Semantic Web

or
Some Statements About Statements

netlabs.org Wintercamp 2008
Pascal Mainini

Get it: http://impressionet.ch/semwebspeech

Why this speech?

Goals?

<Person rdf:about=”http://mainini.ch/pascal”>

Pascal Mainini
pascal <at> impressionet <dot> ch
gurke <at> swissjabber <dot> org (Jabber)
Yes, I can join IRC too.

Open minded, critical hacker
Object-guy and XML evangelist
Software developer
Working as network and security specialist

Introduction

A bit of history...

Some initial work during 1997-1998

1999, February, first recommendation, RDF model and syntax – march: RDF schema proposal

February 2004: A suite of RDF and OWL recommendations, RDF schema recommendation.

Most widely used since for: RSS-feeds
(But not known for that...)

My first contact with it: 2003

Basics – Tripples / Statements

“Everything should be representable, so one needs a common model with great generality”

“Two basic elements:
Assertions
Quotations (statements about assertions)”

[http://www.w3.org/DesignIssues/Semantic.html]

Leads us to:
Everything is a statement, a tripple, or however you call it! - Everything is subject, predicate, object.

Basics – Linked Data

1. Use URIs as names for things

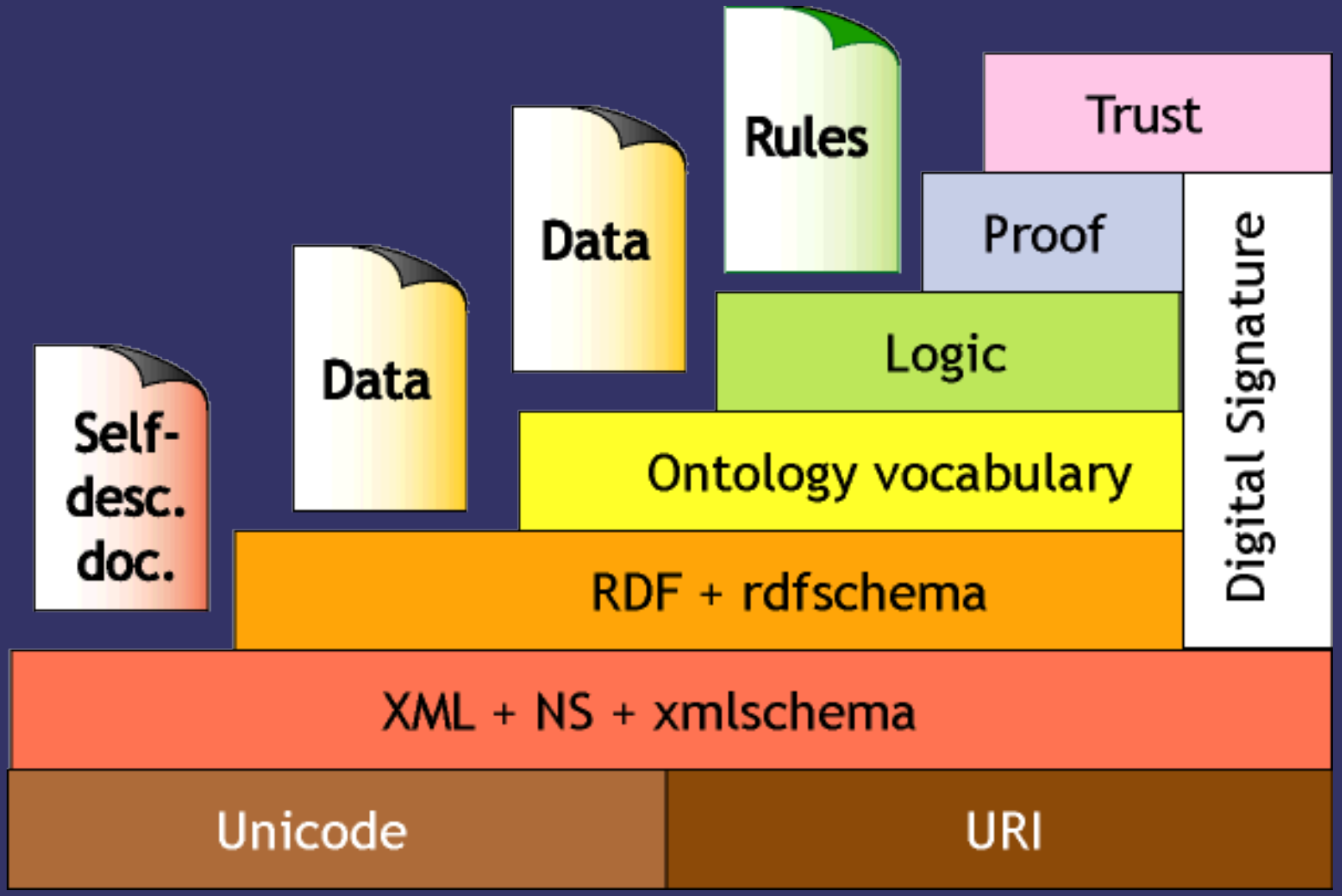
2. Use HTTP URIs so that people can lookup things

3. When someone looks up an URL, provide usefull information

4. Include links to other URIs, so that they can discover other things.

[http://www.w3.org/DesignIssues/LinkedData.html]

The layer cake



[www.w3.org]

RDF
The Ressource Description Framework

RDF - Overview

Pure statements / tripples
Only subject, predicate, object.

Can be expressed using various notations:

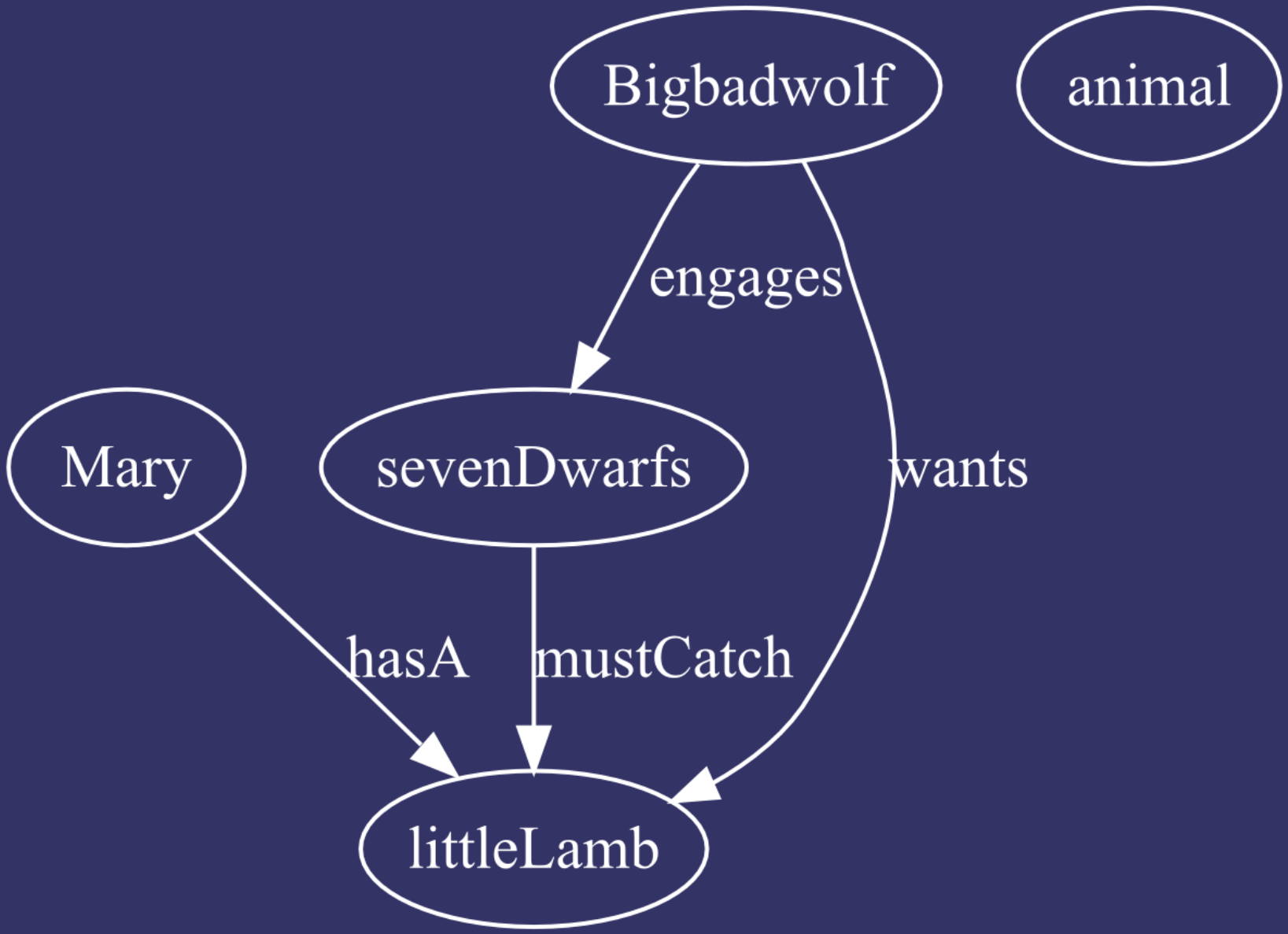
- RDF/XML
- N3
- Turtle
- ntripples
- ...
- natural language ;-)

A simple example

Mary has a lamb.
Big bad wolf wants this lamb.
Big bad wolf engages the seven dwarfs.
The seven dwarfs must catch the lamb.

... and of course, a little lamb is an animal!

A simple example



The example as N3 – hand written

```
Mary hasA littleLamb.
Bigbadwolf wants littleLamb;
    engages sevenDwarfs.
sevenDwarfs mustCatch littleLamb.
littleLamb a animal.
```

The example as N3 – serialized by RDFLib

```
@prefix : <file:///home/gurke/work/os2-
devcon/examples/example.n3#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

 :Bigbadwolf :engages :sevenDwarfs;
     :wants :littleLamb.

 :Mary :hasA :littleLamb.

 :sevenDwarfs :mustCatch :littleLamb.

 :littleLamb a :animal.
```

The example as Turtle

```
@prefix : file:///home/gurke/work/os2-
devcon/examples/example.n3#.
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#.
@prefix rdfs: http://www.w3.org/2000/01/rdf-schema#.
@prefix xml: http://www.w3.org/XML/1998/namespace.

 :Bigbadwolf :engages :sevenDwarfs;
     :wants :littleLamb.

 :Mary :hasA :littleLamb.

 :sevenDwarfs :mustCatch :littleLamb.

 :littleLamb a :animal.
```

The example as RDF/XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
   xmlns="file:///home/gurke/work/os2-devcon/examples/example.n3#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description rdf:about="file:///home/gurke/work/os2-
devcon/examples/example.n3#Mary">
    <hasA rdf:resource="file:///home/gurke/work/os2-
devcon/examples/example.n3#littleLamb"/>
  </rdf:Description>
  <rdf:Description rdf:about="file:///home/gurke/work/os2-
devcon/examples/example.n3#sevenDwarfs">
    <mustCatch rdf:resource="file:///home/gurke/work/os2-
devcon/examples/example.n3#littleLamb"/>
  </rdf:Description>
  <rdf:Description rdf:about="file:///home/gurke/work/os2-
devcon/examples/example.n3#littleLamb">
    <rdf:type rdf:resource="file:///home/gurke/work/os2-
devcon/examples/example.n3#animal"/>
  </rdf:Description>
  <rdf:Description rdf:about="file:///home/gurke/work/os2-
devcon/examples/example.n3#Bigbadwolf">
    <engages rdf:resource="file:///home/gurke/work/os2-
devcon/examples/example.n3#sevenDwarfs"/>
    <wants rdf:resource="file:///home/gurke/work/os2-
devcon/examples/example.n3#littleLamb"/>
  </rdf:Description>
</rdf:RDF>
```

The example as ntripples

```
<file:///home/gurke/work/os2-devcon/examples/example.n3#Mary>
<file:///home/gurke/work/os2-devcon/examples/example.n3#hasA>
<file:///home/gurke/work/os2-devcon/examples/example.n3#littleLamb>.

<file:///home/gurke/work/os2-devcon/examples/example.n3#sevenDwarfs>
<file:///home/gurke/work/os2-devcon/examples/example.n3#mustCatch>
<file:///home/gurke/work/os2-devcon/examples/example.n3#littleLamb>.

<file:///home/gurke/work/os2-devcon/examples/example.n3#littleLamb>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<file:///home/gurke/work/os2-devcon/examples/example.n3#animal>.

<file:///home/gurke/work/os2-devcon/examples/example.n3#Bigbadwolf>
<file:///home/gurke/work/os2-devcon/examples/example.n3#engages>
<file:///home/gurke/work/os2-devcon/examples/example.n3#sevenDwarfs
>.

<file:///home/gurke/work/os2-devcon/examples/example.n3#Bigbadwolf>
<file:///home/gurke/work/os2-devcon/examples/example.n3#wants>
<file:///home/gurke/work/os2-devcon/examples/example.n3#littleLamb>.
```

Summary

RDF as a very simple, broad and general underlying model.

Tripples with subject, predicate, object.

Tripples can reference each other.

Multiple serialization forms:
- N3 / Turtle: Use for manual creation
- XML: Use for automated handling, exchange
- ntripples: the “CSV” of RDF, simplest form

SPARQL
Querying RDF

SPARQL Overview

SPARQL is a query language for RDF graphs similar to SQL for relational databases.

Also, the syntax is similar – but with some differences of course!

SPARQL Query Forms - SELECT

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
schema#>

   SELECT ?dctitle ?dcdate
   WHERE
   {
      ?resource dc:title ?dctitle ;
            dc:date ?dcdate .
   }
```

Returns tabular data with a column for dc:title and a column for dc:date.

SPARQL Query Forms - CONSTRUCT

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
schema#>

   CONSTRUCT
   {
     ?resource rdfs:label ?dctitle .
   }
   WHERE
   {
     ?resource dc:title ?dctitle .
   }
```

Constructs an RDF-graph by using the template in the CONSTRUCT clause.

SPARQL Query Forms - DESCRIBE

```
DESCRIBE <http://test.com/data/#URI>
```

Returns one description of the given resource.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
schema#>

   DESCRIBE ?resource
   WHERE
   {
      ?resource rdfs:label "Yellow label text" .
   }
```

Returns the descriptions of all the resources that match the WHERE-clause.

SPARQL Query Forms - ASK

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>

   ASK
   WHERE
   {
      ?date  dc:date “2008-03-20” .
   }
```

Simply checks if a resource with dc:date of 20th march 2008 exists – and return TRUE if it does.

SPARQL – WHERE Clause

The where clause supports additional operations like OPTIONAL, UNION, FILTER etc.

I won't go into further detail here, but you will find some usefull links in the references...

RDFSchema

Giving it some basic shape

RDF Schema Overview

Defines syntax for RDF-data-exchange, like DTD and XML-schema for XML.

Defines the vocabulary for a certain domain using classes and properties.

If it contains rules for the right usage of ressources defined by it, it's called an ontology. So, RDFS can also be used to formulate (simple) ontologies.

Example: Dublin Core (bibliographical data)

RDF Schema Concepts

RDF has only the type-attribute for specification - we need more: Classes, properties, inheritance...

Classes:
- Class
- Ressource
- Property
- Literal

Properties:
- subClassOf
- subPropertyOf
- domain
- range

RDF Schema – Classes examples

```
ex:MotorVehicle     rdf:type  rdfs:Class.
ex:PassengerVehicle rdf:type  rdfs:Class.
ex:Van              rdf:type  rdfs:Class.
ex:Truck            rdf:type  rdfs:Class.
ex:MiniVan          rdf:type  rdfs:Class.

ex:PassengerVehicle rdfs:subClassOf ex:MotorVehicle.
ex:Van              rdfs:subClassOf ex:MotorVehicle.
ex:Truck            rdfs:subClassOf ex:MotorVehicle.

ex:MiniVan          rdfs:subClassOf ex:Van.
ex:MiniVan          rdfs:subClassOf ex:PassengerVehicle.
```

(from W3C RDF-primer)

RDF Schema – Properties examples

```
ex:Person       rdf:type      rdfs:Class.
ex:author       rdf:type      rdf:Property.
ex:author       rdfs:range    ex:Person.
```

Multiple ranges:

```
ex:hasMother    rdfs:range    ex:Female.
ex:hasMother    rdfs:range    ex:Person.
```

Literal values:

```
ex:age          rdf:type      rdf:Property.
ex:age          rdfs:range    xsd:integer.
```

(from W3C RDF-primer)

OWL
Ontologies – Modelling Knowledge

OWL - Overview

It's the web ontology language (but it's OWL and not WOL)

Describes terms of a domain of knowledge in a way that software agents can “understand” them.

Historically based on DAML+OIL

Uses concepts of predicate logic.

OWL – Language Levels

OWL Lite
Created to be easily implementable, used for simple taxonomies and very simple ontologies.

OWL DL
Most similar to DAML+OIL. DL means Description Logic. Here, RDFSchema hast been limited in some points.

OWL Full
Same as OWL DL but without the given restrictions.

These are upwards compatible – from Lite to Full

OWL – OWL Lite Language Constructs

OWL Lite provides language constructs for:
- RDFSchema Features
- (In-)Equality
- Property Characteristics
- Property Restrictions
- Cardinality Restrictions
- Header Informations
- Class Intersections
- Versioning
- Annotation
- Datatypes

OWL – OWL Lite Language Constructs

Some examples of OWL Lite language constructs:

- rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain
- equivalentClass, equivalentProperty
- ObjectProperty, DatatypeProperty
- minCardinality, maxCardinality
- intersectionOf
- versionInfo, DeprecatedClass, DeprecatedProperty
- AnnotationProperty, OntologyProperty

OWL – OWL DL & OWL Full Language Constructs

Additionally, more constructs are introduced with OWL DL and OWL Full – or existing are expanded

These are:

- oneOf, dataRange, disjointWith, equivalentClass, rdfs:subClassOf
- unionOf, complementOf, intersectionOf
- minCardinality, maxCardinality, cardinality
- hasValue

OWL – A Simple Ontology Example (wikipedia)

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://localhost:8080/OWLBuergerInformation.owl#"
  xml:base="http://localhost:8080/OWLBuergerInformation.owl">

  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="Gender"/>

  <owl:Class rdf:ID="Person"/>

  <owl:Class rdf:ID="Woman">
    <rdfs:subClassOf rdf:resource="#Person"/>
    <owl:equivalentClass>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#Gender"/>
        <owl:hasValue rdf:resource="#female" rdf:type="#Gender"/>
      </owl:Restriction>
    </owl:equivalentClass>
  </owl:Class>
```

Continued on next slide...

OWL – A Simple Ontology Example (wikipedia) ... continued!

```
  <owl:ObjectProperty rdf:ID="gender"
     rdf:type="http://www.w3.org/2002/07/owl#FunctionalProperty">
    <rdfs:range rdf:resource="#Gender"/>
    <rdfs:domain rdf:resource="#Person"/>
  </owl:ObjectProperty>

  <owl:DatatypeProperty rdf:ID="name"
     rdf:type="http://www.w3.org/2002/07/owl#FunctionalProperty">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Person"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="firstname"
     rdf:type="http://www.w3.org/2002/07/owl#FunctionalProperty">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Person"/>
  </owl:DatatypeProperty>

  <Person rdf:ID="STilgner" firstname="Susanne" name="Tilgner">
    <Gender rdf:resource="#female"/>
  </Person>
</rdf:RDF>
```

OWL – Conclusion

OWL is a powerful language for creating and using ontologies.

It's not that simple... We're covering themes from philosophy to mathematics here.

Maybe, some more things will be clear after the Protégé-demo... ;-)

Reasoning

Getting Answers To Complex Questions

Reasoning

Reasoning would be the solution to the Californian Chardonay problem given at the beginning of this speech.

When you have an Ontology – you should be able to make statements about data coming in...

With reasoning, you can automatically tell things about classes and properties thus making assumptions automatically.

Reasoning

In the OWL-section before, we got some idea of it with the example of the person – and what could be automatically assumed about it.

Here is another, natural language example:

- All humans are beeing born.
- Tim Berners Lee was born.

So we can assume: **Tim Berners Lee is a human.**

Reasoning

Protégé has a good support for reasoning.

I won't be able to demonstrate that due to lack of time in preparation – sorry...

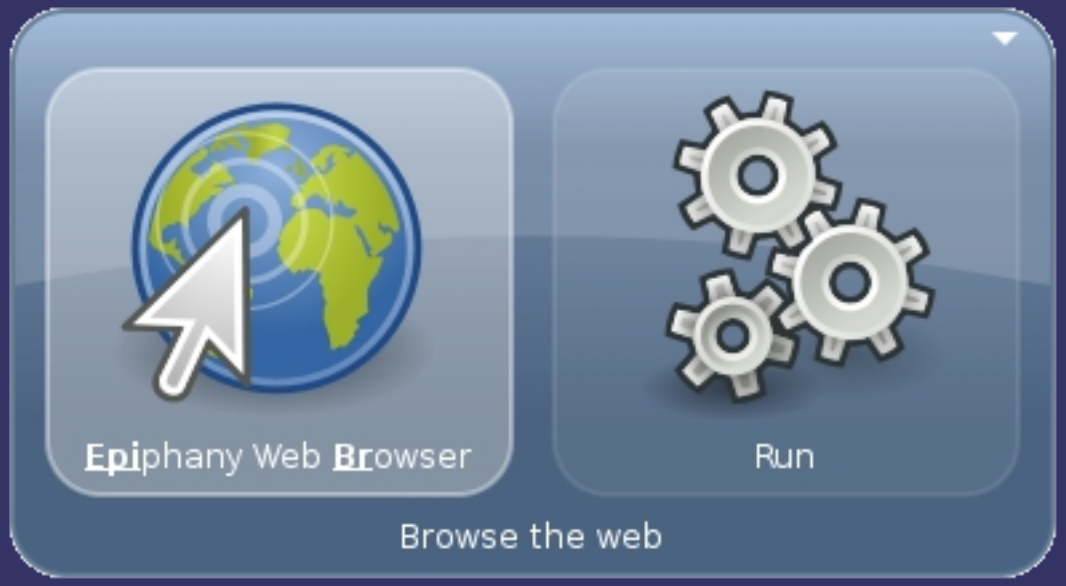
Check out the accompanying ressources to this speech for more information.

If you're really interested, follow the pizza-tutorial from Protégé and the “Advanced Reasoning with OWL”-slides from the ressources.

Case Study

GNOME Do – Maybe A Source of Inspiration?

GNOME DO



[http://do.davebsd.com]

GNOME DO

“A powerful, speedy, and sexy remote control for your GNOME Desktop”
[http://davebsd.com]

(I didn't verify that...)

Could be interesting for Voyager as they use some object model internally which is represented as Ontology.

Check out their whitepaper and wiki for more information!

You want more?!?

Tools, APIs and other useful resources...

You want more?!?

I hope, I was able to wet your appetite a bit...

If you want more information, check out the ressources to this speech (yeah, I know, I said that before..)

Especially:
- Mike Bergman's or the Nodalities-blog
- Dave Becketts HUGE linklist
- The podcast from Talis with Sir Tim Berners Lee
- One or more of the MANY cool projects out there
- Protégé

Demo
Protégé OWL

Thanks!