - Transitive, functional, symmetric, inverse properties
  - Value restrictions

Web Ontology Language (OWL)

- What can be done with OWL?
  - Consistency checks – are there contradictions in the logical model?
  - Satisfiability checks – are there classes that cannot have any instances?
  - Classification – what is the type of a particular instance?

OWL in the Semantic Web stack



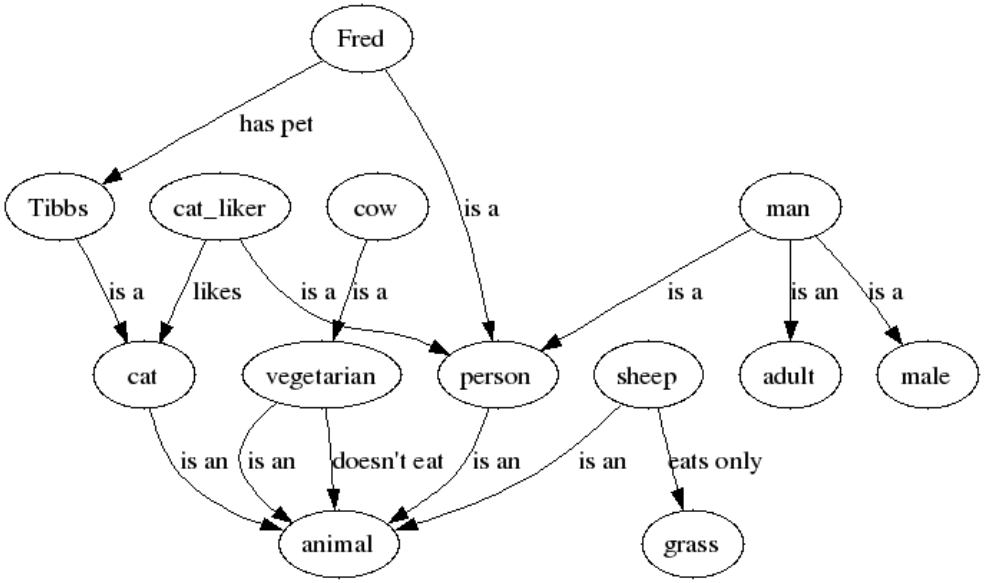
Outline

Description Logics

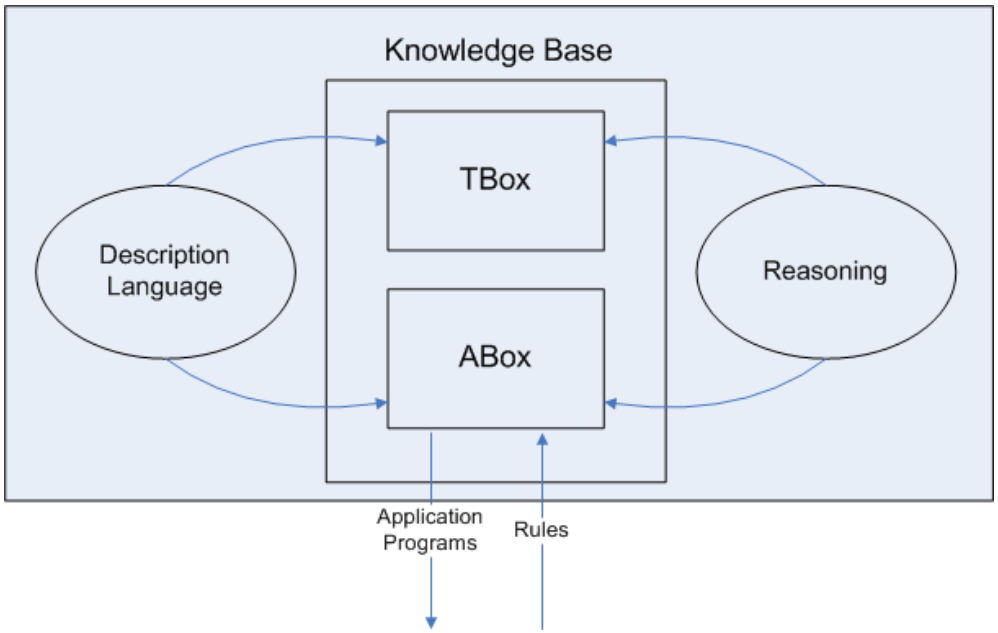
- Logical foundations for Semantic Web ontologies
- Terminological knowledge, taxonomies, complex relations
- Modelling: concepts (classes), roles (properties), objects (individuals)
- Related to: semantic networks & frames
- Formal semantics $\rightarrow$ automated inference
- Various DLs: $\mathcal{ALC}$, $\mathcal{SHOIN}(\mathbf{D})$, $\mathcal{SROIQ}$ – different expressive power

Description Logics examples

- $Man \equiv Person \sqcap Adult \sqcap Male$,
- $CatLiker \sqsubseteq \exists likes.Cat$, $Sheep \sqsubseteq \forall eats.Grass$,
- $Person(fred)$, $Cat(tibbs)$, $hasPet(fred, tibbs)$



Knowledge Representation in DL



Knowledge Base

- TBox – terminology: axioms and definitions
  - $Man \equiv Person \sqcap Adult \sqcap Male$, $CatLiker \sqsubseteq \exists likes.Cat$,
- ABox – world description: concept and role assertions
  - $Person(fred)$, $Cat(tibbs)$, $hasPet(fred, tibbs)$

DL syntax RDFS syntax

- Person(mary)
  - :mary rdf:type :Person .
- Woman $\sqsubseteq$ Person
  - :Woman rdfs:subClassOf :Person .
  - Person $\equiv$ HumanBeing (class equivalence):
    Person $\sqsubseteq$ HumanBeing AND
    HumanBeing $\sqsubseteq$ Person
- hasWife(john,mary)
  - :john :hasWife :mary .
- hasWife $\sqsubseteq$ hasSpouse
  - :hasWife rdfs:subPropertyOf :hasSpouse
  - hasSpouse $\equiv$ marriedWith (class equivalence)

DL syntax FOL syntax

ABox statements

- Person(mary) • Person(mary)
- Woman ⊑ Person • ∀x (Woman(x) → Person(x))
  - Person ≡ HumanBeing (class equivalence)
- hasWife(john,mary) • hasWife(john,mary)
- hasWife ⊑ hasSpouse • ∀x ∀y (hasWife(x,y) → hasSpouse(x,y))
  - hasSpouse ≡ marriedWith (class equivalence)

TBox statements

Special classes and properties

- owl:Thing (RDF syntax)
  - DL-syntax: $\top$
  - contains everything
- owl:Nothing (RDF syntax)
  - DL-syntax: $\bot$
  - empty class
- owl:topProperty (RDF syntax)
  - DL-syntax: U
  - every pair is in U
- owl:bottomProperty (RDF syntax)
  - empty property

Class constructors

- **conjunction**

  $\forall x\,(\text{Mother}(x) \leftrightarrow \text{Woman}(x) \wedge \text{Parent}(x))$

  - **Mother $\equiv$ Woman $\sqcap$ Parent**
    „Mothers are exactly those who are women and parents.“

- **disjunction**

  $\forall x\,(\text{Parent}(x) \leftrightarrow \text{Mother}(x) \vee \text{Father}(x))$

  - **Parent $\equiv$ Mother $\sqcup$ Father**
    „Parents are exactly those who are mothers or fathers.“

- **negation**

  $\forall x\,(\text{ChildlessPerson}(x) \leftrightarrow \text{Person}(x) \wedge \neg\text{Parent}(x))$

  - **ChildlessPerson $\equiv$ Person $\sqcap \neg$Parent**
    „ChildlessPersons are exactly those who are persons and who are not parents.“

Class constructors

- **existential quantification**
  - **only to be used with a role – also called a *property restriction***
  - **Parent ≡ ∃hasChild.Person**
    „Parents are exactly those who have at least one child which is a Person.“

$$\forall x\ (\text{Parent}(x) \leftrightarrow \exists y\ (\text{hasChild}(x,y) \wedge \text{Person}(y)))$$

- **universal quantification**
  - **only to be used with a role – also called a *property restriction***
  - **Person ⊓ Happy ≡ ∀hasChild.Happy**
    „A (person which is also happy) is exactly (something all children of which are happy).“

$$\forall x\ (\text{Person}(x) \wedge \text{Happy}(x) \leftrightarrow \forall y\ (\text{hasChild}(x,y) \rightarrow \text{Happy}(y)))$$

- **Class constructors can be nested arbitrarily**

The Description Logic ALC

The description logic ALC

Complexity: ExpTime

- **ABox expressions:**
  Individual assignments **Father(john)**
  Property assignments **hasWife(john,mary)**

- **TBox expressions**
  subclass relationships $\sqsubseteq$ $\equiv$ **for equivalence**

  conjunction $\sqcap$
  disjunction $\sqcup$
  negation $\neg$ Also: $\top$, $\bot$

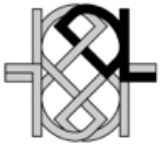
  property restrictions $\forall$
  $\exists$

SROIQ(D) constructors – overview

- ABox assignments of individuals to classes or properties
- ALC: $\sqsubseteq$, $\equiv$ for classes
  $\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$
  $\top$, $\bot$
- SR: + property chains, property characteristics, role hierarchies $\sqsubseteq$
- SRO: + nominals {o}
- SROI: + inverse properties
- SROIQ: + qualified cardinality constraints
- SROIQ(D): + datatypes (including facets)

- + top and bottom roles (for objects and datatypes)
- + disjoint properties
- + Self
- + Keys (not in SROIQ(D), but in OWL)

Complexity of reasoning in DLs



Complexity of reasoning in Description Logics

Note: the information here is (always) incomplete and updated often

Base description logic: *A*ttributive *L*anguage with *C*omplements

$ALC ::= \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C$

Concept constructors:

- ☐ *F* – functionality[2]: $(\leq 1\ R)$
- ☐ *N* – (unqualified) number restrictions: $(\geq n\ R)$, $(\leq n\ R)$
- ☐ *Q* – qualified number restrictions: $(\geq n\ R.C)$, $(\leq n\ R.C)$
- ☐ *O* – nominals: $\{a\}$ or $\{a_1, \ldots, a_n\}$ ("one-of")
- ☐ μ – least fixpoint operator: $\mu X.C$

Forbid complex roles[5] in number restrictions[6]

Role constructors: trans reg

- ☐ *I* – role inverse: R^-
- ☐ ∩ – role intersection[3]: $R \cap S$
- ☐ ∪ – role union: $R \cup S$
- ☐ ¬ – role complement: $\neg R$ full
- ☐ o – role chain (composition): $R \circ S$
- ☐ * – reflexive-transitive closure[4]: R^*
- ☐ *id* – concept identity: $id(C)$

TBox (concept axioms):

- ◉ empty TBox
- ○ acyclic TBox ($A \equiv C$, A is a concept name; no cycles)
- ○ general TBox ($C \sqsubseteq D$, for arbitrary concepts C and D)

RBox (role axioms): OWL-Lite OWL-DL OWL 1.1

- ☐ *S* – role transitivity: $\mathrm{Tr}(R)$
- ☐ *H* – role hierarchy: $R \sqsubseteq S$
- ☐ *R* – complex role inclusions: $R \circ S \sqsubseteq R$, $R \circ S \sqsubseteq S$
- ☐ *S* – some additional features (check it to see)

Reset You have selected a Description Logic: *ALC*

Complexity of reasoning problems[7]

Reasoning problem	Complexity[8]	Comments and references
Concept satisfiability	**PSpace-complete**	• Hardness for *ALC*: see [73]. • Upper bound for *ALCQ*: see [77, Theorem 4.6].
ABox consistency	**PSpace-complete**	• Hardness follows from that for concept satisfiability. • Upper bound for *ALCQO*: see [7, Appendix A].

See http://www.cs.man.ac.uk/~ezolin/dl/

Outline

Outline

DL and OWL

- OWL exploits results of 15+ years of DL research
  - Well defined (model theoretic) semantics
  - Formal properties well understood (complexity, decidability)
  - Known reasoning algorithms
  - Implemented systems (highly optimised)
- OWL ontology equivalent to DL KB (TBox + ABox)

OWL basics

- Statements in OWL normally refer to objects of the world and describe them by putting them into categories (like “Mary is female”) or saying something about their relation (“John and Mary are married”)
- All atomic constituents of statements, be they objects (John, Mary), categories (female) or relations (married) are called entities
- In OWL 2
  - objects are called “individuals”
  - categories are called “classes”
  - relations are called “properties”

OWL basics

- Properties in OWL 2 are further subdivided
  - Object properties relate objects to objects (like a person to their spouse)
  - Datatype properties assign data values to objects (like an age to a person)
  - Annotation properties are used to encode information about (parts of) the ontology itself (like the author and creation date of an axiom) instead of the domain of interest

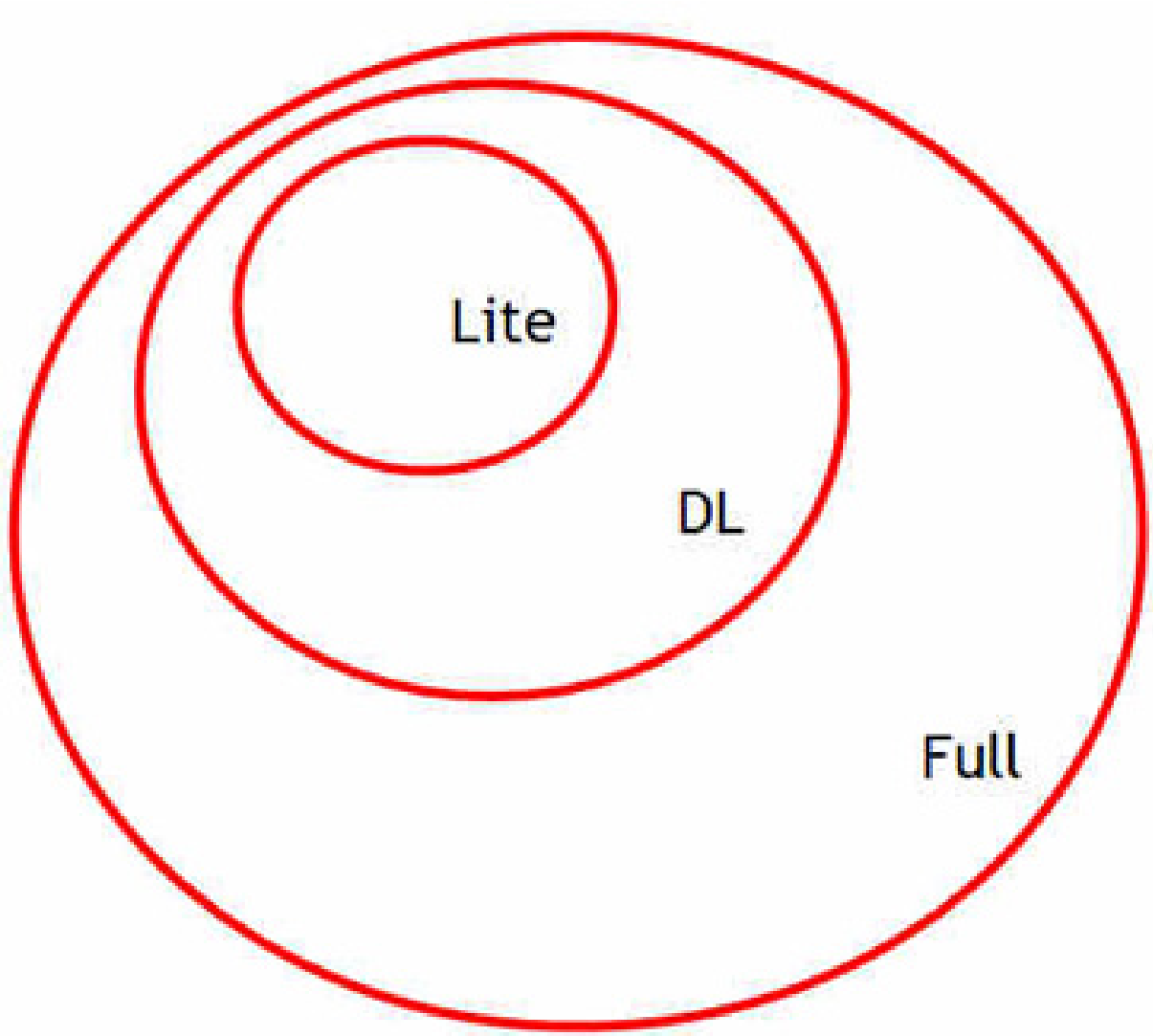
OWL basics

- Names of entities can be combined into expressions using so called constructors
  - As a basic example, the atomic classes “female” and “professor” could be combined conjunctively to describe the class of female professors
  - The latter would be described by an OWL class expression, that could be used in statements or in other expressions
- Expressions can be seen as new entities which are defined by their structure
  - In OWL, the constructors for each sort of entity vary greatly
  - The expression language for classes is very rich and sophisticated
  - The expression language for properties is much less so

Outline

OWL species

OWL (1) languages



3 species of OWL(1)

OWL Full

- union of OWL syntax and RDF
- RDF semantics extended with relevant semantic conditions and axiomatic triples

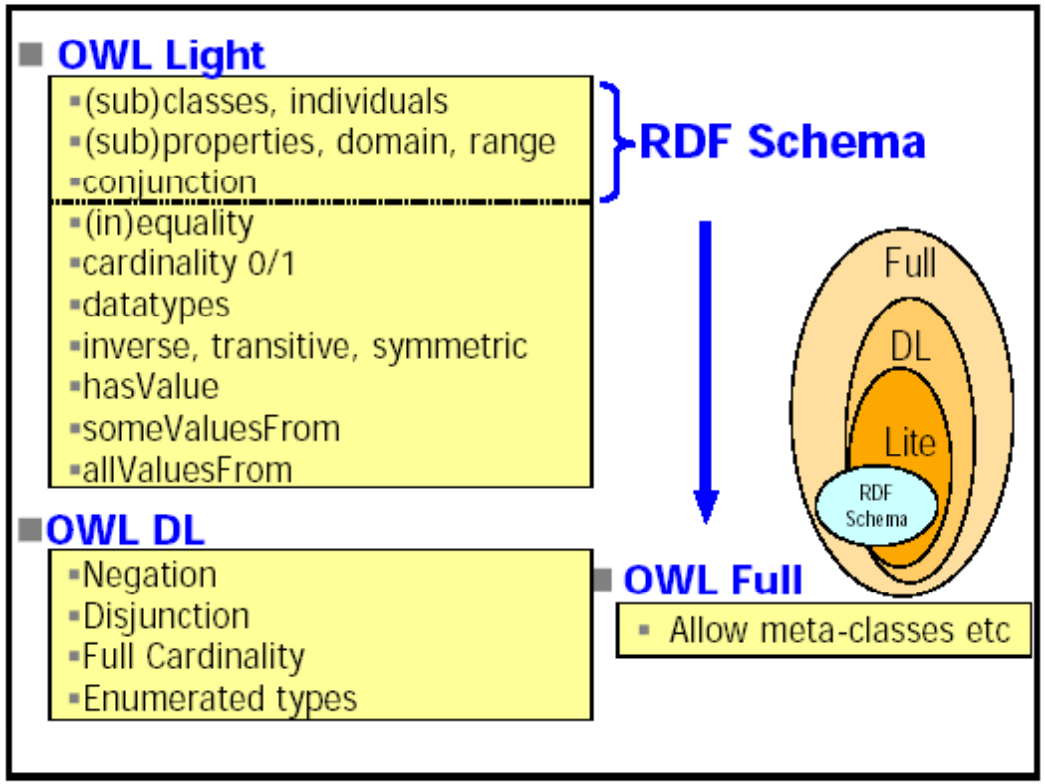
OWL DL

- restricted to FOL fragment
- standard (First Order) model theoretic semantics
- equivalent to $\mathcal{SHOIN}(\mathbf{D})$ DL

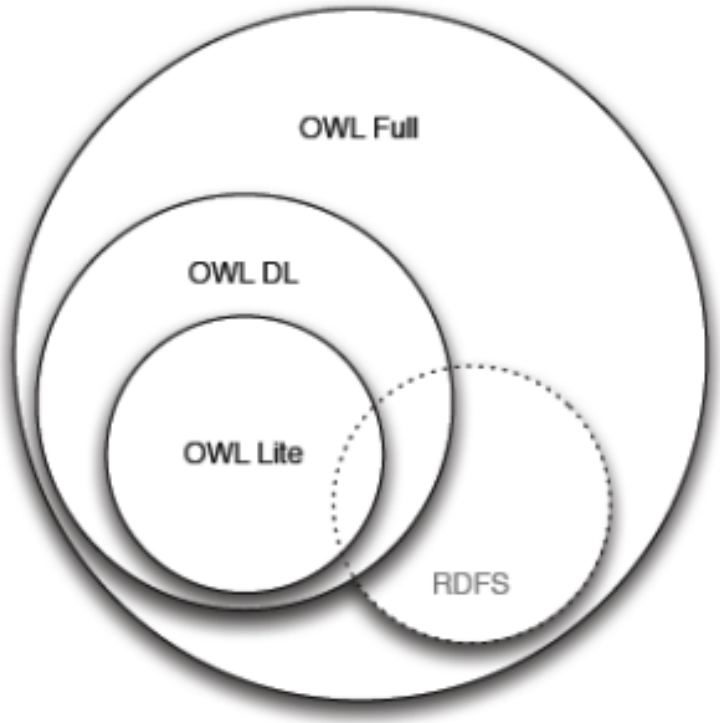
OWL Lite

- restricted subset of OWL DL

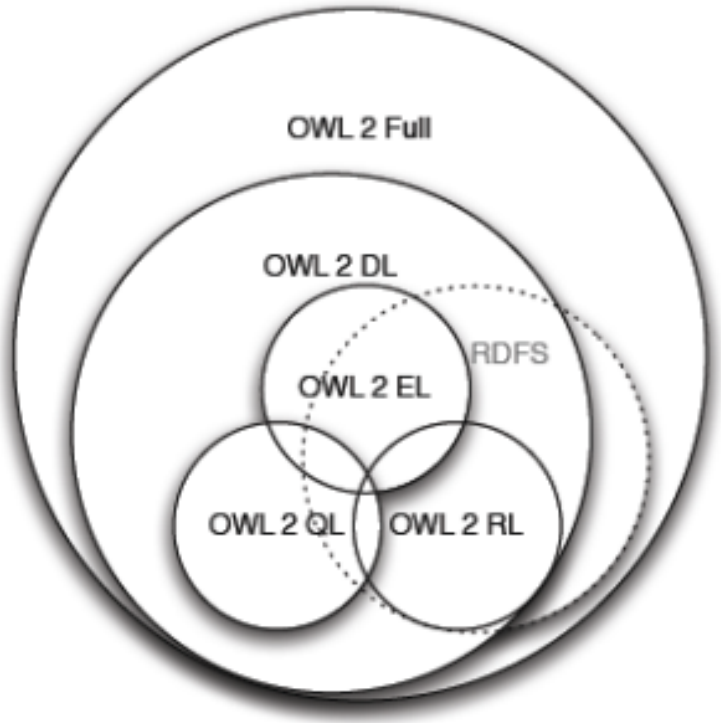
OWL1



OWL 1 vs. OWL 2



OWL 1



OWL 2 Profiles

3 Profiles of OWL 2

OWL 2 EL

- multiple classes/properties
- dedicated tools (e.g. CEL), SNOMED commercial ontology: `http://www.ihtsdo.org/snomed-ct/`

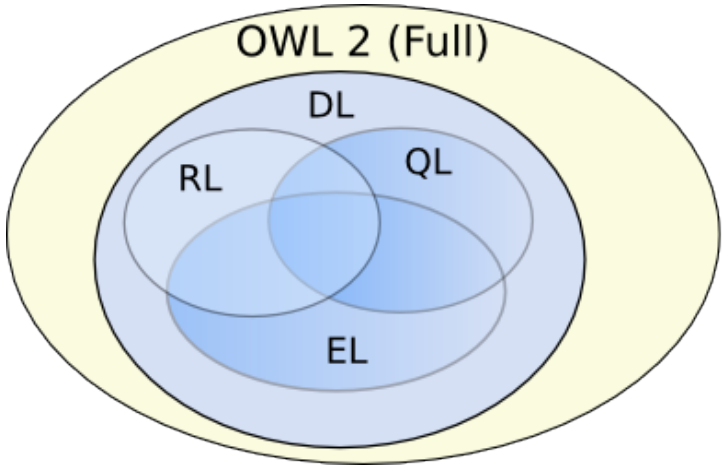
OWL 2 QL

- multiple instances, querying as the most important task
- limited expressive power (but includes most of the UML, ER)
- querying can be implemented with use of standard RDBS

OWL 2 RL

- scalable reasoning, but still expressive
- reasoning can be implemented with rule-based engines
- implemented in Oracle 11g

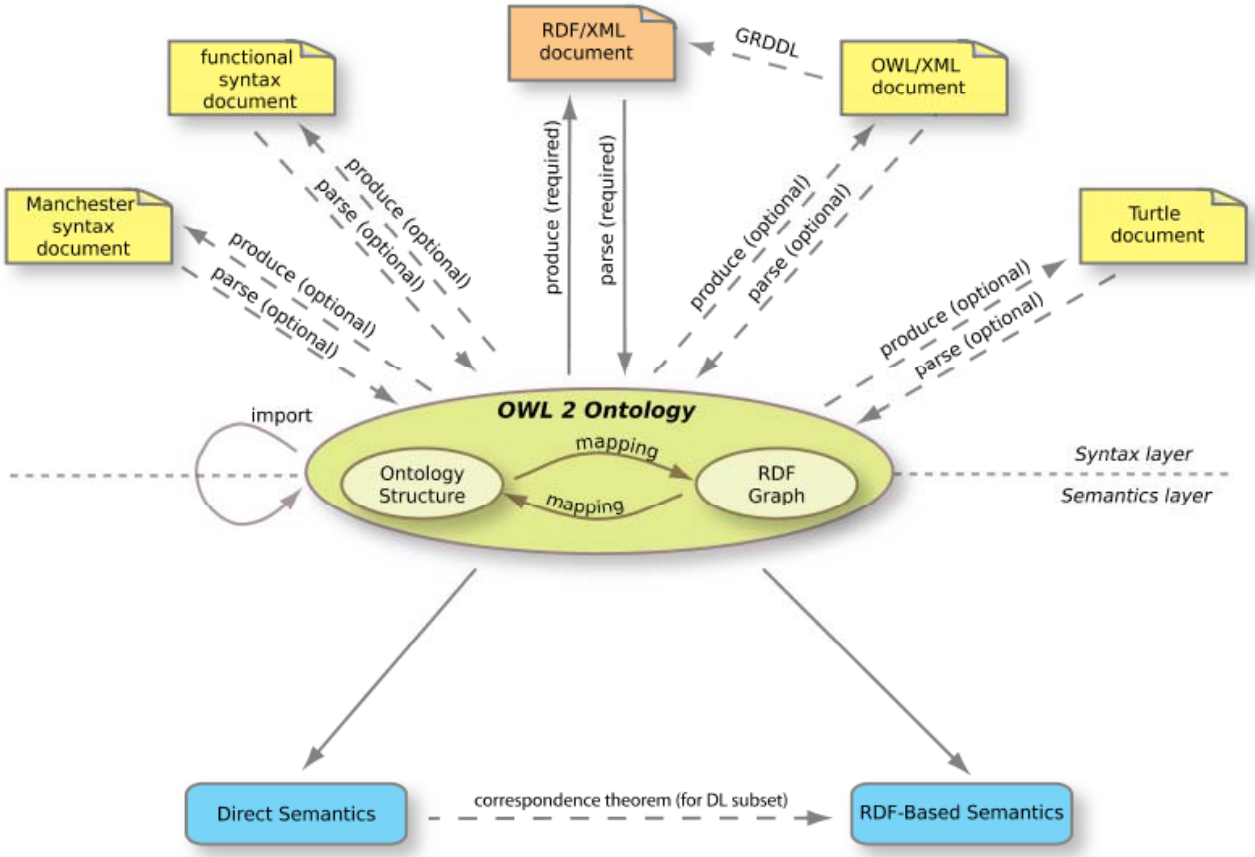
OWL 2 languages



OWL 2 Profiles (EL, QL, RL)

Reasoning in **polynomial time** with respect to the size of data
→ See `http://www.w3.org/TR/owl2-profiles/`

OWL2 structure



OWL2 syntaxes

- Various syntaxes available for OWL, to serve various purposes
- Functional-Style syntax
  - Designed to be easier for specification purposes and to provide a foundation for the implementation of OWL 2 tools such as APIs and reasoners
- RDF/XML syntax
  - Just RDF/XML, with a particular translation for the OWL constructs
  - This is the only syntax that is mandatory to be supported by all OWL 2 tools
- Turtle syntax
  - Turtle serializations for the RDF-based syntax

OWL2 syntaxes

- Manchester syntax
  - Designed to be easier for non-logicians to read
- OWL/XML syntax
  - an XML syntax for OWL defined by an XML schema
- There are tools that can translate between the different syntaxes

OWL2 semantics

- The Direct Semantics and the RDF-Based Semantics provide two alternative ways of assigning meaning to OWL 2 ontologies
  - A correspondence theorem provides a link between the two
- These two semantics are used by reasoners and other tools to answer class consistency, subsumption, instance retrieval queries, …

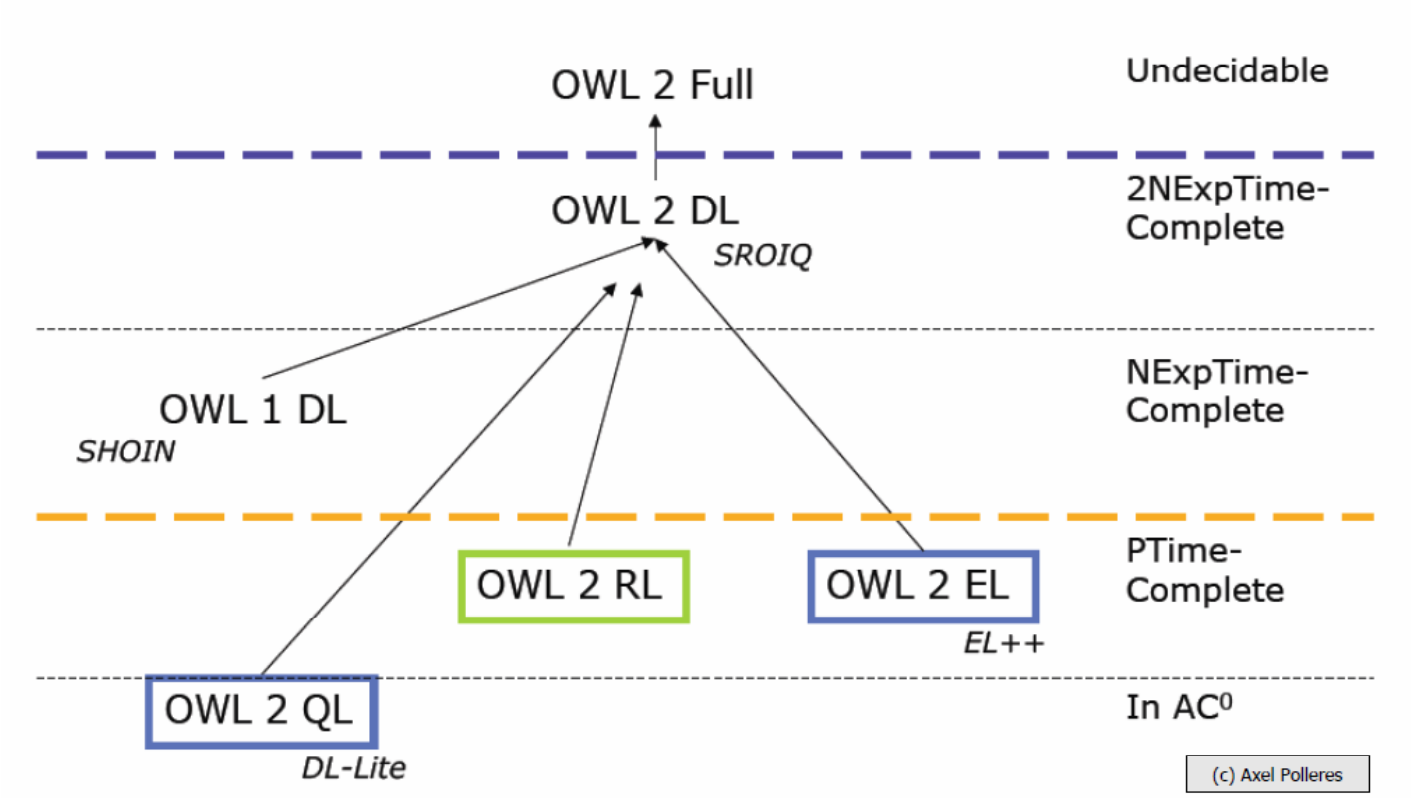
OWL 2 direct semantics

- Assigns meaning directly to ontology structures, resulting in a semantics compatible with the model theoretic semantics of the SROIQ description logic
  - SROIQ is a fragment of first order logic with useful computational properties
- The advantage of this close connection is that the extensive description logic literature and implementation experience can be directly exploited by OWL 2 tools
- However, some conditions must be placed on ontology structures in order to ensure that they can be translated into a SROIQ knowledge base
  - E.g, transitive properties cannot be used in number restrictions
- Ontologies that satisfy these syntactic conditions are called OWL 2 DL ontologies

OWL 2 RDF-based semantics

- Assigns meaning directly to RDF graphs and so indirectly to ontology structures via the Mapping to RDF graphs
- The RDF-Based Semantics is fully compatible with the RDF Semantics, and extends the semantic conditions defined for RDF
- The RDF-Based Semantics can be applied to any OWL 2 Ontology, without restrictions, as any OWL 2 Ontology can be mapped to RDF
- "OWL 2 Full" is used informally to refer to RDF graphs considered as OWL 2 ontologies and interpreted using the RDF-Based Semantics

OWL2 semantics and profiles overview



Outline

Classes and instances

- Example: “Mary is a person”
- Functional-Style Syntax

```
ClassAssertion( :Person :Mary )
```

- RDF/XML Syntax

```
<Person rdf:about="Mary"/>
```

- Turtle Syntax

```
:Mary rdf:type :Person .
```

- Manchester Syntax

```
Individual: Mary
  Types: Person
```

- OWL/XML Syntax

```
<ClassAssertion>
  <Class IRI="Person"/>
  <NamedIndividual IRI="Mary"/>
</ClassAssertion>
```

Class hierarchies

- Example: "Woman is a subclass of Person"
- Functional-Style Syntax

```
SubClassOf( :Woman :Person )
```

- RDF/XML Syntax

```
<owl:Class rdf:about="Woman">
  <rdfs:subClassOf rdf:resource="Person"/>
</owl:Class>
```

- Turtle Syntax

```
:Woman rdfs:subClassOf :Person .
```

- Manchester Syntax

```
Class: Woman
  SubClassOf: Person
```

- OWL/XML Syntax

```
<SubClassOf>
  <Class IRI="Woman"/>
  <Class IRI="Person"/>
</SubClassOf>
```

Class equivalence

- Example: "Person and Human are semantically equivalent"
- Functional-Style Syntax

```
EquivalentClasses( :Person :Human )
```

- RDF/XML Syntax

```
<owl:Class rdf:about="Person">
  <owl:equivalentClass rdf:resource="Human"/>
</owl:Class>
```

- Turtle Syntax

```
:Person owl:equivalentClass :Human .
```

- Manchester Syntax

```
Class: Person
  EquivalentTo: Human
```

- OWL/XML Syntax

```
<EquivalentClasses>
  <Class IRI="Person"/>
  <Class IRI="Human"/>
</EquivalentClasses>
```

Class disjointness

- Example: "Man and women are disjoint classes"
- Functional-Style Syntax

```
DisjointClasses( :Woman :Man )
```

- RDF/XML Syntax

```
<owl:AllDisjointClasses>
  <owl:members rdf:parseType="Collection">
    <owl:Class rdf:about="Woman"/>
    <owl:Class rdf:about="Man"/>
  </owl:members>
</owl:AllDisjointClasses>
```

- Turtle Syntax

```
[] rdf:type owl:AllDisjointClasses ;
   owl:members ( :Woman  :Man ) .
```

- Manchester Syntax

```
DisjointClasses: Woman, Man
```

- OWL/XML Syntax

```
<DisjointClasses>
  <Class IRI="Woman"/>
  <Class IRI="Man"/>
</DisjointClasses>
```

Outline

Object properties

- Example: “Mary is John’s wife”
- Functional-Style Syntax

```
ObjectPropertyAssertion( :hasWife :John :Mary )
```

- RDF/XML Syntax

```
<rdf:Description rdf:about="John">
  <hasWife rdf:resource="Mary"/>
</rdf:Description>
```

- Turtle Syntax

```
:John :hasWife :Mary .
```

- Manchester Syntax

```
Individual: John
  Facts: hasWife Mary
```

- OWL/XML Syntax

```
<ObjectPropertyAssertion>
  <ObjectProperty IRI="hasWife"/>
  <NamedIndividual IRI="John"/>
  <NamedIndividual IRI="Mary"/>
</ObjectPropertyAssertion>
```

Object properties – negative assertion

- Example: "Mary is not Bill's wife"
- Functional-Style Syntax

```
NegativeObjectPropertyAssertion( :hasWife :Bill :Mary )
```

- RDF/XML Syntax

```
<owl:NegativePropertyAssertion>
  <owl:sourceIndividual rdf:resource="Bill"/>
  <owl:assertionProperty rdf:resource="hasWife"/>
  <owl:targetIndividual rdf:resource="Mary"/>
</owl:NegativePropertyAssertion>
```

- Turtle Syntax

```
[] rdf:type owl:NegativePropertyAssertion ;
   owl:sourceIndividual  :Bill ;
   owl:assertionProperty  :hasWife ;
   owl:targetIndividual  :Mary .
```

Object properties – negative assertion

- Example: “Mary is not Bill’s wife”
- Manchester Syntax

```
Individual: Bill
  Facts: not hasWife Mary
```

- OWL/XML Syntax

```
<NegativeObjectPropertyAssertion>
  <ObjectProperty IRI="hasWife"/>
  <NamedIndividual IRI="Bill"/>
  <NamedIndividual IRI="Mary"/>
</NegativeObjectPropertyAssertion>
```

Property hierarchies

- Example: "hasWife is a subproperty of hasSpouse"
- Functional-Style Syntax

```
SubObjectPropertyOf( :hasWife :hasSpouse )
```

- RDF/XML Syntax

```
<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:subPropertyOf rdf:resource="hasSpouse"/>
</owl:ObjectProperty>
```

- Turtle Syntax

```
:hasWife rdfs:subPropertyOf :hasSpouse .
```

- Manchester Syntax

```
ObjectProperty: hasWife
  SubPropertyOf: hasSpouse
```

- OWL/XML Syntax

```
<SubObjectPropertyOf>
  <ObjectProperty IRI="hasWife"/>
  <ObjectProperty IRI="hasSpouse"/>
</SubObjectPropertyOf>
```

Domain and range restrictions

- Example: "if B is the wife of A, B is a woman and A is a man"
- Functional-Style Syntax

```
ObjectPropertyDomain( :hasWife :Man )
ObjectPropertyRange( :hasWife :Woman )
```

- RDF/XML Syntax

```
<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:domain rdf:resource="Man"/>
  <rdfs:range rdf:resource="Woman"/>
</owl:ObjectProperty>
```

- Turtle Syntax

```
:hasWife
  rdfs:domain :Man ;
  rdfs:range  :Woman .
```

Domain and range restrictions

- Example: "if B is the wife of A, B is a woman and A is a man"
- Manchester Syntax

```
ObjectProperty: hasWife
  Domain: Man
  Range: Woman
```

- OWL/XML Syntax

```
<ObjectPropertyDomain>
  <ObjectProperty IRI="hasWife"/>
  <Class IRI="Man"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
  <ObjectProperty IRI="hasWife"/>
  <Class IRI="Woman"/>
</ObjectPropertyRange>
```

Outline

Inequality of individuals

- Example: "John and Bill are not the same individual"
  - Lack of the "unique names assumption"
- Functional-Style Syntax

```
DifferentIndividuals( :John :Bill )
```

- RDF/XML Syntax

```
<rdf:Description rdf:about="John">
  <owl:differentFrom rdf:resource="Bill"/>
</rdf:Description>
```

- Turtle Syntax

```
:John owl:differentFrom :Bill .
```

- Manchester Syntax

```
Individual: John
  DifferentFrom: Bill
```

- OWL/XML Syntax

```
<DifferentIndividuals>
  <NamedIndividual IRI="John"/>
  <NamedIndividual IRI="Bill"/>
</DifferentIndividuals>
```

Equality of individuals

- Example: "James and Jim are the same individual"
- Functional-Style Syntax

```
SameIndividual( :James :Jim )
```

- RDF/XML Syntax

```
<rdf:Description rdf:about="James">
  <owl:sameAs rdf:resource="Jim"/>
</rdf:Description>
```

- Turtle Syntax

```
:James owl:sameAs :Jim .
```

- Manchester Syntax

```
Individual: James
  SameAs: Jim
```

- OWL/XML Syntax

```
<SameIndividual>
  <NamedIndividual IRI="James"/>
  <NamedIndividual IRI="Jim"/>
</SameIndividual>
```

Outline

Semantic Web Ontologies

GEIST

Outline

Introduction
Ontologies in Computer Science
Ontology languages for the Web

Description Logics Basics

OWL Language
Basic Features
OWL Species
Classes
Object Properties
Individuals
Datatype Properties
Complex Class Constructors
Property Restrictions Constructors
Advanced Features
Other Features

Ontology building in Protégé

The End

Datatypes properties

- Example: "John's age is 51"
- Functional-Style Syntax

```
DataPropertyAssertion( :hasAge :John "51"^^xsd:integer )
```

- RDF/XML Syntax

```
<Person rdf:about="John">
  <hasAge rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">51</hasAge>
</Person>
```

- Turtle Syntax

```
:John  :hasAge 51 .
```

- Manchester Syntax

```
Individual: John
  Facts: hasAge "51"^^xsd:integer
```

- OWL/XML Syntax

```
<DataPropertyAssertion>
  <DataProperty IRI="hasAge"/>
  <NamedIndividual IRI="John"/>
  <Literal datatypeIRI="
    http://www.w3.org/2001/
    XMLSchema#integer">51</Literal>
</DataPropertyAssertion>
```

Outline

Complex classes

- Several constructors allow to express more complex knowledge
- Class constructors
  - Intersection of two classes
  - Union of two classes
  - Complement of a class
  - Enumeration of individuals
- Property restrictions
  - Existential quantification
  - Universal quantification
  - Cardinality restriction

Intersection of two classes

- Example: "Mothers are Women that are also Parents"
- Functional-Style Syntax

```
EquivalentClasses(
  :Mother
  ObjectIntersectionOf( :Woman :Parent )
)
```

- RDF/XML Syntax

```
<owl:Class rdf:about="Mother">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Woman"/>
        <owl:Class rdf:about="Parent"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

- Turtle Syntax

```
:Mother owl:equivalentClass [
  rdf:type owl:Class ;
  owl:intersectionOf ( :Woman :Parent )
] .
```

Intersection of two classes

- Example: "Mothers are Women that are also Parents"
- Manchester Syntax

```
Class: Mother
  EquivalentTo: Woman and Parent
```

- OWL/XML Syntax

```
<EquivalentClasses>
  <Class IRI="Mother"/>
  <ObjectIntersectionOf>
    <Class IRI="Woman"/>
    <Class IRI="Parent"/>
  </ObjectIntersectionOf>
</EquivalentClasses>
```

Union of two classes

- Example: “Parents are the union of Mothers and Fathers”
- RDF/XML Syntax

```
<owl:Class rdf:about="Parent">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Mother"/>
        <owl:Class rdf:about="Father"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Complement of a class

- Example: "A ChildlessPerson is a Person that is not a Parent"
- RDF/XML Syntax

```
<owl:Class rdf:about="ChildlessPerson">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Person"/>
        <owl:Class>
          <owl:complementOf rdf:resource="Parent"/>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Use of complex classes in assertions

- Example: "Jack is a Person but not a Parent"
- RDF/XML Syntax

```
<rdf:Description rdf:about="Jack">
  <rdf:type>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="Person"/>
        <owl:Class>
          <owl:complementOf rdf:resource="Parent"/>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </rdf:type>
</rdf:Description>
```

Enumeration of individuals

- A very straightforward way to describe a class is just to enumerate all its instances
  - "closed classes" or enumerated sets
- Example: a class of birthday guests

```
<owl:Class rdf:about="MyBirthdayGuests">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <rdf:Description rdf:about="Bill"/>
        <rdf:Description rdf:about="John"/>
        <rdf:Description rdf:about="Mary"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Outline

Existential quantification

- Property restriction that defines a class as the set of all individuals that are connected via a particular property to another individual which is an instance of a certain class
- Example: "the class of Parents is the class of individuals that are linked to a Person by the hasChild property"
- RDF/XML Syntax

```
<owl:Class rdf:about="Parent">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasChild"/>
      <owl:someValuesFrom rdf:resource="Person"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Universal quantification

- Property restriction used to describe a class of individuals for which all related individuals must be instances of a given class
- Example: "somebody is a happy person if all their children are happy persons"
- RDF/XML Syntax

```
<owl:Class rdf:about="HappyPerson"/>
  <owl:equivalentClass>
      <owl:Restriction>
        <owl:onProperty rdf:resource="hasChild"/>
        <owl:allValuesFrom rdf:resource="HappyPerson"/>
      </owl:Restriction>
    </owl:equivalentClass>
 </owl:Class>
```

Universal quantification

- Example: the class of John's children

```
<owl:Class rdf:about="JohnsChildren">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasParent"/>
      <owl:hasValue rdf:resource="John"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

- Example: narcisistic persons

```
<owl:Class rdf:about="NarcisticPerson">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="loves"/>
      <owl:hasSelf rdf:datatype="&xsd;boolean"> true </owl:hasSelf>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Cardinality restrictions

- Used to specify the number of individuals involved in the restriction
- Example: "John has at most four children who are themselves parents"

```
<rdf:Description rdf:about="John">
  <rdf:type>
    <owl:Restriction>
      <owl:maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">
        4
      </owl:maxQualifiedCardinality>
      <owl:onProperty rdf:resource="hasChild"/>
      <owl:onClass rdf:resource="Parent"/>
    </owl:Restriction>
  </rdf:type>
</rdf:Description>
```

Cardinality restrictions

- Used to specify the number of individuals involved in the restriction
- `owl:maxQualifiedCardinality:` max cardinality
- `owl:minQualifiedCardinality:` min cardinality
- `owl:qualifiedCardinality:` exact cardinality
- `owl:cardinality:` exact cardinality when class is not specified (providing the class is optional)
- Example: "John has 5 children"

```
<rdf:Description rdf:about="John">
  <rdf:type>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
        5
      </owl:cardinality>
      <owl:onProperty rdf:resource="hasChild"/>
    </owl:Restriction>
  </rdf:type>
</rdf:Description>
```

Outline

Advanced use of properties

- Inverse properties

```
<owl:ObjectProperty rdf:about="hasParent">
  <owl:inverseOf rdf:resource="hasChild"/>
</owl:ObjectProperty>
```

- Symmetric and asymmetric properties

```
<owl:SymmetricProperty rdf:about="hasSpouse"/>
...
<owl:AsymmetricProperty rdf:about="hasChild"/>
```

- Disjoint properties
  - Example: parent-child marriages cannot occur

```
<rdf:Description rdf:about="hasParent">
  <owl:propertyDisjointWith rdf:resource="hasSpouse"/>
</rdf:Description>
```

Advanced use of properties

- Reflexive and irreflexive properties

```
<owl:ReflexiveProperty rdf:about="hasRelative"/>
...
<owl:IrreflexiveProperty rdf:about="parentOf"/>
```

- Functional and inverse functional properties
  - Example: every individual can be linked by the hasHusband property to at most one other individual

```
<owl:FunctionalProperty rdf:about="hasHusband"/>
...
<owl:InverseFunctionalProperty rdf:about="hasHusband"/>
```

- Transitive properties

```
<owl:TransitiveProperty rdf:about="hasAncestor"/>
```

Property chains

- Example: definition of the hasGrandparent property
  - We want hasGrandparent to connect all individuals that are linked by a chain of exactly two hasParent properties

```
<rdf:Description rdf:about="hasGrandparent">
  <owl:propertyChainAxiom rdf:parseType="Collection">
    <owl:ObjectProperty rdf:about="hasParent"/>
    <owl:ObjectProperty rdf:about="hasParent"/>
  </owl:propertyChainAxiom>
</rdf:Description>
```

Keys

- A collection of (data or object) properties can be assigned as a key to a class expression
  - This means that each named instance of the class expression is uniquely identified by the set of values which these properties attain in relation to the instance
- Example: the identification of a person by her social security number

```
<owl:Class rdf:about="Person">
  <owl:hasKey rdf:parseType="Collection">
    <owl:ObjectProperty rdf:about="hasSSN"/>
  </owl:hasKey>
</owl:Class>
```

Advanced use of datatypes

- It is possible to express and define new datatypes by constraining or combining existing ones

```
<rdf:Description rdf:about="personAge">
  <owl:equivalentClass>
    <rdfs:Datatype>
      <owl:onDatatype rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
      <owl:withRestrictions rdf:parseType="Collection">
        <rdf:Description>
          <xsd:minInclusive
                  rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
            0</xsd:minInclusive>
        </rdf:Description>
        <rdf:Description>
          <xsd:maxInclusive
                  rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
            150</xsd:maxInclusive>
        </rdf:Description>
      </owl:withRestrictions>
    </rdfs:Datatype>
  </owl:equivalentClass>
</rdf:Description>
```

Advanced use of datatypes

- Moreover, a new datatype can be generated by just enumerating the data values it contains

```
<rdf:Description rdf:about="toddlerAge">
  <owl:equivalentClass>
    <rdfs:Datatype>
      <owl:oneOf>
        <rdf:Description>
          <rdf:first rdf:datatype="&xsd;integer">1</rdf:first>
          <rdf:rest>
            <rdf:Description>
              <rdf:first rdf:datatype="&xsd;integer">2</rdf:first>
              <rdf:rest
                 rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            </rdf:Description>
          </rdf:rest>
        </rdf:Description>
      </owl:oneOf>
    </rdfs:Datatype>
  </owl:equivalentClass>
</rdf:Description>
```

Advanced use of datatypes

- New classes can be defined by restrictions on datatype

```
<owl:Class rdf:about="Teenager">
  <rdfs:subClassOf>
    <owl:Restriction>
    <owl:onProperty rdf:resource="hasAge"/>
      <owl:someValuesFrom>
        <rdfs:Datatype>
          <owl:onDataType rdf:resource="&xsd;integer"/>
          <owl:withRestrictions rdf:parseType="Collection">
            <rdf:Description>
              <xsd:minExclusive rdf:datatype="&xsd;integer">12</xsd:minExclusive>
            </rdf:Description>
            <rdf:Description>
              <xsd:maxInclusive rdf:datatype="&xsd;integer">19</xsd:maxInclusive>
            </rdf:Description>
          </owl:withRestrictions>
        </rdfs:Datatype>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Outline

Annotations

- Associates property-value pairs with parts of an ontology, or the entire ontology itself

```
<owl:Class rdf:about="Person">
  <rdfs:comment>Represents the set of all people.</rdfs:comment>
</owl:Class>
```

Ontology management

- Ontology declaration (XML syntax)

```
<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#">
```

- Ontology metadata (information about the ontology)

```
<owl:Ontology rdf:about="">
   <rdfs:comment>An example OWL ontology</rdfs:comment>
   <owl:priorVersion
        rdf:resource="http://www.mydomain.org/uni-ns-old"/>
   <owl:imports
        rdf:resource="http://www.mydomain.org/persons"/>
   <rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>
```

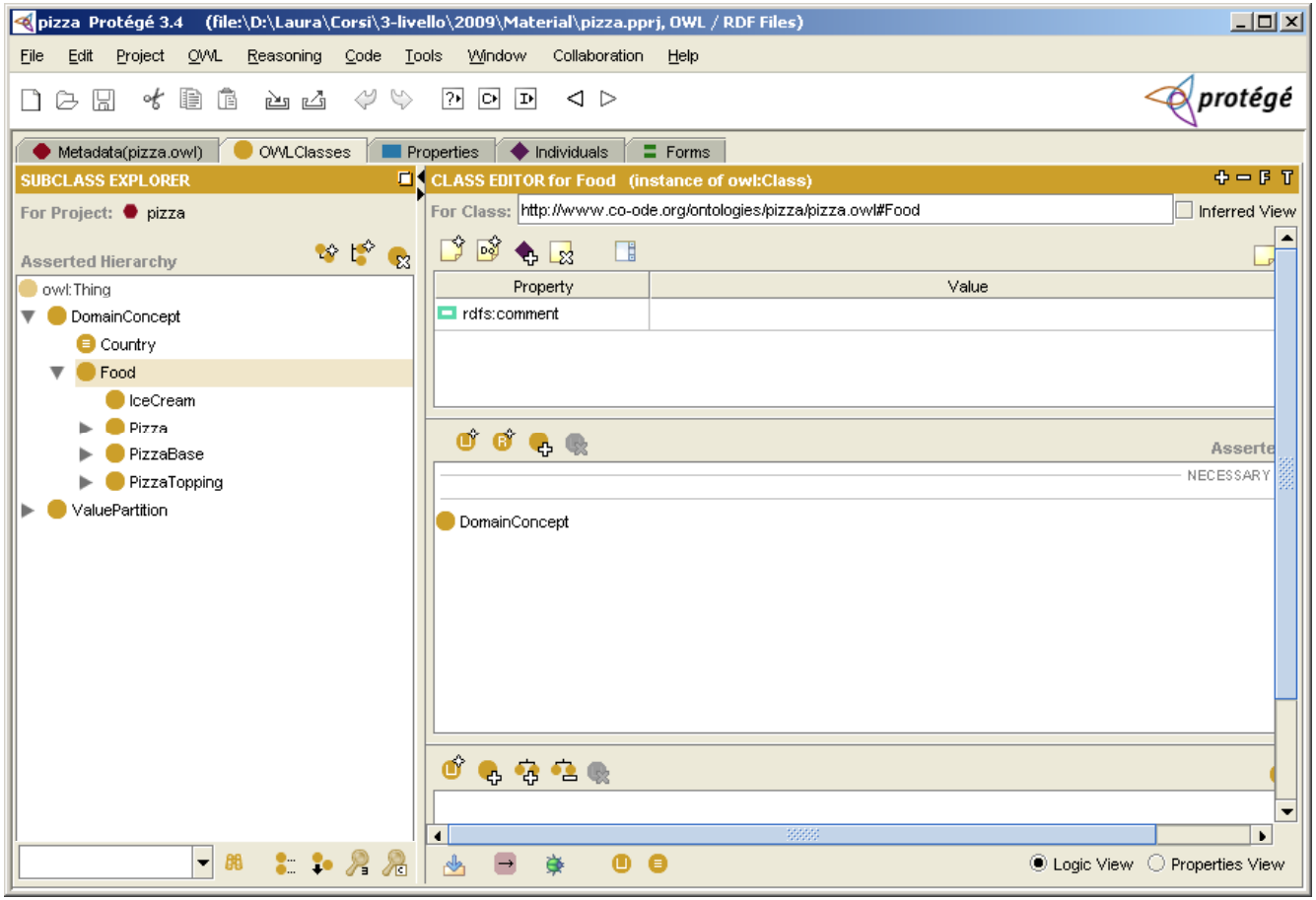
Annotation Properties

named annotation property	A	A
human-readable name	rdfs:label	rdfs:label
human-readable comment	rdfs:comment	rdfs:comment
additional information	rdfs:seeAlso	rdfs:seeAlso
defining agent	rdfs:isDefinedBy	rdfs:isDefinedBy
version information	owl:versionInfo	owl:versionInfo
deprecation	owl:deprecated	owl:deprecated
backwards compatibility	owl:backwardCompatibleWith	owl:backwardCompatibleWith
incompatibility	owl:incompatibleWith	owl:incompatibleWith
prior version	owl:priorVersion	owl:priorVersion

Outline

Protégé

- Open source ontology editor
- Developed by Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine
- http://protege.stanford.edu/

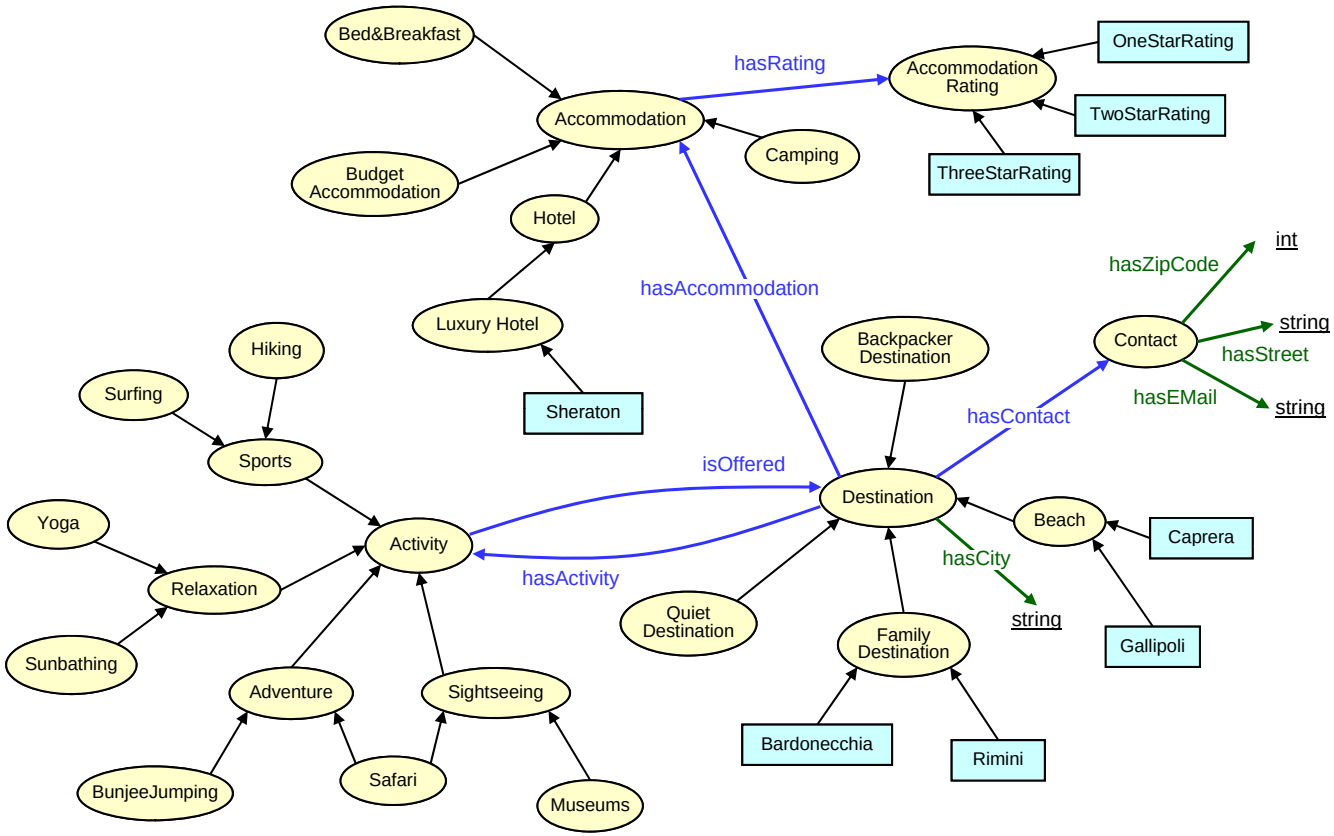


Project steps

0. Conceptual design of the ontology
1. Classes definition
2. Properties definition
3. Individuals definition
4. Restrictions definition

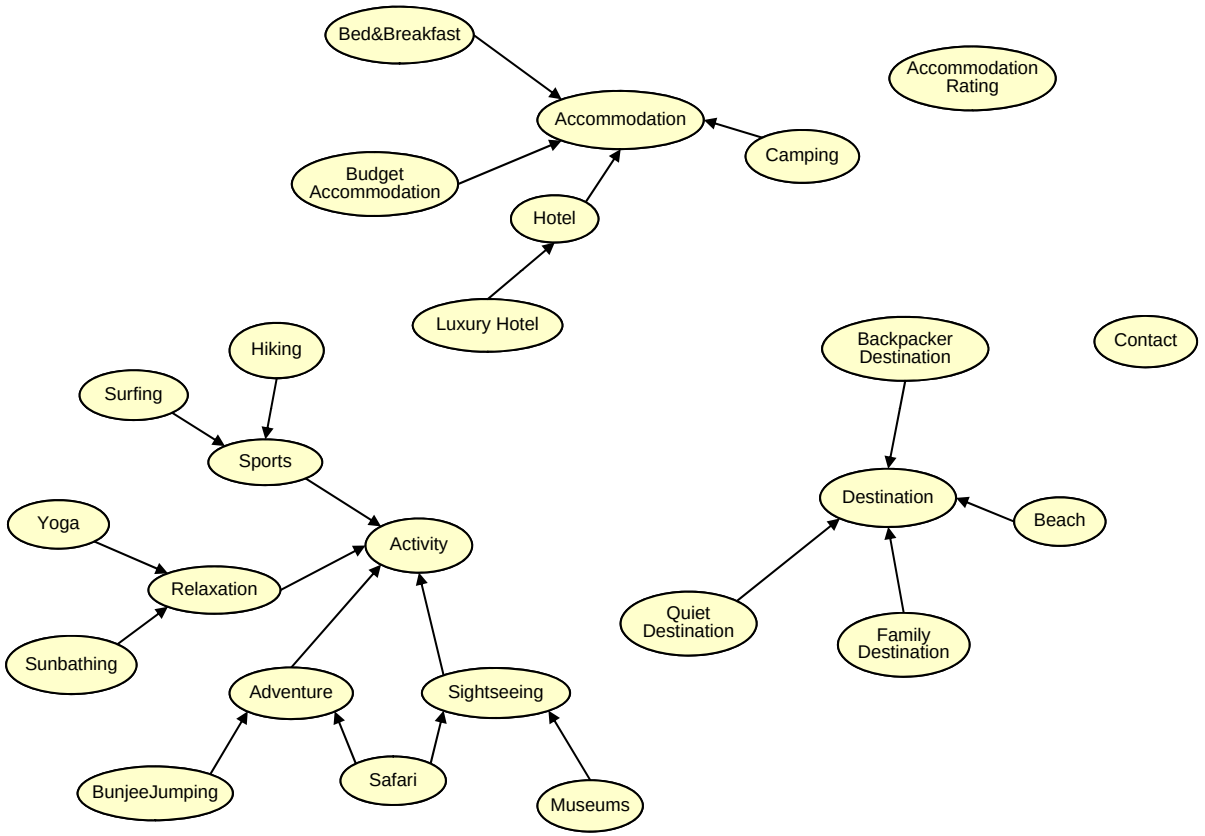
Step 0

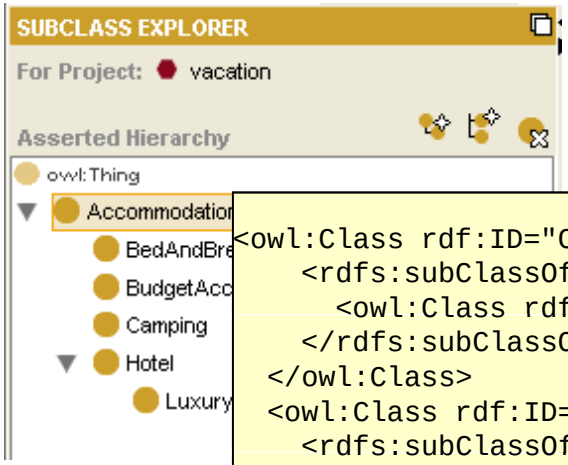
- Conceptual design of the ontology



Step 1a

- Define classes and subclasses
  - Is-a relationship, or subsumption



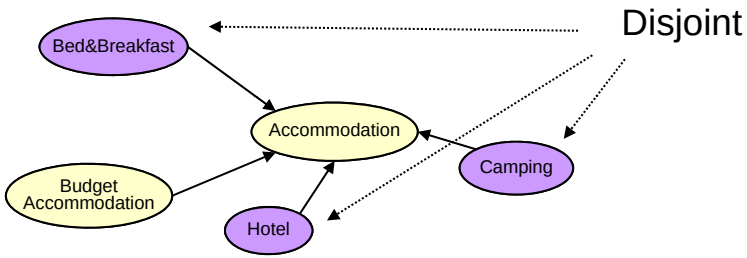


OWL code

```
<owl:Class rdf:ID="Camping">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Accommodation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="BudgetAccommodation">
    <rdfs:subClassOf rdf:resource="#Accommodation"/>
  </owl:Class>
  <owl:Class rdf:ID="BedAndBreakfast">
    <rdfs:subClassOf rdf:resource="#Accommodation"/>
  </owl:Class>
  <owl:Class rdf:ID="LuxuryHotel">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Hotel"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Hotel">
    <rdfs:subClassOf rdf:resource="#Accommodation"/>
  </owl:Class>
```

Step 1b

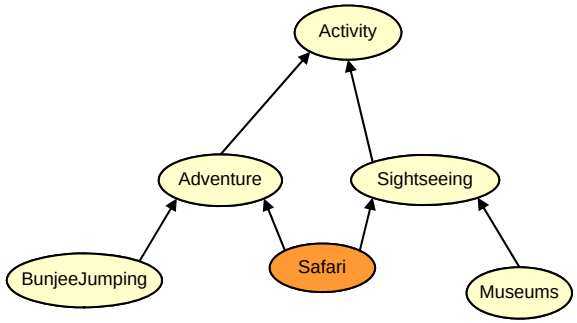
- Define disjoint classes

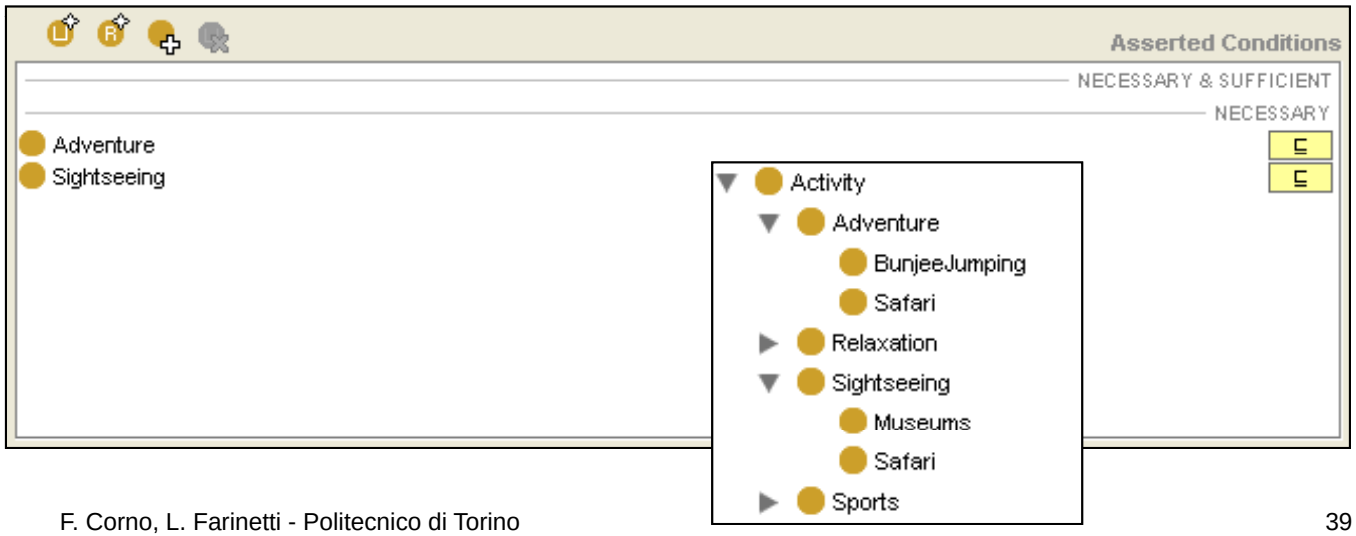



```
<owl:Class rdf:about="#Hotel">
    <owl:disjointWith rdf:resource="#Camping"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#BedAndBreakfast"/>
    </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="#Accommodation"/>
 </owl:Class>
```

Step 1c

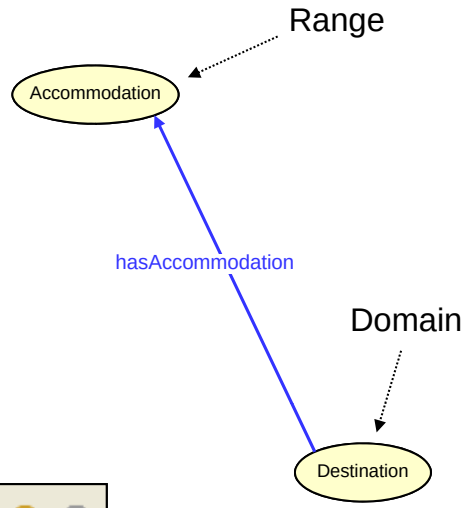
- Define multiple inheritance





Step 2a

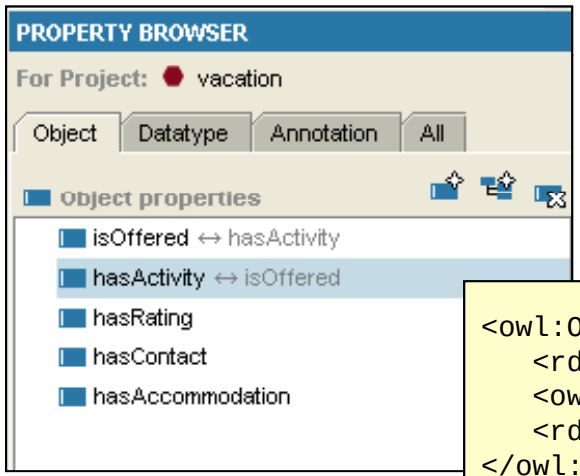
- Define object properties
  - relationships

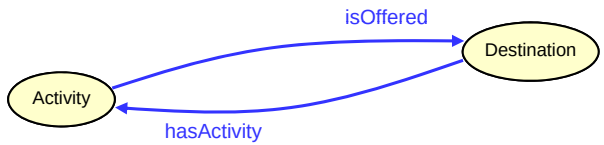



```
<owl:ObjectProperty rdf:ID="hasAccommodation">
   <rdfs:domain rdf:resource="#Destination"/>
   <rdfs:range rdf:resource="#Accommodation"/>
</owl:ObjectProperty>
```

Step 2b

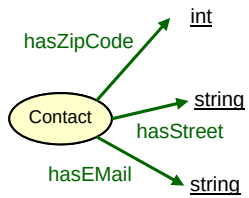
- Define object properties characteristics
  - E.g: define an inverse object property



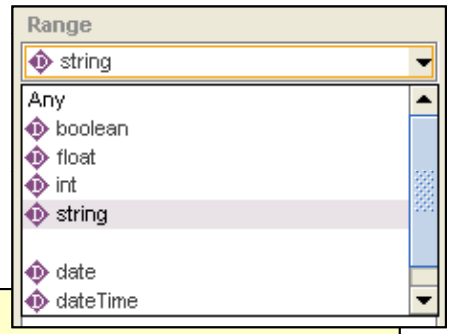



```
<owl:ObjectProperty rdf:about="#isOffered">
   <rdfs:range rdf:resource="#Destination"/>
   <owl:inverseOf rdf:resource="#hasActivity"/>
   <rdfs:domain rdf:resource="#Activity"/>
</owl:ObjectProperty>
```

Step 2c



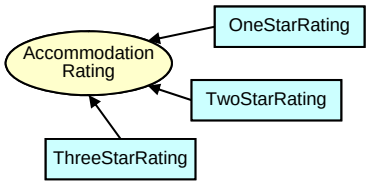
- Define datatype properties
  - The range specifies the data type

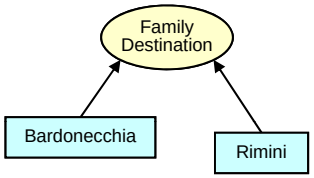



```
<owl:DatatypeProperty rdf:ID="hasEMail">
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   <rdfs:domain rdf:resource="#Contact"/>
 </owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="hasZipCode">
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
   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

Step 3

- Define individuals
  - instances





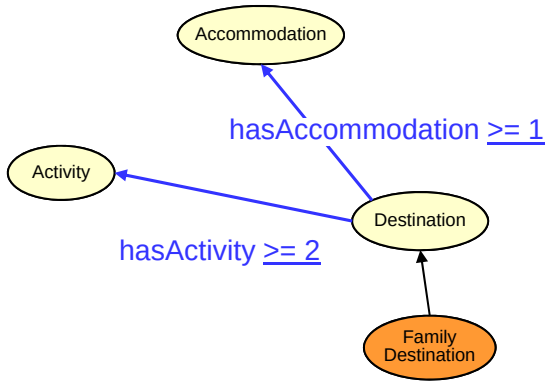

```
<AccommodationRating rdf:ID="OneStarRating"/>
<AccommodationRating rdf:ID="ThreeStarRating"/>
<AccommodationRating rdf:ID="TwoStarRating"/>
<FamilyDestination rdf:ID="Bardonecchia"/>
<FamilyDestination rdf:ID="Rimini"/>
```

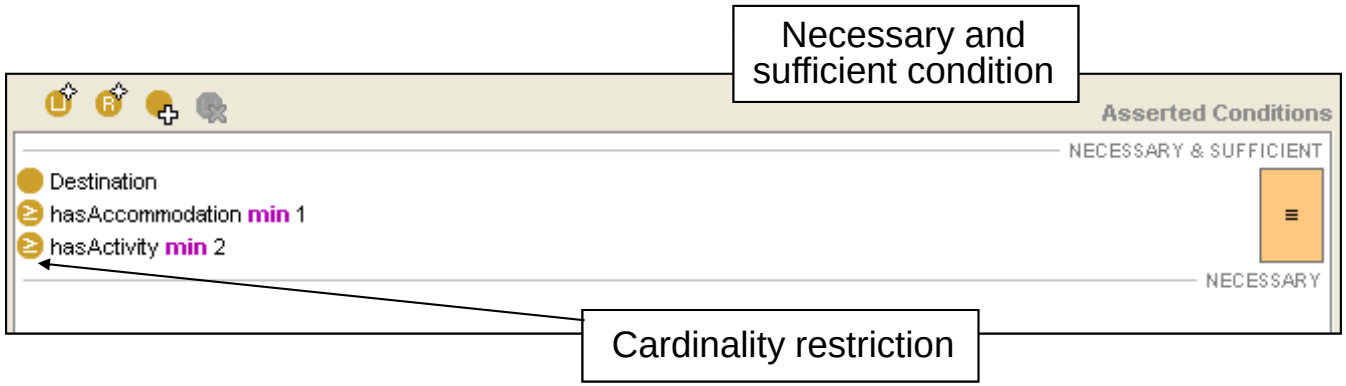
Step 4

- Define restrictions
  - To restrict the individuals that belong to a class
  - Quantifier restrictions (existential, universal quantifiers)
  - Cardinality restrictions
  - hasValue restrictions

Step 4 – example 1

- FamilyDestination is a Destination with at least one accommodation and at least 2 activities





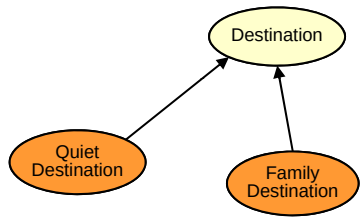
Step 4 – example 1

- FamilyDestination is a Destination with at least one accommodation and at least 2 activities

```
<owl:Class rdf:ID="FamilyDestination">
   <owl:equivalentClass>
     <owl:Class>
       <owl:intersectionOf rdf:parseType="Collection">
         <owl:Restriction>
           <owl:onProperty>
             <owl:ObjectProperty rdf:ID="hasAccommodation"/>
           </owl:onProperty>
           <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
           >1</owl:minCardinality>
         </owl:Restriction>
         <owl:Restriction>
           <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
           >2</owl:minCardinality>
           <owl:onProperty>
             <owl:ObjectProperty rdf:ID="hasActivity"/>
           </owl:onProperty>
         </owl:Restriction>
         <owl:Class rdf:about="#Destination"/>
       </owl:intersectionOf>
     </owl:Class>
   </owl:equivalentClass>
</owl:Class>
```

Step 4 – example 2

- QuietDestination is a Destination that is not chosen by noisy families

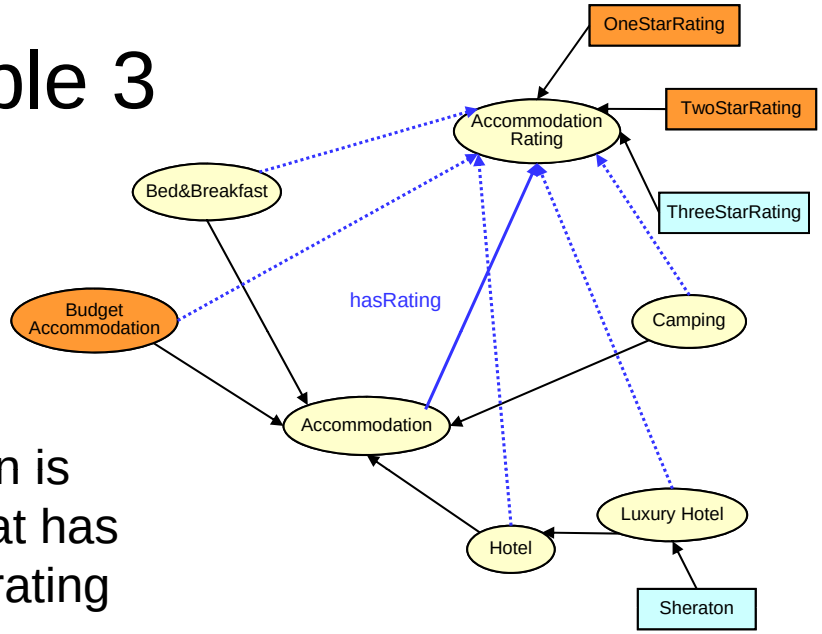


Step 4 – example 2

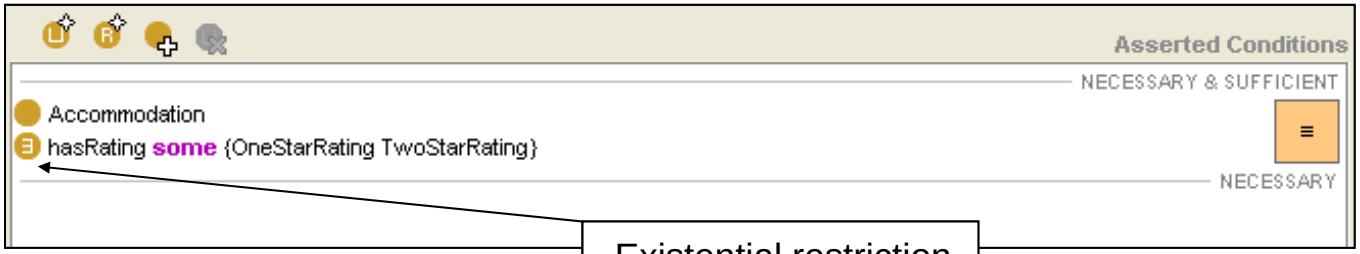
- QuietDestination is a Destination that is not chosen by noisy families

```
<owl:Class rdf:ID="QuietDestination">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class>
          <owl:complementOf rdf:resource="#FamilyDestination"/>
        </owl:Class>
        <owl:Class rdf:about="#Destination"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Step 4 – example 3



- BudgetAccommodation is an Accommodation that has either one or two star rating



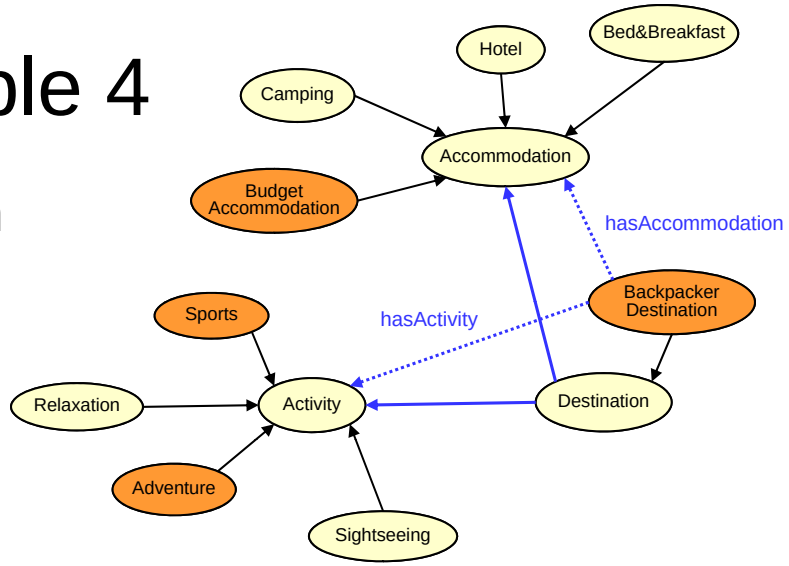
Step 4 – example 3

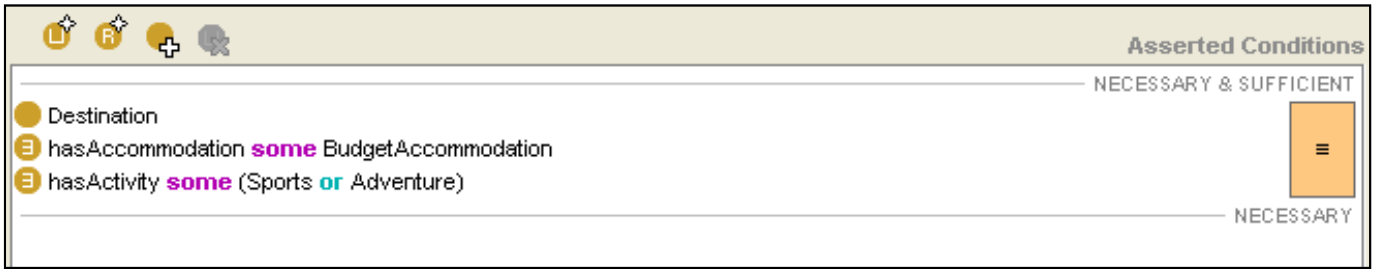
- BudgetAccommodation is an Accommodation that has either one or two star rating

```
<owl:Class rdf:ID="BudgetAccommodation">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasRating"/>
          </owl:onProperty>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:oneOf rdf:parseType="Collection">
                <AccommodationRating rdf:ID="OneStarRating"/>
                <AccommodationRating rdf:ID="TwoStarRating"/>
              </owl:oneOf>
            </owl:Class>
          </owl:someValuesFrom>
        </owl:Restriction>
        <owl:Class rdf:about="#Accommodation"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Step 4 – example 4

- BackpackerDestination is a Destination that provides budget accommodation and offers sport or adventure activities





Step 4 – example 4

- BackpackerDestination is a Destination that provides budget accommodation and offers sport or adventure activities

```
<owl:Class rdf:ID="BackpackerDestination">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasAccommodation"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#BudgetAccommodation"/>
        </owl:Restriction>
        <owl:Class rdf:about="#Destination"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasActivity"/>
          </owl:onProperty>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#Sports"/>
                <owl:Class rdf:about="#Adventure"/>
              </owl:unionOf>
            </owl:Class>
          </owl:someValuesFrom>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

License

- This work is licensed under the Creative Commons **Attribution-Noncommercial-Share Alike** 3.0 Unported License.
- To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

If you want to know more...

Reading:

- N. Guarino: "Formal Ontology and Information Systems"
- http://www.co-ode.org/resources/papers/ekaw2004.pdf
- http://www.betaversion.org/~stefano/linotype/news/57/ - SemWeb for the XML people
- http://www.cs.man.ac.uk/~horrocks/ISWC2003/Tutorial/ - OWL Tutorial from 2003
- http://ontologydesignpatterns.org/wiki/Main_Page - Ontology Design patterns

Tutorials:

- http://www.co-ode.org/resources/tutorials/
- http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/

Additional resources:

- http://www.co-ode.org/ontologies/

If you want to know more...

http://ai.ia.agh.edu.pl/wiki/pl:dydaktyka:semantic_web:start

OWL pictures used according to license:

- Squeezyboy: http://www.flickr.com/photos/squeezyboy/154171735/
- One Speckled Frog / Eric Hoffmann: http://www.flickr.com/photos/speckledfrog
- Larry McQueen - OWLS 1: http://www.natureartists.com/artists/artist_artwork.asp?ArtistID=1374&ArtworkID=18081
- Tasshu Rikimara: http://www.flickr.com/photos/tasshu113/4956888274/
- Johan J.Ingles-Le Nobel: http://www.flickr.com/photos/jingleslenobel/4971486823/
- Mara 1: http://www.flickr.com/photos/flametree/4517245491/
- Gabor Kovacs: http://www.flickr.com/photos/21923568@N00/2520990696/
- Rhys's Piece Is: http://www.flickr.com/photos/rhysasplundh/5105241656/

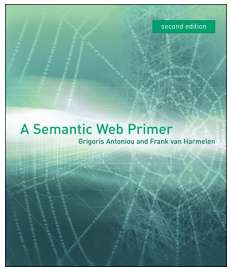
Questions

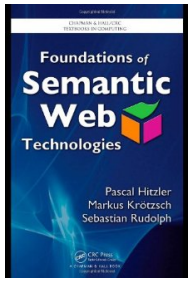
Any questions?

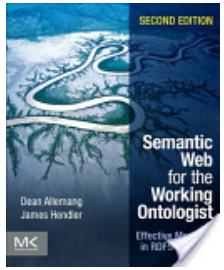
Thank you

Thank you for your attention!

http://geist.agh.edu.pl
GEIST Research Group







Powered by LaTeX