Ontology

Description Logic

- is a formal language at the basis of ontolgies description
- It is decidable
  - is a subset of C2 a decidable subset of FOL (FOL is semi-decidable)
- Many descrition logic languages with increasing expressivity and increasing complexity

Description Logic

A descriptive language **L** consists of three finite sets: **(NC**; **NR**; **Ob)**.

- Elements in **NC** are indicated with letters A, B, ecc. and are called **atomic concepts** of **L**;
- Elements in **NR** are indicated with letters r, s, ecc. and are called **roles** of **L**;
- Elements in **Ob** are indicated with letters a, b, ecc. are called **object (names)** of **L**.

ALC

- ALC = Attribute concept Language with Complements
- A Basic Description Logic
- **Example Axioms**
  - HappyMan ≡ Human ⊓ ¬Female ⊓ ∃ married.Doctor ⊓ ∀ hasChild.(Doctor ⊔ Professor)
  - ∃ hasChild. Human ⊑ Human
  - …
  - NC = {Human, Female, Doctor, Professor,…}
  - NR = {married, hasChild, …}

ALC syntax

- Let **NC** be a **set of concept names** and **NR** be a **set of role names**.
- The set of *ALC-concept descriptions* is the smallest set such that:
  - T, ⊥, and every concept name A ∈ NC is an *ALC-concept description*,
  - if C and D are *ALC-concept descriptions* and r ∈ NR, then C ⊓ D, C ⊔ D, ¬C, ∀r.C, and ∃r.C are *ALC-concept descriptions*.

ALC semantics

- $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ interpretation
- $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for $A \in NC$,
- $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} x \Delta^{\mathcal{I}}$ for $r \in NR$
- $T^{\mathcal{I}} = \Delta^{\mathcal{I}}$
- $\bot^{\mathcal{I}} = \emptyset$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus (C)^{\mathcal{I}}$
- $(C \sqcap D)^{\mathcal{I}} = (C)^{\mathcal{I}} \cap (D)^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} = (C)^{\mathcal{I}} \cup (D)^{\mathcal{I}}$
- $(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}. (x,y) \in r^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$
- $(\exists r.C)^{\mathcal{I}} = \{ x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}. (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$

ALC - TBox

- General concept inclusion (GCI)
  - $C \sqsubseteq D$ (C and D are ALC concepts)
  - $(C \sqsubseteq D)^{\mathcal{I}}$ iff $(C)^{\mathcal{I}} \subseteq (D)^{\mathcal{I}}$
  - $C \equiv D$ iff $C \sqsubseteq D$ and $D \sqsubseteq C$
- T-Box (Terminological Box) a set of axioms describing the concepts relations, a set of GCI
- **Example TBox**
  - Human ⊑ Animal
  - Bird ⊑ Animal ⊓ CanFly
  - Helicopter ⊑ InhanimatedObject ⊓ CanFly
  - Animal ⊓ InhanimatedObject ⊑ ⊥

ALC - ABox

- The ABox (Assertional Box) contains axioms telling:
  - that an object with name *x* belongs to an ALC concept expression:
    - $x : C$
    - $(x : C)^{\mathcal{I}}$ if $x^{\mathcal{I}} \in (C)^{\mathcal{I}}$
  - that two objects are in a role
    - $\langle x,y \rangle : r$
    - $(\langle x,y \rangle : r)^{\mathcal{I}}$ if $(x^{\mathcal{I}},y^{\mathcal{I}}) \in (r)^{\mathcal{I}}$
- **Examples**
  - b1 : Bird
  - h1 : Helicopter
  - <john, jane> : married

ALC - KB

- An ALC KB is made of a **TBox** and an **ABox**
- The TBox states the containment relations among the concepts that typically form a hierarchy of concepts

ALC to FOL

- ALC can be translated to FOL inductively
  - $[A]_x = A(x)$
  - $[C \sqcap D]_x = [C]_x \wedge [D]_x$
  - $[C \sqcup D]_x = [C]_x \vee [D]_x$
  - $[\neg C]_x = \neg [C]_x$
  - $[\exists r.C\,]_x = \exists y.\ r(x, y) \wedge [C]_y$
  - $[\forall r.C\,]_x = \forall y.\ r(x, y) \Rightarrow [C]_y$
  - $[T]_x = T$, $[\perp\,]_x = \perp$
  - $[C \sqsubseteq D] = \forall x.\ [C]_x \Rightarrow [D]_x$
  - $[C \equiv D] = \forall x.\ [C]_x \Leftrightarrow [D]_x$
  - $[a : C] = [C]_a$
  - $[<a,b> : r] = r(a,b)$

Example

- $\exists$ hasChild. Human $\sqsubseteq$ Human
- [$\exists$ hasChild. Human $\sqsubseteq$ Human] =
  $\forall$x. [$\exists$ hasChild. Human]$_x$ $\Rightarrow$ [Human]$_x$ =
  $\forall$x. $\exists$y. hasChild(x, y) $\wedge$ [Human]$_y$ $\Rightarrow$ [Human]$_x$ =
  $\forall$x. $\exists$y. hasChild(x, y) $\wedge$ Human(y) $\Rightarrow$ Human(x)

Many DLs

Many **Descriptive Logics** *(DL) derived from* ***ALC***

$\mathcal{S}$ → ALC + transitive roles (ex. Tr(isPartOf))
$\mathcal{H}$ → role hierarchy (ex., hasDaughter $\sqsubseteq$ hasChild)
$\mathcal{F}$ → functional roles
$\mathcal{O}$ → define concept as enumeration {*a1,…,an*}
$\mathcal{N}$ → cardinality restrictions (es., ≤2hasChild)
$\mathcal{Q}$ → qualified cardinality restrictions (es., ≥3hasChild.Female)
$\mathcal{I}$ → inverse roles (es., isChildOf ≡ hasChild$^-$)

Every *DL* characterized by the use of particular logical operators

OWL-Lite = $\mathcal{SHIF}$
OWL-DL = $\mathcal{SHOIN}$

Concepts

Atomic *(often indicated with letters* **A** *and* **B***)*

WOMAN

intuitively means *"WOMAN"*

Complex *(often indicated with letters* **C** *and* **D***)*

PERSON ⊓ FEMALE

reads *"PERSON and FEMALE"* or *"PERSON intersection FEMALE"*

intuitively means *"person of female genre"*

Often called

terms *or* **classes** *(because represent sets of objects)*

Equivalences

Concepts Equivalence

WOMAN ≡ PERSON ⊓ FEMALE

Intuituvely tells that *"WOMAN equals to PERSON and FEMALE"*

In general

C ≡ D

reads *"C* equals to *D"*

Express the equivalence of *"C"* and *"D"*

Concepts definition

WOMAN ≡ PERSON ⊓ FEMALE

Definition of term *"WOMAN"* from terms *"PERSON"* and *"FEMALE"*

Subsumption (sussunzione)

Concepts Subsumption

GIRL ⊑ WOMAN

Reads *"GIRL is subsumed from WOMAN"* or *"WOMAN subsumes GIRL"*

Intuitively means *"a girl is a woman"*

In general

C ⊑ D

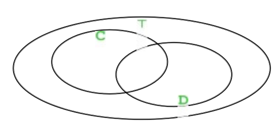
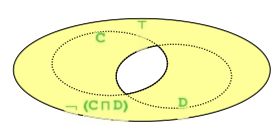
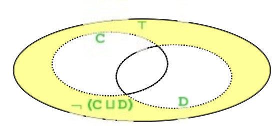
Every individual in *"C"* is also described by *"D"*

Equivalence as double subsumption

C ≡ D is the same as **C ⊑ D** and **D ⊑ C**

DL vs. Boolean Algebra

operators ¬, ⊓, ⊔, ⊤, ⊥ form a boolean algebra:

¬ ⊤	equals to	⊥	
¬ ¬ C	equals to	C	
¬ (C ⊓ D)	equals to	¬ C ⊔ ¬ D	
¬ (C ⊔ D)	equals to	¬ C ⊓ ¬ D	

Existential quantification

Example:

MOTHER ⊑ ∃hasChild

Intuitively means *"every mother has as child at least an individual"*

reads *"MOTHER is subsumed from the set of individuals that have at least one child"*

Translated to FOL

MOTHER ⊑ ∃hasChild becomes

$[\text{MOTHER} \sqsubseteq \exists \text{hasChild}] = \forall x\, ([\text{MOTHER}]_x \rightarrow [\exists \text{hasChild}]_x) =$
$= \forall x\, (\text{MADRE}(x) \rightarrow \exists y\, \text{hasChild}(x, y))$

Qualified Existential Quantification

Example:

∃hasChild.FEMALE

denotes the set of all individuals that have a female as child.

Translated to FOL:

∃hasChild.FEMALE becomes

$[\exists \text{hasChild.FEMALE}]_x = \exists y (\text{hasChild}(x, y) \land [\text{FEMALE}]_y) =$
$= \exists y (\text{hasChild}(x, y) \land \text{FEMALE}(y))$

Universal quantification

Example:

∀hasChild.FEMALE

reads *"the set of all individuals that have only female children"*

Translated to FOL:

∀hasChild.FEMALE becomes

$[\forall \text{hasChild.FEMALE}]_x = \forall y(\text{hasChild}(x, y) \rightarrow [\text{FEMALE}]_y) =$
$= \forall y(\text{hasChild}(x, y) \rightarrow \text{FEMALE}(y))$

Inverse role

We define "*sonOf*" starting from "*hasChild*" using notation "*hasChild –*"

sonOf ≡ hasChild-

Example

∃ hasChild-.FEMALE becomes
$[\exists \text{hasChild-.FEMALE}]_x = \exists y(\text{hasChild}(y, x) \wedge [\text{FEMALE}]_y)$
∀ hasChild-.FEMALE becomes
$[\forall \text{hasChild-.FEMALE}]_x = \forall y(\text{hasChild}(y, x) \rightarrow [\text{FEMALE}]_y)$

In general

R express the relation $R(x, y)$

R- express relation $R(y, x)$

∃ R-.C becomes $[\exists R\text{-}.C]_x = \exists y(R(y, x) \wedge [C]_y)$

∀ R-.C becomes $[\forall R\text{-}.C]_x = \forall y(R(y, x) \rightarrow [C]_y)$

Domain and range (1)

- Roles, in general, have **sense** only for certain subsets of the universe.

Example

hasChild relates in general two people,
while it does not have sense for other inanimate objects.

- We can associate to a role R two sets, called ***domain*** and ***range*** of the role, that represents the sets of individuals used for the variables x and y in expression $R(x,y)$

Domain and range (2)

Definition of domain D and range C of role R

$\top \sqsubseteq \forall R^{-}.D$ *(domain D)* - in FOL $\forall x \forall y\, (R(y, x) \rightarrow [D]_y)$

$\top \sqsubseteq \forall R.C$ *(range C)* - in FOL $\forall x \forall y\, (R(x, y) \rightarrow [C]_y)$

Example

$\top \sqsubseteq \forall$ ownerOf^{-}.PERSON

*"**domain** of ownerOf is the set of Person*

$\top \sqsubseteq \forall$ ownerOf.GOOD

*"**range** of ownerOf is the set of Goods*

In short

R : D → C

ownerOf: PERSON → GOOD

Cardinality constraints

It is possible to express cardinality constraints on roles:

≤nR ≥nR

or qualified cardinality constraints:

≤nR.C ≥nR.C

where n is an integer >= 0

Example:

PARENT3F ≡ ≥3hasChild.FEMALE

"PARENT3F is the set of individuals that are parents of at least 3 daughters"

Cardinality constraints (2)

Translated to FOL

≤nR becomes

$[\leq nR]_x = \exists_{\leq n} y\, R(x, y)$

≥nR becomes

$[\geq nR]_x = \exists_{\geq n} y\, R(x, y)$

≤nR.C becomes

$[\leq nR.C]_x = \exists_{\leq n} y\, (R(x, y) \wedge [C]_y)$

≥nR.C becomes

$[\geq nR.C]_x = \exists_{\geq n} y\, (R(x, y) \wedge [C]_y)$

Cardinality constraints(3)

The last example:

$PARENT3F \equiv \geq 3 hasChild.FEMALE$ becomes:

$\forall x\ (PARENT3F(x) \longleftrightarrow \exists y1, \exists y2, \exists y3,$
$(\ hasChild(x,y1) \wedge FEMALE(y1)$
$\wedge hasChild(x,y2) \wedge FEMALE(y2)$
$\wedge hasChild(x,y3) \wedge FEMALE(y3)$
$\wedge y1 \neq y2$
$\wedge y1 \neq y3$
$\wedge y2 \neq y3\ ).$

Exact cardinality

Definitions:

$=nR \triangleq\ \leq nR\ \sqcap\ \geq nR$

$=nR.C \triangleq\ \leq nR.C\ \sqcap\ \geq nR.C$

Observations:

$\geq 1R.C$	same as	$\exists R.C$
$\leq 0R.C$	same as	$\neg\exists R.C$
$\geq 0R.C$	same as	$\top$

Functional roles (1)

A **functional role** is a binary relation where every element in the domain is in **relation with at most an element** in the range

Example:

wifeOf : WOMAN $\rightarrow$ MAN

domain and range:

$\top \sqsubseteq \forall$ wifeOf.MAN

$\top \sqsubseteq \forall$ wifeOf$^-$.WOMAN

role wifeOf is **functional**
because every wife can have at most a husband

WOMAN $\sqsubseteq \leq 1$wifeOf

Role subsumption

In many DL it is possible to express subsumption and equivalence between **roles** with expressions like::

R $\sqsubseteq$ S becomes in FOL $\forall x \forall y (R(x, y) \rightarrow S(x, y))$

R $\equiv$ S becomes in FOL $\forall x \forall y (R(x, y) \leftrightarrow S(x, y))$

Example

parentOf $\sqsubseteq$ relativeOf

parent is a kind of relationship

sonOf $\equiv$ parentOf$^-$

sonOf is the inverse of *parentOf*

Simmetric property

R $\sqsubseteq$ R$^-$ becomes in FOL $\forall x \forall y (R(x, y) \rightarrow R(y, x))$

siblingOf $\sqsubseteq$ siblingOf$^-$

Roles composition

in some ***DL*** it is possible to build complex roles

using composition operator $\circ$

Given two roles R and S:

$R \circ S$ becomesin FOL $\exists z\,(R(x, z) \wedge S(z, y))$

Role composition is useful but often it is not admmitted for problems with decidibility.

.

Transitive property

Definition

$(R \circ R) \sqsubseteq R$ becomes in FOL

$\forall x\, \forall y\, (\exists z\, (R(x, z) \wedge R(z, y)) \rightarrow R(x, y))$

Many ***DL*** (like $\mathcal{SHOIN}$OWL) do not allow role composition but provide an operator to declare a role as transitive.

$Tr(R)$

Equivalence

$\forall(R \circ S).C$	same as	$\forall(R.(\forall S.C)$
$\exists(R \circ S).C$	same as	$\exists(R.(\exists S.C)$

Example

TBox

Faculty $\sqsubseteq$ *Organization*
FullProfessor $\sqsubseteq$ *Teacher*
FullProfessor $\sqsubseteq \exists$*hasSDS.ScientificDisciplinarySector*
Course $\sqsubseteq \exists$*courseTakenAtTime.Interval*
Person $\sqsubseteq \exists$*hasCompetence.ConceptSkill*
Course $\sqsubseteq \exists$*subject.ConceptSkill*

ABox

Paolo Rossi: FullProfessor,
Knowledge Management and Protection Systems : Course,
Distributed Systems: ConceptSkill,
Cloud Computing: ConceptSkill,
(Paolo Rossi, Knowledge Management and Protection Systems):courseTaken,
(Paolo Rossi, Cloud Computing):hasCompetence,
(Knowledge Management and Protection Systems, Distributed Systems):subject

OWL (1)

http://www.w3.org/TR/owl-ref/

- **OWL** (Web Ontology Language) is the standard proposed by **W3C** for the definitions of ontologies
  - OWL Lite: $\mathcal{SHIF}(\mathcal{D})$;
  - OWL DL: corresponds to $\mathcal{SHION}(\mathcal{D})$;
  - OWL 2 DL: corresponds to $\mathcal{SROIQ}(\mathcal{D})$ and is the “normal” OWL 2 (sublanguage): “maximum” expressivity while keeping reasoning problems decidable—but still very expensive;
  - (Other) profiles are tailored for specific ends, e.g.,
    - OWL 2 QL: is specifically designed for efficient database integration;
    - OWL 2 EL: is a lightweight language with polynomial time reasoning;
    - OWL 2 RL: is designed for compatibility with rule-based inference tools.

OWL (2)

- An **OWL** ontology is made of a **TBox** and an **ABox** **both represented as RDF graphs** *(set of **triples**)*
- *Some **RDFS** constructs are adopted from **OWL***
- **OWL** introduces many constructs not present in RDFS, represented as RDF **triples**

Terminology

In **OWL**

- **Terms** or **concepts** are ***classes***
- The **Operators** to define terms are called **class constructors**
- **Roles** are called ***properties***
- The definitions in the *TBox* are called **class axioms**
- ABox assetions are ***facts***

Identifier

- All **class descriptions** describe a **resource** of type:
  owl:Class
- In the simplest case the description is made of an class identifier *(Unique Resource Identifier = URI)*, corresponding to an atomic term of the DL
- ***RDF Syntax:***

```
<owl:Class  rdf:ID="ClassName"/>
<owl:Class  rdf:about="ClassName"/>
```

Two class identifiers are predefined in OWL

owl:Thing = the set of all individuals ($\top$ for *universal class*)

All OWL classes are subclass of owl:Thing

owl:Nothing = empty set ($\bot$ for empty class)

The owl:Nothing class is a subclass of all classes

Enumeration

- A class can be described from enumeration of a finite set of nominals $a_1, \ldots, a_n$ using operator **owl:oneof**
  - ***DL syntax:***
    $\{a_1, \ldots, a_n\}$
  - ***RDF syntax:***

```
<owl:oneOf rdf:parseType="Collection">
  <owl:Thing rdf:about="#a1" />
  ...
  <owl:Thing rdf:about="#an" />
</owl:oneOf>
```

Intersection

- A class **A** can be described as the *intersection* of a finite number of classes C_1, …, C_n using operator **owl:intersectionOf**
- ***DL syntax:***

  $A \equiv C_1 \sqcap \ldots \sqcap C_n$
- ***RDF syntax:***

```
<owl:Class rdf:ID="A">
  <owl:intersectionOf rdf:parseType="Collection">
   <owl:Class rdf:about="#C1" />
   ...
   <owl:Class rdf:about="#Cn" />
  </owl:intersectionOf>
</owl:Class>
```

Union

- A class **A** can be described as *union* of a finite number of classes C_1, …, C_n using operator **owl:unionOf**
- ***DL syntax:***

  $A \equiv C_1 \sqcup \ldots \sqcup C_n$
- ***RDF syntax:***

```
<owl:Class rdf:ID="A">
  <owl:unionOf rdf:parseType="Collection">
   <owl:Class rdf:about="#C1" />
   ...
   <owl:Class rdf:about="#Cn" />
  </owl:unionOf>
</owl:Class>
```

Complement

- A class **A** can be described as *complement* of class **B** using operator **owl:complementOf**
- ***DL syntax:***

  $A \equiv \neg B$
- ***RDF syntax:***

```
<owl:Class rdf:ID="A">
  <owl:complementOf>
    <owl:Class rdf:about="#B" />
  </owl:complementOf>
</owl:Class>
```

Class axioms: *Subclasses*

- Between two class descriptions **C** e **D** can be defined the subclass relation using operator **rdfs:subClassOf**

- ***DL syntax:***

  $C \sqsubseteq D$
- ***RDF syntax:***

```
<owl:Class  rdf:about="#C">
  <rdfs:subClassOf  rdf:resource="#D"  />
</owl:Class>
```

Class axioms: *Equivalence*

- Between two call descriptions **C** and **D** can be defined the equality relation using operator **owl:equivalentClass**
- ***DL syntax:***

  $C \equiv D$
- ***RDF syntax:***

```
<owl:Class rdf:about="#C">
  <owl:equivalentClass rdf:resource="#D" />
</owl:Class>
```

Property restrictions (1)

- A class can be described as a restriction on a property, the restriction can be the existential qualified role using operator **owl:someValuesFrom**
- ***DL syntax:***

  $\exists R.C$
- ***RDF syntax:***

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:someValuesFrom rdf:resource="#C" />
</owl:Restriction>
```

Property restriction (2)

- A class can be described as a restriction on a property, the restriction can be the universal qualified role using operator **owl:allValuesFrom**
- *DL syntax:*

  $\forall R.C$
- *RDF syntax:*

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:allValuesFrom rdf:resource="#C" />
</owl:Restriction>
```

Property restrictions(3)

Examples (W3C OWL Guide: http://www.w3.org/TR/owl-ref/)

```
<owl:Restriction>
      <owl:onProperty rdf:resource="#hasParent" />
      <owl:someValuesFrom rdf:resource="#Doctor" />
</owl:Restriction>

 <owl:Restriction>
      <owl:onProperty rdf:resource="#hasParent" />
      <owl:allValuesFrom rdf:resource="#Human" />
 </owl:Restriction>
```

Property restrictions(4)

- A class can be described as a restriction of all individuals that associate throught a role R to a single individual a using operator **owl:hasValue**
- *DL syntax*
  $\forall R.\{a\}$
- *RDF syntax*

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:hasValue rdf:resource="#a" />
</owl:Restriction>
```

Property restriction (5)

Example (W3C OWL Guide: http://www.w3.org/TR/2004/REC-owl-guide-20040210/)

```
<owl:Class rdf:ID="Burgundy">
  ...
  <rdfs:subClassOf rdf:resource="Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTaste" />
      <owl:hasValue rdf:resource="#Dry" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

Property restrictions (6)

- A class can be described as a restriction on a set of individuals with role R with a maximum number n of associated individuals using operator **owl:maxCardinality**
- ***DL syntax:***
  $\leq nR$
- ***RDF syntax:***

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:maxCardinality rdf:datatype=
    "&xsd;nonNegativeInteger">n</owl:maxCardinality>
</owl:Restriction>
```

Property restriction (7)

- A class can be described as a restriction on a set of individuals with role R with at least n associated individuals, using operator **owl:minCardinality**
- ***DL syntax:***
  $\geq nR$
- ***RDF syntax:***

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:minCardinality rdf:datatype=
    "&xsd;nonNegativeInteger">n</owl:minCardinality>
</owl:Restriction>
```

Property restriction (8)

- A class can be described as a restriction on a set of individuals with role R associated exactly to n individuals, using operator **owl:cardinality**
  - *DL syntax:*
    =nR
  - *RDF syntax:*

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:cardinality rdf:datatype=
    "&xsd;nonNegativeInteger">n</owl:cardinality>
</owl:Restriction>
```

Property restriction (9)

Examples (W3C OWL Guide: http://www.w3.org/TR/owl-ref/)

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">2</owl:maxCardinality>
</owl:Restriction>

<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">2</owl:minCardinality>
</owl:Restriction>

<owl:Restriction>
  <owl:onProperty rdf:resource="#hasIDFiscalCode" />
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction>
```

Class axioms: *Disjunction*

- Between two classes **C** and **D** can be declared a disjoint relation using operator **owl:disjointWith**

- ***DL syntax:***
  $C \sqcap D \equiv \perp$
- ***RDF syntax:***

```
<owl:Class rdf:about="#C">
  <owl:disjointWith rdf:resource="#D" />
</owl:Class>
```

Properties

- Coherently with *RDFS*, in **OWL** also properties *(as roles in DL)* can be seen as ***particular classes***

- Can have sub properties and can be combined using some constructors

- As every class is an **owl:Class** resource, all properties are resources of type **rdf:Property**

- In OWL properties can be resources of three types
  - **owl:ObjectProperty** (property between two objects)
  - **owl:DatatypeProperty** (property between an object and a datatype)
  - **owl:AnnotationProperty** (property used to associate metadata)

Subproperties

A property **R** can be defined as a subproperty of property **S** using operator **rdfs:subPropertyOf**

- ***DL syntax:***

  $R \sqsubseteq S$

- ***RDF syntax:***

```
<owl:ObjectProperty rdf:ID="R">
  <rdfs:subPropertyOf rdf:resource="#S" />
</owl:ObjectProperty>
```

Domain

- Of property **R** can be specified the domain **D** using operator **rdfs:domain**
- ***DL syntax:***

  $\top \sqsubseteq \forall R^{-}.D$

- ***RDF syntax:***

```
<owl:ObjectProperty rdf:ID="R">
  <rdfs:domain>
    <owl:Class rdf:about="#D" />
  </rdfs:domain>
</owl:ObjectProperty>
```

Range

- Of property **R** can be specified the range **C** using operator **rdfs:range**
- *DL syntax:*

  $\top \sqsubseteq \forall R.C$

- *RDF syntax:*

```
<owl:ObjectProperty rdf:ID="R">
  <rdfs:range>
   <owl:Class rdf:about="#C" />
  </rdfs:range>
</owl:ObjectProperty>
```

Equivalent property

- A property **R** can be defined as equivalent to another property **S** using operator **owl:equivalentProperty**
- *DL syntax:*

  $R \equiv S$

- *RDF syntax:*

```
<owl:ObjectProperty rdf:ID="R">
  <owl:equivalentProperty rdf:resource="#S" />
</owl:ObjectProperty>
```

Inverse property

- Given property **R** the inverse property **S** can be defined using operator **owl:inverseOf**
- *DL syntax:*

  $S \equiv R^-$
- *RDF syntax:*

```
<owl:ObjectProperty rdf:ID="S">
  <owl:inverseOf rdf:resource="#R" />
</owl:ObjectProperty>
```

Functional property (1)

- A property **R** is functional if it associate at max 1 individual for each element in the domain, expressed from operator **owl:FunctionalProperty**
- *DL syntax:*

  $\top \sqsubseteq \leq 1 R$
- *RDF syntax:*

```
<owl:FunctionalProperty rdf:about="#R" />
…
<owl:ObjectProperty rdf:ID="R">
  <rdfs:domain rdf:resource="#D" />
  <rdfs:range rdf:resource="#C" />
</owl:ObjectProperty>
```

Functional property (2)

Example (W3C OWL Guide: http://www.w3.org/TR/2004/REC-owl-guide-20040210/):

```
<owl:ObjectProperty rdf:ID="hasHusband">
  <rdf:type    rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Woman" />
  <rdfs:range  rdf:resource="#Man" />
</owl:ObjectProperty>
```

Inverse functional property

- A property **R** is inverse functional if the inverse property is functional, thus each element in the range is associated from max 1 individual in the domain, expressed from operator **owl:InverseFunctionalProperty**
- ***DL Syntax:***
  $\top \sqsubseteq \leq 1 R^{-}$
- ***RDF Syntax:***

```
<owl:InverseFunctionalProperty rdf:ID="R">
  <rdfs:domain rdf:resource="#D" />
  <rdfs:range rdf:resource="#C" />
</owl:InverseFunctionalProperty>
```

Transitive property (1)

- In OWL it is possible to declare that a property is transitive using operator **owl:TransitiveProperty**
- ***DL syntax:***

  Tr(R)

- ***RDF syntax:***

```
<owl:TransitiveProperty rdf:ID="R">
  <rdfs:domain rdf:resource="#D" />
  <rdfs:range rdf:resource="#D" />
</owl:TransitiveProperty>
```

Transitive property (2)

Example (W3C OWL Guide: http://www.w3.org/TR/2004/REC-owl-guide-20040210/):

```
<owl:ObjectProperty rdf:ID="subRegionOf">
  <rdf:type rdf:resource="&owl;TransitiveProperty"/>
  <rdfs:domain rdf:resource="#Region"/>
  <rdfs:range  rdf:resource="#Region"/>
</owl:ObjectProperty>
```

Symmetric property (1)

- In OWL it is possible to declare a symmetric property **owl:SymmetricProperty**
- ***DL syntax:***

  $R \sqsubseteq R^{-}$
- ***RDF syntax:***

```
<owl:SymmetricProperty rdf:ID="R">
  <rdfs:domain rdf:resource="#D" />
  <rdfs:range rdf:resource="#D" />
</owl:SymmetricProperty>
```

Symmetric property (2)

Example (W3C OWL Guide: http://www.w3.org/TR/2004/REC-owl-guide-20040210/):

```
<owl:SymmetricProperty rdf:ID="friendOf">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range  rdf:resource="#Human"/>
</owl:SymmetricProperty>
```

Example

```
<owl:ObjectProperty rdf:about="&UniFI#isCoordinatorOf">
        <rdfs:subPropertyOf rdf:resource="&UniFI#isWorkingFor"/>
        <rdfs:domain rdf:resource="&UniFI#Coordinator"/>
        <owl:inverseOf rdf:resource="&UniFI#hasCoordinator"/>
        <rdfs:range>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="&UniFI#Center"/>
                    <rdf:Description rdf:about="&UniFI#ResearchProject"/>
                </owl:unionOf>
            </owl:Class>
        </rdfs:range>
</owl:ObjectProperty>
```

Class member

- In OWL it is possible to declare that an individual **a** belong to class **C**
- ***DL syntax:***

  `C(a)`
- ***RDF syntax:***

```
<C rdf:ID="a">              <C rdf:about="a">
  ...                         ...
</C>                        </C>
```

Property instances

- In OWL it is possible to specify that property **R** of an individual **a** has value **b**
- ***DL syntax:***

```
R(a,b)
```

- ***RDF syntax:***

```
<C rdf:ID="a">
  <R rdf:resource="#b" />
  ...
</C>
```

Unique names?

- Two distinct names are necessary referring to two different objects?
- Depends... We have two possibilities:
  - We use **Unique Name Assumption** (UNA), every object has a unique name
  - or **Not Unique Name Assumption** (NUNA), the same object can be identified using two or more different names
- OWL and semantic web adopt NUNA, this implies that it should be needed a way to express that two objects are the same or that are different

Object identity (1)

- **OWL** allows to assert that two names refers to the same individual using operator `owl:sameAs`
- *DL syntax:*
  a = b
- *RDF syntax:*

```
<rdf:Description rdf:about="#a">
  <owl:sameAs rdf:resource="#b" />
</rdf:Description>
```

Object identity (2)

- It is also possible to assert that two names refer to different individuals using operators `owl:differentFrom`
- *RDF syntax:*

```
<C rdf:ID="a">
  <owl:differentFrom rdf:resource="#b"/>
  …
</C>
```

Object identity (3)

- It is also possible to express that **n** names refer to all different individuals using operator **owl:AllDifferent**
- ***RDF syntax:***

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
   <C rdf:about="#a1"/>
   ...
   <C rdf:about="#an"/>
  </owl:distinctMembers>
</owl:AllDifferent>
```

Open or closed world?

- A term that is not explicitly asserted and that cannot be derived as true, is it false?
  - **Closed World Assumption (CWA)** → yes, is false
  - **Open World Assumption (OWA)** → we don't know it could be true or false
- **OWL and semantic web adopt the OWA that make it difficult to check for concistency:**
  - **TBox:** $A \equiv =2R$
  - **ABox:** $A(a1)$, $R(a1,x1)$

OWL2 (1)

- OWL2 introduces some new features:
  - **DisjointUnion** of classes
  - **DisjointClasses** (all classes are pair-wise disjoint)
  - Negative object and data properties assertions
  - Self restriction on a property
    - `∃R.Self` = {x : R(x,x) }
    - `Narcisist ≡ Person ⊓ ∃loves.Self`
  - Reflexive and irreflexive properties
    - `⊤⊑ ∃R.Self`
    - `⊤⊑ ¬ ∃R.Self`
  - Disjoint properties
    - `Disjoint(R1,R2,...)`
  - Asymmetric properties
    - `Disjoint(R,R-)`

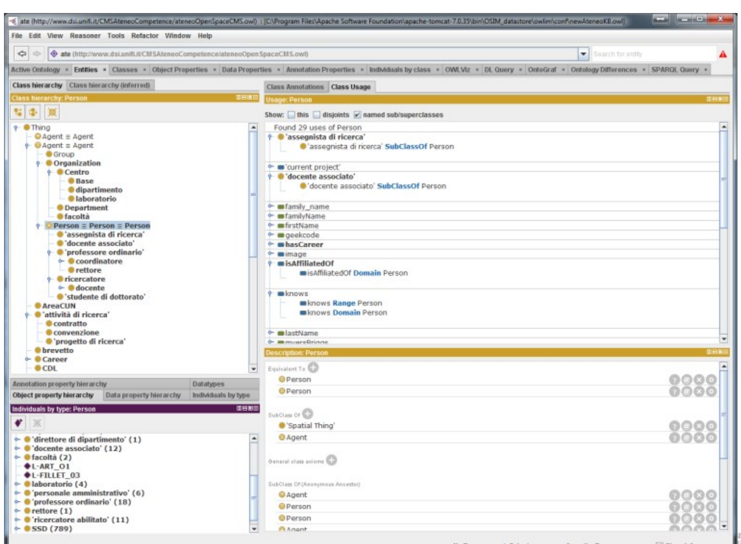
OWL2 (2)

- OWL2 introduces some new features:
  - Property chains inclusion
    - role composition can be used to define a new property
    - `R1 o R2 o ... o Rn ⊑ CR`
    - Example:
    - `parentOf o parentOf ⊑ grandParentOf`
  - Qualified cardinality restrictions
    - `≤nR.C`
    - `=nR.C`
    - `≥nR.C`
  - Keys of a class
    - A set of properties uniquely identifies an individual of a class

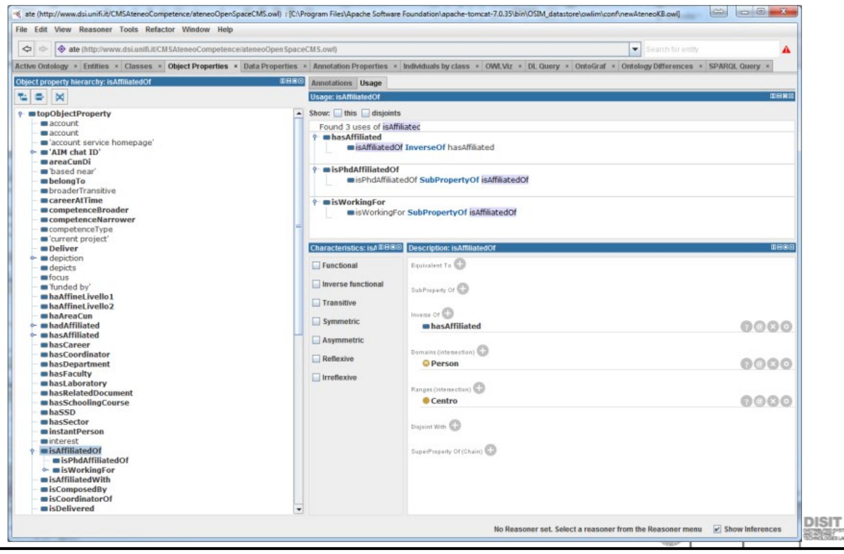
Protégé

- **Protégé** (http://protege.stanford.edu/), is an open-source editor for KBs and Ontologies (RDF, OWL, NT…) developed by Stanford University
- Allows the visualization, creation, editing of:
  - Entities
  - Classes
  - Properties (*Object Property* & *Data Property*)
  - Instances (*Individuals*).
- SPARQL Query Interface to query the KB.
- Graphic visualizations of classes, properties and instances.

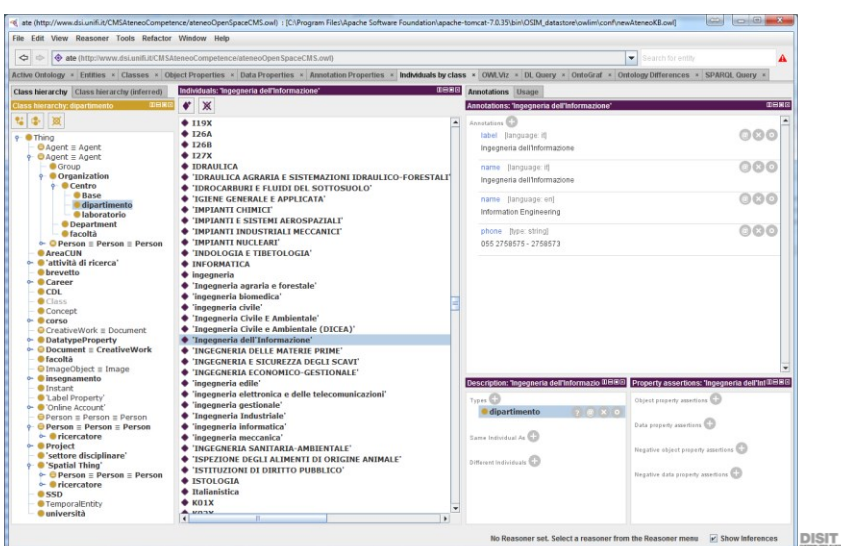
Protégé – *Classes*



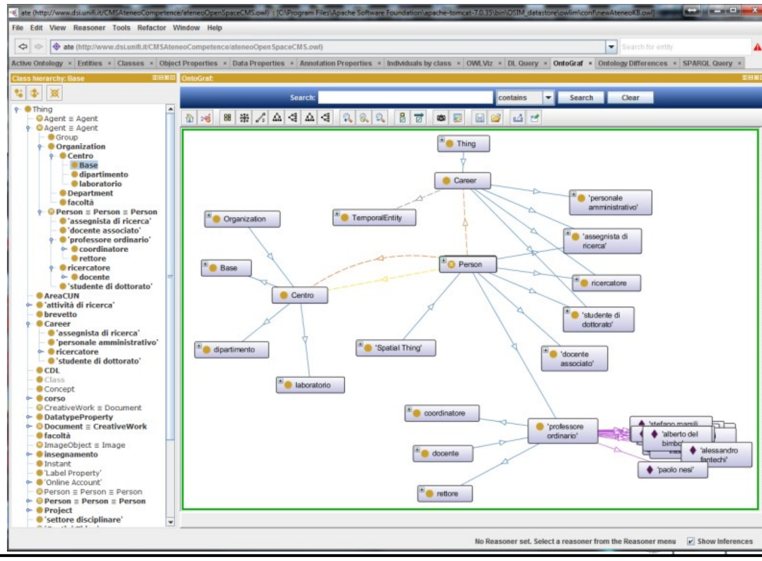
Protégé – *Object Properties*



Protégé – *Instances*



Protégé – *OntoGraph*



Manchester syntax

- Used in Protégé to express restrictions on a class
  - C **and** D — $C \sqcap D$
  - C **or** D — $C \sqcup D$
  - **not** C — $\neg C$
  - p **some** C — $\exists p.C$
  - p **only** C — $\forall p.C$
  - p **exactly** n C — $=n.C$
  - p **min** n C — $\geq np.C$
  - p **max** n C — $\leq np.C$
  - p **value** v — $\forall p.\{v\}$
  - { v1, v2, … vn }
  - **inverse** p — p^-

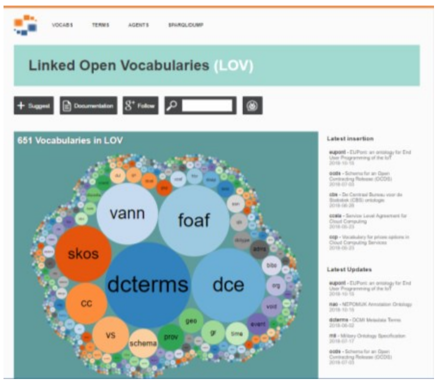
Examples

- PERSON **and** (parentOf **some** PERSON)
- Human **and not** Female **and** (married **some** Doctor) **and** (hasChild **only** (Doctor **or** Professor))
- Person **and** (sonOf **exactly** 2 Person)

LOV

- Linked Open Vocabulary
- Vocabulary descriptions should be available via Linked Data
- If you open the URL of a property or class with a browser it forwards to the HTML documentation of the vocabulary
- If you request (Accept) rdfxml or turtle formats it provides a machine readable description

lov.linkeddata.es



- a repository where are available links to vocabularies (ontologies) available as linked open data.
- contains a RDF description of vocabularies

SKOS

- Used for Taxonomy representation
- Taxonomies are used for classification, are hierarchies of concepts
- SKOS = Simple Knowledge Organization System
- Describes a set of **skos:Concepts**
- Related via properties:
  - broader (e.g. <Ontology> *skos:broader* <KnowledgeRepresentation>),
    - read "has broader"
  - narrower (e.g. <Animal> *skos:narrower* <Mammifer>),
    - read "has narrower"
  - related (e.g. <Human> *skos:related* <Philosophy>),
  - …

SKOS

- *broader* & *narrower* build hierarchies, *related* used to associate concepts from different hierarchies
- broader & narrower are not transitive
  - *broader* ⊑ *broaderTransitive* ⊑ *semanticRelation*
  - *narrower* ⊑ *narrowerTransitive* ⊑ *semanticRelation*
  - *related* ⊑ *semanticRelation*
  - x *brd* y, y *brd* z ➔ x *brdTr* y, y *brdTr* z ➔ x *brdTr* z
  - *broaderTransitive* disjoint with *related*
  - *related* is symmetric
- *semanticRelation*: Concept→Concept
- *narrower* inverse of *broader*
- *narrowerTransitive* inverse of *broaderTransitive*

SKOS

- Can represent different ConceptSchemes
  - cs rdf:type **ConceptScheme**
  - c1 **skos:inScheme** cs
  - cs **skos:hasTopConcept** cx
  - cx **skos:topConceptOf** cs
  - **skos:topConceptOf** ⊑ **skos:inScheme**
  - **skos:topConceptOf** inverse of **skos:hasTopConcept**

SKOS

- concepts from different concept schemes can be associated using specific properties:
  - *exactMatch*
  - *closeMatch*
  - *broadMatch*
  - *narrowMatch*
  - *relatedMatch*

SKOS

- properties:
  - *exactMatch* ⊑ *closeMatch* ⊑ *mappingRelation*
  - *mappingRelation* ⊑ *semanticRelation*
  - *broadMatch* ⊑ *mappingRelation*
  - *narrowMatch* ⊑ *mappingRelation*
  - *relatedMatch* ⊑ *mappingRelation*
  - *broadMatch* ⊑ *broader*
  - *narrowMatch* ⊑ *narrower*
  - *relatedMatch* ⊑ *related*
  - *exactMatch*, *closeMatch*, *relatedMatch* are symmetric
  - *exactMatch* is transitive
  - *narrowMatch* inverse of *broadMatch*

SKOS labels

- SKOS introduces some annotationProperties for labelling resources
  - skos:prefLabel – one for each language
  - skos:altLabel – other alternative labels
  - skos:hiddenLabel – labels used for search (e.g. mispelled names)

SKOS example

- US library of congress publish as linked data the **Library of Congress Subject Headings**
  - http://id.loc.gov/authorities/subjects.html
  - Example Concept "Security measures"
    - http://id.loc.gov/authorities/subjects/sh99005297

Dublin Core

- 15 properties used to associate content with the usual bibliographic descriptions
  - title,
  - creator,
  - contributor,
  - subject,
  - language,
  - publisher,
  - identifier,
  - date,
  - description,
  - rights,
  - coverage,
  - format,
  - relation,
  - source,
  - type

Dublin Core

- Two prefixes
  - dc: <http://purl.org/dc/elements/1.1/>
  - dcterms: <http://purl.org/dc/terms/>
- The first is the legacy one
- "dc" properties can be used as both data properties and object properties
- Example:
  - <divinacommedia> dc:creator "Dante Alighieri"
  - <divinacommedia> dc:creator <http://dbpedia.org/resource/DanteAlighieri>

Dublin Core

- "dcterms" properties introduce many sub-properties:
  - abstract ⊑ description
  - license ⊑ rights
  - spatial ⊑ coverage
  - available ⊑ date
  - bibliographicCitation ⊑ identifier
  - conformsTo ⊑ relation
  - created ⊑ date
  - dateAccepted ⊑ date
  - dateCopyrighted ⊑ date
  - dateSubmitted ⊑ date
  - extent ⊑ format
  - hasPart ⊑ relation
  - isPartOf ⊑ relation
  - …

FOAF

- FOAF = Friend Of a Friend
- allows to describe people and their relations

FOAF Core

-
  - Agent
  - Person
  - name
  - title
  - img
  - depiction (depicts)
  - familyName
  - givenName
  - knows
  - based_near
  - age
  - made (maker)
  - primaryTopic (primaryTopicOf)
-
  - Project
  - Organization
  - Group
  - member
-
  - Document
  - Image

Social Web

- nick
- mbox
- homepage
- weblog
- openid
- jabberID
- mbox_sha1sum
- interest
- topic_interest
- topic (page)
- workplaceHomepage
- workInfoHomepage
- schoolHomepage
- publications
- currentProject
- pastProject
- account
- OnlineAccount
- accountName
- accountServiceHomepage
- PersonalProfileDocument
- tipjar
- sha1
- thumbnail
- logo

schema.org

- Supported by search engines (google, yahoo)
- Used to describe content to be indexed
- Many classes and properties

link link link link link link link link

- Given two vocabularies, we can build a mapping between:
  - **classes** using:
    - subClassOf
    - equivalentClass
  - **properties** using:
    - subPropertyOf
    - equivalentProperty
- Aka Ontology Alignment

link link link link link link link link

- Objects can be linked using:
  - **owl:sameAs**
    - to state that two URL refers exacly to the same entity
  - **rdfs:seeAlso**
    - to state that two URL refers to related entities
- Use **sameAs** relations with care, is a transitive, symmetric, reflexive property