Semantic Web: from theory to reality

Natalia Villanueva-Rosales
School of Computer Science
Carleton University
nvillanu@scs.carleton.ca

Motivation

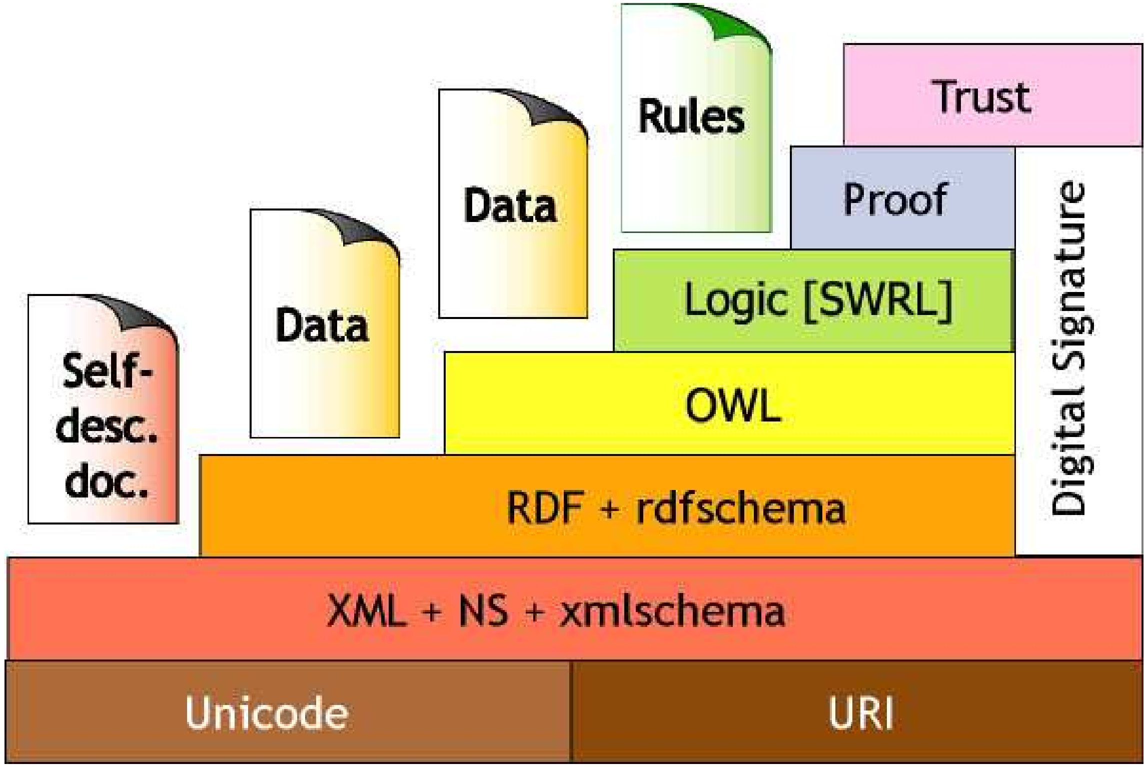
- Huge amount of information in web resources and distributed information systems.
- Information can have different formats (i.e. text, databases, documents) and platforms (i.e. operative systems, programming languages).
- Information is syntactically manipulated so far.
- Challenge: access and integration of such information.

Semantic web

- Extension of the current www
- Idea: To publish and query machine understandable information using semantics
- Towards the use of expressive and appropriate (Standard Markup) languages for in such a way that the document carries its information and meta-information: the data and its meaning ...
- That additional semantics can be used when data is published and queried on the web
- Iterchange of data (not only documents) on the web
- Project guided by the W3C.

Semantic web (2)

Semantic Web Languages so far:[a]



[a]Modified from Semantic Web talk by Tim Berners-Lee at XML 2000

XML Document Syntax

- XML documents contain elements .
- Elements must have an opening and closing tag.
- Elements can have child elements (ordered) and attributes (not ordered).
- Elements must be properly nested and can be extensible .

Example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<message dateSent="03/01/06">
    <sender>Jack<sender>
    <recipient>Mary</recipient>
    <content>What is all this talk about?</content>
</message>
```

XML Syntax (2)

Example 2:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<message dateSent="03/01/06">
    <title>Question</title>
    <sender>
        <title>Mr<title> <name>Jack</name>
    </sender>
    <recipient>
        <title>Miss</title> <name>Julie</name>
    </recipient>
    <content>What is all this talk about?</content>
</message>
```

- An XML document can contain elements from different sources, as it is widely used for data exchange and data integration.
- Problem: Recognition and collision of element names. In example 2: The first ”title” refers to the message title, the second and the third to the sender and recipient title respectively.
- Solution: namespaces .

XML NameSpaces

Example 2 (modified):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<message xmlns:person="www.example.com/Person" dateSent="03/01/06">
    <title>Question</title>
    <sender>
        <person:title>Mr</person:title> <person:name>Jack</person:name>
    </sender>
    <recipient>
        <person:title>Miss</person:title> <person:name>Julie</person:name>
    </recipient>
    <content>What is all this talk about?</content>
</message>
```

- The element name title is now distinguished using a prefix (i.e. person:title and title represent different elements).
- The namespace attribute is placed in the start tag of an element (line 2).
- Qualified names have scopes beyond their containing document.
- This is a way to group element and attribute definitions from different contexts and reuse them unambiguously in the same document.

RDF ”Resource Description Framework”

RDF allows to create models of data in terms of resources and relations between them, called *statements* .

It is possible to give a simple semantics to those data models, without making assumptions about the structure (as XML).

Basic elements in RDF are:

- *Resources*
  - Anything represented by an URI (a person, a painting, an e-store).
  - Nodes in a graph representation.
  - Example:
    `http://www.dumontierlab.com/myFirstOntology.owl/Joe`

RDF Elements

- *Properties*
  - Also represented by an URI.
  - Edges in a graph representation.
  - Binary relations between two resources.
  - Example:
    `http://www.dumontierlab.com/myFirstOntology.owl/hasPet`
- *Literals*
  - Concrete data values.
  - Nodes in a graph representation.
  - Can use XML Schema data types (but checking data types is not done in RDF).
  - Example: `"2"`, `"blue"`, `"10-12-2006"`

RDF Syntax

RDF databases can be seen as graph databases , whose building blocks are triples of the form

Here *Joe* is the subject , *hasPet* is the property , and *Benji* is the object
It is also represented as a real *subject/property/object*- triple:

$$(Joe, hasPet, Benji)$$

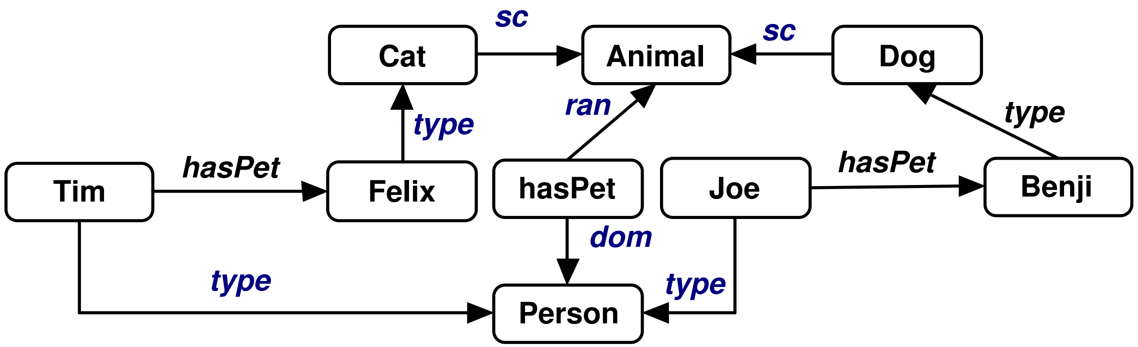
RDF Syntax (2)

The RDF/XML sytax is the most used in applications.

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
xmlns="http://www.dumontierlab.com/myFirstOntology.owl#"
xmlns:base="http://www.dumontierlab.com/myFirstOntology.owl">
<rdf:Description about="Joe">
    <hasPet rdf:resource="#Benji"/>
</rdf:Description>
</rdf:RDF>
```

It is possible to use namespaces and XML Schema data types.
Containers, collections and bags are also defined in RDF.

RDFS "RDF Schema"



RDF Graph

- Adds the notion of classes to RDF (and instances of classes).
- Allows hierarchies of classes being a subclass of and properties being a subproperty of
- Triples can be collected together into a graph-like database, that now (with RDFS) include "semantic" links as indicated above
- Some of the links have a fixed semantics, e.g. *sc* (for subclass), *dom* (for domain), *ran* (for range), ...

RDFS "RDF Schema" (2)

- Properties of a class are inherited by instances that belong (via *dom*) to a subclass.
- RDFS is recognized as a language for ontologies. However, RDFS is not the right foundation for the Semantic Web yet
- It is still too weak to describe resources in sufficient detail. From the point of view of ontologies, we miss some useful constructs to build new concepts (classes)
- The semantics is still rather basic
- A more expressive language is needed: Web Language Ontology (OWL).

OWL

- The first level above RDF required for the Semantic Web is an ontology language what can formally describe the meaning of terminology used in Web resources (and real world) :
  Web Ontology Language

- OWL provides greater machine interpretability of Web content than the one supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics for describing ontologies

- Formally specified and “adequate” expressive power (complexity vs. expressivity) with “support” for automated reasoning

OWL (2)

- An ontology is a “(shared) specification of a (knowledge) conceptualization”[a]
- An ontology contains statements like *Pet Owner is a subclass of people that have some pet*
- Has the same interpretation of some RDF statements (i.e. subsumption, domain and range) with a richer set of primitives, e.g. relations between classes (e.g. disjointness: *a cat is not a rabbit*), cardinality (e.g. exactly one: *a one pet owner is a person that has exactly one pet*), richer typing of properties, characteristics of properties (e.g. symmetry, transitivity), equality, and enumerated classes
- The namespace `http://www.w3.org/2002/07/owl` contains basic OWL vocabulary.

[a] Gruber, 1993

Three sublanguages of OWL

- Lite Supports classification hierarchy and simple constraints. (It supports cardinality constraints with 0 and 1)
- DL Based on Description Logics, decidable fragment of First Order Logic (FOL) (allow us to use reasoners). Maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions
- Full

  Maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

OWL RDF/XML Syntax

Define the class OnePetOwner , as a subclass of Person and equivalent to the class of individuals that have only one pet.

Assume the prefix `owl` for the owl namespace, `&xsd` for XML-Schema entity

```
<owl:Class rdf:ID="OnePetOwner">
        <rdfs:subClassOf rdf:resource="#Person"/>
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasPet"/>
                <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinalit
            </owl:Restriction>
        </owl:equivalentClass>
    </owl:Class>
```

OWL Tools

- Manage (i.e. create new, save, edit) ontologies.
- To use reasoners.
- Protégé , SWOOP, TopQuadrant (commercial).

An OWL-DL example

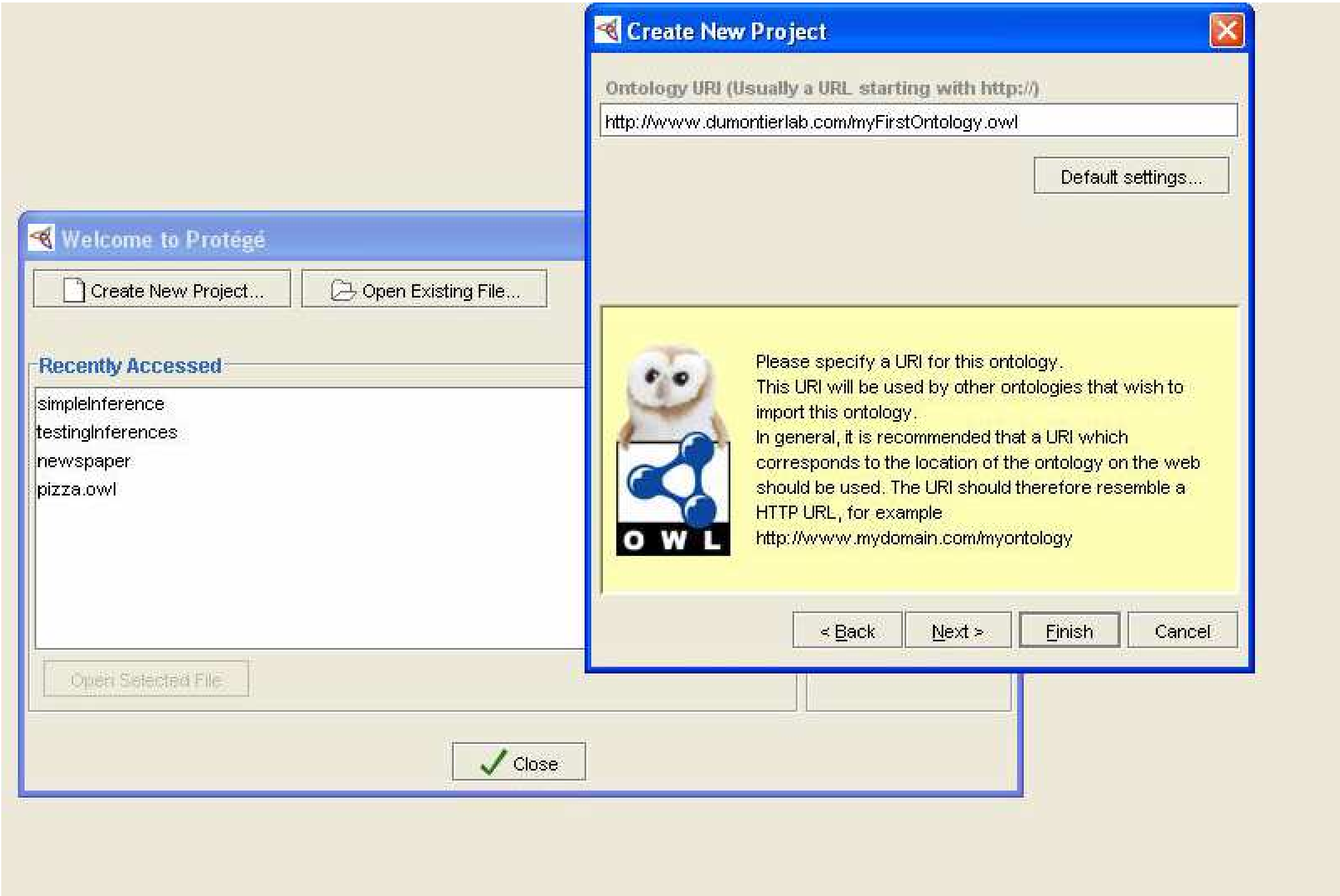
We will create an ontology to describe people and their pets. A person can have an animal pet. An animal can be a dog or a cat or a rabbit, but not a dog and a cat, nor a car and a rabbit. We want to group people depending on the number of pets they have : OnePetOwner, ManyPetsOwner. We also want to group people depending on the type of pet they have: PetOwner, DogOwner, CatOwner,RabitOwner,

Protégé

- Created in Stanford University, initially to handle “Frames” data models.
- A free, open source java plugin
- One of the most used tools for creating ontologies
- Friendly GUI

Protégé - Create New Project

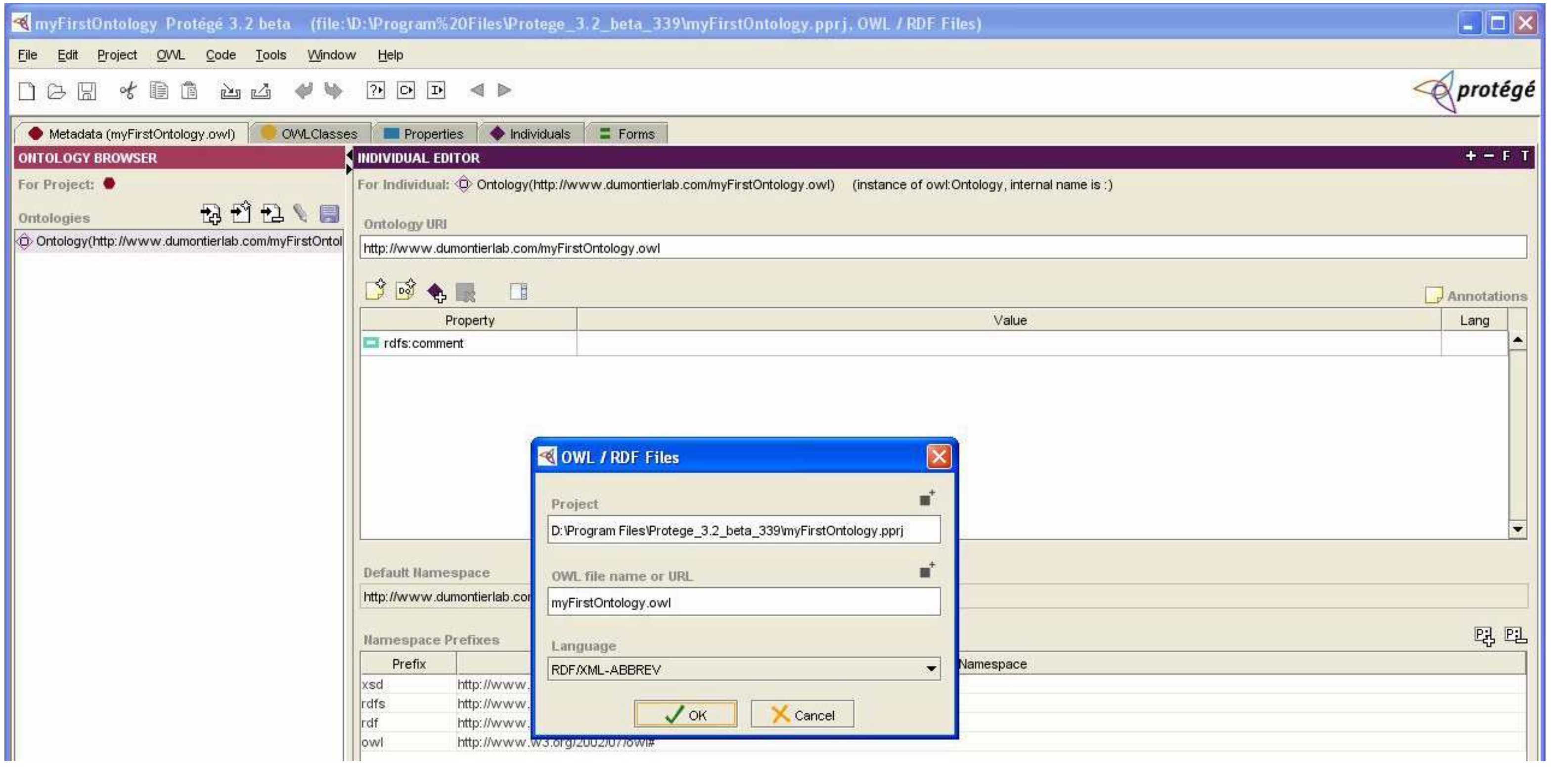
- Run Protégé
- Define a URI to identify our ontology (normally a URL starting with http://) In our example: `http://www.dumontierlab.com/myFirstOntology.owl`
- Our language profile will be OWL-DL
- Choose LogicView



Create New Project
Ontology URI (Usually a URL starting with http://)
http://www.dumontierlab.com/myFirstOntology.owl
Default settings...
Please specify a URI for this ontology.
This URI will be used by other ontologies that wish to import this ontology.
In general, it is recommended that a URI which corresponds to the location of the ontology on the web should be used. The URI should therefore resemble a HTTP URL, for example
http://www.mydomain.com/myontology
< Back
Next >
Finish
Cancel
Welcome to Protégé
Create New Project...
Open Existing File...
Recently Accessed
simpleInference
testingInferences
newspaper
pizza.owl
Open Selected File
Close

Protégé - Saving Projects

File → Save Project → myFirstOntology.pprj Note that myFirstOntology.owl is also created.



Protégé - OWL Tabs

To select the visible tabs: Project → Configure → TabWidget We will be working with:

- OWLClasses
- Properties
- Individuals
- QueriesTab

Note: To select a tab just check the corresponding box

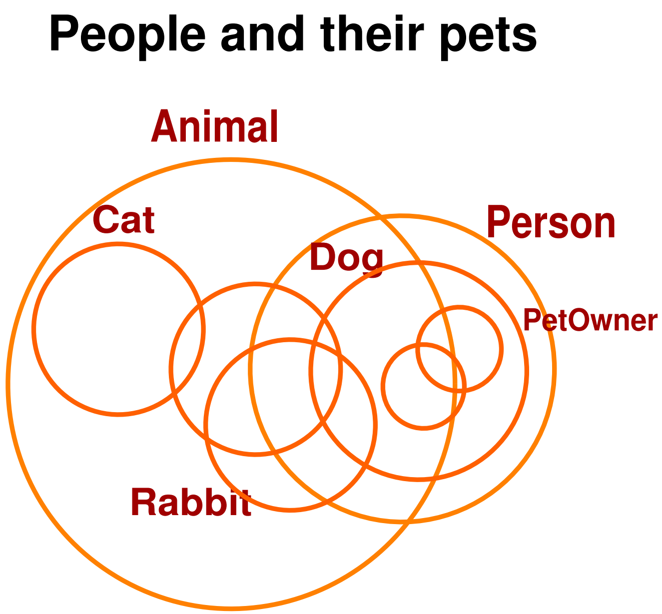
OWL Clases

- E.g. Person, Animal, Cat, Rabbit
- A class is a set of individuals.
- Membership of a class depends on its logical description
- Classes can have a name or be anonymous e.g. *People that have some pet*
- Also called "Type", "Concept", "Category" (in TBox).
- In protégé, we will create our class hierarchy based on our domain in the OWL Classes Tab.
- Logical Subsumption, owl:Thing is the root class.
- In our example, create the class hierarchy shown in the next slide.

Protégé - Class Hierarchy

Protégé - Class Hierarchy

Disjointness of classes is not assumed, so far we have [a]:

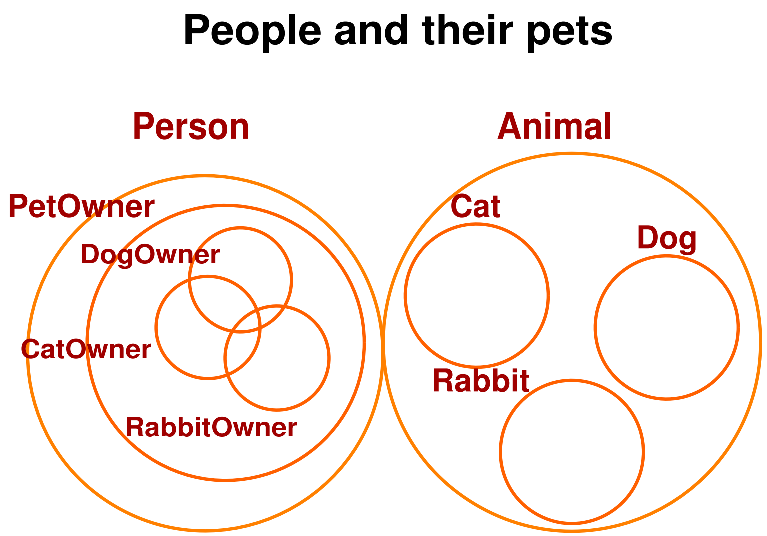


[a] Adapted from the Manchester Pizza Tutorial

Protégé - Disjoint Axioms

We need to add disjoint axioms (i.e. explicitly specify which classes are disjoint). E.g. The class Person is disjoint with the class Animal (i.e. no individual can be Person and Animal). For consistency checking, a reasoner is needed.

After adding the disjoint axioms (in the disjoint section of OWLClasses Tab) we have[a]:

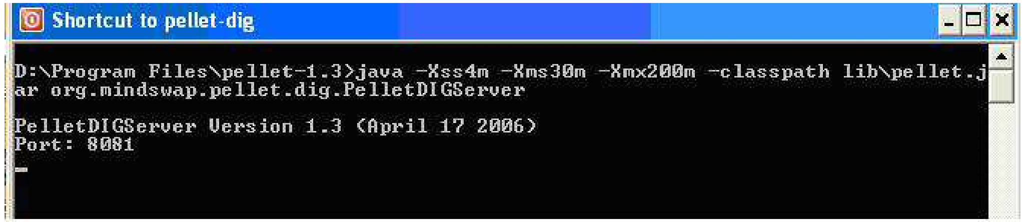


[a] Adapted from the Manchester Pizza Tutorial

Reasoners

- Reasoners are used to "infer" implicit information
- Programs like protégé communicate with the reasoner at runtime using DIG interface.
- Users normally offer:
  - Consistency checking.
  - Subsumption checking.
  - Equivalence checking.
  - Realization (Finds the most specific class that an instance belongs to).
- In practice, you should download a reasoner (e.g. Pellet), run it and set up the port in protégé to communicate with the reasoner: OWL→Preferences → Reasoner URL → `http://localhost:8081` (for Pellet, 8080 for Racer).

Reasoners

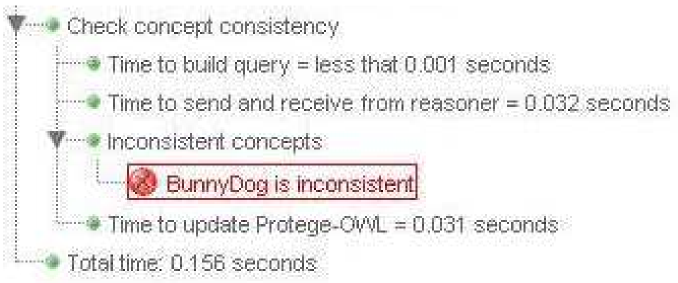


In protégé we can use the reasoner to:

- Check class consistency. Use `?>` button.
- Classify taxonomy. Use `C>` button
- Compute inferred types for all the instances. Use `I>` button. Note: Alternatively, you can right click on one individual and compute its types

Reasoners

Check consistency of myFirstOntology.
What if we add a BunnyDog (that is a subclass of Rabbit and Dog).
Why is inconsistent? Note: Inconsistent classes are red.



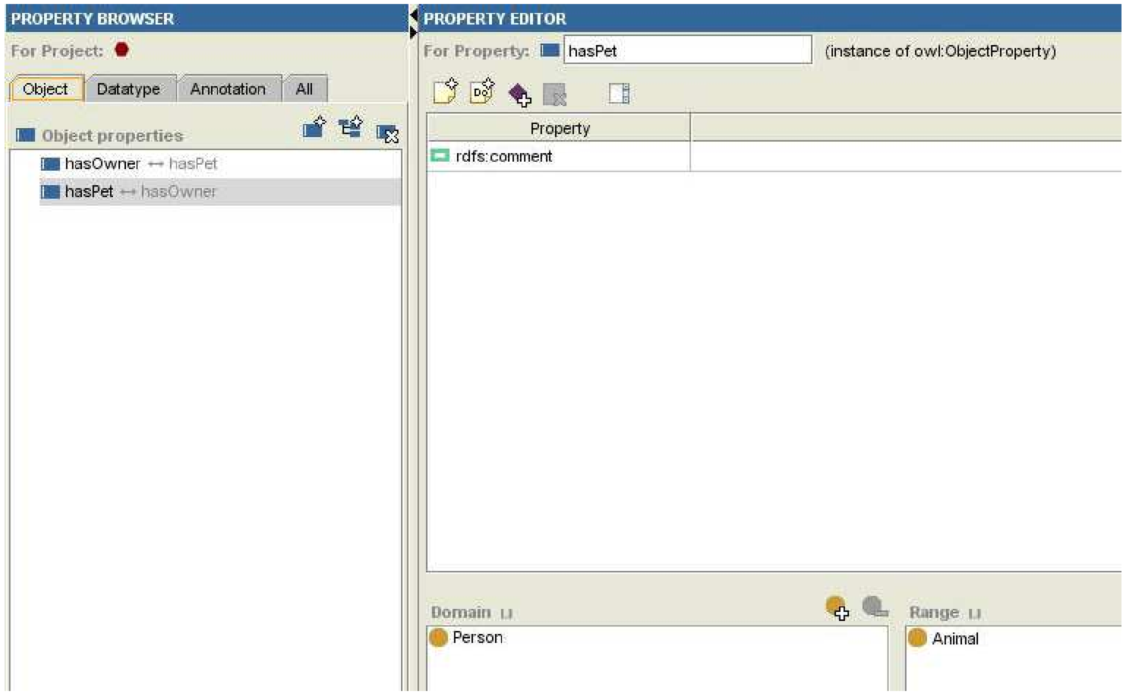
Properties

In protégé, the properties tab will allow us to add properties in our description.

- E.g. hasPet, hasOwner, wasBornIn
- Properties that relate individuals to individuas are called object properties . E.g. Joe hasPet Benji.
- Properties that relate individuals to data values are called datatype properties .E.g. Joe wasBornIn "12/10/1977".
- Aka "Roles" in DL
- A property can be (in the logical sense): functional, inverse, transitive, symmetric, inverse symmetric
- A property has a domain and range (just like predicates in RDF).

In our example we'll add property hasPet to relate Person with Animals with inverse hasOwner. *What does this mean?*

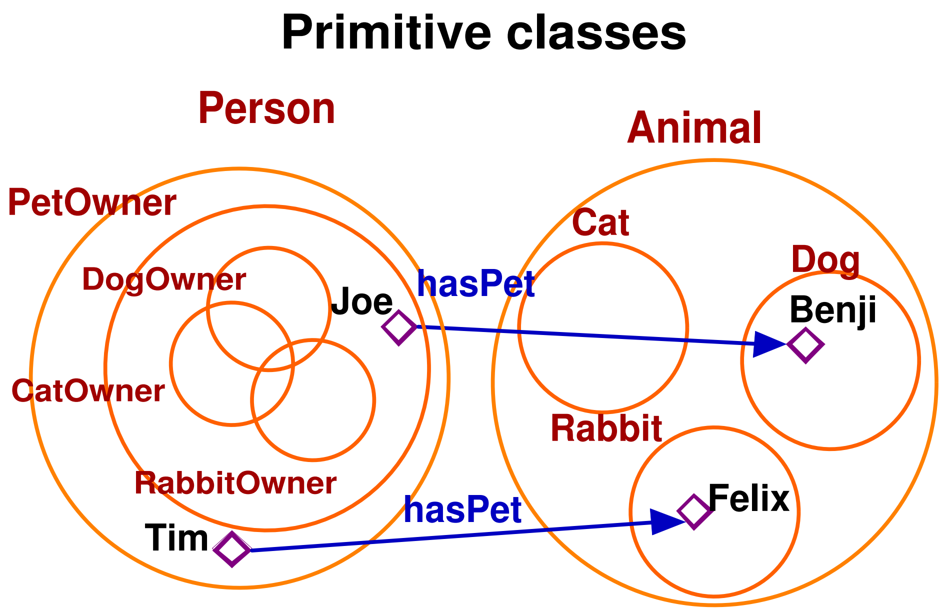
Properties


PROPERTY BROWSER
For Project:
Object
Datatype
Annotation
All
Object properties
hasOwner ↔ hasPet
hasPet ↔ hasOwner
PROPERTY EDITOR
For Property:
hasPet
(instance of owl:ObjectProperty)
Property
rdfs:comment
Domain
Person
Range
Animal

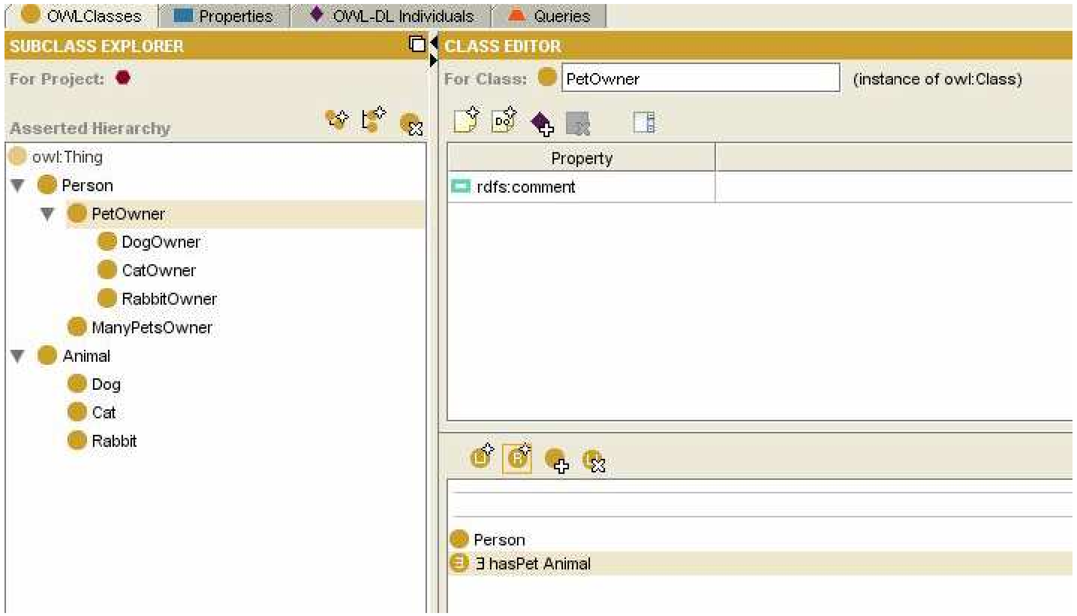
A better definition of classes

We can add logical descriptions of classes using property restrictions. For instance, it is necessary that all individuals of the class Pet Owner must have some pet.

Note: Properties are inherited

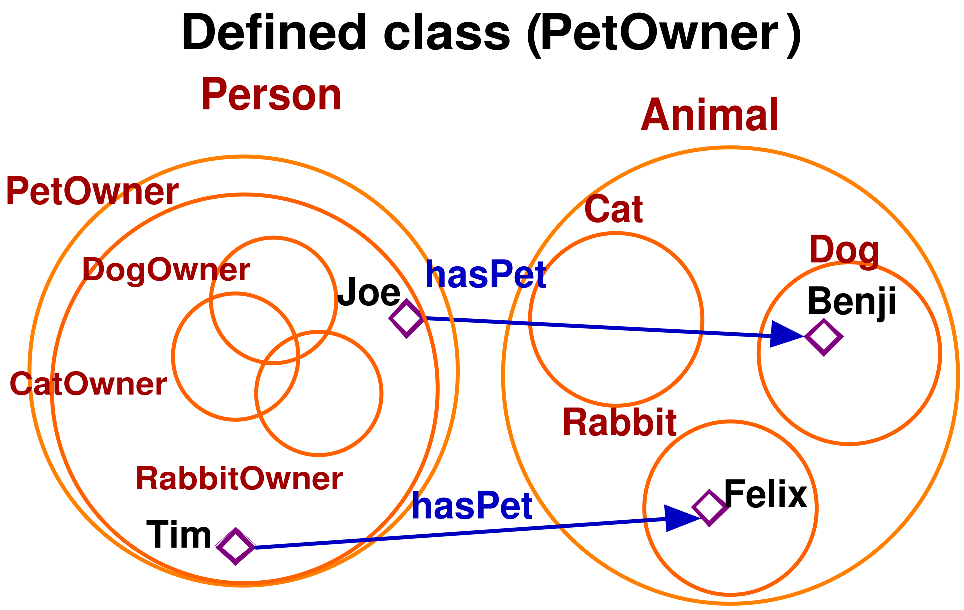


A better definition of classes



A better definition of classes

E.g. When defining the class using "necessary and sufficient" conditions we can say that all individual that has some pet is member of the class Pet Owner.



A better definition of classes

Define the classes:

- Dog Owner $\equiv$ Person that has some Dog as pet.
- Cat Owner $\equiv$ Person that has some Cat as pet.
- Rabbit Owner $\equiv$ Person that has some Rabbit as pet.
- Many Pets Owner $\equiv$ Person that has some more than 2 pets.

Note: ManyPetsOwner is now moved as subclass of PetOwner by the reasoner

OWL Individuals

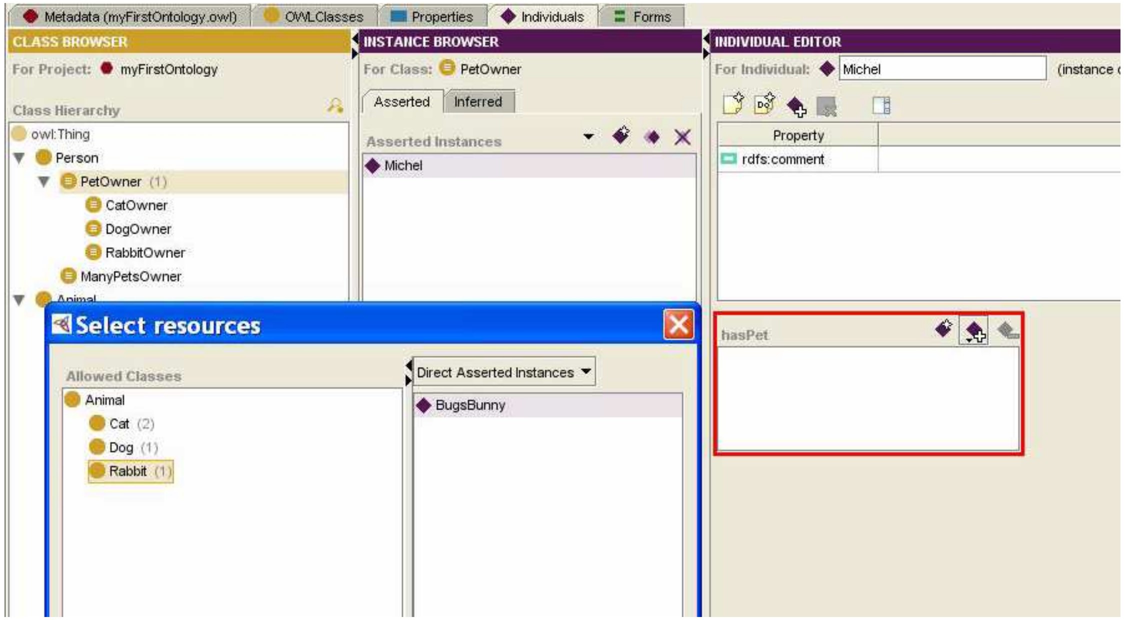
- E.g. Maria, Joe, Michel, Leo, you, me
- Aka “Objects”,”Instances”, (Abox in DL)
- Individuals can (and usually are) members of multiple classes.

OWL Individuals

Now we will add individuals (ABox in DL) to our ontology using the InstanceTab and also the following assertions:

- Joe, TimToBeInferredCatOwner are persons.
- Michel is a PetOwner, he has a rabbit pet called BugsBunny.
- Felix and HelloKitty are cats.
- Joe has HelloKitti and Benji as pets.
- TimToBeInferredCatOwner has Felix as pet.

Individuals



Note the necessary conditions

Reasoning about individuals

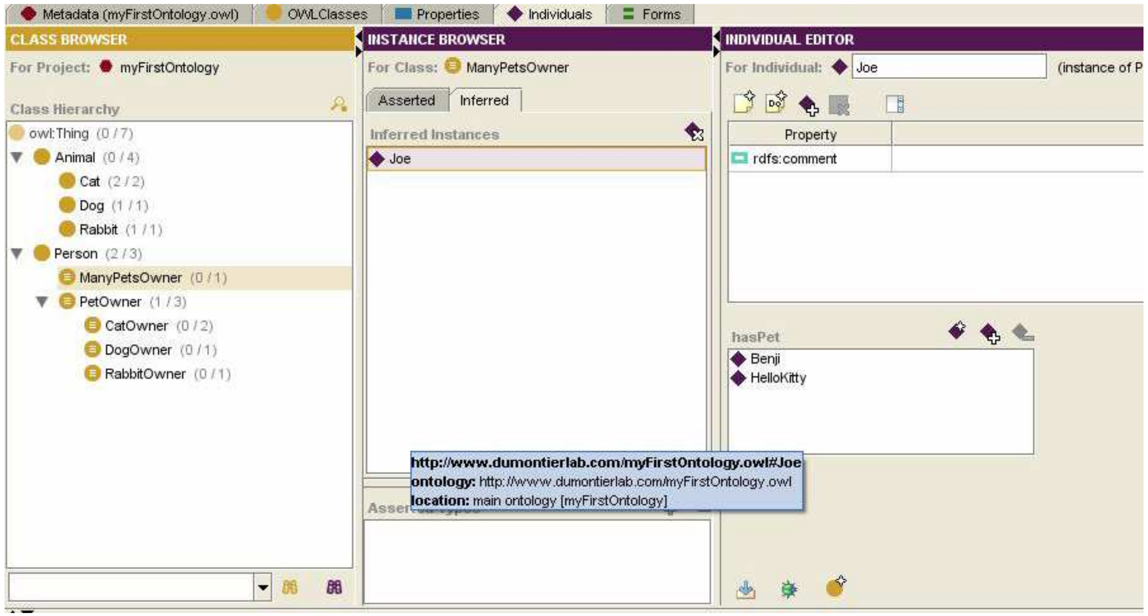
When "asking" the reasoner for inferred types of our instances, we have the following result:

- Persons: Joe, Michel, TimToBeInferredCatOwner
- PetOwner: Joe, Michel, TimToBeInferredCatOwner
- CatOwner: Joe, TimToBeInferredCatOwner
- DogOwner: Joe
- RabbitOwner: Michel
- ManyPetsOwner: Joe
- Felix and HelloKitty are cats.
- Joe has HelloKitti and Benji as pets.
- TimToBeInferredCatOwner has Felix as pet.

Not Unique Name Assumption

Asserting that Michel also owns Cotton (a rabbit), the inference expected is: Michel is a member of ManyPetsOwner.

Result:

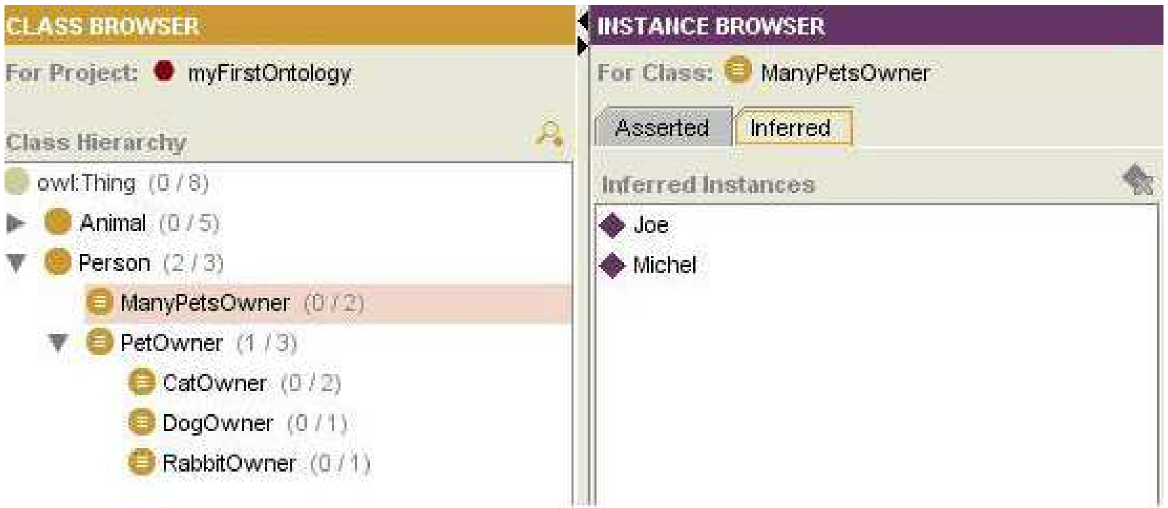


Not Unique Name Assumption (2)

Why?

OWL does not have the **U**nique **N**ame **A**ssumption (i.e. different names represent different individuals in the real world.

Solution: Using the construct owl:allDifferent to explicitly specify which individuals are diferent, doing this, we have the expected result.



Open World Assumption

OWL has the Open World Assumption ,i.e. we can not *assume* that something we don't know (we can not prove) is *false*.
E.g. Add the class OnePetOwner: people with only one pet.
The expected inference is that TimToBeInferredCatOwner should be a member of OnePetOwner. *Is it?*
Why? Just because we do not know (can not prove) if Tim... has more than one pet, we can not *assume* that he doesn't (e.g. he has another pet in another source we are not aware of).
Solutions? Some "tricks".

Applying Semantic Web in Bioinformatics

Research under the supervision of:

Dr. Leopoldo Bertossi, School of Computer Science
Dr. Michel Dumontier, Department of Biology

Motivation

Lack of efficient management of biological information. Nature of data:

- Huge amounts of biological data in the web,
- Updated everyday (what was true yesterday, might not be tomorrow)
- In different formats/platforms
- Many application dependent identifier for the same entity

Current applications are still limited to simple syntactic search with no intuitive GUI's and application dependent languages for semantic annotations.

Ongoing work

Goal: Represent, integrate, manage and query biological data in a transparent way using semantic web languages.
Interdisciplinary team.
From the Computer Science perspective: knowledge representation and knowledge engineering,implementation skills, problem solving, etc.
From the Biology perspective: domain expertise, modelling, methodology for domain knowledge discovery.

Ongoing work (2)

So far, we have extended biochemical ontologies with logical description for automated reasoning from the chemical and biochemical domain. Some of the ontologies extended include OBO ontologies, in particular Gene Ontology.

More information: www.dumontierlab.com

References

- Gruber, T.A.
  A translation approach to portable ontology specifications
  Knowl. Acquis., Vol. 5, No. 2. (June 1993), pp. 199-220

- OWL
  Web Ontology Language
  Available at http://www.w3.org/TR/2004/REC-owl-features-20040210/

- Bechhofer S., Horrocks I., Patel-Schneider P.F.
  OWL Tutorial
  Available at
  http://www.cs.man.ac.uk/ horrocks/ISWC2003/Tutorial/examples.pdf

- Berners-Lee T.
  Pellet OWL Reasoner
  Available at http://www.mindswap.org/2003/pellet/

References(2)

- Protegé
  Ontology Editor and Knowledge Base Framework
  Available at http://protege.stanford.edu/
- *Semantic Web on XML*
  XML 2000 Washington DC,
  Available at
  http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html
- *World Wide Web Consortium*
  http://www.w3.org.
- w3Schools Tutorials
  Web building tutorials
  http://www.w3schools.com

References(3)

- XML
  Extensible Markup Language
  http://www.w3.org/XML/
- Horridge M.,Knublauch H., Rector A., Stevens R., Wroe C.
  A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools
  Available at
  www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf