OWL and SWRL

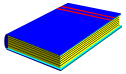
Oscar Corcho, María del Carmen Suárez de Figueroa Baonza
ocorcho@fi.upm.es, mcsuarez@delicias.dia.fi.upm.es
http://www.oeg-upm.net/

Ontological Engineering Group
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn,
28660 Boadilla del Monte, Madrid, Spain

Main References

Gómez-Pérez, A.; Fernández-López, M.; Corcho, O. **Ontological Engineering.** *Springer Verlag. 2003*

Capítulo 4: Ontology languages



Baader F, McGuinness D, Nardi D, Patel-Schneider P (2003)
The Description Logic Handbook: Theory, implementation and applications.
Cambridge University Press, Cambridge, United Kingdom

Dean M, Schreiber G (2004) *OWL Web Ontology Language Reference.* W3C Recommendation. *http://www.w3.org/TR/owl-ref/*

Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M (2004) *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission. *http://www.w3.org/Submission/SWRL/*

Jena web site:	*http://jena.sourceforge.net/*
Jena API:	*http://jena.sourceforge.net/tutorial/RDF_API/*
Jena tutorials:	*http://www.ibm.com/developerworks/xml/library/j-jena/index.html*
	http://www.xml.com/pub/a/2001/05/23/jena.html

Pellet:	http://pellet.owldl.com/
RACER:	http://www.racer-systems.com/
FaCT++:	http://owl.man.ac.uk/factplusplus/

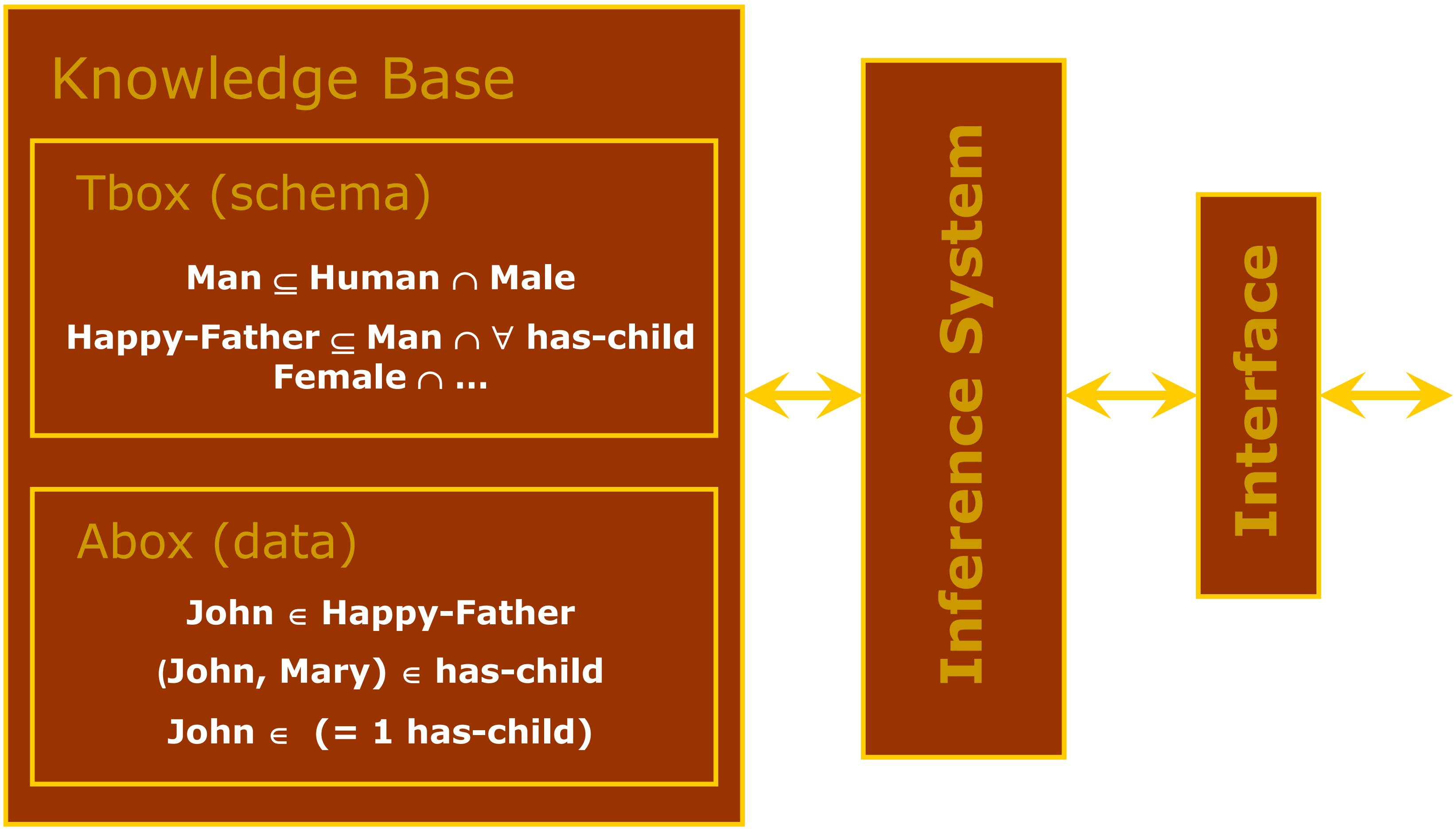
Table of Contents

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
    - Specific languages characterised by the constructors and axioms used to assert knowledge about classes, roles and individuals.
    - Example: ALC (the least expressive language in DL that is propositionally closed)
      - Constructors: boolean (and, or, not)
      - Role restrictions

- **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 Architecture

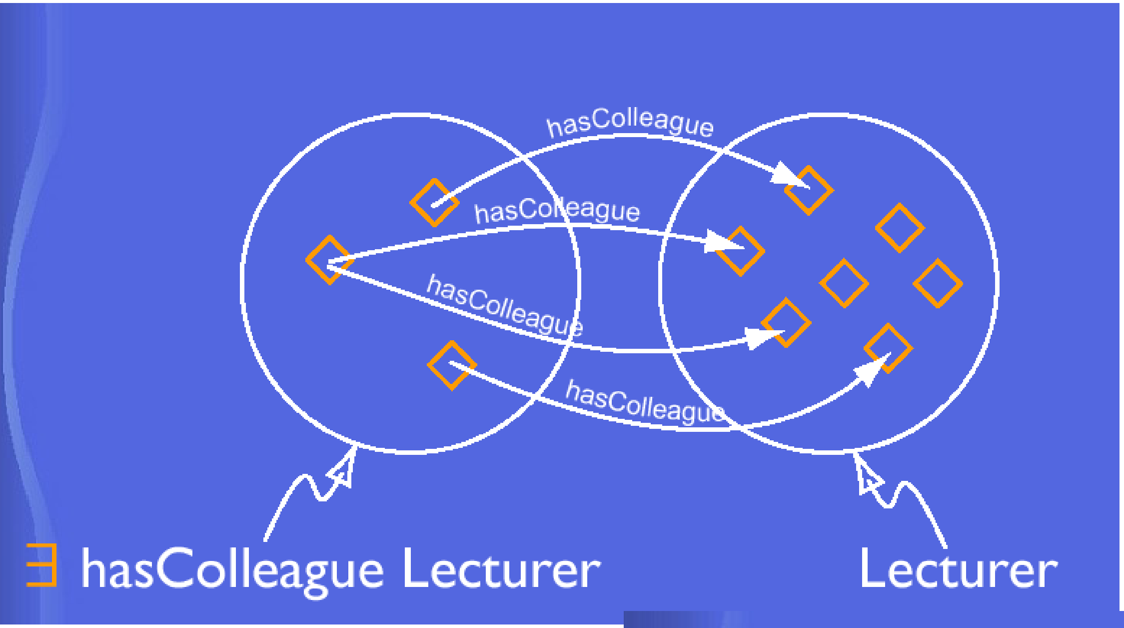


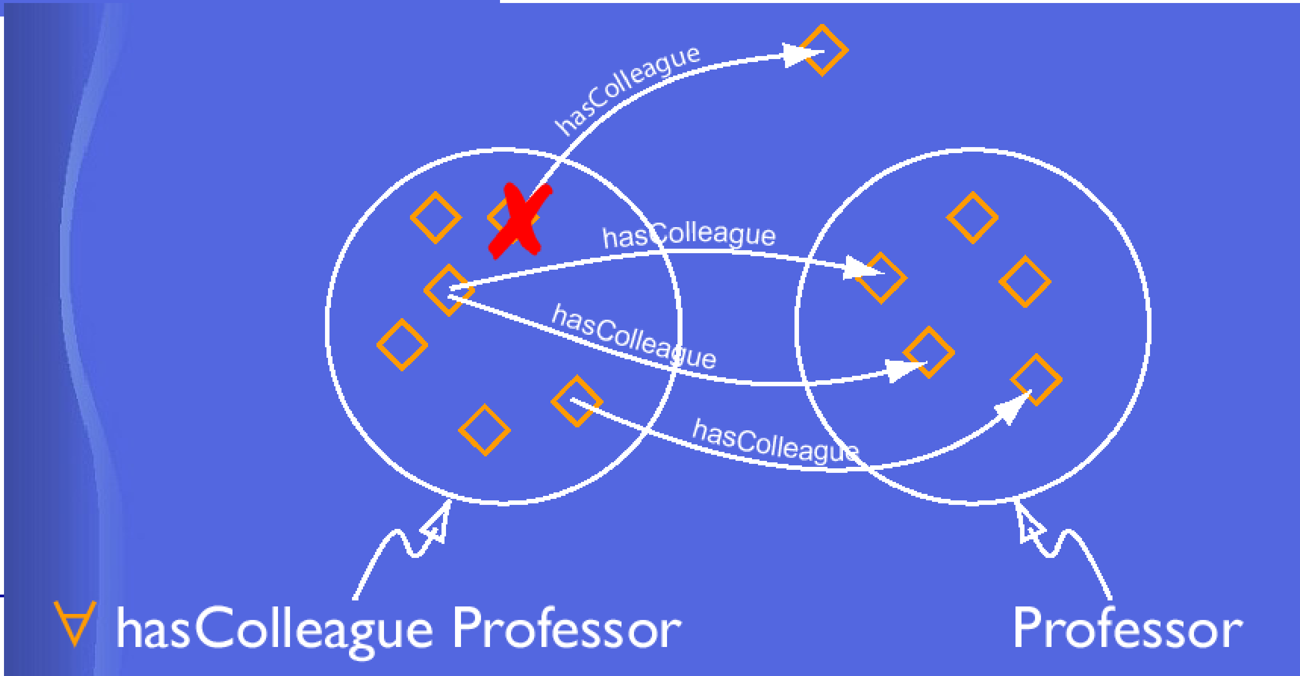
Most common constructors in class definitions

- **Intersection: $C_1 \cap ... \cap C_n$** — **Human $\cap$ Male**
- **Union: $C_1 \cup ... \cup C_n$** — **Doctor $\cup$ Lawyer**
- **Negation: $\neg$C** — **$\neg$Male**
- **Nominals: $\{x_1\} \cup ... \cup \{x_n\}$** — **{john} $\cup$ ... $\cup$ {mary}**
- **Universal restriction: $\forall$P.C** — **$\forall$hasChild.Doctor**
- **Existential restriction: $\exists$P.C** — **$\exists$hasChild.Lawyer**
- **Maximum cardinality: $\leq$nP** — **$\leq$3hasChild**
- **Minimum cardinality: $\geq$nP** — **$\geq$1hasChild**
- **Specific Value: $\exists$P.{x}** — **$\exists$hasColleague.{Matthew}**

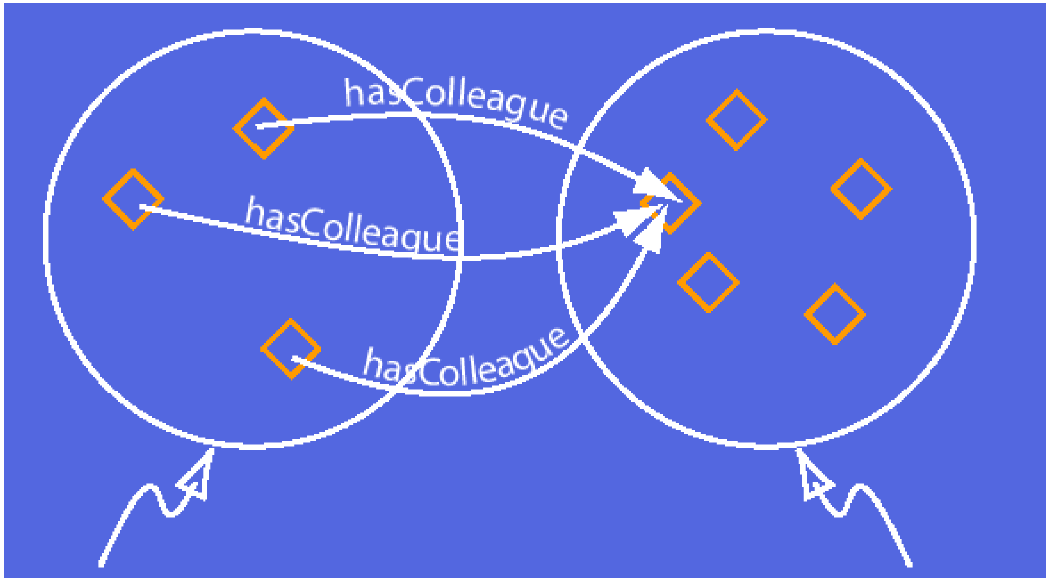
- **Nesting of constructors can be arbitrarily complex**
  - Person $\cap$ $\forall$hasChild.(Doctor $\cup$ $\exists$hasChild.Doctor)
- **Lots of redundancy**
  - A$\cup$B is equivalent to $\neg$($\neg$ A $\cap$ $\neg$B)
  - $\exists$P.C is equivalent to $\neg\forall$P. $\neg$C

Existential and Universal Restrictions





Specific value



∃hasColleague.{Matthew} **Persons**

Most common axioms in class, property and individual definitions

- **Classes**

– Subclass	$C1 \subseteq C2$	$Human \subseteq Animal \cap Biped$
– Equivalence	$C1 \equiv C2$	$Man \equiv Human \cap Male$
– Disjointness	$C1 \cap C2 \subseteq \bot$	$Male \cap Female \subseteq \bot$

- **Properties/roles**

– Subproperty	$P1 \subseteq P2$	$hasDaughter \subseteq hasChild$
– Equivalence	$P1 \equiv P2$	$cost \equiv price$
– Inverse	$P1 \equiv P2^{-}$	$hasChild \equiv hasParent^{-}$
– Transitive	$P+ \subseteq P$	$ancestor+ \subseteq ancestor$
– Functional	$T \subseteq \leq 1P$	$T \subseteq \leq 1hasMother$
– InverseFunctional	$T \subseteq \leq 1P^{-}$	$T \subseteq \leq 1hasPassportID^{-}$

- **Individuals**

– Equivalence	$\{x1\} \equiv \{x2\}$	{oeg:OscarCorcho} $\equiv$ {img:Oscar}
– Different	$\{x1\} \equiv \neg\{x2\}$	$\{john\} \equiv \neg\{peter\}$

- **Most axioms are reducible to inclusion ($\cup$)**
  - $C \equiv D$ iff both $C \subseteq D$ and $D \subseteq C$
  - C disjoint D iff $C \subseteq \neg D$

DL constructors and DL languages

<table>
<tr><th>Construct</th><th>Syntax</th><th colspan="4">Language</th></tr>
<tr><td>Concept</td><td>A</td><td rowspan="4">FL_0</td><td rowspan="7">FL^-</td><td rowspan="8">AL</td><td rowspan="11">S[14]</td></tr>
<tr><td>Role name</td><td>R</td></tr>
<tr><td>Intersection</td><td>$C \cap D$</td></tr>
<tr><td>Value restriction</td><td>$\forall R.C$</td></tr>
<tr><td>Limited existential quantification</td><td>$\exists R$</td><td rowspan="3"></td></tr>
<tr><td>Top or Universal</td><td>$\top$</td></tr>
<tr><td>Bottom</td><td>$\bot$</td></tr>
<tr><td>Atomic negation</td><td>$\neg A$</td><td colspan="2"></td></tr>
<tr><td>Negation[15]</td><td>$\neg C$</td><td colspan="3">C</td></tr>
<tr><td>Union</td><td>$C \cup D$</td><td colspan="3">U</td></tr>
<tr><td>Existential restriction</td><td>$\exists R.C$</td><td colspan="3">E</td></tr>
<tr><td>Number restrictions</td><td>$(\geq n\ R)\ \ (\leq n\ R)$</td><td colspan="3">N</td><td></td></tr>
<tr><td>Nominals</td><td>$\{a_1 \ldots a_n\}$</td><td colspan="3">O</td><td></td></tr>
<tr><td>Role hierarchy</td><td>$R \subseteq S$</td><td colspan="3">H</td><td></td></tr>
<tr><td>Inverse role</td><td>R^-</td><td colspan="3">I</td><td></td></tr>
<tr><td>Qualified number restriction</td><td>$(\geq n\ R.C)\ \ (\leq n\ R.C)$</td><td colspan="3">Q</td><td></td></tr>
</table>

OWL is SHOIN(D+)

{Colombia, Argentina, México, ...} → MercoSur countries

≤2 hasChild.Female, ≥1 hasParent.Male

[12] Names previously used for Description Logics were: terminological knowledge representation languages, concept languages, term subsumption languages, and KL-ONE-based knowledge representation languages.
[13] In this table, we use A to refer to atomic concepts (concepts that are the basis for building other concepts), C and D to any concept definition, R to atomic roles and S to role definitions. FL is used for structural DL languages and AL for attributive languages (Baader et al., 2003).
[14] S is the name used for the language ALC_{R+}, which is composed of ALC plus transitive roles.
[15] ALC and ALCUE are equivalent languages, since union (U) and existential restriction (E) can be represented using negation (C).

Other:

Concrete datatypes: hasAge.(<21)

Transitive roles: hasChild* (descendant)

Role composition: hasParent o hasBrother (uncle)

Some basic DL modelling guidelines

- **X must be Y, X is an Y that...** → $X \subseteq Y$
- **X is exactly Y, X is the Y that...** → $X \equiv Y$
- **X is not Y (*not the same as X is whatever it is not Y*)** → $X \subseteq \neg Y$
- **X and Y are disjoint** → $X \cap Y \subseteq \bot$
- **X is Y or Z** → $X \subseteq Y \cup Z$
- **X is Y for which property P has only instances of Z as values** → $X \subseteq Y \cap (\forall P.Z)$
- **X is Y for which property P has at least an instance of Z as a value** → $X \subseteq Y \cap (\exists P.Z)$
- **X is Y for which property P has at most 2 values** → $X \subseteq Y \cap (\leq 2.P)$
- **Individual X is a Y** → $X \in Y$

Description Logics Formalisation

Develop a sample ontology in the domain of people, pets, vehicles, and newspapers

- Understand how to formalise knowledge in description logics

Chunk 1. Formalize in DL

1. **Concept definitions**:

   Grass and trees must be plants. Leaves are parts of a tree but there are other parts of a tree that are not leaves. A dog must eat bones, at least. A sheep is an animal that must only eat grass. A giraffe is an animal that must only eat leaves. A mad cow is a cow that eats brains that can be part of a sheep.

2. **Restrictions**:

   Animals or part of animals are disjoint with plants or parts of plants.

3. **Properties**:

   Eats is applied to animals. Its inverse is eaten_by.

4. **Individuals**:

   Tom.
   Flossie is a cow.
   Rex is a dog and is a pet of Mick.
   Fido is a dog.
   Tibbs is a cat.

Chunk 2. Formalize in DL

1. **Concept definitions**:

   Bicycles, buses, cars, lorries, trucks and vans are vehicles. There are several types of companies: bus companies and haulage companies.

   An elderly person must be adult. A kid is (exactly) a person who is young. A man is a person who is male and is adult. A woman is a person who is female and is adult. A grown up is a person who is an adult. And old lady is a person who is elderly and female. Old ladies must have some animal as pets and all their pets are cats.

2. **Restrictions**:

   Youngs are not adults, and adults are not youngs.

3. **Properties**:

   Has mother and has father are subproperties of has parent.

4. **Individuals**:

   Kevin is a person.
   Fred is a person who has a pet called Tibbs.
   Joe is a person who has at most one pet. He has a pet called Fido.
   Minnie is a female, elderly, who has a pet called Tom.

Chunk 3. Formalize in DL

1. **Concept definitions**:

   A magazine is a publication. Broadsheets and tabloids are newspapers. A quality broadsheet is a type of broadsheet. A red top is a type of tabloid. A newspaper is a publication that must be either a broadsheet or a tabloid.

   White van mans must read only tabloids.

2. **Restrictions**:

   Tabloids are not broadsheets, and broadsheets are not tabloids.

3. **Properties**:

   The only things that can be read are publications.

4. **Individuals**:

   Daily Mirror

   The Guardian and The Times are broadsheets

   The Sun is a tabloid

Chunk 4. Formalize in DL

1. **Concept definitions**:

   A pet is a pet of something. An animal must eat something. A vegetarian is an animal that does not eat animals nor parts of animals. Ducks, cats and tigers are animals. An animal lover is a person who has at least three pets. A pet owner is a person who has animal pets. A cat liker is a person who likes cats. A cat owner is a person who has cat pets. A dog liker is a person who likes dogs. A dog owner is a person who has dog pets.

2. **Restrictions**:

   Dogs are not cats, and cats are not dogs.

3. **Properties**:

   Has pet is defined between persons and animals. Its inverse is is_pet_of.

4. **Individuals**:

   Dewey, Huey, and Louie are ducks.
   Fluffy is a tiger.
   Walt is a person who has pets called Huey, Louie and Dewey.

Chunk 5. Formalize in DL

1. **Concept definitions**

   A driver must be adult. A driver is a person who drives vehicles. A lorry driver is a person who drives lorries. A haulage worker is who works for a haulage company or for part of a haulage company. A haulage truck driver is a person who drives trucks ans works for part of a haulage company. A van driver is a person who drives vans. A bus driver is a person who drives buses. A white van man is a man who drives white things and vans.

2. **Restrictions**:

   --

3. **Properties**:

   The service number is an integer property with no restricted domain

4. **Individuals**:

   Q123ABC is a van and a white thing.
   The42 is a bus whose service number is 42.
   Mick is a male who read Daily Mirror and drives Q123ABC.

Chunk 1. Formalisation in DL

$$grass \subseteq plant$$

$$tree \subseteq plant$$

$$leaf \subseteq \exists partOf.tree$$

$$dog \subseteq \exists eats.bone$$

$$sheep \subseteq animal \cap \forall eats.grass$$

$$giraffe \subseteq animal \cap \forall eats.leaf$$

$$madCow \equiv cow \cap \exists eats.(brain \cap \exists partOf.sheep)$$

$$(animal \cup \exists partOf.animal) \cap (plant \cup \exists partOf.plant) \subseteq \bot$$

Chunk 2. Formalisation in DL

$$bicycle \subseteq vehicle; bus \subseteq vehicle; car \subseteq vehicle; lorry \subseteq vehicle; truck \subseteq vehicle$$

$$busCompany \subseteq company; haulageCompany \subseteq company$$

$$elderly \subseteq person \cap adult$$

$$kid \equiv person \cap young$$

$$man \equiv person \cap male \cap adult$$

$$woman \equiv person \cap female \cap adult$$

$$grownUp \equiv person \cap adult$$

$$oldLady \equiv person \cap female \cap elderly$$

$$oldLady \subseteq \exists hasPet.animal \cap \forall hasPet.cat$$

$$young \cap adult \subseteq \perp$$

$$hasMother \subseteq hasParent$$

$$hasFather \subseteq hasParent$$

Chunk 3. Formalisation in DL

$$magazine \subseteq publication$$

$$broadsheet \subseteq newspaper$$

$$tabloid \subseteq newspaper$$

$$qualityBroadsheet \subseteq broadsheet$$

$$redTop \subseteq tabloid$$

$$newspaper \subseteq publication \cap (broadsheet \cup tabloid)$$

$$whiteVanMan \subseteq \forall reads.tabloid$$

$$tabloid \cap broadsheet \subseteq \bot$$

Chunk 4. Formalisation in DL

$$pet \equiv \exists isPetOf.\mathrm{T}$$

$$animal \subseteq \exists eats.\mathrm{T}$$

$$vegetarian \equiv animal \cap \forall eats.\neg animal \cap \forall eats.\neg(\exists partOf.animal)$$

$$duck \subseteq animal; cat \subseteq animal; tiger \subseteq animal$$

$$animalLover \equiv person \cap (\geq 3 hasPet)$$

$$petOwner \equiv person \cap \exists hasPet.animal$$

$$catLike \equiv person \cap \exists likes.cat; catOwner \equiv person \cap \exists hasPet.cat$$

$$dogLike \equiv person \cap \exists likes.dog; dogOwner \equiv person \cap \exists hasPet.dog$$

$$dog \cap cat \subseteq \bot$$

Chunk 5. Formalisation in DL

$$driver \subseteq adult$$

$$driver \equiv person \cap \exists drives.vehicle$$

$$lorryDriver \equiv person \cap \exists drives.lorry$$

$$haulageWorke \equiv \exists worksFor.(haulageCompany \cup \exists partOf.haulageCompany)$$

$$haulageTruckDriver \equiv person \cap \exists drives.truck \cap \exists worksFor.(\exists partOf.haulageCompany)$$

$$vanDriver \equiv person \cap \exists drives.van$$

$$busDriver \equiv person \cap \exists drives.bus$$

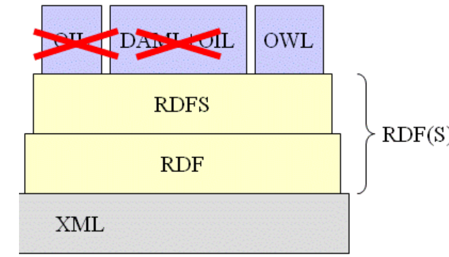
$$whiteVanMan \equiv man \cap \exists drives.(whiteThing \cap van)$$

Table of Contents

OWL

Web Ontology Language

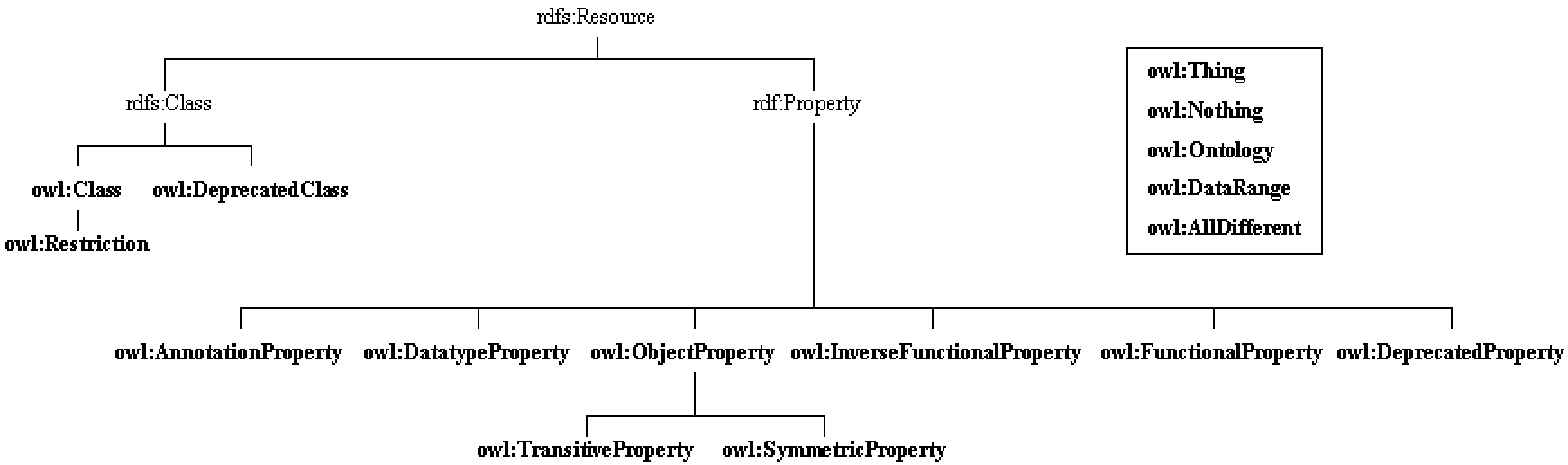
Built on top of RDF(S) and renaming DAML+OIL primitives



3 layers:
- OWL Lite
  - A small subset of primitives
  - Easier for frame-based tools to transition to
- OWL DL
  - Description logic
  - Decidable reasoning
- OWL Full
  - RDF extension, allows metaclasses

Several syntaxes:
- Abstract syntax
- Manchester syntax
- RDF/XML

Class taxonomy of the OWL KR ontology



Property list of the OWL KR ontology

Property name	domain	range
owl:intersectionOf	owl:Class	rdf:List
owl:unionOf	owl:Class	rdf:List
owl:complementOf	owl:Class	owl:Class
owl:oneOf	owl:Class	rdf:List
owl:onProperty	owl:Restriction	rdf:Property
owl:allValuesFrom	owl:Restriction	rdfs:Class
owl:hasValue	owl:Restriction	*not specified*
owl:someValuesFrom	owl:Restriction	rdfs:Class
owl:minCardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite: {0,1} OWL DL/Full: {0,..,N}
owl:maxCardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite: {0,1} OWL DL/Full: {0,..,N}
owl:cardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite: {0,1} OWL DL/Full: {0,..,N}
owl:inverseOf	owl:ObjectProperty	owl:ObjectProperty
owl:sameAs	owl:Thing	owl:Thing
owl:equivalentClass	owl:Class	owl:Class
owl:equivalentProperty	rdf:Property	rdf:Property
owl:sameIndividualAs	owl:Thing	owl:Thing
owl:differentFrom	owl:Thing	owl:Thing
owl:disjointWith	owl:Class	owl:Class
owl:distinctMembers	owl:AllDifferent	rdf:List
owl:versionInfo	*not specified*	*not specified*
owl:priorVersion	owl:Ontology	owl:Ontology
owl:incompatibleWith	owl:Ontology	owl:Ontology
owl:backwardCompatibleWith	owl:Ontology	owl:Ontology
owl:imports	owl:Ontology	owl:Ontology

OWL: Most common constructors in class definitions and axioms for classes, properties and individuals

Intersection:	$C_1 \cap ... \cap C_n$	intersectionOf	Human ∩ Male
Union:	$C_1 \cup ... \cup C_n$	unionOf	Doctor ∪ Lawyer
Negation:	$\neg C$	complementOf	¬Male
Nominals:	$\{x_1\} \cup ... \cup \{x_n\}$	oneOf	{john} ∪ ... ∪ {mary}
Universal restriction:	$\forall P.C$	allValuesFrom	∀hasChild.Doctor
Existential restriction:	$\exists P.C$	someValuesFrom	∃hasChild.Lawyer
Maximum cardinality:	$\leq nP$	maxCardinality	≤3hasChild
Minimum cardinality:	$\geq nP$	minCardinality	≥1hasChild
Specific Value:	$\exists P.\{x\}$	hasValue	∃hasColleague.{Matthew}

Subclass	$C1 \subseteq C2$	subClassOf	Human ⊆ Animal ∩ Biped
Equivalence	$C1 \equiv C2$	equivalentClass	Man ≡ Human ∩ Male
Disjointness	$C1 \cap C2 \subseteq \bot$	disjointWith	Male ∩ Female ⊆ ⊥
Subproperty	$P1 \subseteq P2$	subPropertyOf	hasDaughter ⊆ hasChild
Equivalence	$P1 \equiv P2$	equivalentProperty	cost ≡ price
Inverse	$P1 \equiv P2-$	inverseOf	hasChild ≡ hasParent-
Transitive	$P+ \subseteq P$	TransitiveProperty	ancestor+ ⊆ ancestor
Functional	$T \subseteq \leq 1P$	FunctionalProperty	T ⊆ ≤1hasMother
InverseFunctional	$T \subseteq \leq 1P-$	InverseFunctionalProperty	T ⊆ ≤1hasPassportID-
Equivalence	$\{x1\} \equiv \{x2\}$	sameIndividualAs	{oeg:OscarCorcho}≡{img:Oscar}
Different	$\{x1\} \equiv \neg\{x2\}$	differentFrom, AllDifferent	{john} ≡ ¬{peter}

OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom

Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*; restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*; restricted to {0,1}),
owl:cardinality (*daml:cardinality*; restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*)
owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom

Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*; restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*; restricted to {0,1}),
owl:cardinality (*daml:cardinality*; restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*)
owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

R

$R \subseteq S$

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

$C \subseteq D$

OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom

Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*; restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*; restricted to {0,1}),
owl:cardinality (*daml:cardinality*; restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*)
owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*; restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*; restricted to {0,1}),
owl:cardinality (*daml:cardinality*; restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

$\forall R.C$

$\exists R.C$

$\leq nR$

$\geq nR$

$= nR$

$C \cap D$

OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValue

Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*; restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*; restricted to {0,1}),
owl:cardinality (*daml:cardinality*; restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*)
owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*)
owl:Nothing (*daml:Nothing*)

$(+)R$

$\top$

$\bot$

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

R^{-1}

$C \equiv D$

$R \equiv S$

Abox

OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom
Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom
Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

$\exists R.\{x\}$

$\{x_1,\ldots,x_n\}$

$\neg C$

$C \cup D$

$C \cap D \subseteq \bot$

owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

Table of Contents

Basic Inference Tasks

- **Subsumption – check knowledge is correct (captures intuitions)**
  - Does C subsume D w.r.t. ontology O? (in *every* model I of O, $C^I \subseteq D^I$)
- **Equivalence – check knowledge is minimally redundant (no unintended synonyms)**
  - Is C equivalent to D w.r.t. O? (in *every* model I of O, $C^I = D^I$)
- **Consistency – check knowledge is meaningful (classes can have instances)**
  - Is C satisfiable w.r.t. O? (there exists *some* model I of O s.t. $C^I \neq \emptyset$)
- **Instantiation and querying**
  - Is x an instance of C w.r.t. O? (in *every* model I of O, $x^I \in C^I$)
  - Is (x,y) an instance of R w.r.t. O? (in *every* model I of O, $(x^I,y^I) \in R^I$)
- **All reducible to KB satisfiability or concept satisfiability w.r.t. a KB**
- **Can be decided using highly optimised tableaux reasoners**

Tableaux Algorithms

- **Try to prove satisfiability of a knowledge base**
- **How do they work**
  - They try to build a model of input concept C
    - Tree model property
      - If there is a model, then there is a tree shaped model
    - If no tree model can be found, then input concept unsatisfiable
  - Decompose C syntactically
    - Work on concepts in negation normal form (De Morgan's laws)
    - Use of tableaux expansion rules
    - If non-deterministic rules are applied, then there is search
  - Stop (and backtrack) if clash
    - E.g. $A(x), \neg A(x)$
  - Blocking (cycle check) ensures termination for more expressive logics

- **The algorithm finishes when no more rules can be applied or a conflict is detected**

Tableaux rules for ALC and for transitive roles

$x \bullet \{C_1 \sqcap C_2, \ldots\}$	$\rightarrow_\sqcap$	$x \bullet \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$
$x \bullet \{C_1 \sqcup C_2, \ldots\}$	$\rightarrow_\sqcup$	$x \bullet \{C_1 \sqcup C_2, C, \ldots\}$ for $C \in \{C_1, C_2\}$
$x \bullet \{\exists R.C, \ldots\}$	$\rightarrow_\exists$	$x \bullet \{\exists R.C, \ldots\}$ $R \downarrow$ $y \bullet \{C\}$
$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{\ldots\}$	$\rightarrow_\forall$	$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{C, \ldots\}$
$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{\ldots\}$	$\rightarrow_{\forall_+}$	$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{\forall R.C, \ldots\}$

Tableaux example

- Example
  - $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)$

Description Logics Reasoning with Tableaux

Use tableaux algorithms to determine whether the following formulae are satisfiable or not

Exercise 1: $\exists R.(\exists R.D) \land \exists S.\neg D \land \forall S.(\exists R.D)$

Exercise 2: $\exists R.(C \lor D) \land \forall R.\neg C \land \neg \exists R.D$

Description Logics Reasoning

Develop a sample ontology in the domain of people, pets, vehicles, and newspapers

- Understand the basic reasoning mechanisms of description logics

Subsumption

Automatic classification: an ontology built collaboratively

Instance classification

Detecting redundancy

Consistency checking: unsatisfiable restrictions in a Tbox (are the classes coherent?)

Interesting results (I). Automatic classification

And old lady is a person who is elderly and female.
Old ladies must have some animal as pets and all their pets are cats.

$$elderly \subseteq person \cap adult$$

$$woman \equiv person \cap female \cap adult$$

$$catOwner \equiv person \cap \exists hasPet.cat$$

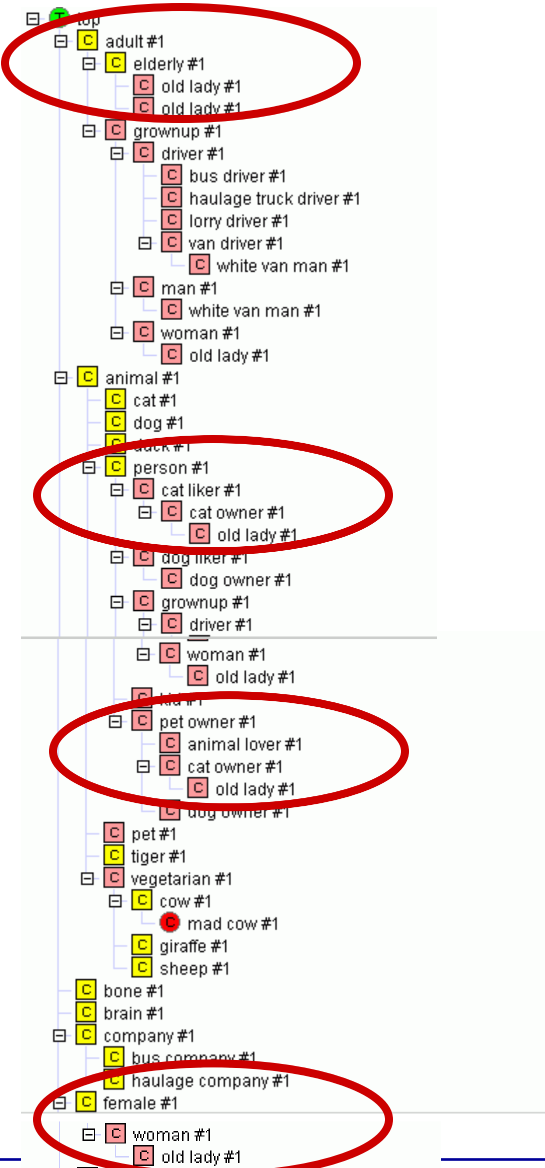
$$oldLady \equiv person \cap female \cap elderly$$

$$oldLady \subseteq \exists hasPet.animal \cap \forall hasPet.cat$$

We obtain:

Old ladies must be women.
Every old lady must have a pet cat
Hence, every old lady must be a cat owner

$$oldLady \subseteq woman \cap elderly \cap catOwner$$



Interesting results (II). Instance classification

A pet owner is a person who has animal pets
Old ladies must have some animal as pets and all their pets are cats.
Has pet has domain person and range animal
Minnie is a female, elderly, who has a pet called Tom.

$$petOwner \equiv person \cap \exists hasPet.animal$$

$$oldLady \subseteq \exists hasPet.animal \cap \forall hasPet.cat$$

$$hasPet \subseteq (person, animal)$$

$$Minnie \in female \cap elderly$$

$$hasPet(Minnie, Tom)$$

We obtain:

Minnie is a person
Hence, Minnie is an old lady
Hence, Tom is a cat

$$Minnie \in person; Tom \in animal$$

$$Minnie \in petOwner$$

$$Minnie \in oldLady$$

$$Tom \in cat$$

Interesting results (III). Instance classification and redundancy detection

An animal lover is a person who has at least three pets
Walt is a person who has pets called Huey, Louie and Dewey.

$$animalLover \equiv person \cap (\geq 3 hasPet)$$

$$Walt \in person$$

$$hasPet(Walt, Huey)$$

$$hasPet(Walt, Louie)$$

$$hasPet(Walt, Dewey)$$

We obtain:

Walt is an animal lover
Walt is a person is redundant

$$Walt \in animalLover$$

Interesting results (IV). Instance classification

A van is a type of vehicle
A driver must be adult
A driver is a person who drives vehicles
A white van man is a man who drives vans and white things
White van mans must read only tabloids
Q123ABC is a white thing and a van
Mick is a male who reads Daily Mirror and drives Q123ABC

$$van \subseteq vehicle$$

$$driver \subseteq adult$$

$$driver \equiv person \cap \exists drives.vehicle$$

$$whiteVanMan \equiv man \cap \exists drives.(van \cap whiteThing)$$

$$whiteVanMan \subseteq \forall reads.tabloid$$

$$Q123ABC \in whiteThing \cap van$$

$$Mick \in male$$

$$reads(Mick, DailyMirror)$$

$$drives(Mick, Q123ABC)$$

We obtain:

Mick is an adult
Mick is a white van man
Daily Mirror is a tabloid

$$Mick \in adult$$

$$Mick \in whiteVanMan$$

$$DailyMirror \in tabloid$$

Interesting results (V). Consistency checking

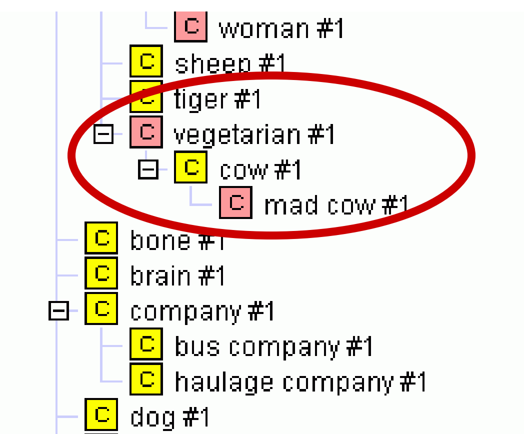
Cows are vegetarian.
A vegetarian is an animal that does not eat animals nor parts of animals.
A mad cow is a cow that eats brains that can be part of a sheep

$$cow \subseteq vegetarian$$

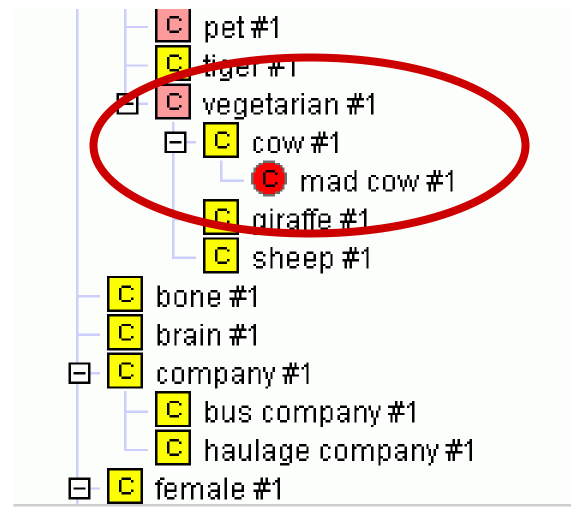
$$vegetarian \equiv animal \cap \forall eats.\neg animal \cap$$
$$\forall eats.\neg(\exists partOf.animal))$$

$$madCow \equiv cow \cap \exists eats.(brain \cup \exists partOf.sheep)$$

$$(animal \cup \exists partOf.animal) \cap (plant \cup \exists partOf.plant) \subseteq \perp$$



We obtain:
Mad cow is unsatisfiable



When to use a classifier

1. **At author time (pre-coordination): As a compiler**
   - Ontologies will be delivered as “pre-coordinated” ontologies to be used without a reasoner
   - To make extensions and additions quick, easy, and responsive, distribute developments, empower users to make changes
   - Part of an ontology life cycle

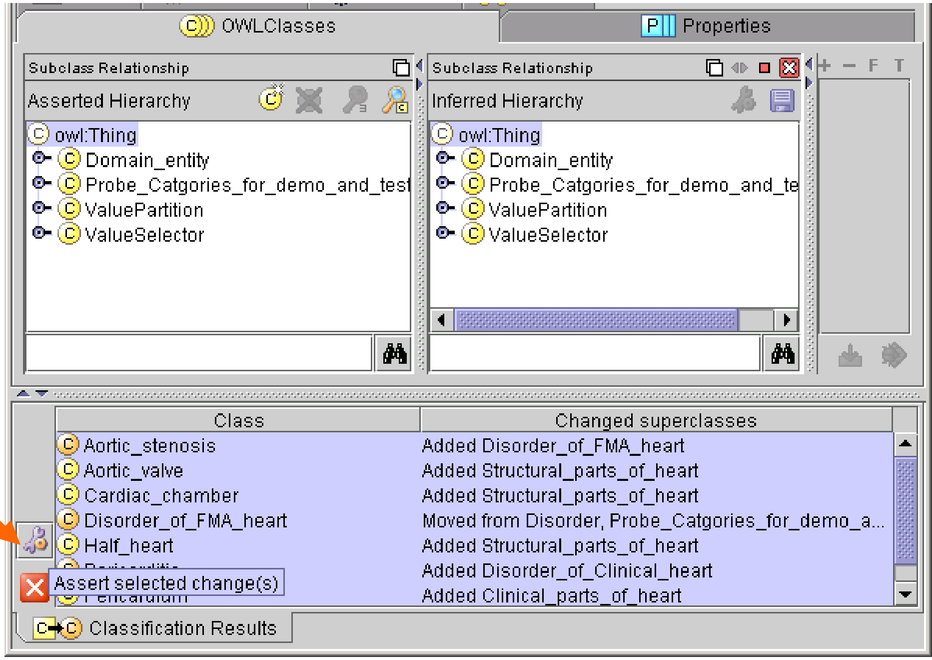
2. **At delivery time (post-coordination): as a normalisation service**
   - Many fixed ontologies are too big and too small
     - Too big to find things; too small to contain what you need
       - Create them on the fly
   - Part of an ontology service

3. **At application time (inference): as a reasoner**
   - Decision support, query optimisation, schema integration, etc.
   - Part of a reasoning service

1. Pre-coordinated delivery: classifier as compiler

- **Develop an ontology**
  - A classifier can be used to detect and correct inconsistencies
- **Classify the ontology**
- **Commit classifier results to a pre-coordinated ontology**

Assert ("Commit") changes inferred by classifier



- **Deliver it**
  - In OWL-Lite or RDFS
- **Use RDQL, SPARQL, or your favourite RDF(S) query tool**

2. Post Coordination: classifier as a service

- **Logic based ontologies act as a conceptual lego**
  - Modularisation/Normalisation is needed to make them easier to maintain

hand

extremity

body

gene

protein

cell

expression

chronic

acute

abnormal
normal

Lung
inflammation
infection

bacterial

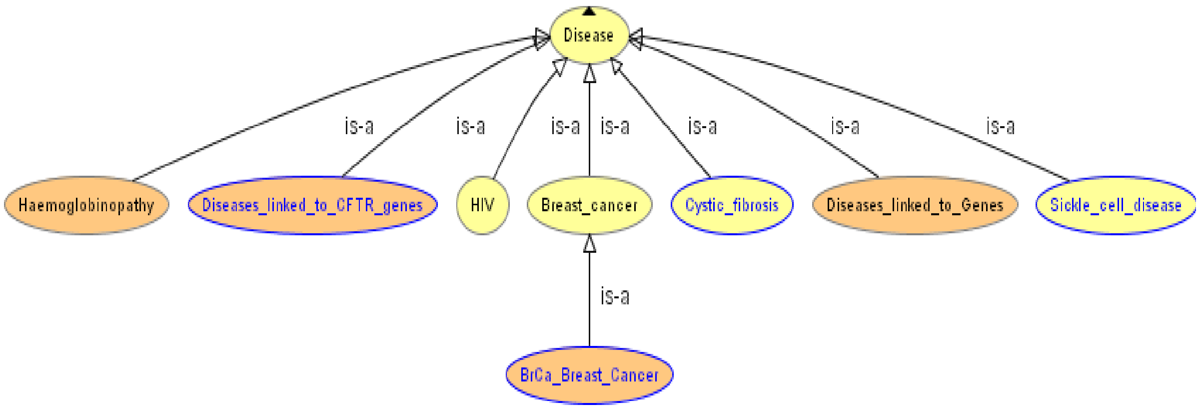
ischaemic

deletion

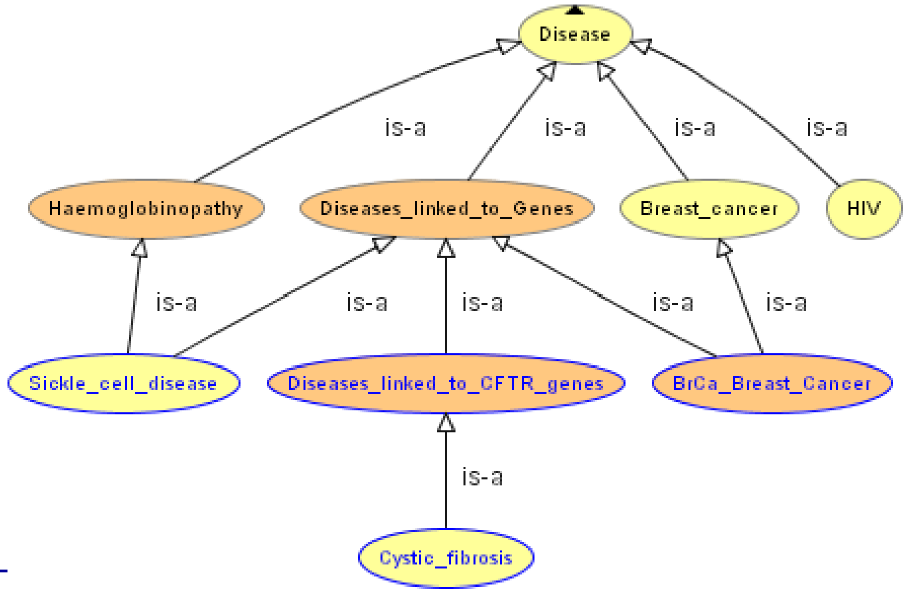
polymorphism

2. Post Coordination. Example (I)

- **Build a simple tree**

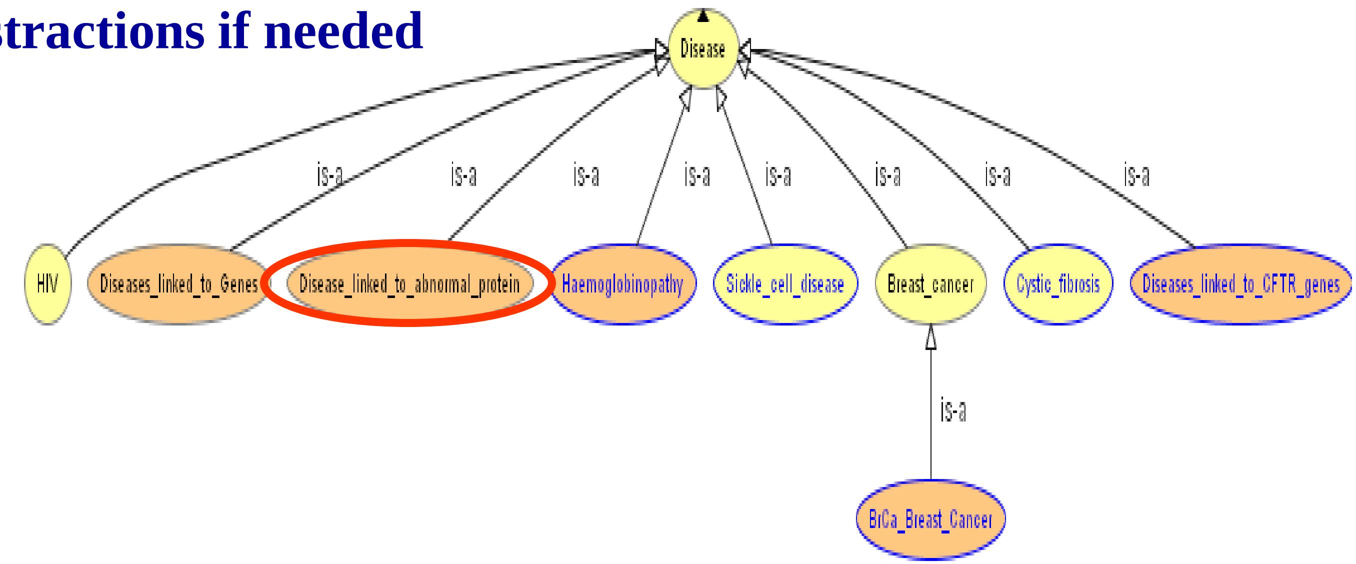


- **Let the classifier organise it and check consistency**

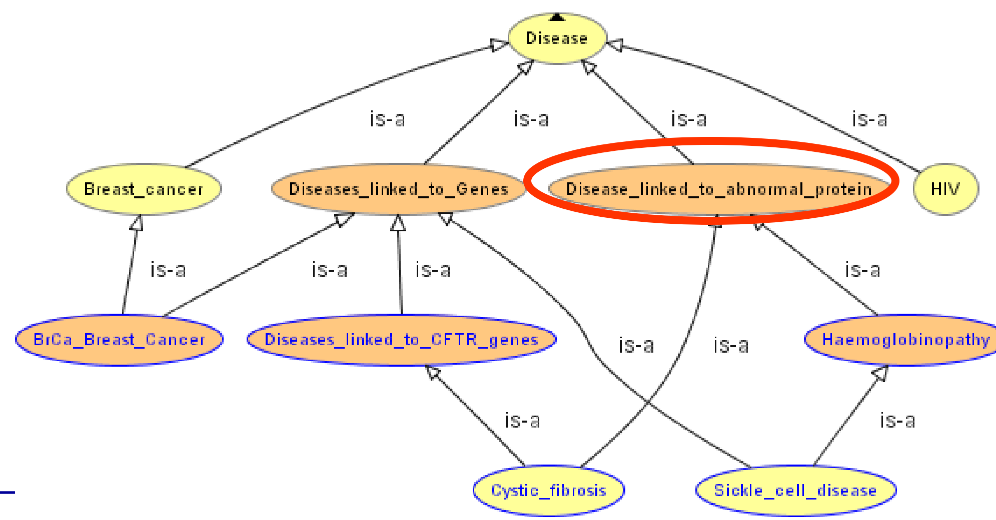


2. Post Coordination. Example (II)

- **Add more abstractions if needed**



- **Let the classifier organise it again and check consistency!!**



3. Inference at application run-time

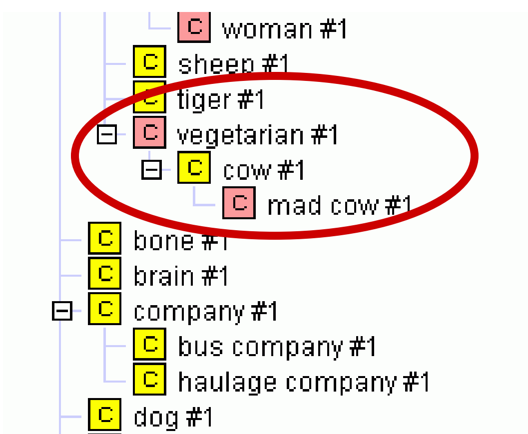
Cows are vegetarian.
A vegetarian is an animal that does not eat animals nor parts of animals.
A mad cow is a cow that eats brains that can be part of a sheep

$$cow \subseteq vegetarian$$

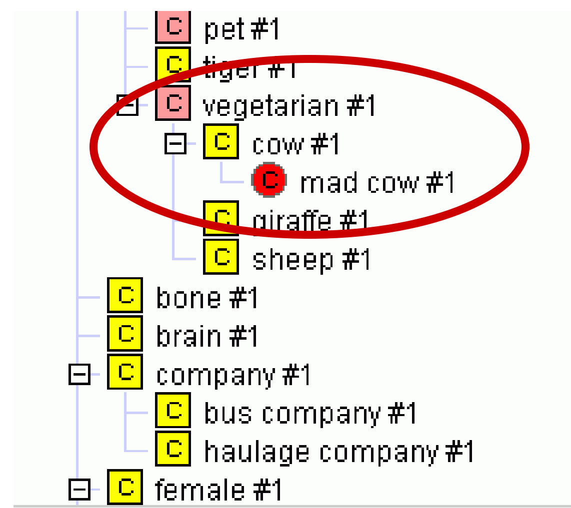
$$vegetarian \equiv animal \cap \forall eats.\neg animal \cap$$
$$\forall eats.\neg(\exists partOf.animal))$$

$$madCow \equiv cow \cap \exists eats.(brain \cup \exists partOf.sheep)$$

$$(animal \cup \exists partOf.animal) \cap (plant \cup \exists partOf.plant) \subseteq \perp$$



We obtain:
Mad cow is unsatisfiable



OWL Classifier limitations

- **Numbers and strings**
  - Simple concrete data types in spec
  - User defined XML data types enmeshed in standards disputes
  - No standard classifier deals with numeric ranges
    - Although several experimental ones do

- **is-part-of and has-part**
  - Totally doubly-linked structures scale horridly

- **Handling of individuals**
  - Variable with different classifiers
  - oneOf works badly with all classifiers at the moment

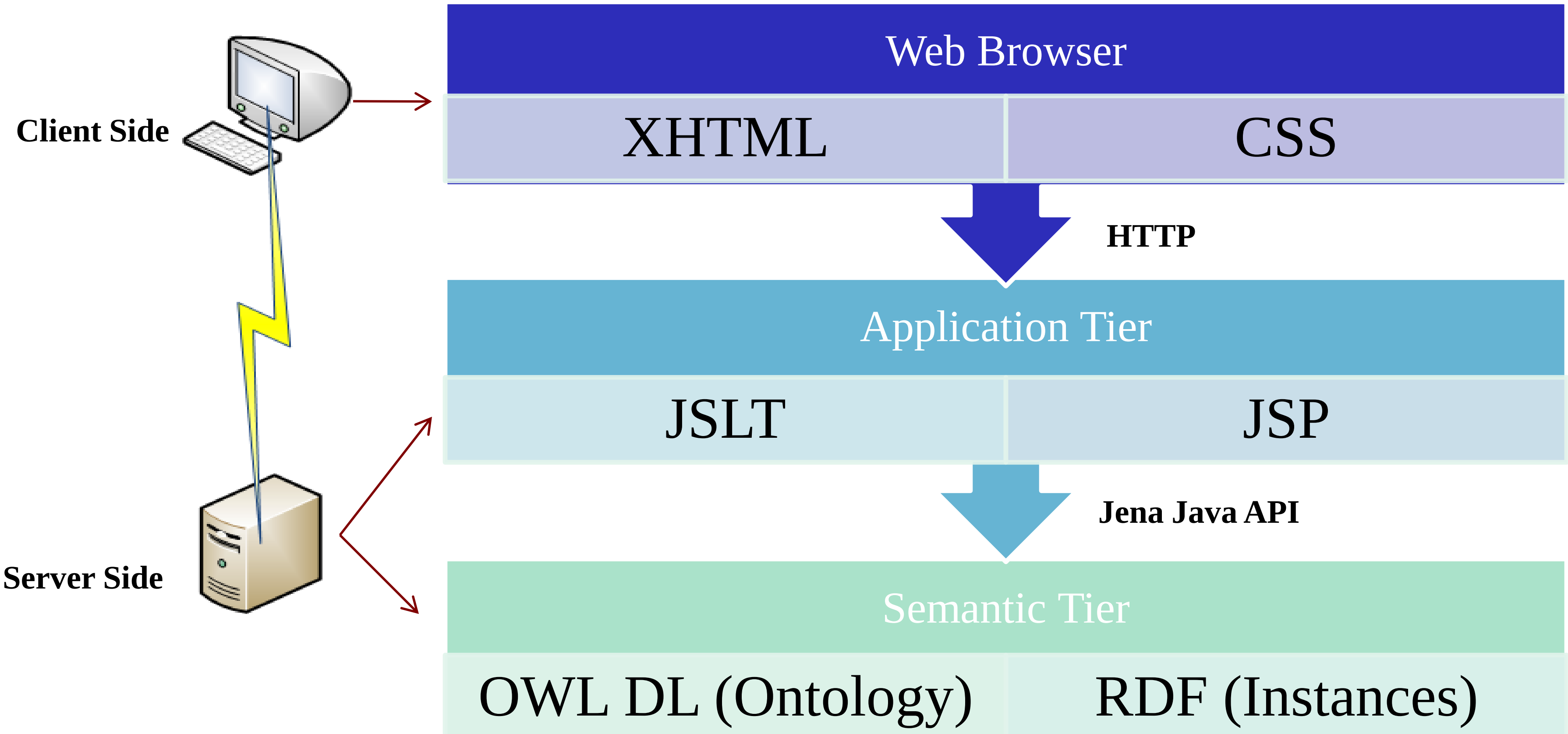
Table of Contents

Table of Contents

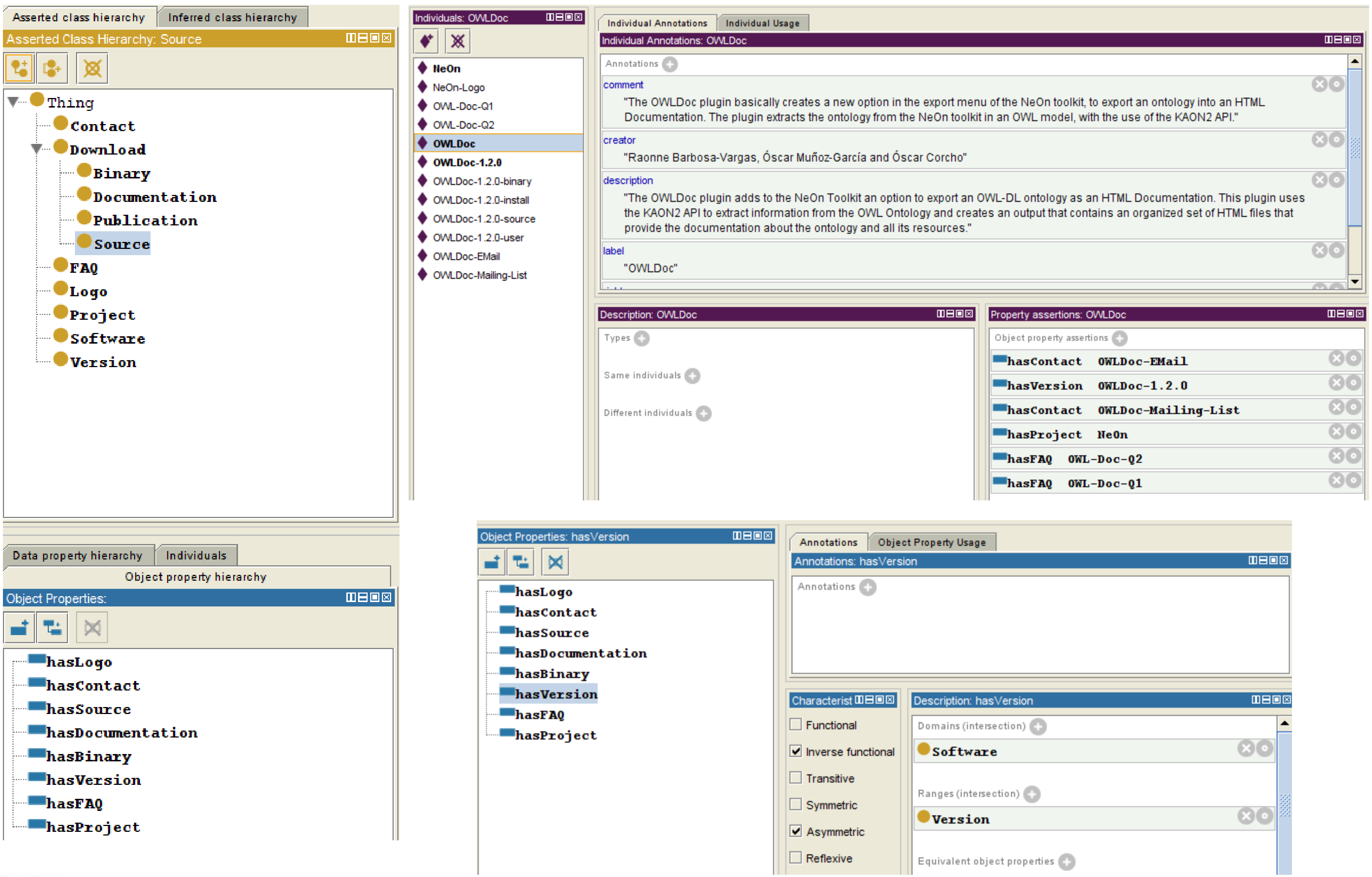
Añadir cosas de OWL APIs

Table of Contents

Application Architecture



Software Ontology



Example Code (logo.jsp)

```
<%@ page contentType="text/html; charset=utf-8" language="java" import="com.hp.hpl.jena.ontology.*, com.hp.hpl.jena.rdf.model.*"
 errorPage="" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page isELIgnored="false" %>

<%
    Individual project = (Individual) session.getAttribute("project");
    pageContext.setAttribute("title", project.getPropertyValue((AnnotationProperty) application.getAttribute("title")));
    ObjectProperty hasLogo = (ObjectProperty) application.getAttribute("hasLogo");
    AnnotationProperty identifier = (AnnotationProperty) application.getAttribute("identifier");
    OntResource logo = (OntResource) project.getPropertyValue(hasLogo);
    Literal logoIdentifier = (Literal) logo.getPropertyValue(identifier).as(Literal.class);
    pageContext.setAttribute("logoIdentifier", logoIdentifier);
    session.setAttribute("label", project.getLabel(""));
%>
```

Jena Code

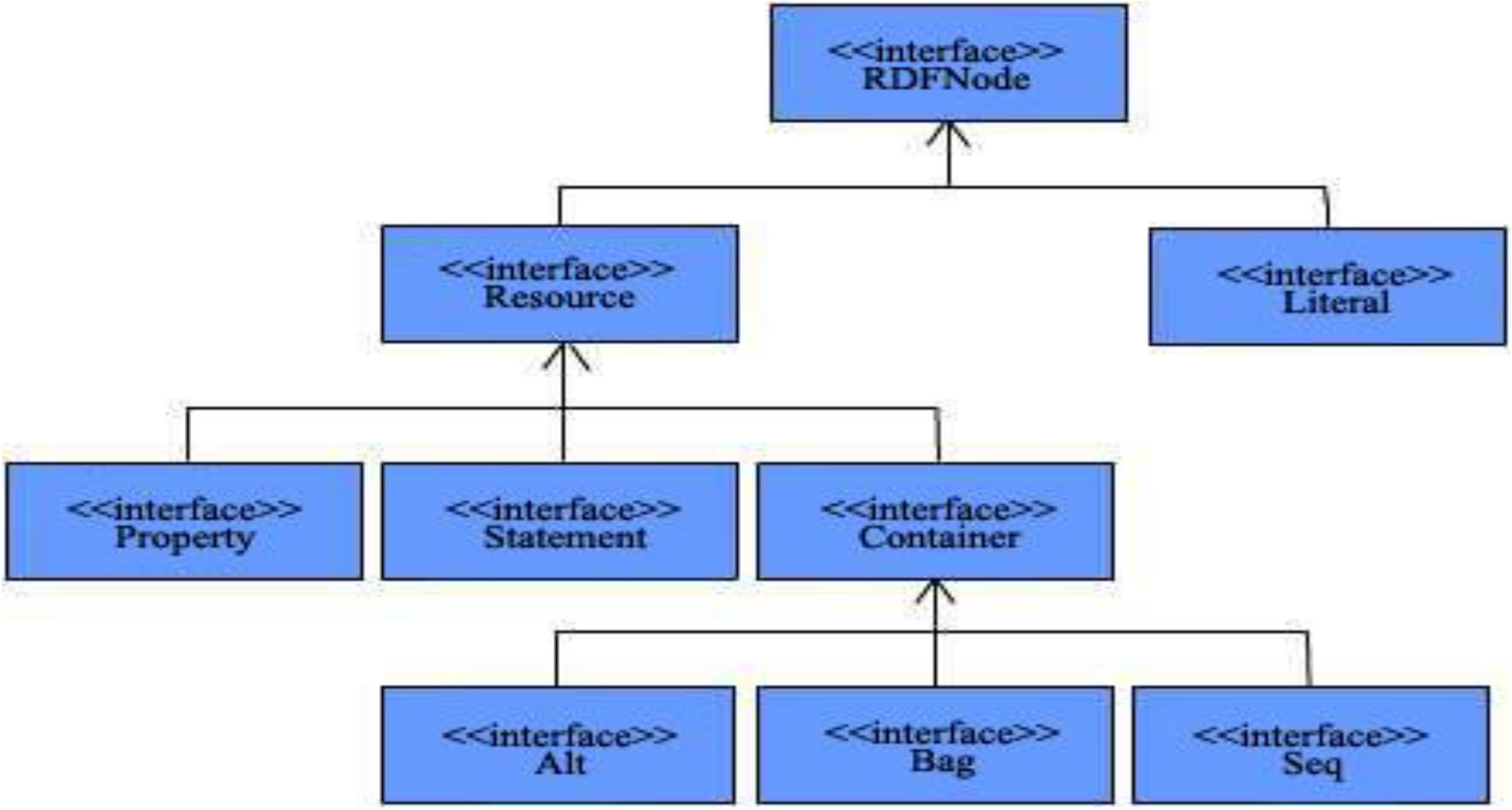
```
<a href="<c:url value="${projectIdentifier.string}"/>">
    <img src="<c:url value="${logoIdentifier.string}"/>"
        alt="<c:out value="${label}"/>"
        longdesc="<c:out value="${title.string}"/>"
        style="border-style: none"/>
</a>
```

JSLT Code

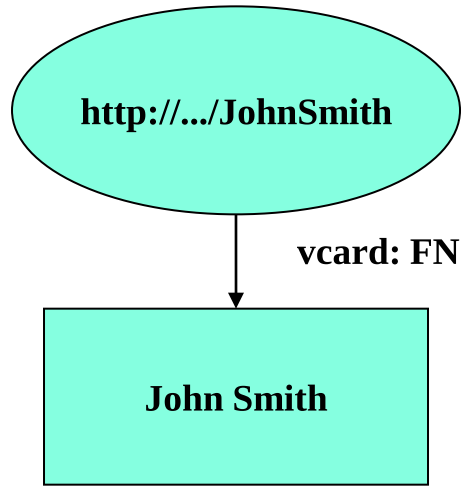
Screenshot


OEG Software
Archivo Edición Visualización Historial Favoritos Ventana Ayuda
http://localhost:8080/oegsw/index.jsp;jsessionid=10196974EF6D3C20FB411C3EAEEC9F58?sw=http%3a%2f%2fwww.oeg-upm.net%2fsoftw
Google
Apple Yahoo! Google Maps YouTube Wikipedia News (310) Popular
Ontology Engineer ingGroup
OEG SOFTWARE
OWLDoc
NeOn
NeOn OWL-Doc Plugin
The OWLDoc plugin adds to the NeOn Toolkit an option to export an OWL-DL ontology as an HTML Documentation. This plugin uses the KAON2 API to extract information from the OWL Ontology and creates an output that contains an organized set of HTML files that provide the documentation about the ontology and all its resources.
NeOn OWL-Doc is open source and published under the XXXXX license.
More Information
Authors: Raonne Barbosa-Vargas, Óscar Muñoz-García and Óscar Corcho
Mailing List
ocorcho@fi.upm.es
Download
Version | Date | Binaries | Source | Documentation
NeOn OWL-DOC 1.2.0 | 7/4/2008 | Usage Manual How to install
FAQ
Is it possible to generate the documentation for an ontology in F-Logic?
No, it isn't. Although it is possible to create a F-Logic ontology using the NeOn Toolkit, OWL-Doc only generates documentation for ontologies expressed in OWL.
Can I use NeOn OWL-Doc as a stand-alone application?
No, you can't. NeOn OWL-Doc is a plugin that can only be deployed inside the NeOn Toolkit.
© NeOn: Lifecycle Support for Networked Ontologies (all rights reserved). This product development has been supported by NeOn, an Integrated Project funded by the European Commission. Further information available from http://www.neon-project.org. The information in this website is subject to a disclaimer notice.
W3C XHTML 1.1
W3C CSS

Jena API Structure

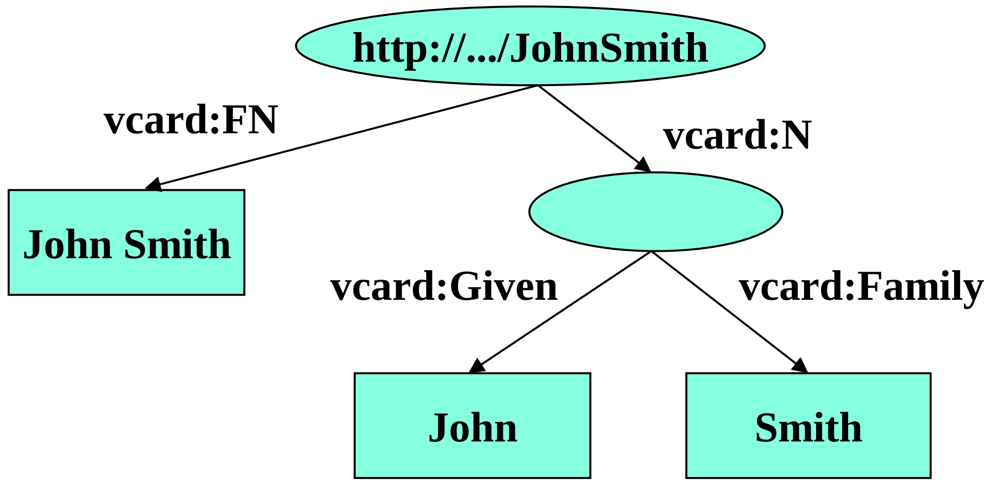


Data Model




```
// some definitions
static String personURI = "http://somewhere/JohnSmith";
static String fullName = "John Smith";
// create an empty Model
Model model = ModelFactory.createDefaultModel();
// create the resource
Resource johnSmith = model.createResource(personURI);
// add the property
johnSmith.addProperty(VCARD.FN, fullName);
```

Another data model




```
// some definitions
String personURI = "http://somewhere/JohnSmith"; String givenName = "John";
String familyName = "Smith";
String fullName = givenName + " " + familyName;
// create an empty
Model Model model = ModelFactory.createDefaultModel();
// create the resource
// and add the properties cascading style
Resource johnSmith = model.createResource(personURI)
    .addProperty(VCARD.FN, fullName)
    .addProperty(VCARD.N, model.createResource()
    .addProperty(VCARD.Given, givenName)
    .addProperty(VCARD.Family, familyName));
```

Statements

```
// list the statements in the Model
StmtIterator iter = model.listStatements();
// print out the predicate, subject and object of each statement
while (iter.hasNext())
{
   Statement stmt = iter.nextStatement(); // get next statement
   Resource subject = stmt.getSubject(); // get the subject
   Property predicate = stmt.getPredicate(); // get the predicate
   RDFNode object = stmt.getObject(); // get the object
   System.out.print(subject.toString());
   System.out.print(" " + predicate.toString() + " ");
   if (object instanceof Resource) {
      System.out.print(object.toString());
    }
    else { // object is a literal
      System.out.print(" \"" + object.toString() + "\"");
    }
    System.out.println(" .");
} // end of while
```

```
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#N anon:14df86:ecc3dee17b:-7fff

anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Family "Smith"

anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Given "John"

http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#FN
    “John Smith" .
```

Writing RDF

```
Model model = ModelFactory.createDefaultModel();
Resource jsmith =
model.createResource(“http://somewhere/johnsmith”)
    .addProperty(VCARD.FN, “John Smith”)
    .addProperty(VCARD.N, model.createResource()
    .addProperty(VCARD.Given,”John”)
    .addProperty(VCARD.Family, “Smith”));
model.write(new PrintWriter(System.out));
```

```
model.write(System.out, "RDF/XML-ABBREV");

model.write(System.out, "N-TRIPLE");
```

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
 >
  <rdf:Description rdf:nodeID='A0'>
    <vcard:Given>John</vcard:Given>
    <vcard:Family>Smith</vcard:Family>
  </rdf:Description>
  <rdf:Description rdf:about='http://somewhere/johnsmith'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID='A0'/>
  </rdf:Description>
</rdf:RDF>
```

Reading RDF

```
// create an empty model
Model model = ModelFactory.createDefaultModel();

// use the FileManager to find the input file
InputStream in = FileManager.get().open( inputFileName );
if (in == null) {
    throw new IllegalArgumentException("File not found");
}

// read the RDF/XML file
model.read(in, "");

// write it to standard out
model.write(System.out);
```

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
 >
  <rdf:Description rdf:nodeID="A0">
    <vcard:Family>Smith</vcard:Family>
    <vcard:Given>John</vcard:Given>
  </rdf:Description>
  <rdf:Description rdf:about='http://somewhere/JohnSmith/'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID="A0"/>
  </rdf:Description>
...
</rdf:RDF>
```


Navigating a model

```
// retrieve the John Smith vcard resource from the model
Resource vcard = model.getResource(johnSmithURI);
```

Three ways of retrieving property values:

```
// retrieve the value of the N property
Resource name = (Resource) vcard.getProperty(VCARD.N).getObject();

// retrieve the value of the N property
Resource name = vcard.getProperty(VCARD.N).getResource();

// retrieve the given name property
String fullName = vcard.getProperty(VCARD.N).getString();
```

Multiple values in properties

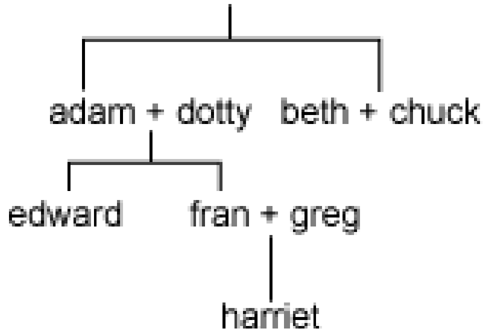
```
// add two nickname properties to vcard
vcard.addProperty(VCARD.NICKNAME, "Smithy")
     .addProperty(VCARD.NICKNAME, "Adman");

// set up the output
System.out.println("The nicknames of \"" + fullName + "\" are:");

// list the nicknames
StmtIterator iter = vcard.listProperties(VCARD.NICKNAME);
while (iter.hasNext()) {
  System.out.println(" " + iter.nextStatement() .getObject() .toString());
}
```

Querying a model

```
// select all the resources with a VCARD.FN property
ResIterator iter = model.listSubjectsWithProperty(VCARD.FN);
if (iter.hasNext()) {
  System.out.println("The database contains vcards for:");
  while (iter.hasNext()) {
    System.out.println(" " + iter.nextStatement()
                                        .getProperty(VCARD.FN) .getString());
  }
} else {
  System.out.println("No vcards were found in the database");
}
```

```
The database contains vcards for:
 Sarah Jones
 John Smith
 Matt Jones
 Becky Smith
```

Create resources




```
// URI declarations
String familyUri = "http://family/";
String relationshipUri = "http://purl.org/vocab/relationship/";

// Create an empty Model
Model model = ModelFactory.createDefaultModel();

// Create a Resource for each family member, identified by their URI
Resource adam = model.createResource(familyUri+"adam");
Resource beth = model.createResource(familyUri+"beth");
Resource dotty = model.createResource(familyUri+"dotty");
// and so on for other family members

// Create properties for the different types of relationship to represent
Property childOf = model.createProperty(relationshipUri,"childOf");
Property parentOf = model.createProperty(relationshipUri,"parentOf");
Property siblingOf = model.createProperty(relationshipUri,"siblingOf");
Property spouseOf = model.createProperty(relationshipUri,"spouseOf");

// Add properties to adam describing relationships to other family members
adam.addProperty(siblingOf,beth);
adam.addProperty(spouseOf,dotty);
adam.addProperty(parentOf,edward);

// Can also create statements directly . . .
Statement statement = model.createStatement(adam,parentOf,fran);

// but remember to add the created statement to the model
model.add(statement);
```

Querying a model

```
// List everyone in the model who has a child:
ResIterator parents = model.listSubjectsWithProperty(parentOf);

// Because subjects of statements are Resources, the method returned a ResIterator
while (parents.hasNext()) {

  // ResIterator has a typed nextResource() method
  Resource person = parents.nextResource();

  // Print the URI of the resource
  System.out.println(person.getURI());
}

// Can also find all the parents by getting the objects of all "childOf" statements
// Objects of statements could be Resources or literals, so the Iterator returned
// contains RDFNodes
NodeIterator moreParents = model.listObjectsOfProperty(childOf);

// To find all the siblings of a specific person, the model itself can be queried
NodeIterator siblings = model.listObjectsOfProperty(edward, siblingOf);

// But it's more elegant to ask the Resource directly
// This method yields an iterator over Statements
StmtIterator moreSiblings = edward.listProperties(siblingOf);
```

Using selectors to query a model

```
// Find the exact statement "adam is a spouse of dotty"
model.listStatements(adam,spouseOf,dotty);

// Find all statements with adam as the subject and dotty as the object
model.listStatements(adam,null,dotty);

// Find any statements made about adam
model.listStatements(adam,null,null);

// Find any statement with the siblingOf property
model.listStatements(null,siblingOf,null);
```

Exercise

- **Objective**
  - **Understand how to use an RDF(S) management API**
- **Tasks**
  - **Read an ontology in RDF(S) from two files:**
    - **GP_Santiago.rdf (conceptualization)**
    - **GP_Santiago.rdfs (instances)**
  - **Write the class hierarchy of the ontology, including the instances of each class**

Hands-on

- **To read an ontology in RDF(S) from two files:**
  - GP_Santiago.rdf (conceptualization)
  - GP_Santiago.rdfs (instances)
- **To write the class hierarchy of the ontology, including the instances of each class:**

```
Class Practica2:MedioTransporte
  Class Practica2:Tren
  Class Practica2:Bicicleta
    Instance Practica2:GP_Santiago_Instance_70
  Class Practica2:Automovil
  Class Practica2:AutoBus
  Class Practica2:APie
Class Practica2:InfraEstructuraTransporte
  Class Practica2:ViaFerrea
  Class Practica2:Sendero
  Class Practica2:Carretera
    Instance Practica2:A6
...
```


Set up

- **Requirements:**
  - Java JDK 5
  - Eclipse (optional)

- **Create a directory for your project**
- **Install Jena from the USB:**
  - Unzip *Jena-2.5.5.zip/**lib*** in the project directory
- **Copy the ontologies from the USB:**
  - Copy *ontologies/**rdf*** in the project directory

Or copy the JenaProjectTemplate directory in your computer

- **In Eclipse:**
  - Create a new Java project (from existing source)
  - Append the Jena libraries to your classpath if needed (check JDK libs)
  - Write Java code using the Jena API

    http://jena.sourceforge.net/javadoc/index.html
  - Compile
  - Run

Hints

- **Create ontology model:**

```
public static OntModel
  createOntologyModel(OntModelSpec spec)
```

- **Read the ontology in the file**

```
Model read(java.lang.String url)
```

- **Add all the statements in another model to this model**

```
Model add(Model m)
```

More hints

- **List root classes**

```
ExtendedIterator listHierarchyRootClasses()
```

- **List subclasses of a class**

```
ExtendedIterator listSubClasses(boolean
  direct)
```

- **List instances of a class**

```
ExtendedIterator listInstances(boolean direct)
```