An Introduction to Ontology Development

Stefano Bragaglia

A (Really) Brief History of the Semantic Web

"The World-Wide Web (W3) was developed to be a pool of human knowledge, which would allow collaborators in remote sites to share their ideas and all aspects of a common project."

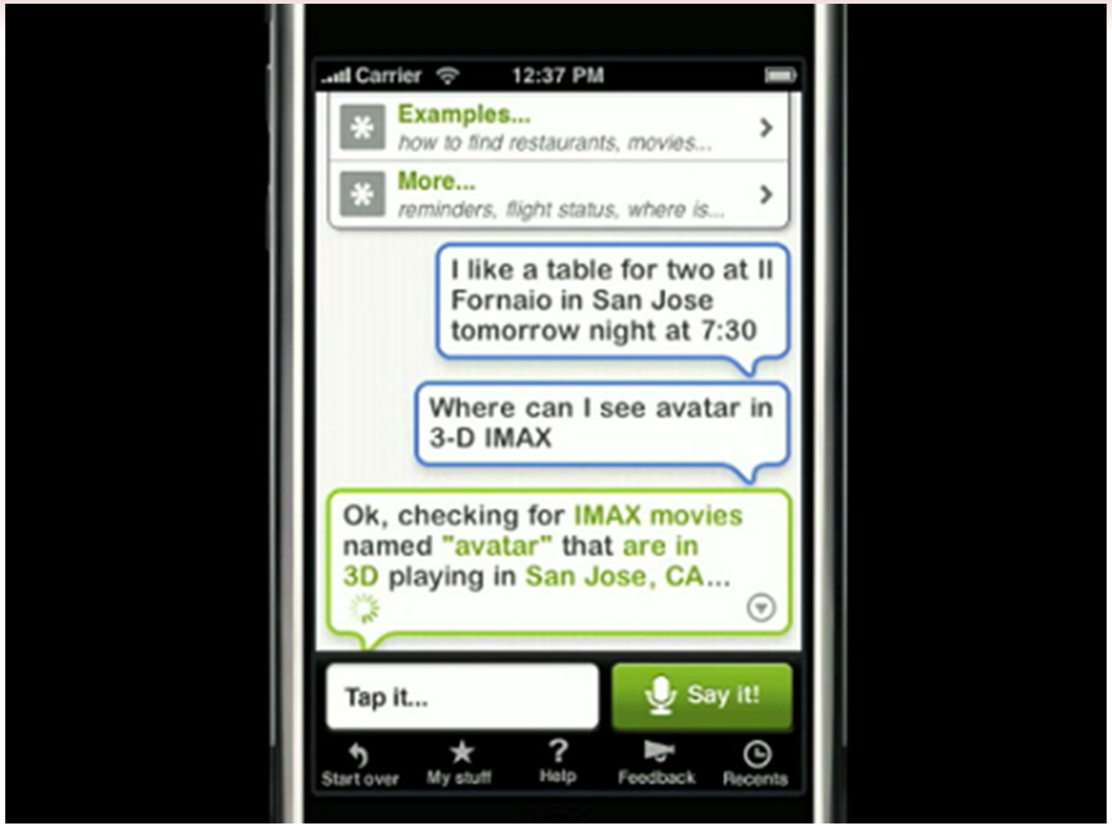
Sir Timothy Berners-Lee

- **Christmas 1990:** with the implementation of the *HyperText Transfer Protocol*, the *HyperText Markup Language*, the *CERN httpd* (an HTTP server) and *WorldWideWeb* (a web editor/browser) the first web page is transmitted.
- **September 1994:** Berners-Lee founds the *World-Wide Web Consortium* to create standards and recommendations to improve quality of the Web.
- **Early 2002:** new ideas for sharing contents arises, the new paradigm aims to democratize the Web by means of *social networks* (*Web 2.0*)
- **Mid 2006:** Berners-Lee introduces the *Semantic Web* to turn the Web from *machine-representable* to *machine-understandable*, thus allowing *automatic reasoning*

A Winning Example of Semantic Web Application

SIRI
a free personal assistant for iPhone

- Awarded during this year's *SXSW Festival* as **"Most Innovative Web Technology"**
- Integrates a growing number of third-party web services
- Understands user's needs, automatically discovers solutions and provides answers



Examples... how to find restaurants, movies...

More... reminders, flight status, where is...

I like a table for two at Il Fornaio in San Jose tomorrow night at 7:30

Where can I see avatar in 3-D IMAX

Ok, checking for IMAX movies named "avatar" that are in 3D playing in San Jose, CA...

A Few Considerations on the Semantic Web

- Automatic reasoning will be possible if the data on Internet will be enriched with semantic annotations

Who should do it? How difficult is it?

- Software that (ascertainably) draws conclusions from semantic annotations is also required (work in progress)

Is it feasible? Will it push Web annotation?

- The first partial results were collected only now, after years of large investments

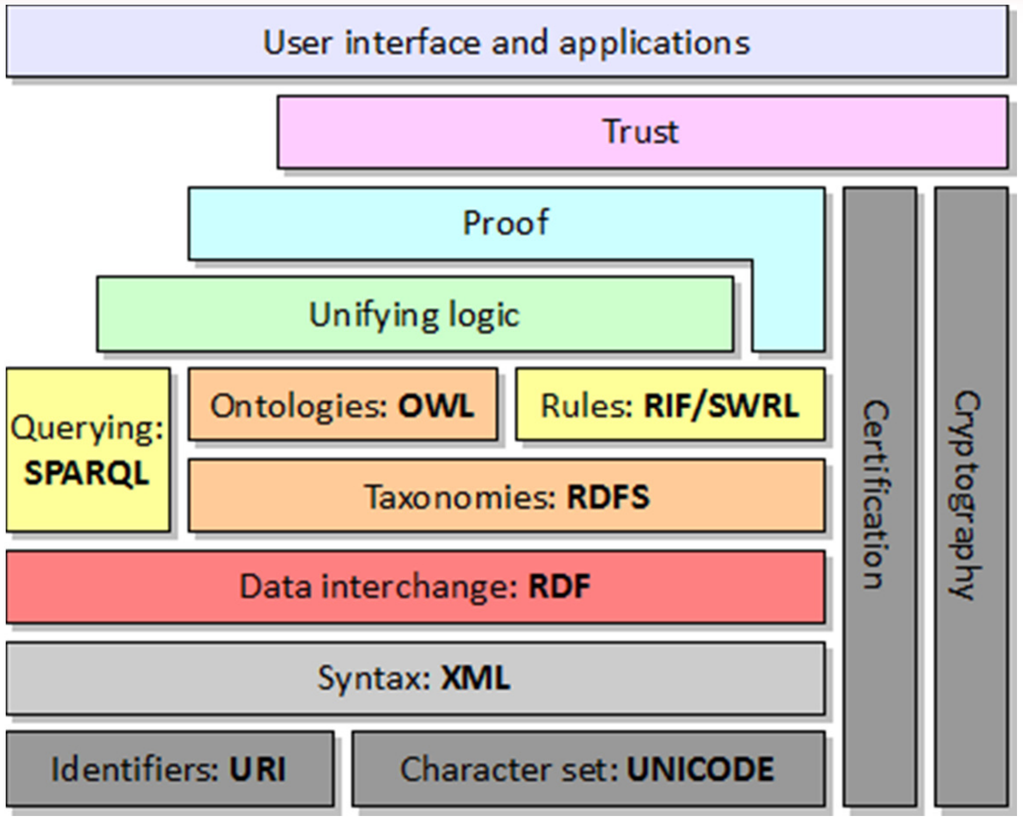
Is the Semantic Web really needed?

- Thus automatic reasoning has been postponed in favor of data interoperability (Data Web vs. Semantic Web), but…

The Semantic Web Cake

Architecture:

The Semantic Web is complex and several layers are required, each addressing specific issues



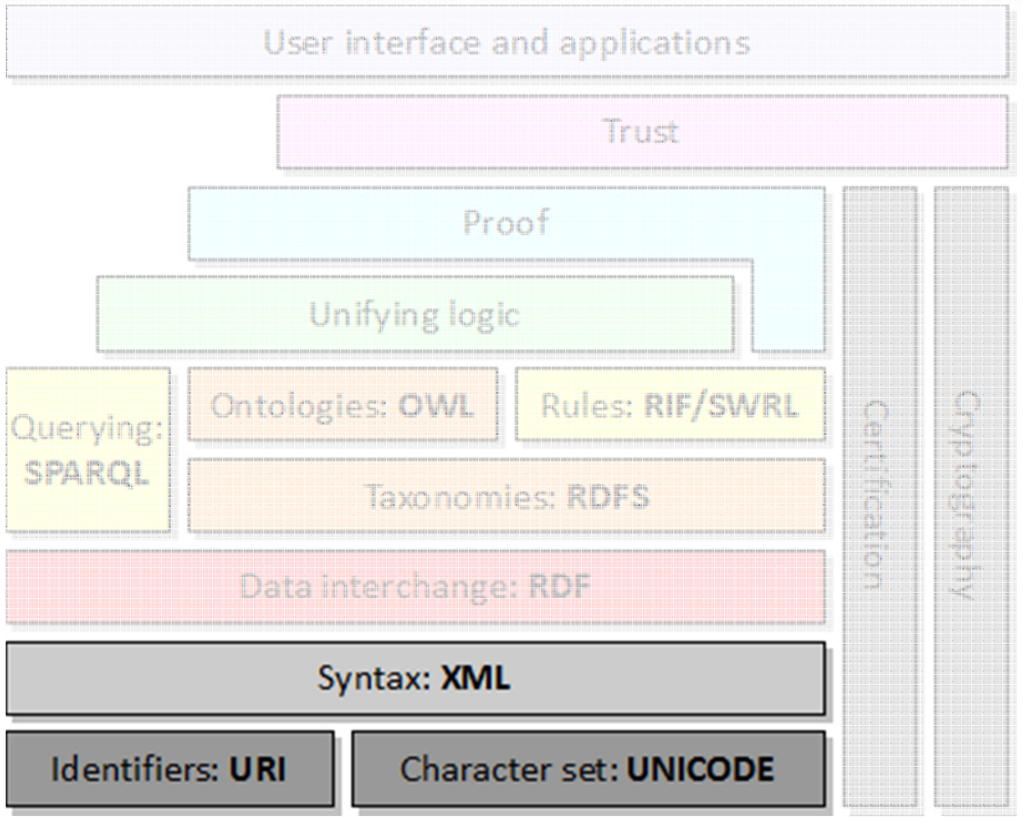
The Semantic Web Cake

Requirements:

URI: a mechanism to refer things (similar to URL)

UNICODE: a universal character set to grant communication

XML: elemental syntax and content structure for documents (semantics delegated to higher levels)

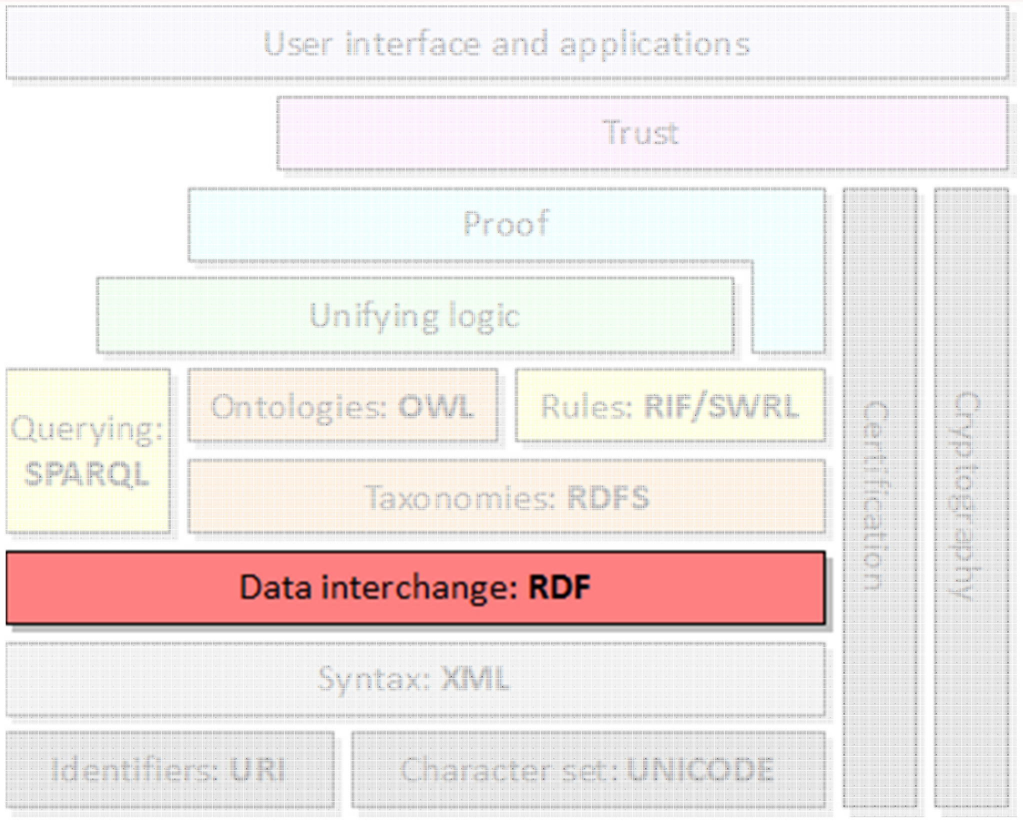


THE SEMANTIC WEB CAKE

Data Interchange:

Resource Description Framework:

- Based on XML
- (Almost) as expressive as *E-R Diagrams* or *Class Diagrams*
- Describes the domain by triples : *< subject, predicate, object >* or *< resource, attribute, value >*
- Defines "roles" like: *subject*, *predicate*, *object*, *type*, *value*, *ecc.*

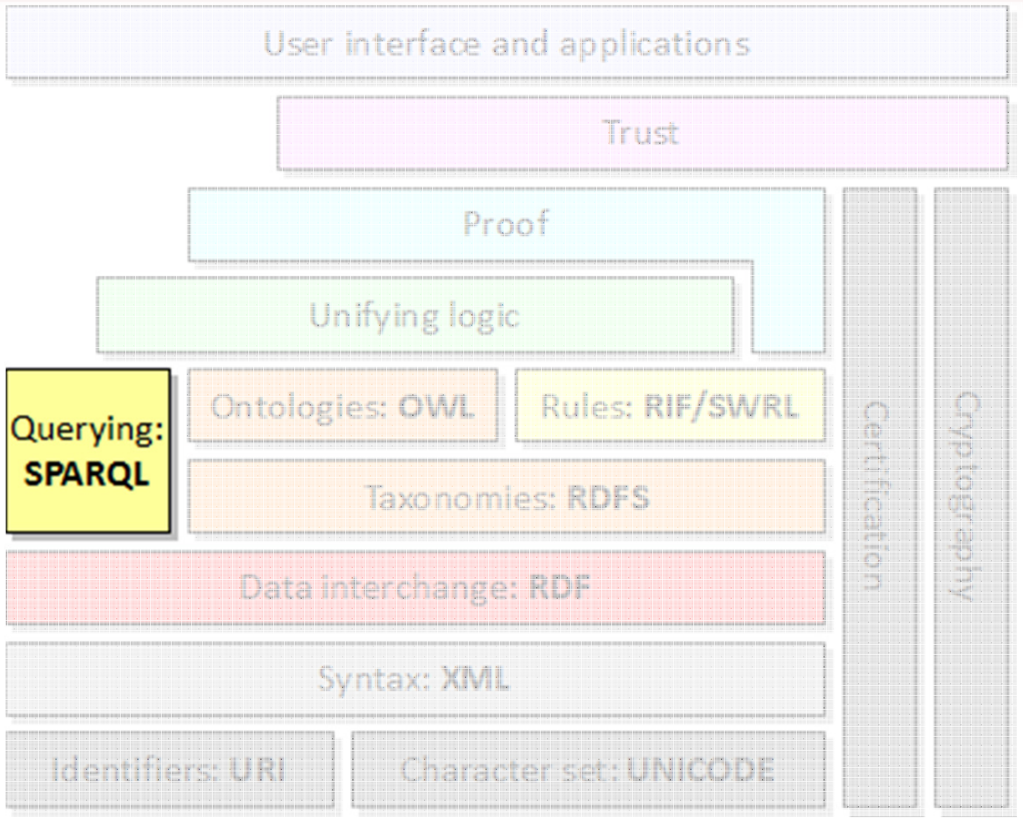


The Semantic Web Cake

Querying:

SPARQL Protocol and RDF Query Language:

- Inherits from both Prolog and SQL
- Interacts with RDF triples



The Semantic Web Cake

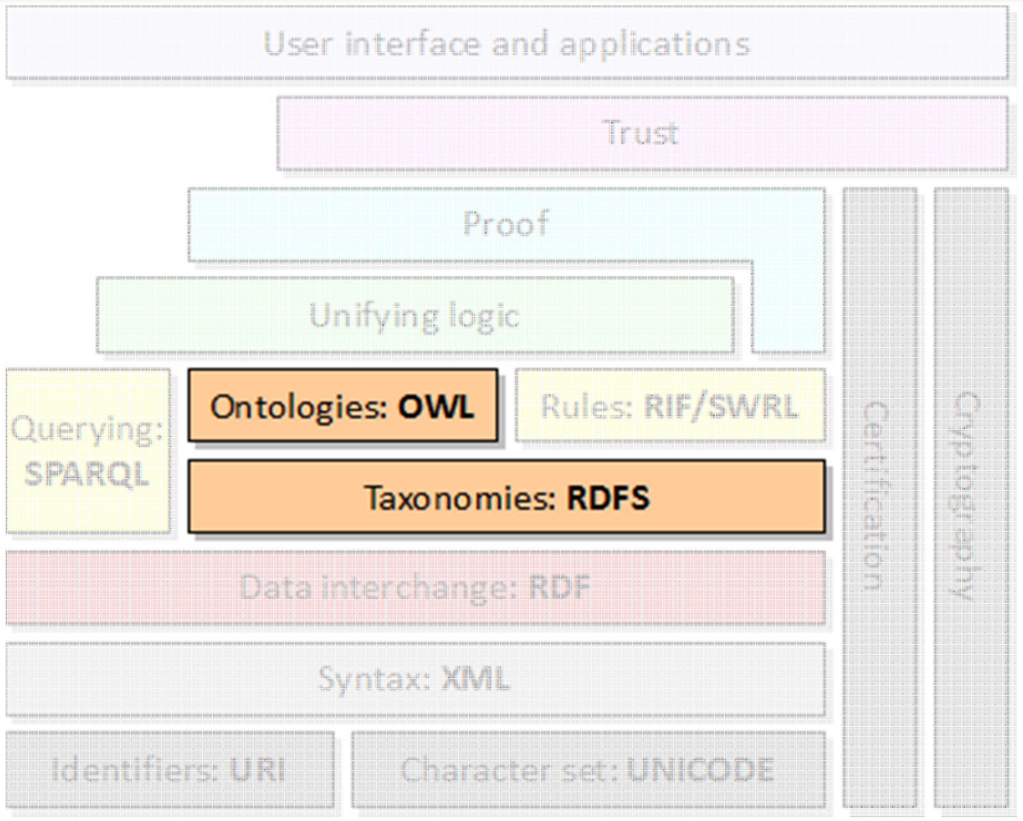
Knowledge Representation:

RDF Schema:

- Extends RDF with *subClassOf*, *Datatype*, etc.
- Good for taxonomies

OWL:

- Extends RDFS with *disjointWith*, *equivalentProperty*, *InverseOf*, ecc.
- Good for ontologies



The Semantic Web Cake

Rules Representation:

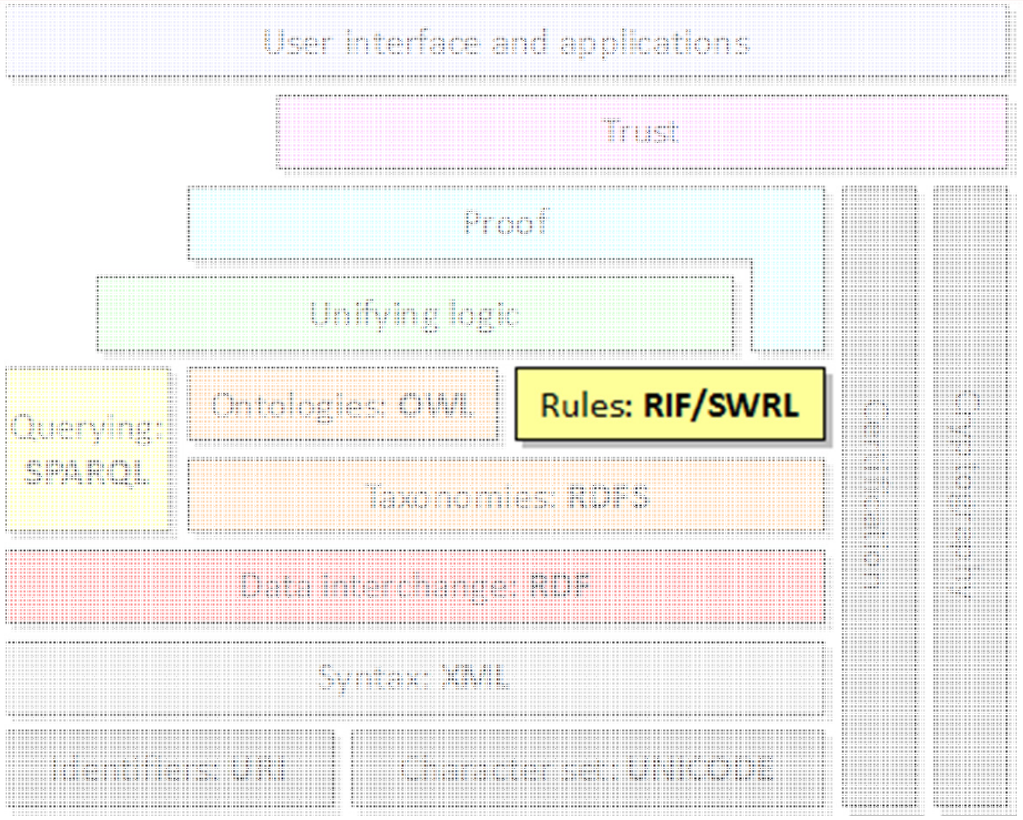
Semantic Web rule language:

- Applies rules to data: *a hasParent b /\ b hasBrother c => a hasUncle c*

Rule Interchange Format:

- Allows portability of rules between languages

NB: Currently undergoing a standardization process



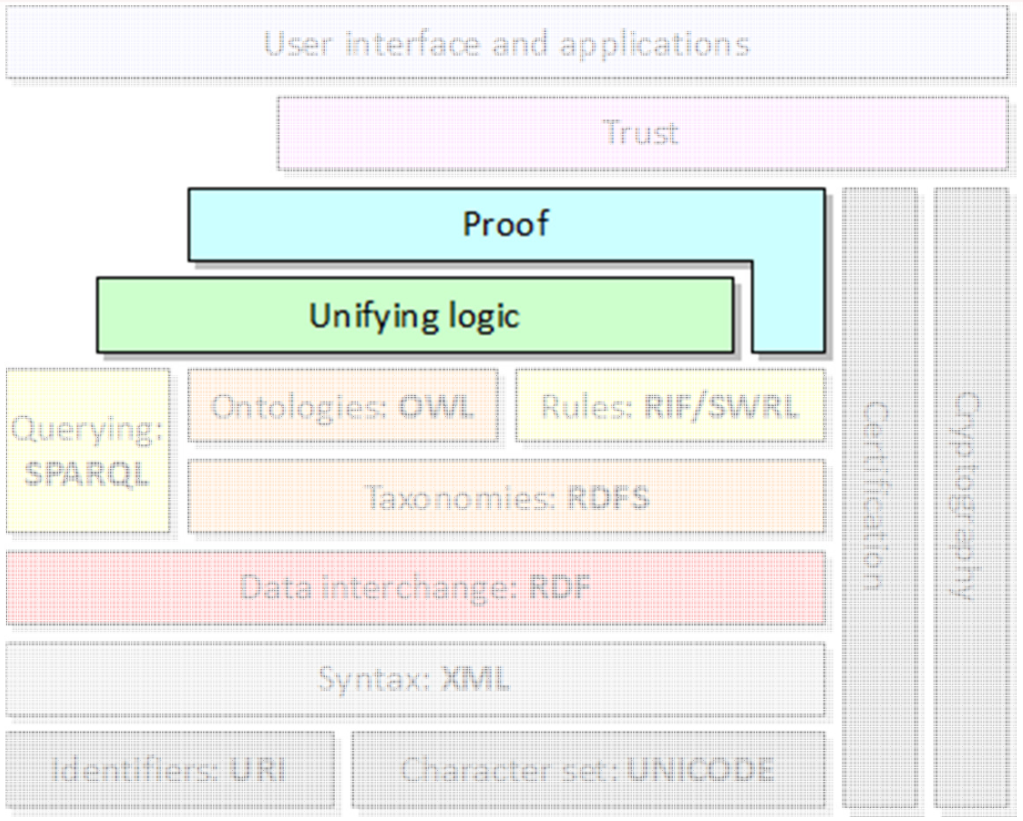
The Semantic Web Cake

Reasoning:

Unifying logic: a mediator layer between data querying, knowledge and rule representation

Proof: exploits the underlying unified logic to draw new conclusions from available data

NB: Currently undergoing active research



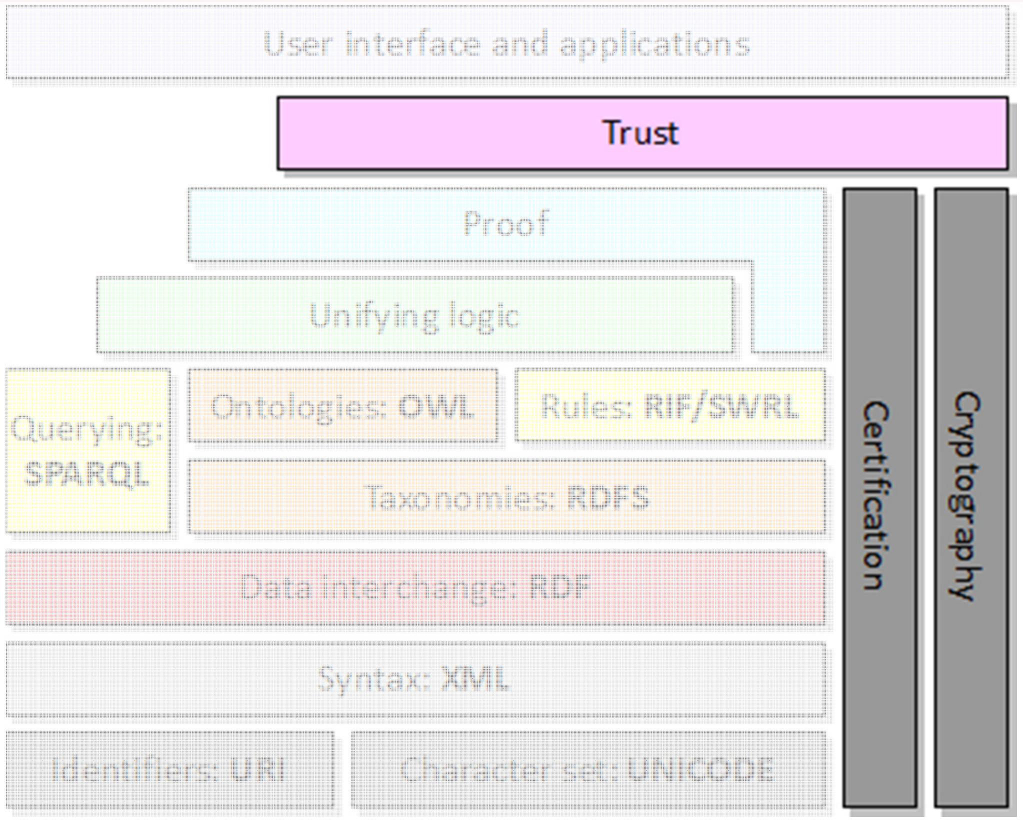
The Semantic Web Cake

Trust:

Cryptography: founded on underlying platform, ensures that data are kept confidential

Certification: founded on underlying platform, ensures that data come from an entrusted source

NB: Currently undergoing active research

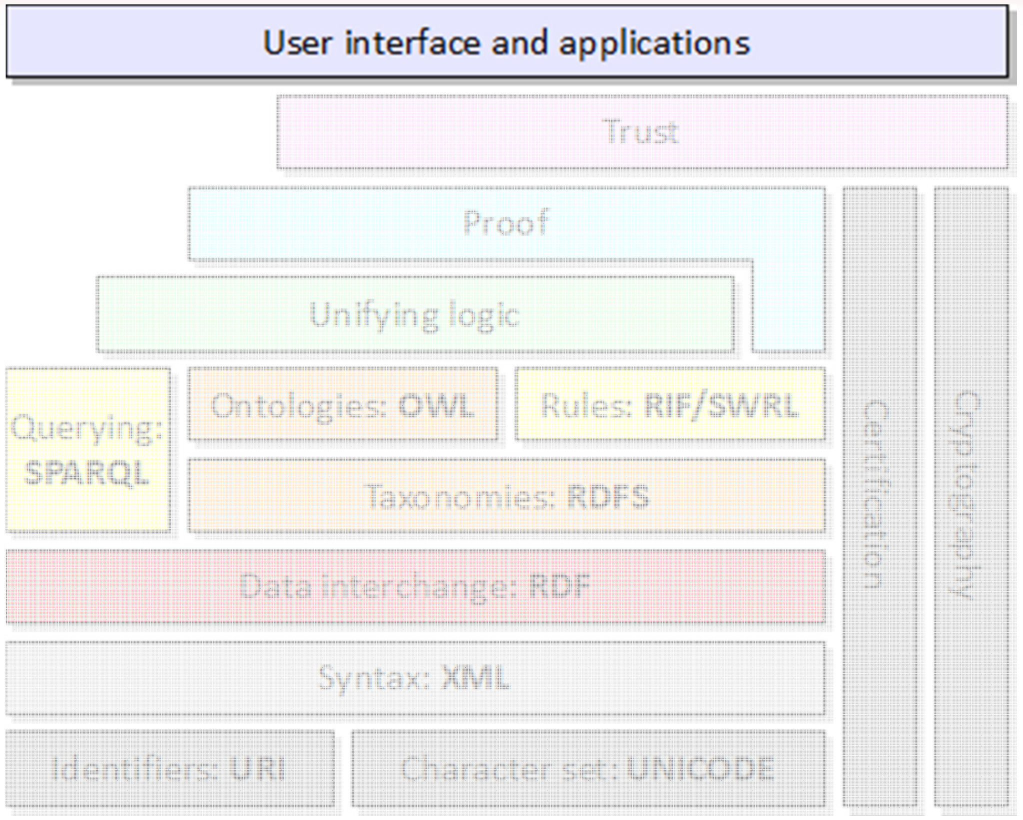


THE SEMANTIC WEB CAKE

Presentation:

User interface: provides an environment to present application to final users

NB: Currently undergoing active research



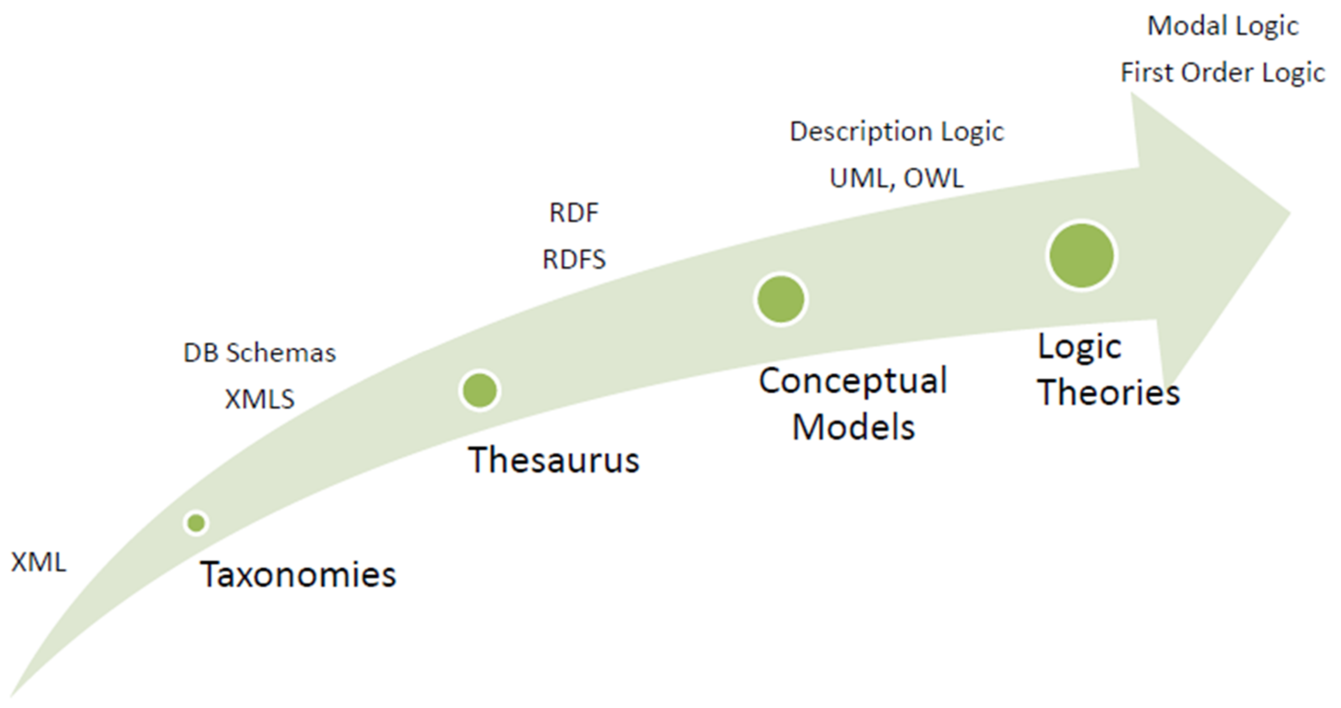
Expressiveness of Semantic Models

The languages seen so far can be arranged by increasing expressiveness.

Consider the complexity of what you aim to model when choosing one:

- **Taxonomy:** a set of terms hierarchically organized
  Ex.: IEEE Computer Society Keywords
- **Thesaurus:** a set of terms with linguistic relations among them
  Ex.: Princeton University's WordNET
- **Conceptual Model:** a set of organized concepts and specific relations among them describing a specific domain
- **Logic Theory:** basically a conceptual model supported by inference mechanisms

  Ex: PizzaOntology, FOAF, etc. just to name a few



Ontologies and the Semantic Web

- **Ontology:** a formal, explicit description of a domain of interest (by means of concepts and relations among them)

  Ontology + Instances = Knowledge Base

- Several proposal within the Semantic Web Initiative:
  - **RDF Schema:** RDF extensions with proper terms for ontological concepts; good for taxonomies
  - …

Ontologies and the Semantic Web

- Several proposal within the Semantic Web Initiative:
  - **OWL 1.1 (Ontology Web Language):** RDFS extension with three variants of increasing expressiveness:
    - **OWL-Lite:** limited support for certain features (ex.: cardinality), good for thesauri or hierarchies; intended to be easily computable, tools development is difficult as per other dialects
    - **OWL-DL:** maximum expressiveness possible, computationally complete, decidable and availability of practical reasoning algorithms; named after *Description Logic* that studies the logics of its formal foundation
    - **OWL-Full:** minimal compatibility with RDF-S, has different semantics (classes as both collections of individuals and individuals), complete reasoning support is unlikely
  - **OWL 2:** introduces "profiles" such as ***OWL 2-EL*** (fragment with polynomial time reasoning complexity), ***OWL 2-QL*** (simplifies support to queries) and ***OWL 2-RL*** (OWL subset meant to handle rules)

OWL and Description Logics

- Description Logics, as First Order Logic, is a family of logics
- Each fragment of logics depends on which operators are supported
- The more supported operators, the higher the complexity

- OWL-DL supports the following operators:

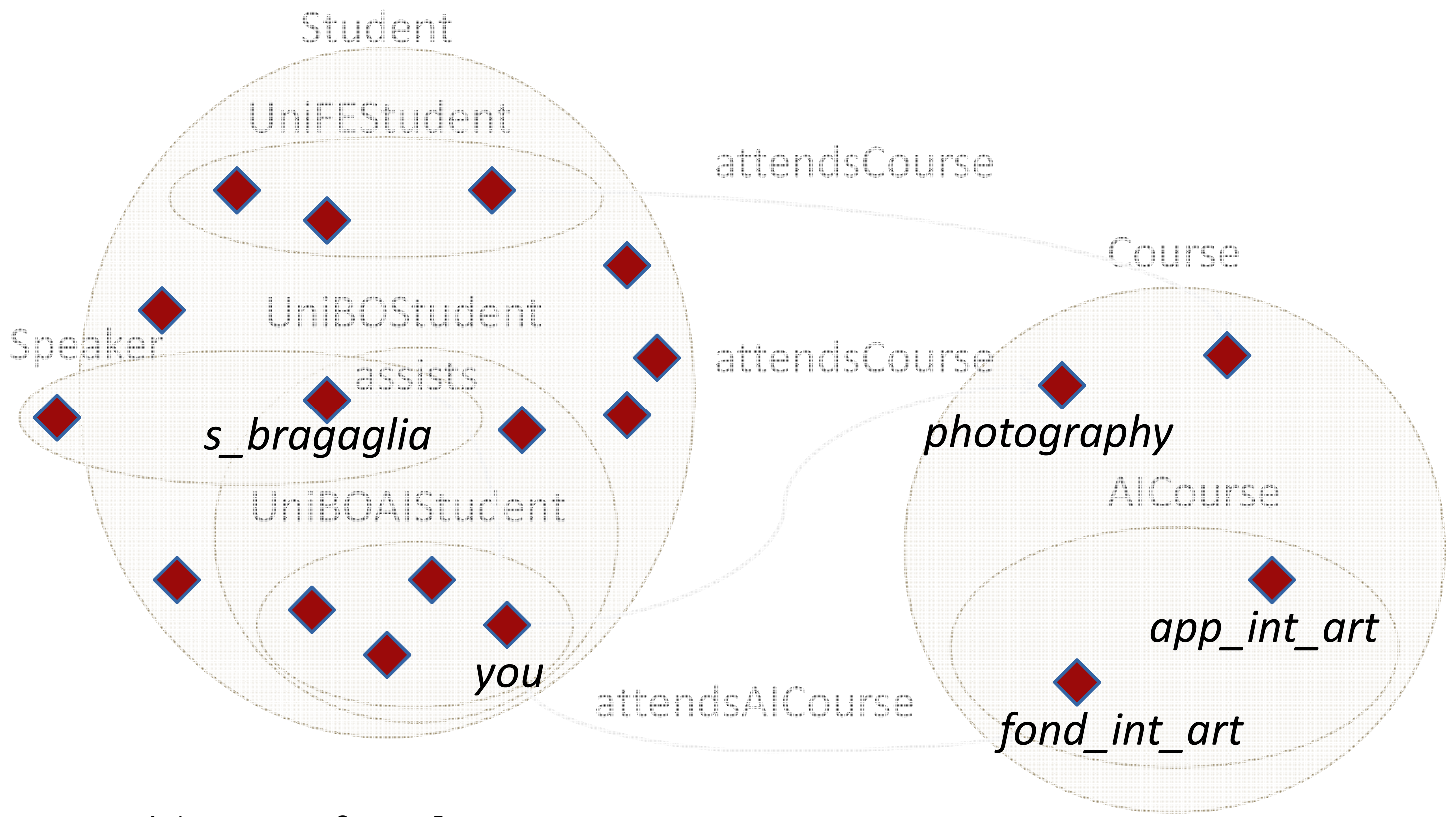
Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq$ $\neg$Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq$ $\neg${peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leqslant 1 P$	$\top \sqsubseteq \leqslant$1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1 P^-$	$\top \sqsubseteq \leqslant$1hasSSN$^-$

OWL Ontologies

Terminology:

Instance: a specific object of the domain (same as DL's *individual*)

- An instance may pertain to none, one or more classes.
- An instance may have none, one or more properties.
- Instances cannot be members of ***owl:Nothing***.

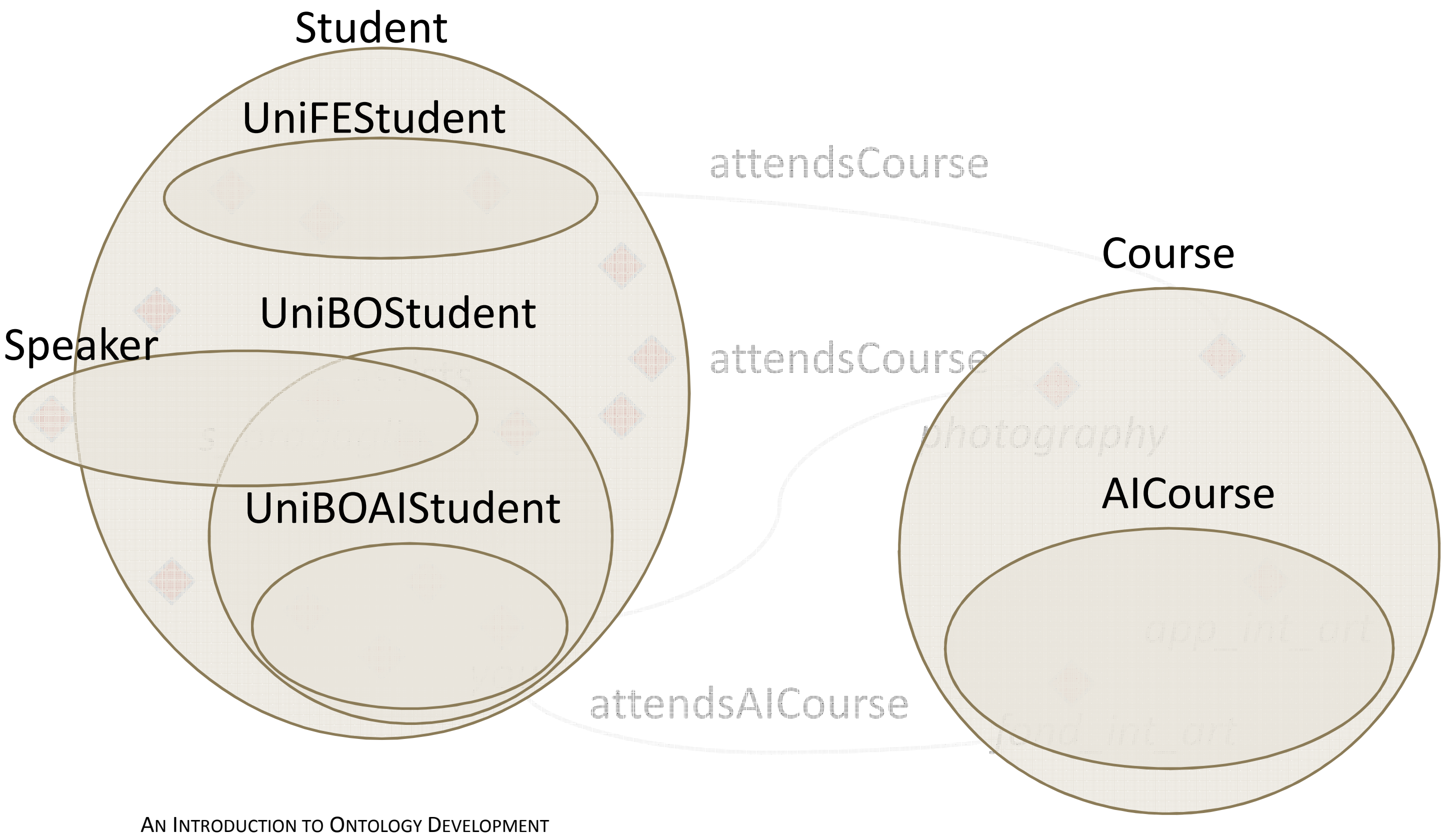


OWL Ontologies

Terminology:

Class: a collection of objects with similar characteristics (same as DL's *concept*)

- It may be a ***subclass*** of another, inheriting characteristics from its parent ***superclass*** (*logical subsumption*, DL's *concept inclusion*).
- All classes are subclasses of ***olw:Thing*** (the root class) and are subclassed by ***owl:Nothing*** (the empty class)

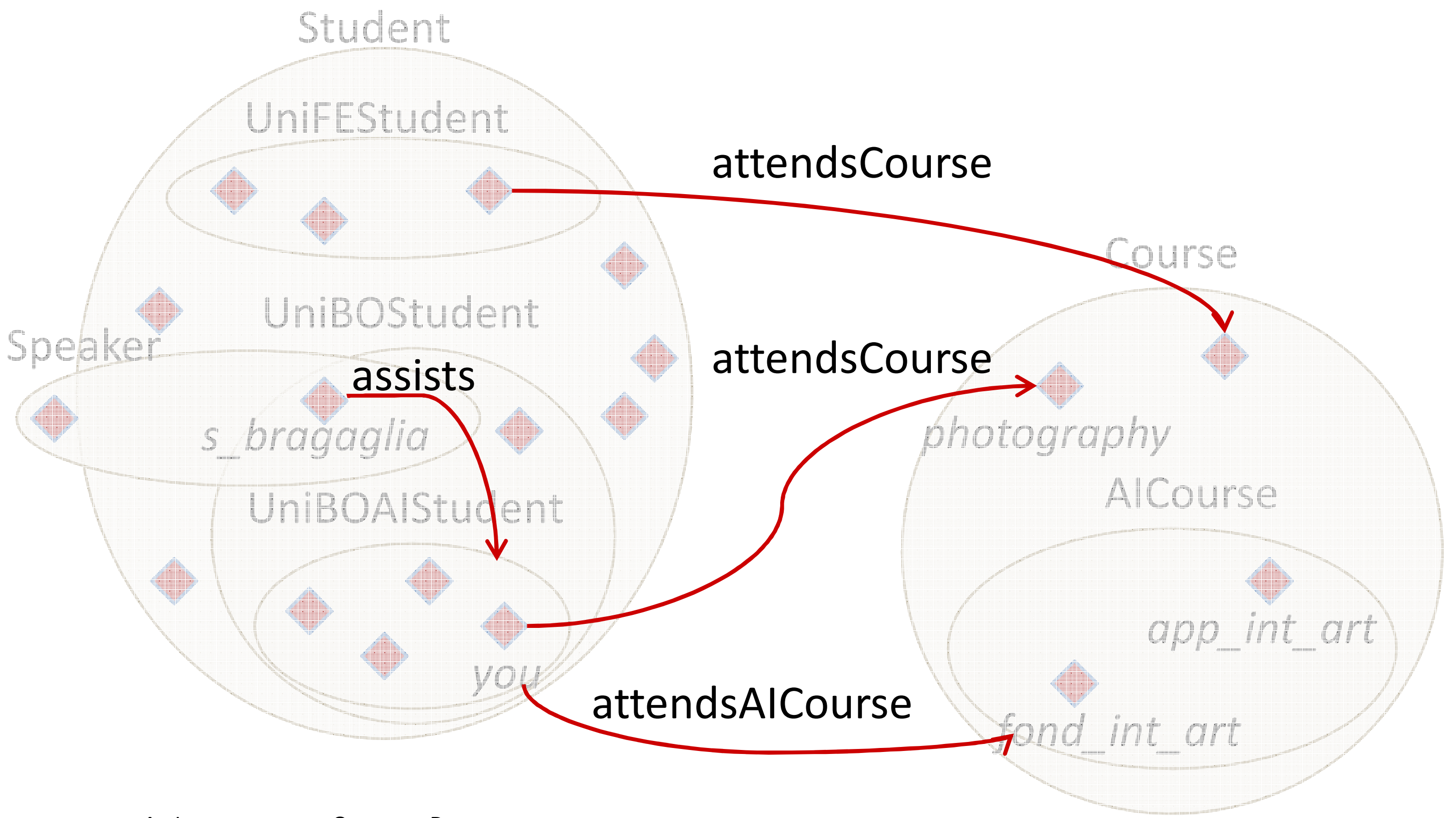


OWL Ontologies

Terminology:

Property: a direct binary relation that specifies class characteristics (same as DL's *role*)

- It may have ***domain*** and ***range***.
- It may be a ***subproperty*** of another, inheriting characteristics from its parent ***superproperty***
- As attribute of instances, it links to RDF literals (***datatype property***, ex.: hasName)
- Often it links two instances (***object property***); it may have logical capabilities such as being ***transitive***, ***symmetric***, ***inverse*** and ***functional***.

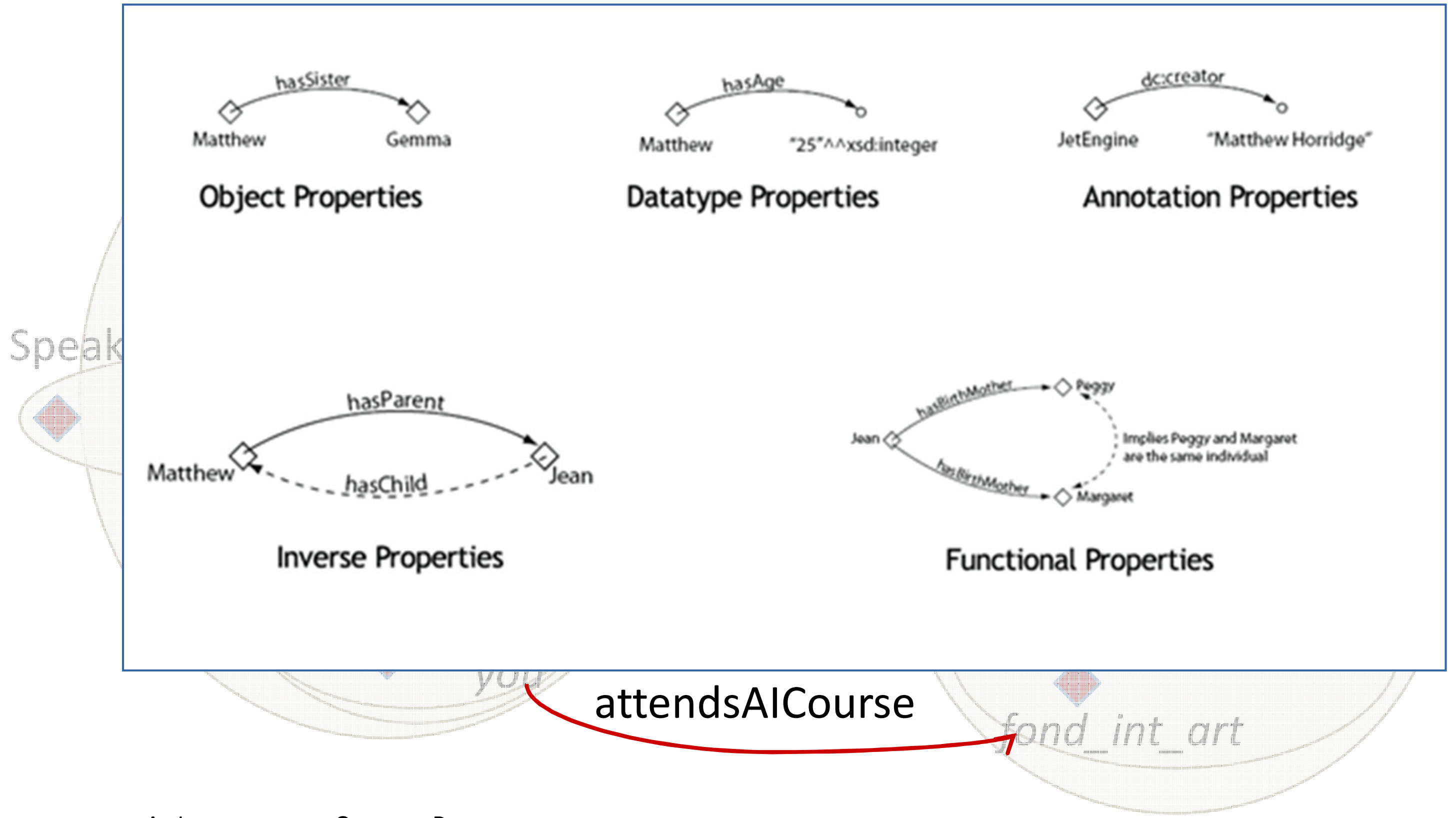


OWL Ontologies

Terminology:

Property: a direct binary relation that specifies class characteristics (same as DL's *role*)

- It may have ***domain*** and ***range***.
- It may be a ***subproperty*** of another, inheriting characteristics from its parent ***superproperty***
- As attribute of instances, it links to RDF literals (***datatype property***, ex.: hasName)
- Often it links two instances (***object property***); it may have logical capabilities such as being ***transitive***, ***symmetric***, ***inverse*** and ***functional***.

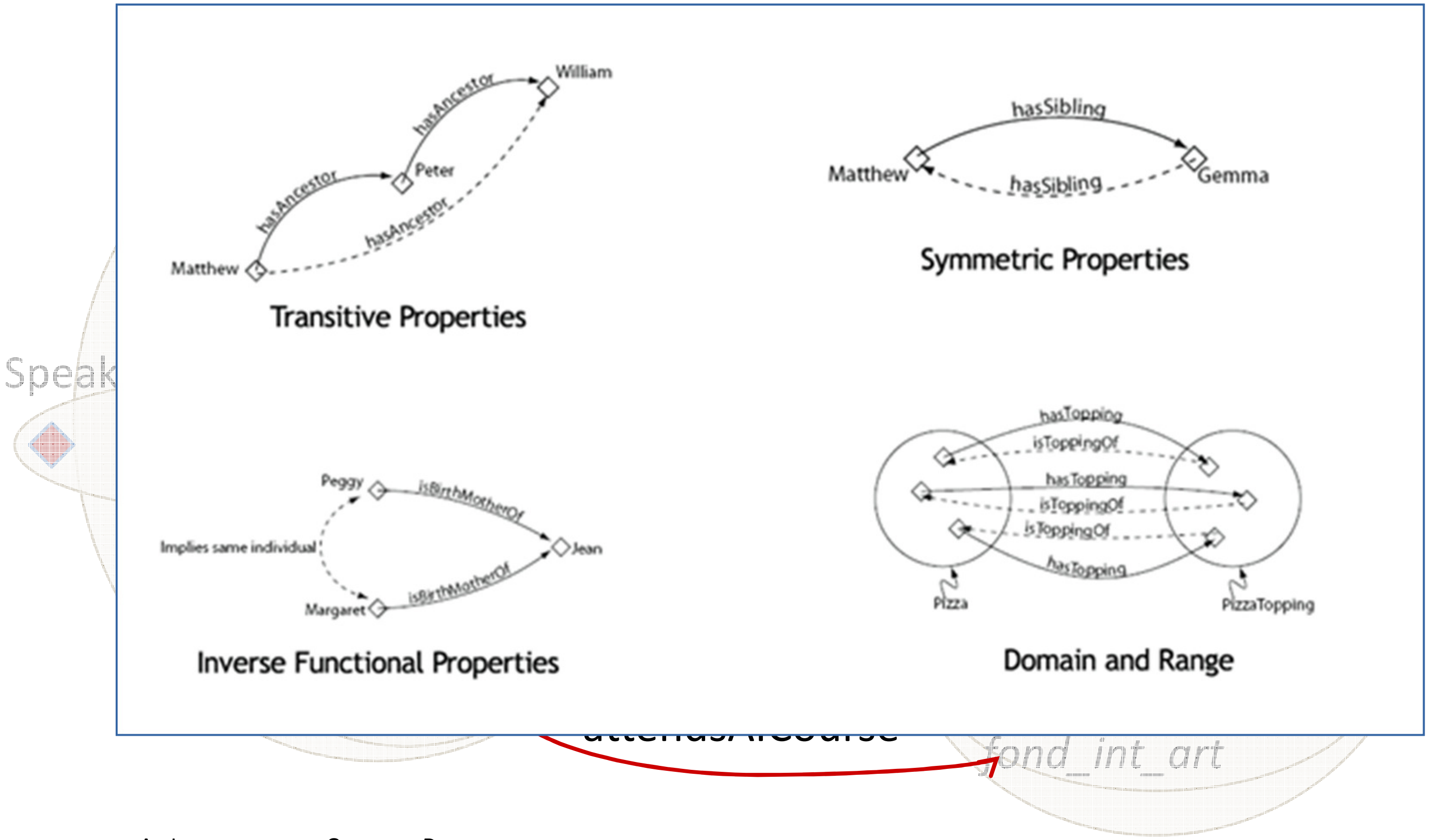


OWL Ontologies

Terminology:

Property: a direct binary relation that specifies class characteristics (same as DL's *role*)

- It may have ***domain*** and ***range***.
- It may be a ***subproperty*** of another, inheriting characteristics from its parent ***superproperty***
- As attribute of instances, it links to RDF literals (***datatype property***, ex.: hasName)
- Often it links two instances (***object property***); it may have logical capabilities such as being ***transitive***, ***symmetric***, ***inverse*** and ***functional***.



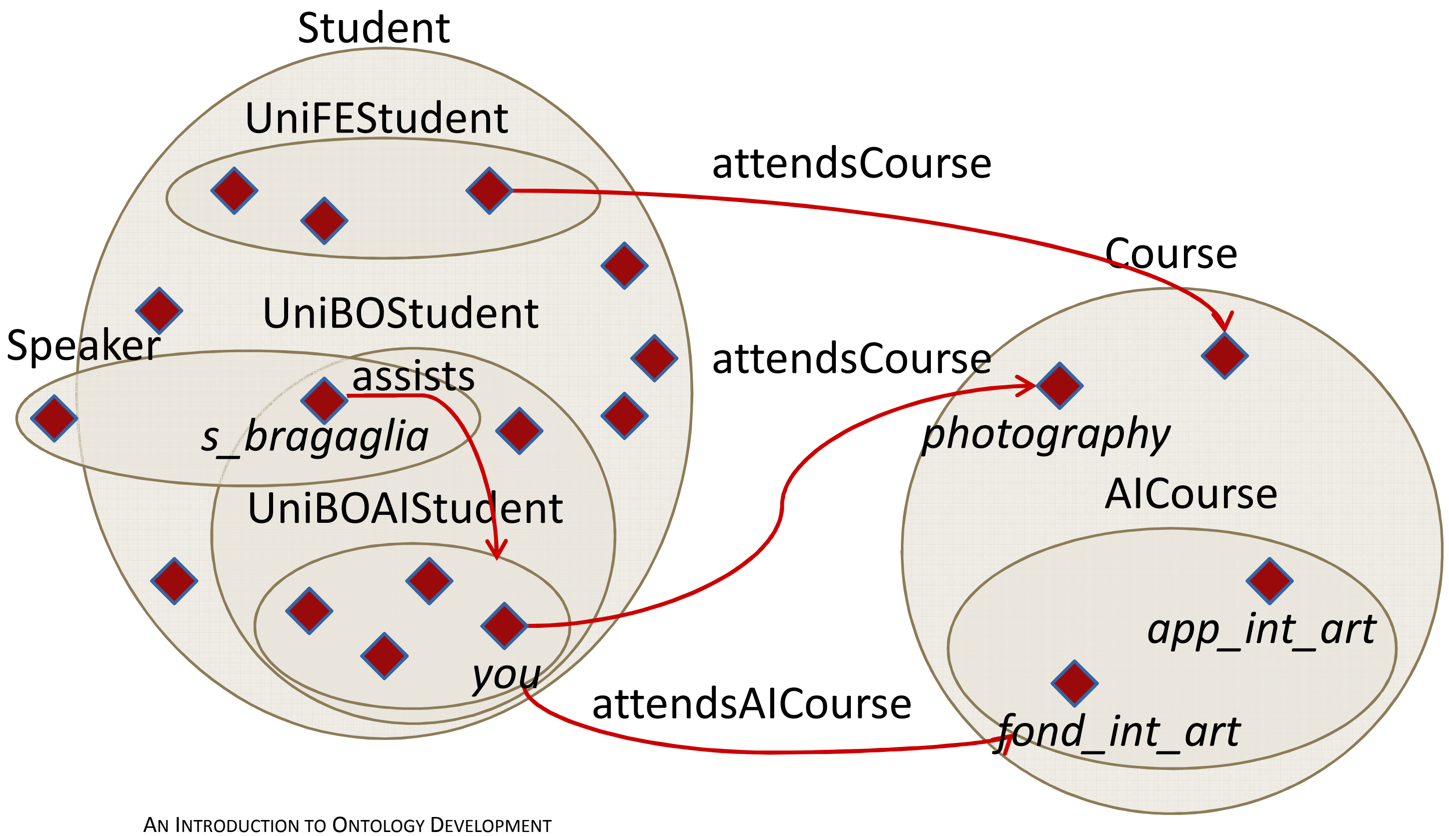
OWL Ontologies

Terminology:

Operator: OWL language supports various operations on classes such as ***union***, ***intersection*** and ***complement***

Enumeration, ***cardinality*** and ***disjointness*** are also supported.

Such operations are usually referred as ***restrictions*** (and ***expressions***, backward compatibility)



Developing Ontologies

A possible development process involves:

1. Analyze the **domain** and the **goal** of the ontology
2. Determine the **key concepts** of the domain
   - Glossary of terms
   - Competency questions
3. Consider to **reuse existing ontologies**
   - **Upper** / **medium** / **lower** ontologies (Greek temple model)
   - Ontologies **patterns**
4. Organize concepts in **classes** e hierarchies among classes
5. Determine the **properties** of the classes
6. Add **constraints** (allowed values) on the properties
7. Create the **instances**
8. Assign **values** to the properties for each instance
9. Verify the ontology and release it

DEVELOPING ONTOLOGIES

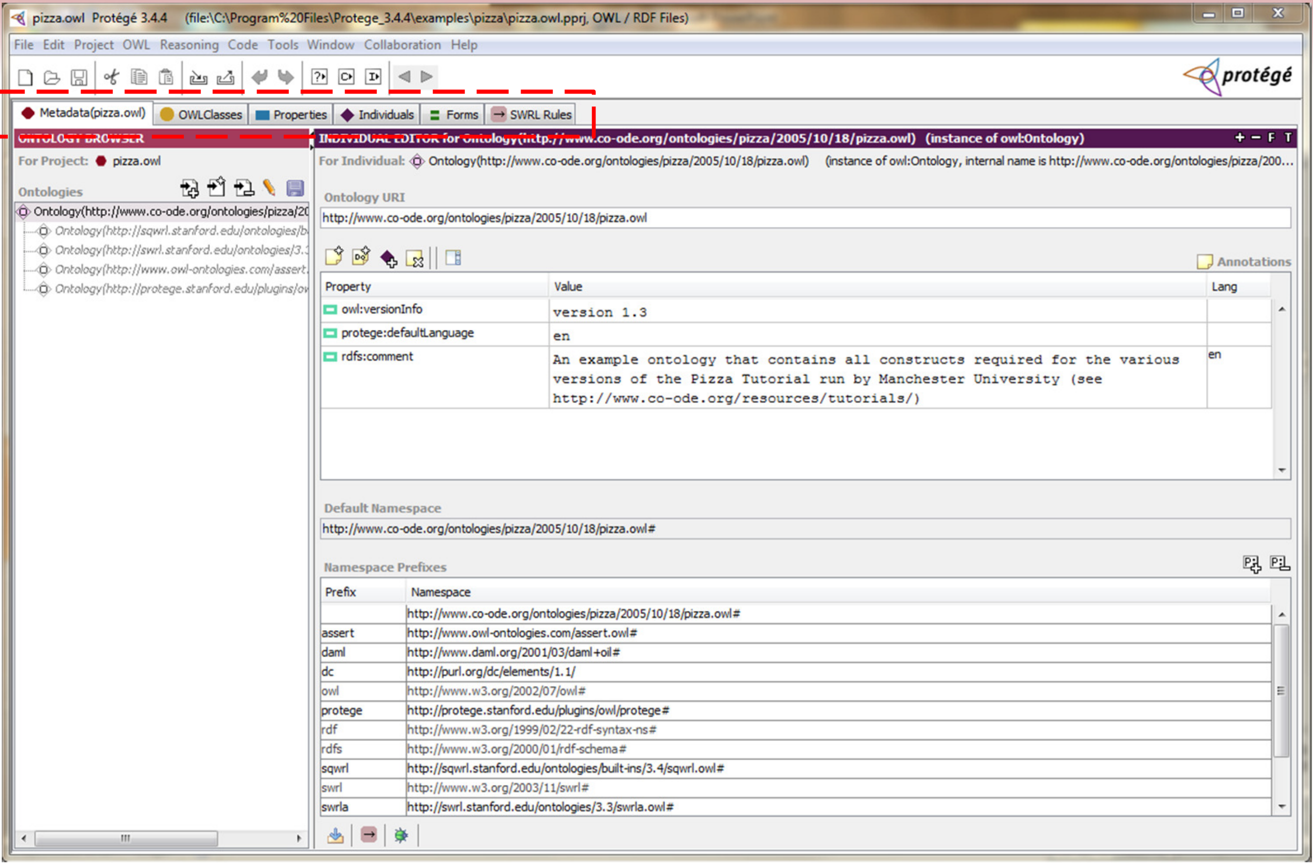
- ***Purposes:***
  - Export (data export, knowledge export, etc.)
  - Modeling (formalization of steps during development, etc.)
  - Interoperability between different systems

- ***Tools:***
  - Too much, at the moment! Almost any competitor proposes its own tool due to the big expectations…
  - Remarkable editors are (just to name a few): *Protégé*, *TopBraid Composer*, *Swoop*, *DOME*, *NeOn*, *WSMT*, etc.

Protégé 3.4.4

Editing functions are split in different tabs:

- **Metadata:** overview of the ontology
- **OWL Classes:** concepts insertion and organization
- **Properties:** roles definition and customization
- **Individuals:** object introduction and composition
- **Forms:** form definition for data insertion
- **SWRL Rules:** rule definition that triggers on existing data


pizza.owl Protégé 3.4.4 (file:\C:\Program%20Files\Protege_3.4.4\examples\pizza\pizza.owl.pprj, OWL / RDF Files)
File Edit Project OWL Reasoning Code Tools Window Collaboration Help
protégé
Metadata(pizza.owl)
OWLClasses
Properties
Individuals
Forms
SWRL Rules
ONTOLOGY BROWSER
INDIVIDUAL EDITOR for Ontology(http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl) (instance of owl:Ontology)
For Project: pizza.owl
Ontologies
Annotations

Ontology(http://www.co-ode.org/ontologies/pizza/20
Ontology(http://sqwrl.stanford.edu/ontologies/b
Ontology(http://swrl.stanford.edu/ontologies/3.3
Ontology(http://www.owl-ontologies.com/assert.
Ontology(http://protege.stanford.edu/plugins/ow

For Individual: Ontology(http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl) (instance of owl:Ontology, internal name is http://www.co-ode.org/ontologies/pizza/200...

Ontology URI

http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl

Property	Value	Lang
owl:versionInfo	version 1.3	
protege:defaultLanguage	en	
rdfs:comment	An example ontology that contains all constructs required for the various versions of the Pizza Tutorial run by Manchester University (see http://www.co-ode.org/resources/tutorials/)	en

Default Namespace

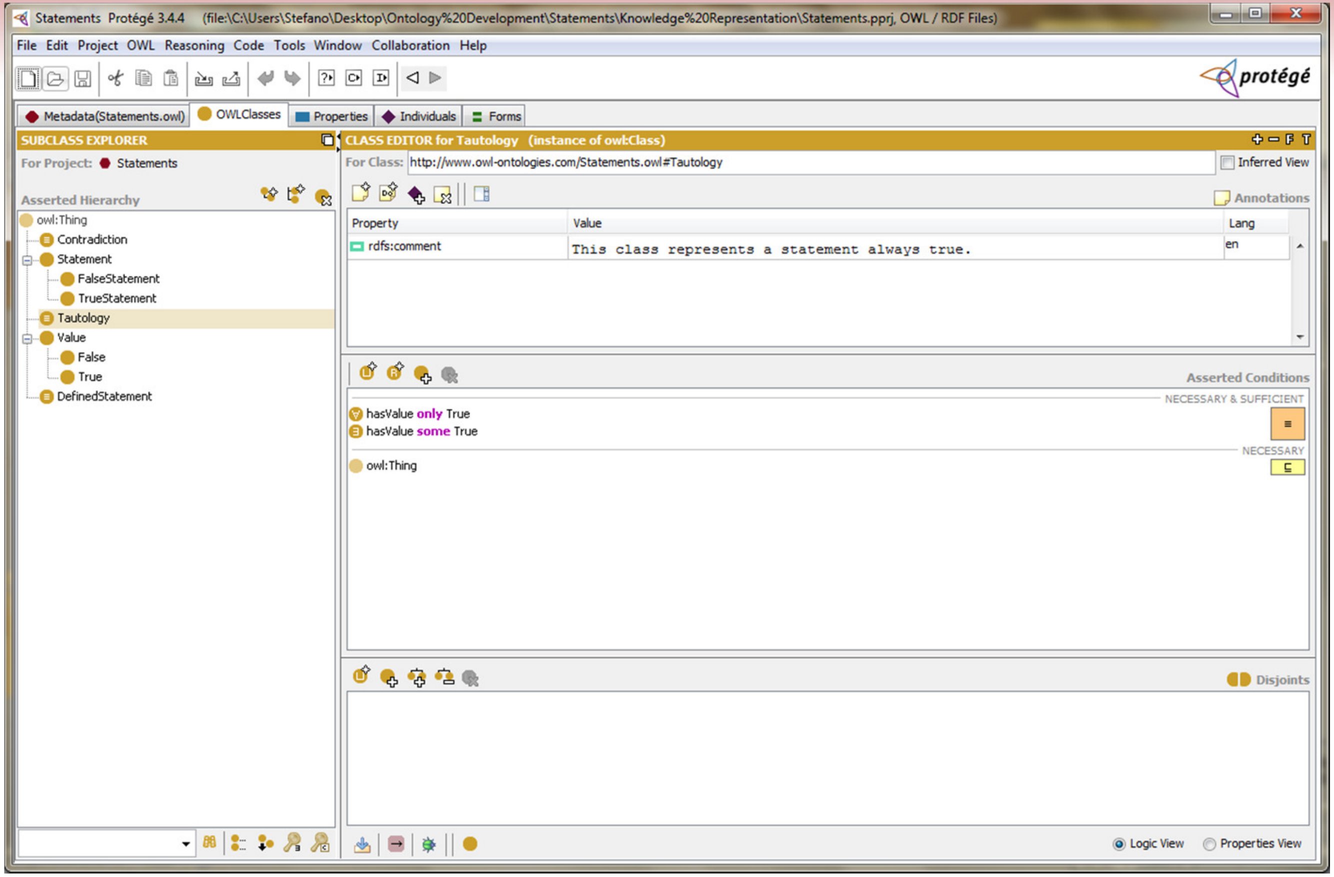
http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl#

Namespace Prefixes

Prefix	Namespace
	http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl#
assert	http://www.owl-ontologies.com/assert.owl#
daml	http://www.daml.org/2001/03/daml+oil#
dc	http://purl.org/dc/elements/1.1/
owl	http://www.w3.org/2002/07/owl#
protege	http://protege.stanford.edu/plugins/owl/protege#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
sqwrl	http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#
swrl	http://www.w3.org/2003/11/swrl#
swrla	http://swrl.stanford.edu/ontologies/3.3/swrla.owl#

http://protege.stanford.edu/download/registered.html

OWL Classes Tab


Statements Protégé 3.4.4 (file:\C:\Users\Stefano\Desktop\Ontology%20Development\Statements\Knowledge%20Representation\Statements.pprj, OWL / RDF Files)
File Edit Project OWL Reasoning Code Tools Window Collaboration Help
protégé
Metadata(Statements.owl)
OWLClasses
Properties
Individuals
Forms
SUBCLASS EXPLORER
For Project: Statements
Asserted Hierarchy
CLASS EDITOR for Tautology (instance of owl:Class)
For Class: http://www.owl-ontologies.com/Statements.owl#Tautology
Inferred View
Annotations
Asserted Conditions
NECESSARY & SUFFICIENT
NECESSARY
Disjoints
Logic View
Properties View

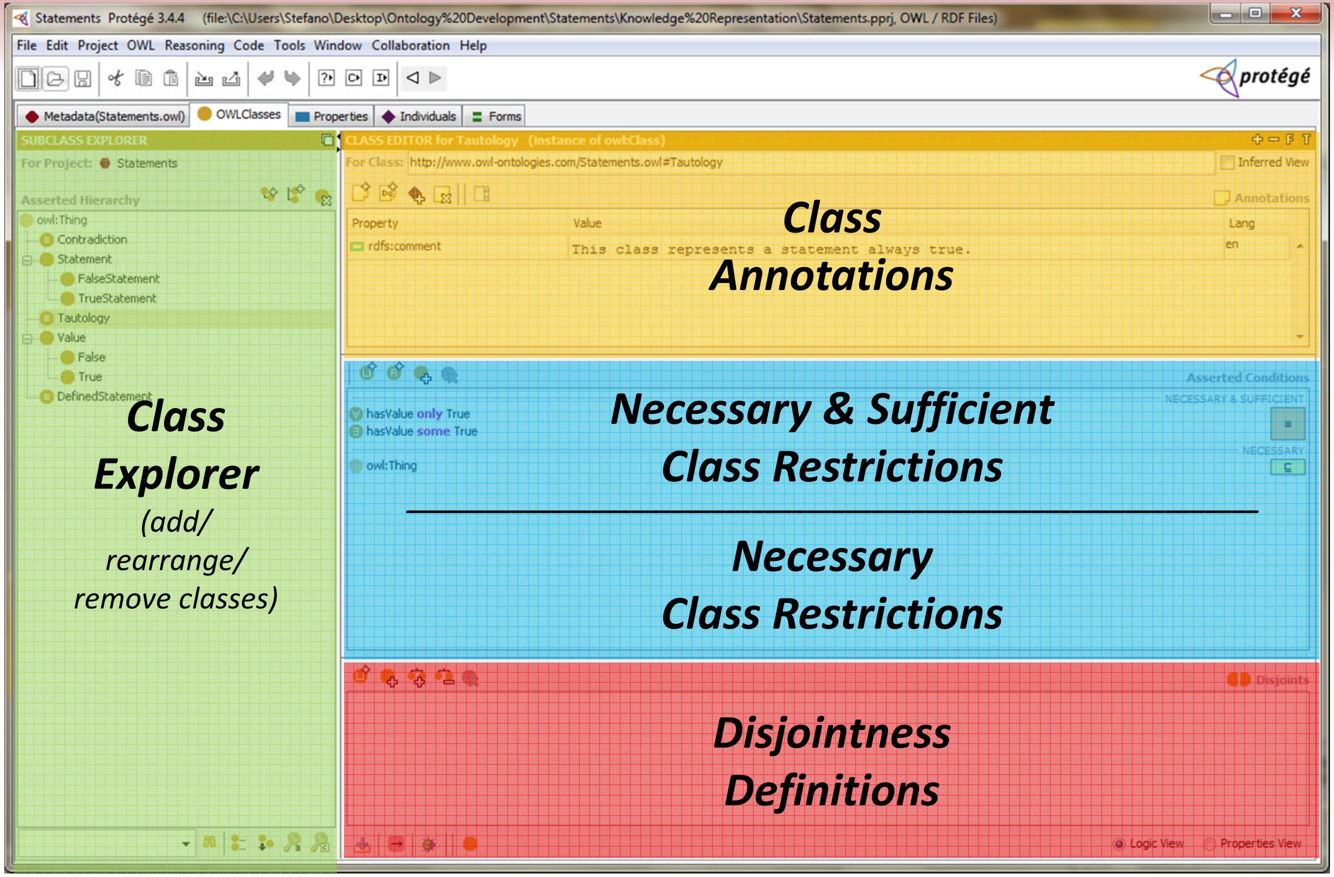
owl:Thing
- Contradiction
- Statement
  - FalseStatement
  - TrueStatement
- Tautology
- Value
  - False
  - True
- DefinedStatement

Property	Value	Lang
rdfs:comment	This class represents a statement always true.	en

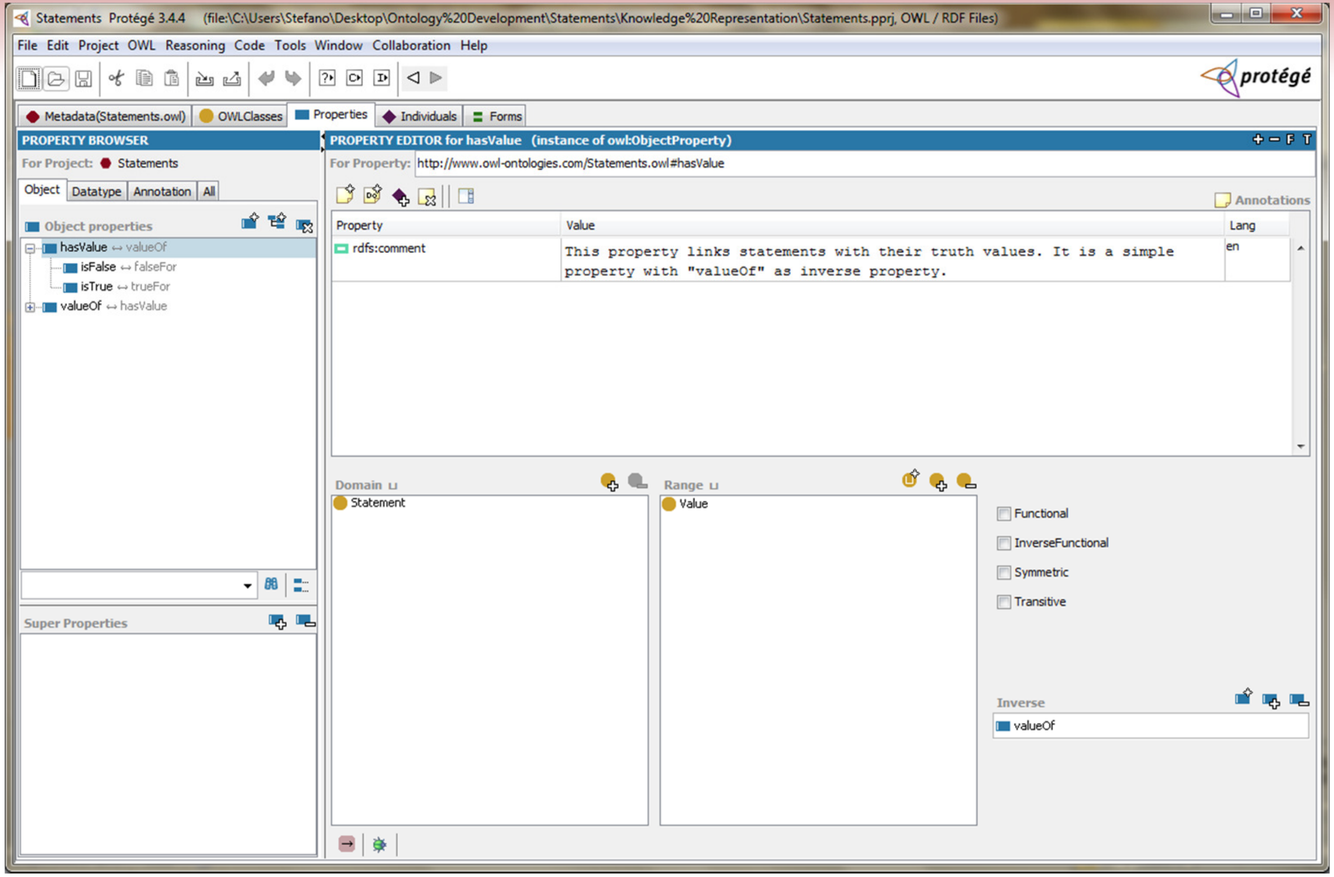
hasValue **only** True
hasValue **some** True

owl:Thing

OWL Classes Tab


Statements Protégé 3.4.4 (file:\C:\Users\Stefano\Desktop\Ontology%20Development\Statements\Knowledge%20Representation\Statements.pprj, OWL / RDF Files)
File Edit Project OWL Reasoning Code Tools Window Collaboration Help
protégé
Metadata(Statements.owl)
OWLClasses
Properties
Individuals
Forms
SUBCLASS EXPLORER
For Project: Statements
Asserted Hierarchy
owl:Thing
Contradiction
Statement
FalseStatement
TrueStatement
Tautology
Value
False
True
DefinedStatement
CLASS EDITOR for Tautology (instance of owl:Class)
For Class: http://www.owl-ontologies.com/Statements.owl#Tautology
Inferred View
Annotations
Property
Value
Lang
rdfs:comment
This class represents a statement always true.
en
Asserted Conditions
NECESSARY & SUFFICIENT
hasValue only True
hasValue some True
NECESSARY
owl:Thing
Disjoints
Logic View
Properties View
Class Explorer
(add/ rearrange/ remove classes)
Class Annotations
Necessary & Sufficient Class Restrictions
Necessary Class Restrictions
Disjointness Definitions

Properties Tab



Statements Protégé 3.4.4 (file:\C:\Users\Stefano\Desktop\Ontology%20Development\Statements\Knowledge%20Representation\Statements.pprj, OWL / RDF Files)

PROPERTY BROWSER

For Project: Statements

Object properties

- hasValue ↔ valueOf
  - isFalse ↔ falseFor
  - isTrue ↔ trueFor
- valueOf ↔ hasValue

Super Properties

PROPERTY EDITOR for hasValue (instance of owl:ObjectProperty)

For Property: http://www.owl-ontologies.com/Statements.owl#hasValue

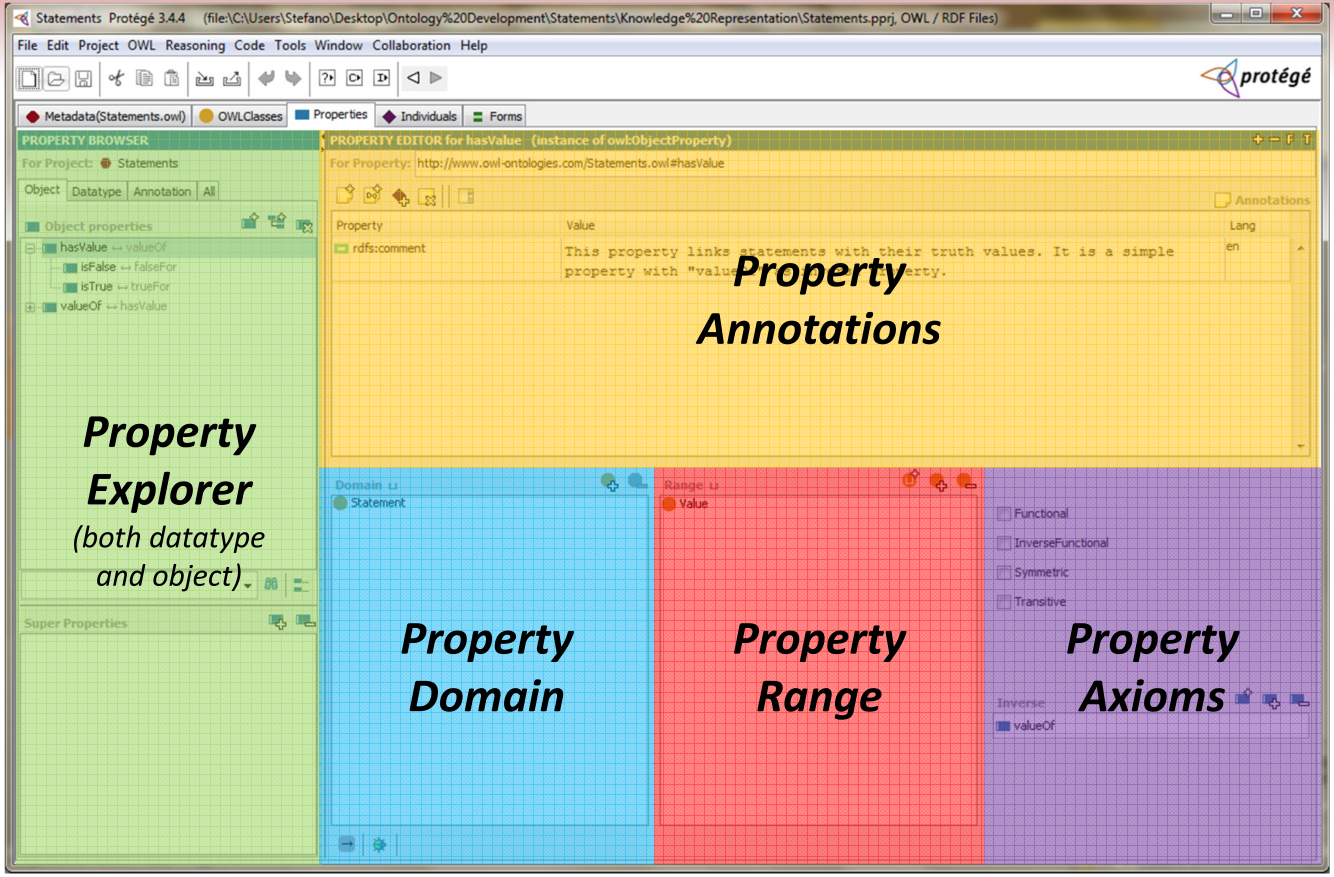
Property	Value	Lang
rdfs:comment	This property links statements with their truth values. It is a simple property with "valueOf" as inverse property.	en

Domain: Statement

Range: Value

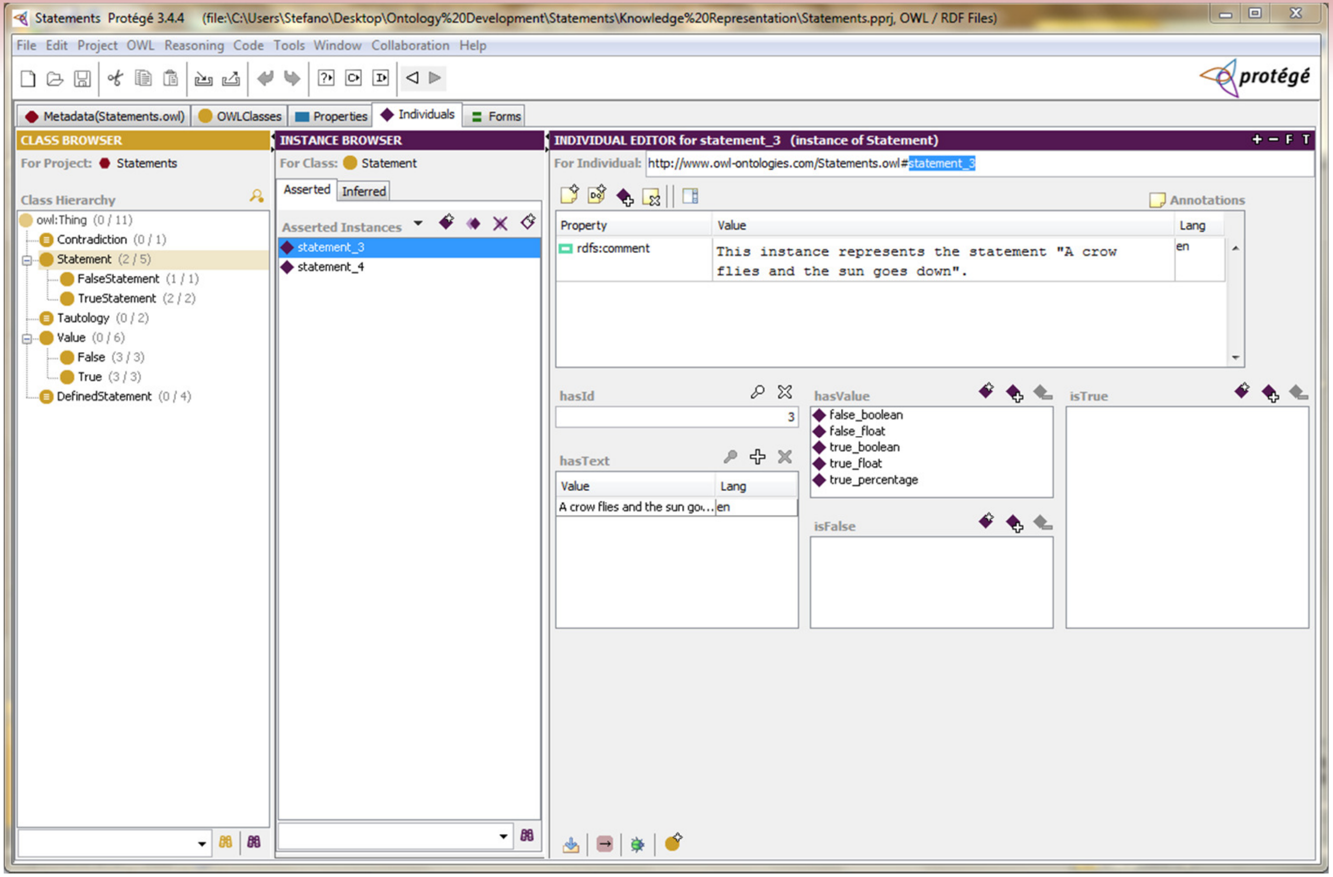
Inverse: valueOf

PROPERTIES TAB



This property links statements with their truth values. It is a simple property with "value" property.

INDIVIDUALS TAB


Statements Protégé 3.4.4 (file:\C:\Users\Stefano\Desktop\Ontology%20Development\Statements\Knowledge%20Representation\Statements.pprj, OWL / RDF Files)
File Edit Project OWL Reasoning Code Tools Window Collaboration Help
protégé
Metadata(Statements.owl) OWLClasses Properties Individuals Forms
CLASS BROWSER
INSTANCE BROWSER
Asserted Inferred
Asserted Instances

For Project: Statements

Class Hierarchy
- owl:Thing (0 / 11)
  - Contradiction (0 / 1)
  - Statement (2 / 5)
    - FalseStatement (1 / 1)
    - TrueStatement (2 / 2)
  - Tautology (0 / 2)
  - Value (0 / 6)
    - False (3 / 3)
    - True (3 / 3)
  - DefinedStatement (0 / 4)

For Class: Statement

- statement_3
- statement_4

INDIVIDUAL EDITOR for statement_3 (instance of Statement)

For Individual: http://www.owl-ontologies.com/Statements.owl#statement_3

Annotations

Property	Value	Lang
rdfs:comment	This instance represents the statement "A crow flies and the sun goes down".	en

hasId: 3

hasText

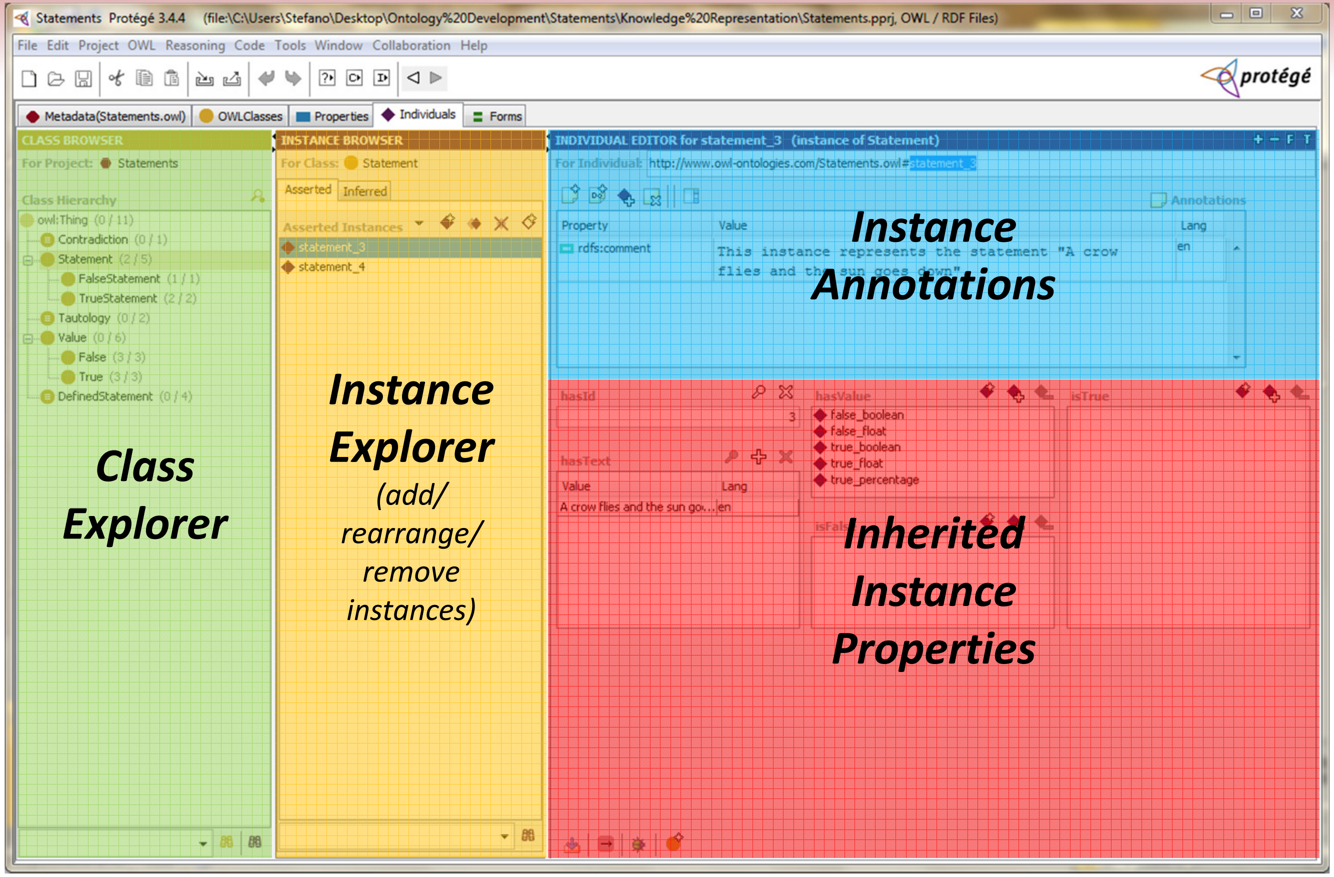
Value	Lang
A crow flies and the sun go...	en

hasValue
- false_boolean
- false_float
- true_boolean
- true_float
- true_percentage

isFalse

isTrue

Individuals Tab


Statements Protégé 3.4.4 (file:\C:\Users\Stefano\Desktop\Ontology%20Development\Statements\Knowledge%20Representation\Statements.pprj, OWL / RDF Files)
File Edit Project OWL Reasoning Code Tools Window Collaboration Help
protégé
Metadata(Statements.owl) OWLClasses Properties Individuals Forms
CLASS BROWSER
For Project: Statements
Class Hierarchy
owl:Thing (0 / 11)
Contradiction (0 / 1)
Statement (2 / 5)
FalseStatement (1 / 1)
TrueStatement (2 / 2)
Tautology (0 / 2)
Value (0 / 6)
False (3 / 3)
True (3 / 3)
DefinedStatement (0 / 4)
INSTANCE BROWSER
For Class: Statement
Asserted Inferred
Asserted Instances
statement_3
statement_4
INDIVIDUAL EDITOR for statement_3 (instance of Statement)
For Individual: http://www.owl-ontologies.com/Statements.owl#statement_3
Annotations
Property Value Lang
rdfs:comment This instance represents the statement "A crow flies and the sun goes down" en
hasId 3
hasText
Value Lang
A crow flies and the sun go... en
hasValue
false_boolean
false_float
true_boolean
true_float
true_percentage
isTrue
isFalse
Class Explorer
Instance Explorer (add/ rearrange/ remove instances)
Instance Annotations
Inherited Instance Properties

An Introductory Example

- Suppose the domain to capture is the following:

  In logic, each statement has a text and has a truth value depending on the value of its elements.
  Statements are identified with unique numeric ids.
  Allowed truth values are true and false.
  A tautology is a statement that is true by definition, for any value of its elements.
  Contradictions are the opposite of tautologies.

- For example *"A crow flies when the sun goes down."* is a statement, *"All crows are black or there is one that is not."* is a tautology and *"Every crow is either completely black and completely white."* is a contradiction.

An Introductory Example

- The following **text analysis** is possible:

  In logic, each statement has a text and has a truth value depending on the value of its elements.
  Statements are identified with unique numeric ids.
  Allowed truth values are true and false.
  A tautology is a statement that is true by definition, for any value of its elements.
  Contradictions are the opposite of tautologies.

- The identified terms and attributes should be reported into a **glossary**.

An Introductory Example

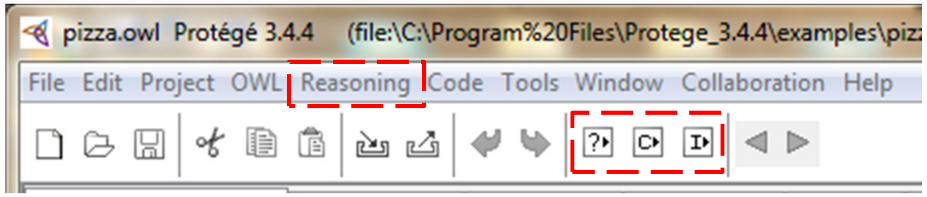
- The given text may lead to the following **competency questions**:
  - What defines a statement? A text.
  - What identifies a statement? A numeric id.
  - What resolves a statement to? A truth value.
  - What truth values are valid? True and false.
  - What is a tautology? A statement always true.
  - What is a contradiction? The opposite of a tautology.

Note: this approach considers a bit of information at a time instead of the whole at once. Development is easier but end result is less uniform and reuse is more difficult.

An Introductory Example

- Due to the nature and size of the domain, no existing ontology will be used.
- Create a new ontology and introduce the identified items:
  - **Classes:** Statement, Value
  - **Properties:** hasId, hasText, hasValue
  - **Instances:** a few examples like the proposed one.
  - **Values:** strings, integers, etc.

An Introductory Example
Reasoning Tasks


pizza.owl Protégé 3.4.4 (file:\C:\Program%20Files\Protege_3.4.4\examples\piz
File Edit Project OWL Reasoning Code Tools Window Collaboration Help

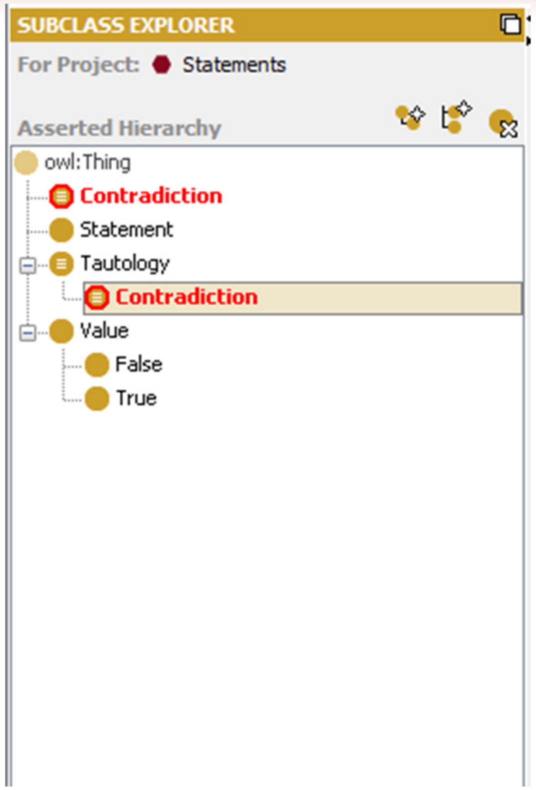
- Under “Reasoning” menu, select “Pellet 1.5.2” as default reasoner and
  - Check the consistency of current ontology
  - Classify automatically the classes entered so far
  - Compute inferred types for instances

Note: Ontology should be *consistent* and *explicit*.

Detecting Consistency Errors

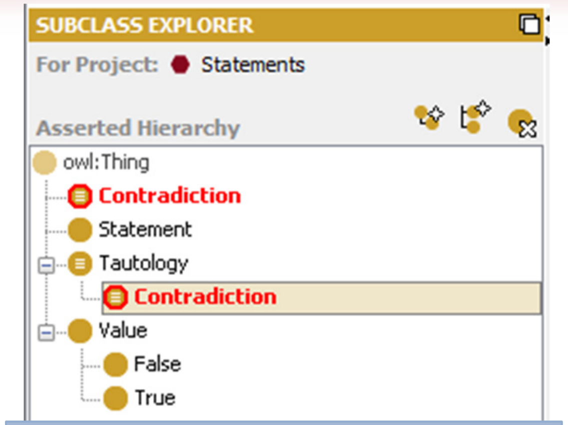
- Sometimes it happens that two or more concepts of the ontology inadvertently conflicts with each other
- Detecting and fixing such conflicts may be difficult, especially when dealing with large projects
- This is where the consistency check comes in handy!
- As an example, introduce ***Tautology*** as a top-level class restriction that *hasValue only True* and ***Contradiction*** as a ***Tautology*** that instead *hasValue only False*

Detecting Consistency Errors



- If you run the ***consistency check*** task again, the inconsistency is detected and highlighted in red!
- To fix the error, you have to identify the colliding concepts in the editor panel of inconsistent class or property and modify them properly
- An extension which directly identifies the colliding concepts is being discussed

Detecting Consistency Errors



In this case, the error is due to the definition of ***Contradiction***:
- *hasValue only False*
- *hasValue only True* (inherited)

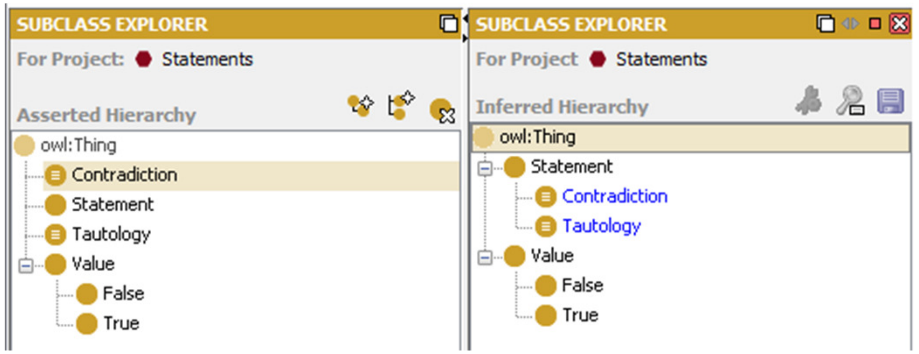
and
- *True disjoint with False*

The error is fixed by simply stating that it is a ***Statement*** and not a ***Tautology***!

- If you run the ***consistency check*** task again, the inconsistency is detected and highlighted in red!
- To fix the error, you have to identify the colliding concepts in the editor panel of inconsistent class or property and modify them properly
- An extension which directly identifies the colliding concepts is being discussed

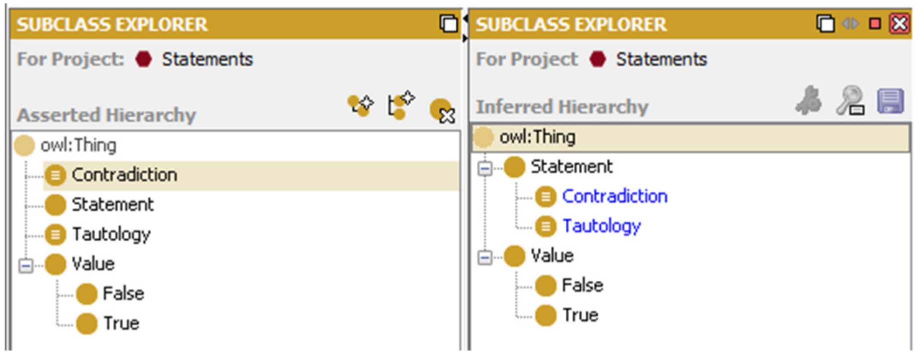
Classifying Classes in Taxonomies

- Despite being top-level classes, ***Tautology*** and ***Contradiction*** are somehow related with ***Statement***
- The ***classify taxonomy*** task successfully recognizes such dependency and rearrange them as sub-classes of Statement



CLASSIFYING CLASSES IN TAXONOMIES

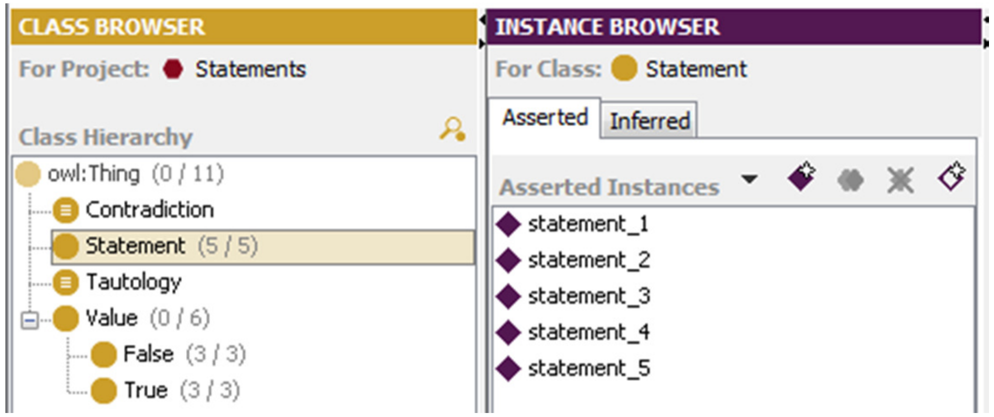
- Despite being top-level classes, ***Tautology*** and ***Contradiction*** are somehow related with ***Statement***
- The ***classify taxonomy*** task successfully recognizes such dependency and rearrange them as sub-classes of Statement



This is due to the fact that they both use *hasValue* property whose domain is actually ***Statement***

Inferring Types for Instances

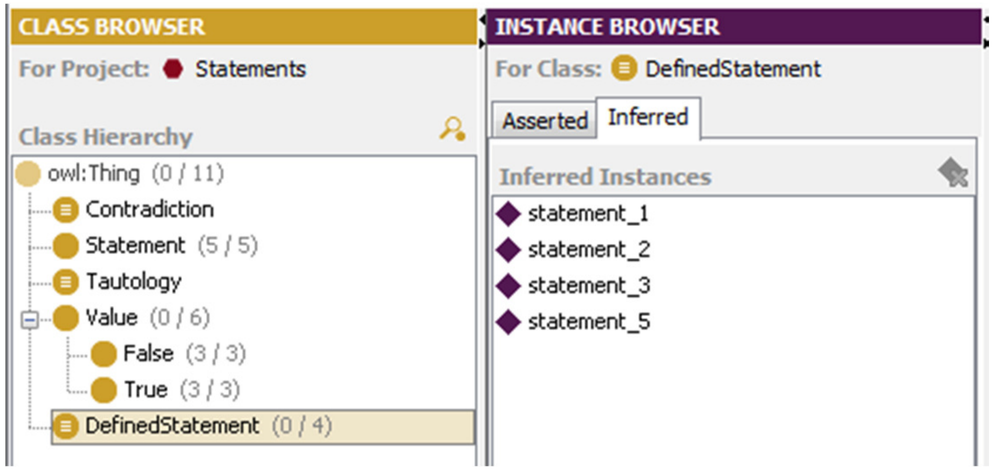
- Having introduced instances that appear to be a ***Tautology*** and a ***Contradiction***, we expect that the ***compute inferred types*** task recognizes them accordingly…
- …but all the instances are still reported only as ***Statements***! What's wrong?



INFERRING TYPES FOR INSTANCES

- Let's have another try:
  Introduce ***DefinedStatement*** as a top-level restriction class that *hasValue some (True or False)* and run the ***compute inferred types*** task again…

- …this time it seems to work! What's wrong?



CLASS BROWSER
For Project: Statements
Class Hierarchy
owl:Thing (0 / 11)
Contradiction
Statement (5 / 5)
Tautology
Value (0 / 6)
False (3 / 3)
True (3 / 3)
DefinedStatement (0 / 4)

INSTANCE BROWSER
For Class: DefinedStatement
Asserted | Inferred
Inferred Instances
statement_1
statement_2
statement_3
statement_5

Open World vs. Close World Assumption

- Unlike Prolog, SQL and many other languages, OWL (and DL in general) adopts the ***Open World Assumption*** instead of the ***Close World Assumption***
  - CWA implies that everything not explicitly asserted is false,
  - OWA states that we cannot speculate on something that is not explicitly asserted since it is undefined
- **Ex:** asserting that *"Huey, Dewey and Louie are Donald Duck's nephews"* means that Donald Duck has no other nephews in CWA, while this is not due in OWA

Open World vs. Close World Assumption

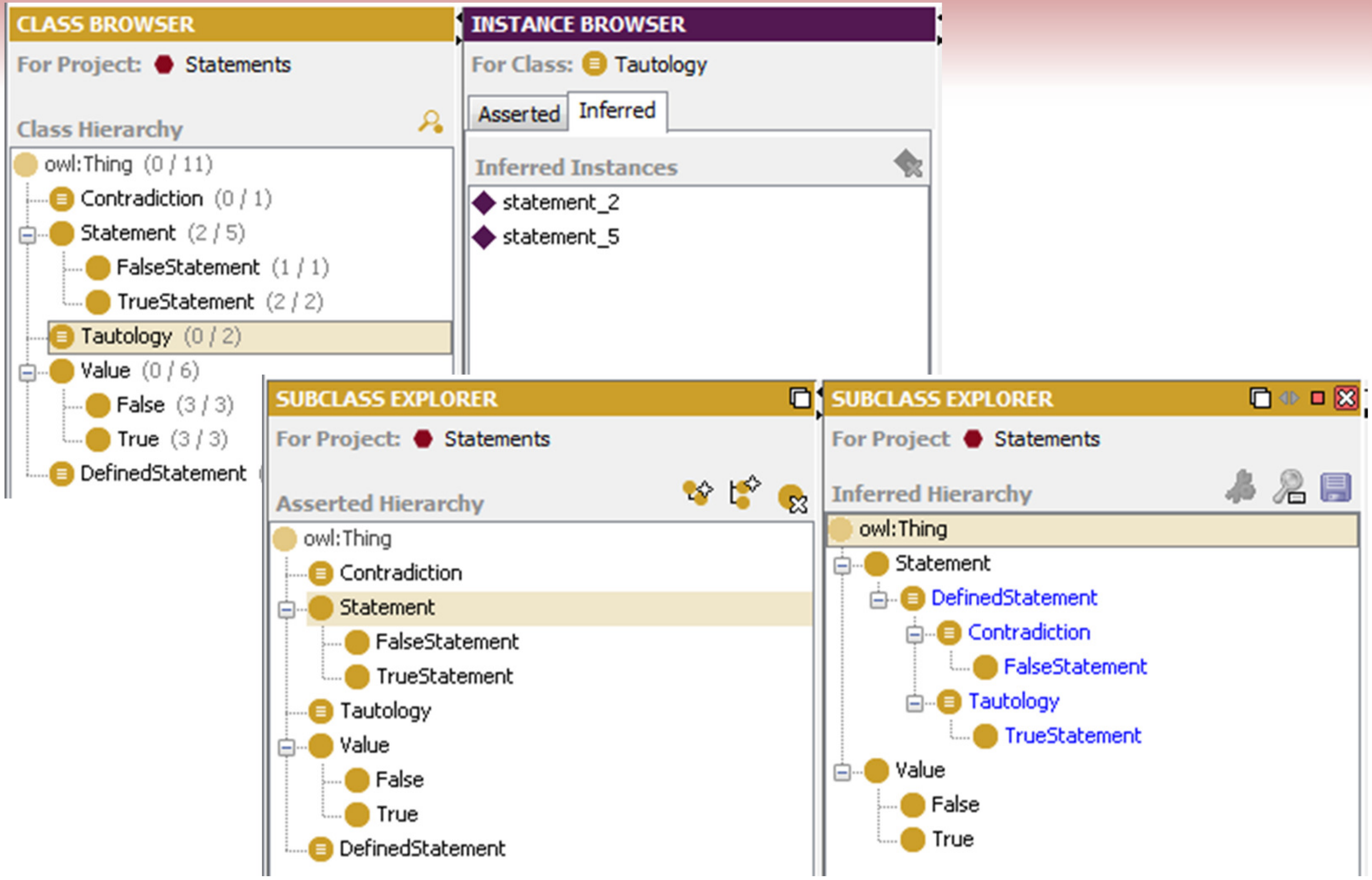
- Due to the Open World Assumption, every restriction involving the keywords ***only***, ***exactly*** and ***max*** cannot help inferring types for instances
- The reasoner, in fact, cannot perform such inference because it suppose the presence of information that may violate the restrictions even if it is not aware of it
- This limitation is overcome by introducing the so-called ***Closure Axioms*** involving
  - The definition of classes with the same necessary restrictions
  - The introduction of instances as members of such classes

Note: This method makes the model correct but some attempts of deducing things become quite useless…

Open World vs. Close World Assumption

Inferred types for instances are correctly recognized after the introduction of ***closure axioms***

Also note the final ***taxonomy*** for the model as classified by the reasoner


CLASS BROWSER
For Project: Statements
Class Hierarchy

```
owl:Thing (0 / 11)
    Contradiction (0 / 1)
    Statement (2 / 5)
        FalseStatement (1 / 1)
        TrueStatement (2 / 2)
    Tautology (0 / 2)
    Value (0 / 6)
        False (3 / 3)
        True (3 / 3)
    DefinedStatement
```

INSTANCE BROWSER
For Class: Tautology
Asserted | Inferred
Inferred Instances

```
statement_2
statement_5
```

SUBCLASS EXPLORER
For Project: Statements
Asserted Hierarchy

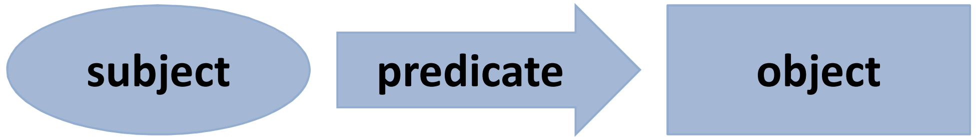
```
owl:Thing
    Contradiction
    Statement
        FalseStatement
        TrueStatement
    Tautology
    Value
        False
        True
    DefinedStatement
```

SUBCLASS EXPLORER
For Project: Statements
Inferred Hierarchy

```
owl:Thing
    Statement
        DefinedStatement
            Contradiction
                FalseStatement
            Tautology
                TrueStatement
    Value
        False
        True
```

RDF & SPARQL Queries

- Any *RDF graph* is a collection of triples:



- Each part of the triples is said **node**
- Nodes referring to classes, properties or instances are said **resources**
- Resources are represented with URIs, which can be abbreviated as **prefixed names**
- Objects can also be **literals** (strings, integers, booleans, etc.)

RDF & SPARQL Queries

Every SPARQL query comprises:

- **Prefix declaration** *to abbreviate URIs (opt.)*
- **Dataset definition** *to include RDFs (opt.)*
- **Result clause** *to state expected data*
- **Query pattern** *to specify context to query*
- **Solution modifiers** *to rearrange results (opt.)*

SPARQL query

RDF & SPARQL Queries

Every SPARQL query comprises:

```
# prefix declarations
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE {
     ...
}
# solution modifiers
ORDER BY ...
```

SPARQL query

RDF & SPARQL Queries

Ex.: Find the name of all the persons in current ontology

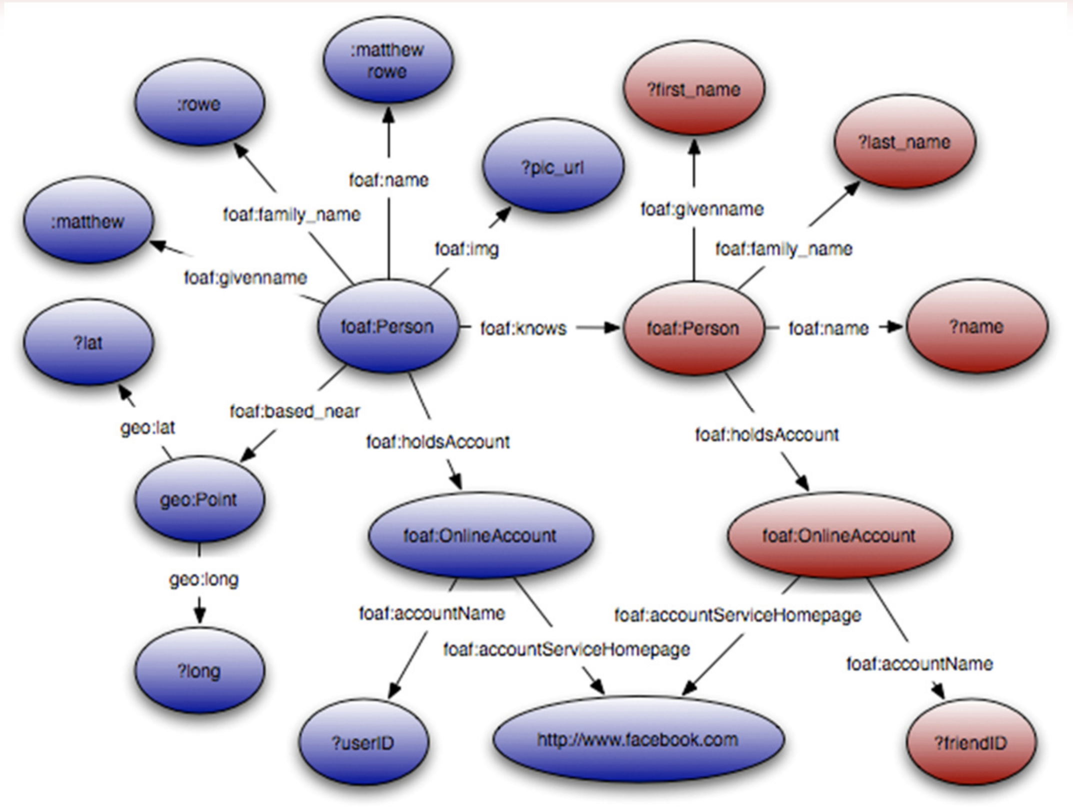
```
SELECT ?name
WHERE {
    ?person :hasName ?name .
}
```

- ***SPARQL variables*** start with a ***?*** and can match any node (resource or literal) in the RDF dataset
- ***Triple patterns*** are just like triples, except that its parts can be replaced with variables
- Queries may have several triple patterns, each ending with ***.***
- **SELECT** returns a table of values that satisfy the query

RDF & SPARQL Queries

FOAF Ontology

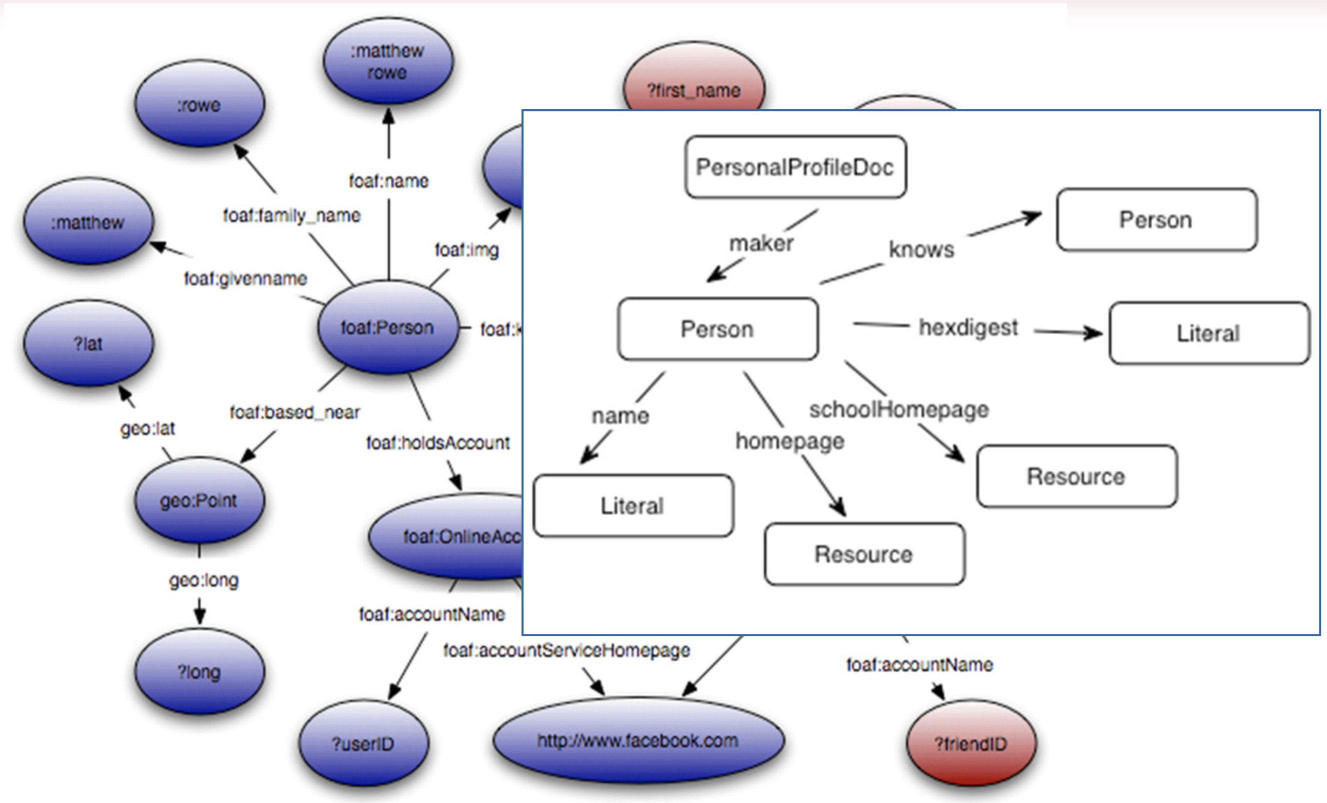
- Revolves around the class ***Person***
- Aggregates data around persons, in terms of both ***resources*** and ***literals***
- ***Associates persons***, each to another



RDF & SPARQL Queries

FOAF Ontology

- Revolves around the class ***Person***
- Aggregates data around persons, in terms of both ***resources*** and ***literals***
- ***Associates persons***, each to another



Prefix Declaration

Ex.: Find all the people that have name ***and*** email address using the external ***FOAF*** (Friend of a Friend) ontology

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT *
WHERE {
 ?person foaf:name ?name ;
         foaf:mbox ?email . # ?person foaf:mbox ?email .
}
```

- The ******* selects all the variables that are mentioned in the query
- The ***;*** allows two consecutive triple patterns to share the subject
- Multiple triple patterns with the same subject explore multiple properties about the resource ***at the same time***

Dataset Definition

Ex.: Find the homepage of anyone known by ***Tim Berners-Lee***

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX card: <http://www.w3.org/People/Berners-Lee/card#>
SELECT ?homepage
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
 card:i foaf:knows ?known .
 ?known foaf:homepage ?homepage .
}
```

- ***FROM*** allows to specify which dataset (instances) to consider
- Two consecutive triple patterns sharing a variable as object and subject respectively allows to ***traverse links*** in the graph

Result Clause

- This is the most important section of SPARQL queries together with the other non-optional section Query Pattern
- Several ways to customize it:
  - **Alternatives:** **`CONSTRUCT`**, **`ASK`**, **`DESCRIBE`**
  - **Modifiers:** **`DISTINCT`**
  - **Aggregate functions:** **`COUNT`**, **`MIN`**, **`MAX`**, **`SUM`**, etc.

RESULT CLAUSE: DISTINCT

Ex.: Find the name of all the persons with an email ***or*** an homepage

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name
WHERE {
 { ?person foaf:mbox ?mbox }
   UNION
 { ?person foaf:homepage ?homepage }
 ?person foaf:name ?name .
}
```

- If a person has both email and homepage, its name will appear twice: ***DISTINCT*** allows to remove duplicates

Result Clause: CONSTRUCT

Ex.: Convert all the ***FOAF*** data from Tim Berners-Lee ontology into ***VCard*** data

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT {
  ?X vCard:FN ?name .
  ?X vCard:URL ?url .
  ?X vCard:TITLE ?title .
}
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  OPTIONAL { ?X foaf:name ?name .    FILTER isLiteral(?name) . }
  OPTIONAL { ?X foaf:homepage ?url . FILTER isURI(?url) . }
  OPTIONAL { ?X foaf:title ?title .  FILTER isLiteral(?title) . }
}
```

- ***CONSTRUCT*** returns an RDF graph instead of a table of values
- The equivalent ***SELECT*** query is performed and the returned values are used to fill the template
- Query patterns involving ***unbound*** variables are ***discarded***

Result Clause: ASK

Ex.: Check out if Tim Berners-Lee knows me

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX card: <http://www.w3.org/People/Berners-Lee/card#>
ASK
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
 card:i foaf:knows :stefano .
}
```

- ***ASK*** returns true or false depending on the query pattern
- ***WHERE*** is always optional (***ASK*** queries usually drop it to ease the reading)

Result Clause: DESCRIBE

Ex.: Retrieve any piece of information about the Ford Motor Company

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DESCRIBE ?ford
WHERE {
  ?ford foaf:name "FORD MOTOR CO" .
}
```

- ***DESCRIBE*** returns all the triples involving the given resource(s)
- Returned ***information may vary*** depending on the implementation

Result Clause: Aggregate Functions

Ex.: What are the top interests of ***LiveJournal*** users interested in ***Harry Potter***?

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?interest COUNT(*) AS ?count
WHERE {
    ?person foaf:interest
     <http://www.livejournal.com/interests.bml?int=harry+potter> .
    ?person foaf:interest ?interest
}
GROUP BY ?interest ORDER BY DESC(COUNT(*)) LIMIT 10
```

- Only supported by some SPARQL implementations
- Aggregate functions calculate single values from sets of results
- They include: ***COUNT***, ***MIN***, ***MAX***, ***SUM***, etc.
- Sometimes used with clauses that ***breaks the results into groups before applying the aggregate function***(s)

Query Pattern

- As said, this non-optional section plays an important role in SPARQL queries together with Result Clause
- Several ways to customize it:
  - **Operators: `FILTER`, `OPTIONAL`, `UNION`**
  - **Named Graphs: GRAPH**
  - **Nested clauses: `SELECT`**, etc.

QUERY PATTERN: FILTER

Ex.: Find the name of all the person located ***only near*** a given geospatial point

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geo: <http://www.geonames.org/ontology/ontology_v2.0_Full.rdf>
SELECT ?name
WHERE {
    ?person foaf:name ?name .
    ?person foaf:based_near ?loc .
    ?loc geo:lat ?lat .
    ?loc geo:long ?long .
    FILTER ( ?lat > 50 && ?lat < 100 &&
             ?long > 25 && ?long < 75 ) .
}
```

- ***FILTER*** uses boolean contraints to ignore unwanted results
- It *may* lead to ***duplicates*** in results (how to handle that?)

Query Pattern: FILTER

- Many operators are allowed in a **`FILTER`** declarations:
  - **Logical:** `!`, `&&`, `||`
  - **Math:** `+`, `-`, `*`, `/`
  - **Comparison:** `=`, `!=`, `>`, `<`, etc.
  - **SPARQL tests:** *`isURI`*, *`isBlank`*, *`isLiteral`*, *`bound`*
  - **SPARQL accessors:** *`str`*, *`lang`*, *`datatype`*
  - **Other:** *`sameTerm`*, *`langMatches`*, *`regex`*

Query Pattern: OPTIONAL

Ex.: Find the name of all the persons along with their image, homepage and location

```
# The wrong way...
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?img ?home ?loc
WHERE {
    ?person foaf:name ?name ;
            foaf:img ?img .
            foaf:homepage ?home .
            foaf:based_near ?loc .
}
```

- This query does not return all the expected values: what is wrong?
- The pattern above requires ***every*** piece of data to be available…

QUERY PATTERN: OPTIONAL

Ex.: Find the name of all the persons along with their image, homepage and location

```
# The right way...
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?img ?home ?loc
WHERE {
    ?person foaf:name ?name ;
    OPTIONAL { ?person foaf:img ?img }
    OPTIONAL { ?person foaf:homepage ?home }
    OPTIONAL { ?person foaf:based_near ?loc }
}
```

- Not everyone has an image, homepage or location
- OPTIONAL tries to match the pattern, but does not fail the whole query in the specific match fails
- In case of failure, any unassigned variable in the pattern remain unbound

Query Pattern: UNION

Ex.: Find the name of all the persons with an email ***or*** an homepage

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name
WHERE {
 { ?person foaf:mbox ?mbox }
   UNION
 { ?person foaf:homepage ?homepage }
 ?person foaf:name ?name .
}
```

- ***UNION*** forms a disjunction of two graph patterns
- Solution to both sides are ***included*** in the results

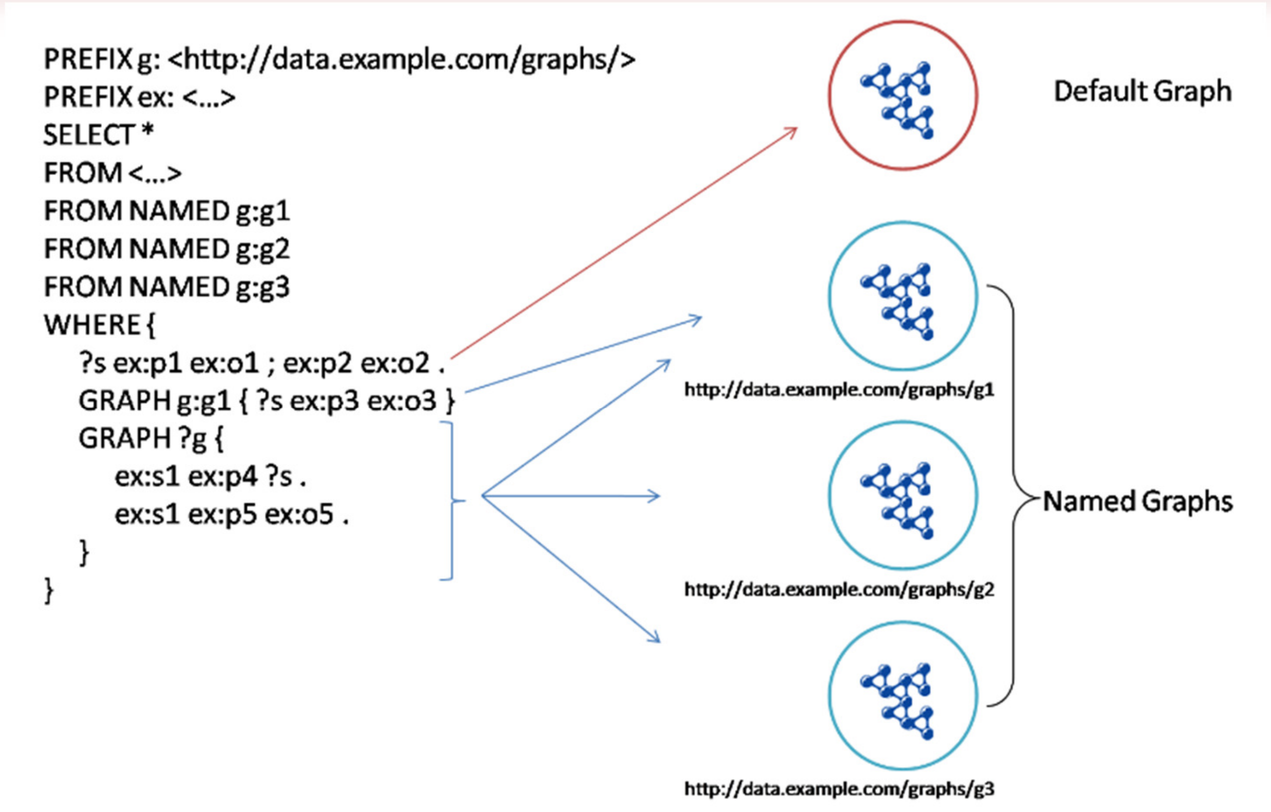
Query Pattern: GRAPH

Ex.: Find the name of all the people involved in at least three distinct events

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX g: <http://data.example.com/graphs/>
SELECT DISTINCT ?name
FROM NAMED g:g1
FROM NAMED g:g2
FROM NAMED g:g3
WHERE {
    ?person foaf:name ?name .
    GRAPH ?g1 { ?person a foaf:Person }
    GRAPH ?g2 { ?person a foaf:Person }
    GRAPH ?g3 { ?person a foaf:Person }
    FILTER ( ?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3 ) .
}
```

- So far queries have been again a single graph (***default graph***)
- Additional graphs may be added with ***FROM NAMED*** clauses
- ***GRAPH*** then allows portions of the query to match against the named graphs
- ***a*** is a shortcut for the `rdf:type` predicate

Query Pattern: GRAPH



```
PREFIX g: <http://data.example.com/graphs/>
PREFIX ex: <...>
SELECT *
FROM <...>
FROM NAMED g:g1
FROM NAMED g:g2
FROM NAMED g:g3
WHERE {
    ?s ex:p1 ex:o1 ; ex:p2 ex:o2 .
    GRAPH g:g1 { ?s ex:p3 ex:o3 }
    GRAPH ?g {
        ex:s1 ex:p4 ?s .
        ex:s1 ex:p5 ex:o5 .
    }
}
```

Default Graph
http://data.example.com/graphs/g1
http://data.example.com/graphs/g2
http://data.example.com/graphs/g3
Named Graphs

Query Pattern: Nested Clauses

Ex.: Retrieve the ***second page*** of names and emails of people in Tim Berners-Lee's FOAF file, given that ***each page has 10 people***

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  { SELECT DISTINCT ?person ?name
    WHERE {
      ?person foaf:name ?name
    } ORDER BY ?name LIMIT 10 OFFSET 10
  }
  OPTIONAL { ?person foaf:mbox ?email }
}
```

- The sub-query ***limits the amount of person to consider in advance*** instead of pruning the returned values
- Not available on all the implementations

Solution Modifiers

- Solution modifiers affect the values returned by the query in several ways
- Two class of solution modifiers are available:
  - **ORDER BY** is used to ***sort the solution*** on values of one or more variables
  - **`OFFSET`** and **`LIMIT`** are used to ***take a slice of the solution***
- Specific examples have been introduced in previous sections

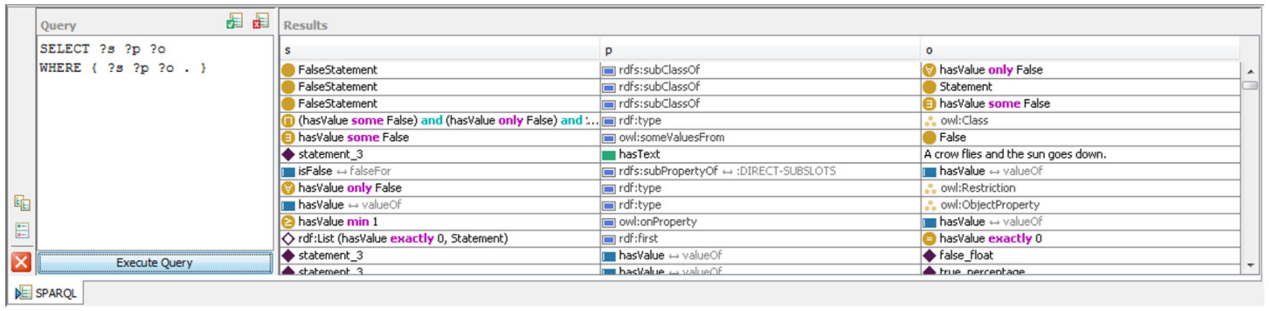
Future Directions for SPARQL

- Unfortunately, a few pieces of SPARQL are not yet standard: *full-text search, parameters passing, querying more graphs at a time, direct support of XML*
- A new version of SPARQL was chartered in March 2009; planned features are: *insert/update/delete capabilities, negation, improvements on aggregate functions, query with constants/functions/expressions, standard for sub-query*
- The following features are also being discussed: *interaction with OWL/RDF-S/RIF, recursive property paths, basic federated query*

SPARQL SUCCEEDS WHERE OWL FAILS?

- SPARQL is complementary to OWL and is based on Closed World Assumption
- Thus it can be used to overcome the limitations seen before

Note: ***SPARQL Query panel*** can be accessed under "Reasoning" menu with "Open SPARQL Query panel" command:



Query

```
SELECT ?s ?p ?o
WHERE { ?s ?p ?o . }
```

Execute Query

Results

s	p	o
FalseStatement	rdfs:subClassOf	hasValue only False
FalseStatement	rdfs:subClassOf	Statement
FalseStatement	rdfs:subClassOf	hasValue some False
(hasValue some False) and (hasValue only False) and :...	rdf:type	owl:Class
hasValue some False	owl:someValuesFrom	False
statement_3	hasText	A crow flies and the sun goes down.
isFalse ↔ falseFor	rdfs:subPropertyOf ↔ :DIRECT-SUBSLOTS	hasValue ↔ valueOf
hasValue only False	rdf:type	owl:Restriction
hasValue ↔ valueOf	rdf:type	owl:ObjectProperty
hasValue min 1	owl:onProperty	hasValue ↔ valueOf
rdf:List (hasValue exactly 0, Statement)	rdf:first	hasValue exactly 0
statement_3	hasValue ↔ valueOf	false_float
statement_3	hasValue ↔ valueOf	true_percentage

SPARQL

SPARQL SUCCEDS WHERE OWL FAILS?

Ex.: Find the text of all the statements that are tautology

```
SELECT ?text
WHERE {
  # find statements that has value true
  ?statement a :Statement ;
             :hasValue True .
  OPTIONAL {
    # find out if it also has value false, but use a different name
    ?statement2 a :Statement ;
                :hasValue False .
    FILTER ( ?statement2 = ?statement ) .
  }
  # keep only the statements that failed to have value false
  FILTER ( !bound( ?statemnt2 ) ) .
  ?statement :hasText ?text .
}
```

- Together, ***OPTIONAL*** and the ***!bound(...)*** allow to query for things that are ***not asserted*** in the dataset
- A similar technique allows for other types of universally quantified queries (such as max)

Exercise 0

1. Download, extract and open the pizza ontology in the Protégé editor
2. Add a new class, the “ChesaniPizza”.
   - It must have some cheese
   - It must have a toping based on meat
   - It must be an “interesting pizza”
   - It must be vegetarian
3. Check the consistency of this pizza… and solve possible problems!
4. Then try to transform it into a definition
   - Which pizza are subsumed by ChesaniPizza?

EXERCISE 0

INDIVIDUALS

1. Create an individual, e.g., “pizzaTonight”
2. Add some topping:
   - Mozz_bufala
   - Luganega (o lucaneca)
   - Pomodoro_di_pachino
   - Melanzane
3. Add these toppings to pizzaTonight (create the needed individuals…)
4. Try to classify it: to which class this pizza belong? Why?

EXERCISE 0

DOMAIN/RANGE ASSERTIONS

1. Create the “Cake” class, as subclass of Food
2. Specify that Cake “hasTopping some FruitTopping”
3. Invoke the reasoner… where it is classified? Why?
4. Specify “Cake disjoint with Pizza”, and re-invoke the reasoner

Exercise 0

SPARQL

1. Create topping “Mortadella”
2. Create instance m1 of Mortadella
3. Create two instances p1 and p2 of pizza
4. Specify “p1 hasTopping m1”
5. Check the following queries:

Exercise 0

SPARQL

1. All the instances of Pizza

```
SELECT ?p
WHERE { ?p rdf:type :Pizza . }
```

2. Find the pizza that has m1 as topping

```
SELECT ?p
WHERE {
 ?p :hasTopping :m1 .
}
```

3. Find all the pizza that has some mortadella…

```
SELECT ?p
WHERE {
 ?p :hasTopping ?t .
 ?t rdf:type  :Mortadella .
}
```

Exercise 1

Ex.: Design an ontology containing statements representing the following sentences:

- A doll is a kind of toy
- Young women are defined as young persons that are also female
- A young person cannot be both a young man and a young woman
- Young persons are either young men or young women
- Lenore and Emily are famous dolls
- All young women play with some doll
- Young women play only with famous dolls
- Young men play with at least one toy

Exercise 1

Ex.: Design an ontology containing statements representing the following sentences:

- Clara is a young woman
- Clara and Laura are different individuals
- Lalu is the same person as Laura
- To dress a toy is a special case of playing, where the toy is a doll
- To be dressed by a young person is the inverse relation of a young person dressing a doll
- To be ancestor of a person is a transitive relation that holds between persons

Exercise 2

Context
An online music database wishes to semantically represent their data about musicians, albums, and performances, in order to be able to provide better search functions to their users, i.e. by querying the knowledge base instead of using keyword queries. Below is an example of what they typically would like to store, and at the bottom you find the competency questions developed as requirements for the ontology.

Story: music and bands
The current configuration of the "Red Hot Chili Peppers" are: Anthony Kiedis (vocals), Flea (bass, trumpet, keyboards, and vocals), John Frusciante (guitar), and Chad Smith (drums). The line-up has changed a few times during they years, Frusciante replaced Hillel Slovak in 1988, and when Jack Irons left the band he was briefly replaced by D.H. Peligro until the band found Chad Smith. In addition to playing guitars for Red hot Chili Peppers Frusciante also contributed to the band "The Mars Volta" as a vocalist for some time.
From September 2004, the Red Hot Chili Peppers started recording the album "Stadium Arcadium". The album contains 28 tracks and was released on May 5 2006. It includes a track of the song "Hump de Bump", which was composed in January 26, 2004. The critic Crian Hiatt defined the album as "the most ambitious work in his twenty-three-year career". On August 11 (2006) the band gave a live performance in Portland, Oregon (US), featuring songs from Stadium Arcadium and other albums.

Exercise 2

Competency questions (CQs) and contextual statements of music and bands

1. What instruments does a certain person play?
2. What are the members of a certain band during a certain time period?
3. What role does a certain person have in a certain band during a certain time?
4. During what time period was a certain album recorded?
5. How many tracks does a particular album contain?
6. When was a certain album released?
7. What song is a specific track a recording of?
8. When was a certain song composed?
9. What does a certain critic say about a certain album?
10. When did a certain performance take place?
11. What songs were played in a certain performance?
12. Where did a certain performance take place?
13. In what region is a certain city located?
14. In what country is a certain region located?

Contextual statement:

- An album always contains at least one track.

References

- W3C:
  - http://www.w3.org/
- Protégé:
  - http://protege.stanford.edu/
- OWL:
  - http://www.w3.org/TR/owl-features/
  - http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/
- SPARQL:
  - http://www.w3.org/TR/rdf-sparql-query/
  - http://jena.sourceforge.net/ARQ/Tutorial/

Contacts

- If you have any questions, please send me an email or call me:

E-mail:	stefano.bragaglia@unibo.it
Tel:	051-20.93086