Semantic Web

Resource Description Framework (RDF)

What Are RDF and RDF Schema?

- RDF
  - Resource Description Framework
  - Data model
    - Syntax (XML)
  - Domain independent
    - Vocabulary is defined by RDF Schema

- RDF Schema
  - RDF Vocabulary Description Language
  - Captures the *semantic model* of a domain

RDF Basics

- RDF is a language that enable to describe making statements on resources
  - John is father of Bill
- Statement (or triple) as a logical formula P(x, y), where the binary predicate P relates the object x to the object y
- Triple data model:

  `<subject, predicate, object>`

  - **Subject**: Resource or blank node
  - **Predicate**: Property
  - **Object**: Resource (or collection of resources), literal or blank node
- Example:

  `<ex:john, ex:father-of, ex:bill>`

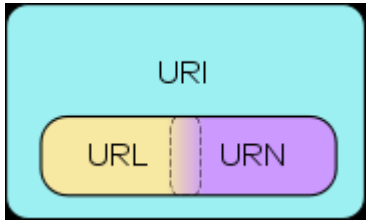
- RDF offers only binary predicates (properties)

Resources

- A resource may be:
  - Web page (e.g. `http://www.w3.org`)
  - A person (e.g. `http://www.fensel.com`)
  - A book (e.g. `urn:isbn:0-345-33971-1`)
  - Anything denoted with a URI!
- A URI is an *identifier* and **not** a location on the Web
- RDF allows making statements about resources:
  - `http://www.w3.org` **has the format** text/html
  - `http://www.fensel.com` **has first name** Dieter
  - `urn:isbn:0-345-33971-1` **has author** Tolkien

URI, URN, URL

- A Uniform Resource Identifier (URI) is a string of characters used to identify a name or a resource on the Internet

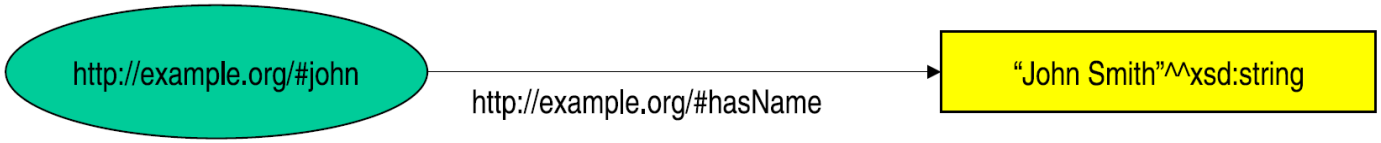


- A URI can be a URL or a URN
- A Uniform Resource Name (URN) defines an item's identity
  - the URN *urn:isbn:0-395-36341-1* is a URI that specifies the identifier system, i.e. International Standard Book Number (ISBN), as well as the unique reference within that system and allows one to talk about a book, but doesn't suggest where and how to obtain an actual copy of it
- A Uniform Resource Locator (URL) provides a method for finding it
  - the URL *http://www.sti-innsbruck.at/* identifies a resource (STI's home page) and implies that a representation of that resource (such as the home page's current HTML code, as encoded characters) is obtainable via HTTP from a network host named www.sti-innsbruck.at

Literals

- Plain literals
  - E.g. `"any text"`
  - Optional language tag, e.g. `"Hello, how are you?"@en-GB`
- Typed literals
  - E.g. `"hello"^^xsd:string`, `"1"^^xsd:integer`
  - Recommended datatypes:
    - XML Schema datatypes
- Only as *object* of a triple, e.g.:

```
⟨<http://example.org/#john>,
                    <http://example.org/#hasName>,
                                        "John Smith"^^xsd:string⟩
```



Datatypes

- One pre-defined datatype: **`rdf:XMLLiteral`**
  - Used for embedding XML in RDF

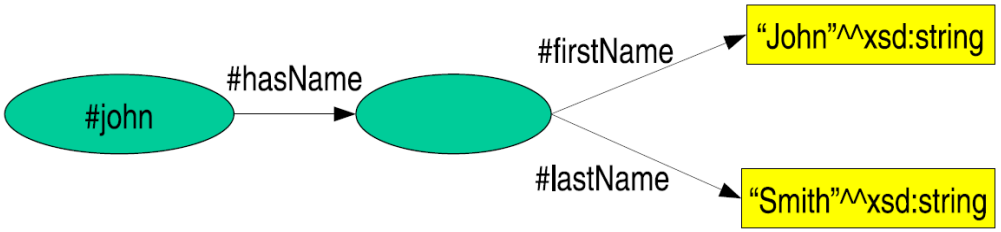
- Recommended datatypes are XML Schema datatypes, e.g.:
  - **`xsd:string`**
  - **`xsd:integer`**
  - **`xsd:float`**
  - **`xsd:anyURI`**
  - **`xsd:boolean`**

Blank Nodes I

- Blank nodes are nodes without a URI
  - Unnamed resources
  - More complex constructs
- Representation of blank nodes is syntax-dependent
  - *Blank node identifier*

- For example:

```
⟨<#john>, <#hasName>, _:johnsname⟩
⟨_:johnsname, <#firstName>, "John"^^xsd:string⟩
⟨_:johnsname, <#lastName>, "Smith"^^xsd:string⟩
```



Blank Nodes II

- **Representation of complex data**

  A blank node can be used to indirectly attach to a resource a consistent set of properties which together represent a complex data

- **Anonymous classes in OWL**

  The ontology language OWL uses blank nodes to represent anonymous classes such as unions or intersections of classes, or classes called restrictions, defined by a constraint on a property

RDF Containers

- Grouping property values:

“The lecture is attended by John, Mary and Chris” **`Bag`**

“[RDF-Concepts] is edited by Graham and Jeremy (in that order)” **`Seq`**

“The source code for the application may be found at ftp1.example.org, ftp2.example.org, ftp3.example.org” **`Alt`**

RDF Containers 2

- Three types of containers:
  - **`rdf:Bag`** - unordered set of items
  - **`rdf:Seq`** - ordered set of items
  - **`rdf:Alt`** - set of alternatives

- Every container has a triple declaring the **`rdf:type`**

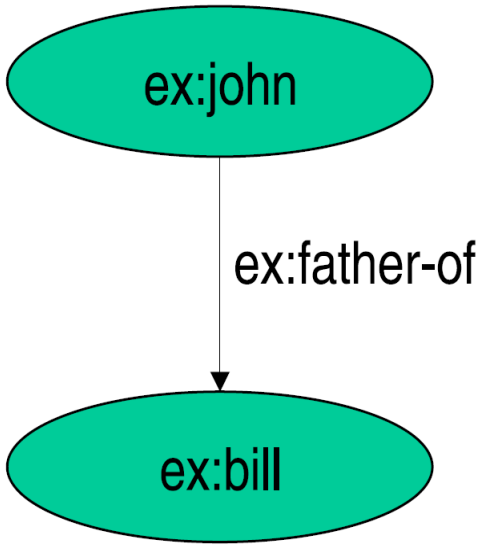
- Items in the container are denoted with
  - **`rdf:_1`**, **`rdf:_2`**, . . . ,**`rdf:_n`**

RDF Containers 2

- Three types of containers:
  - `rdf:Bag` - unordered set of items
  - `rdf:Seq` - ordered set of items
  - `rdf:Alt` - set of alternatives
- Every container has a triple declaring the `rdf:type`
- Items in the container are denoted with
  - `rdf:_1, rdf:_2, . . . ,rdf:_n`
- Limitations:
  - Semantics of the container is up to the application
  - What about closed sets?
    - How do we know whether Graham and Jeremy are the only editors of [RDF-Concepts]?

RDF Triple Graph Representation

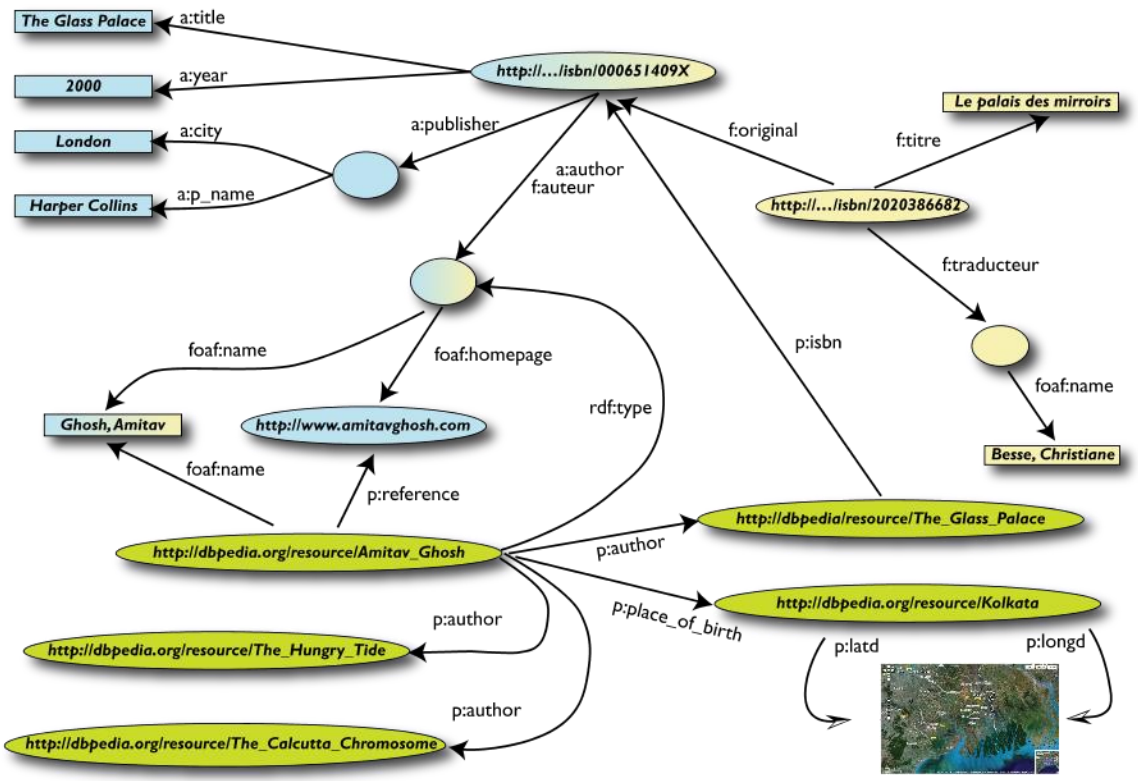
- The triple data model can be represented as a graph
- Such graph is called in the Artificial Intelligence community a **semantic net**



- Labeled, directed graphs
  - **Nodes**: resources, literals
  - **Labels**: properties
  - **Edges**: statements

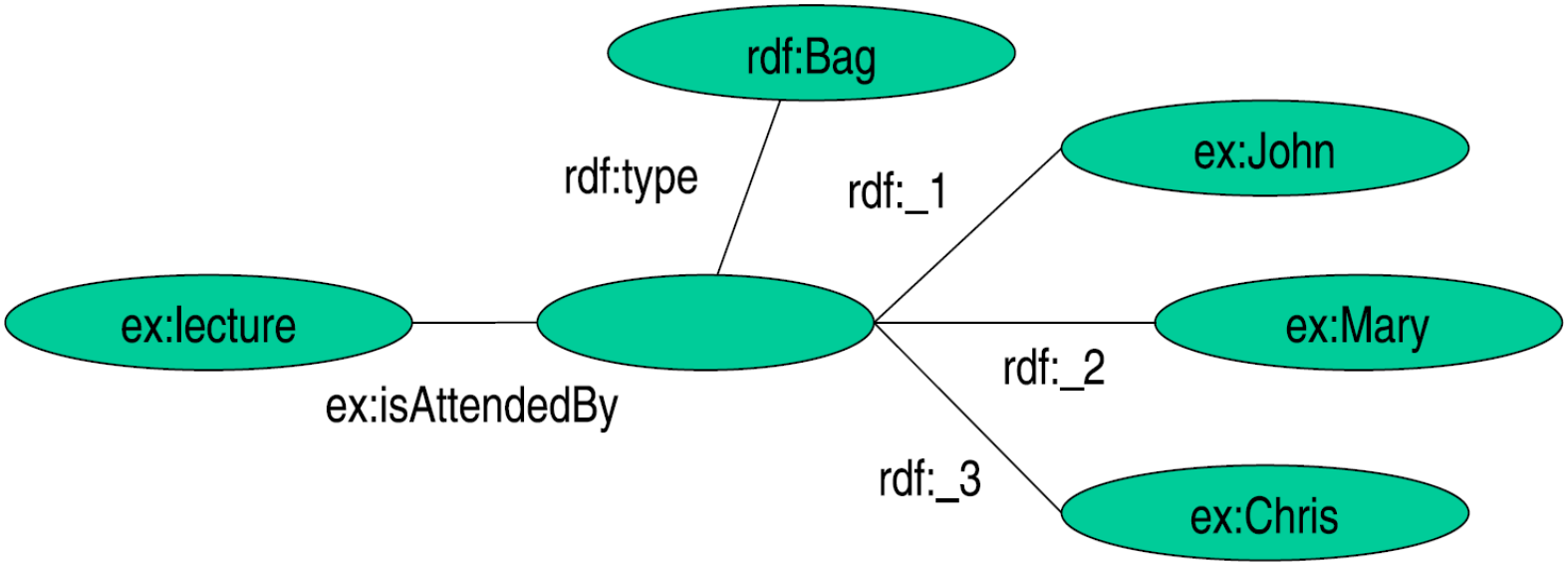
RDF: a Direct Connected Graph based Model

- Different interconnected triples lead to a more complex graphic model
- Basically a RDF document is a direct connect graph
  - http://en.wikipedia.org/wiki/Connectivity_%28graph_theory%29



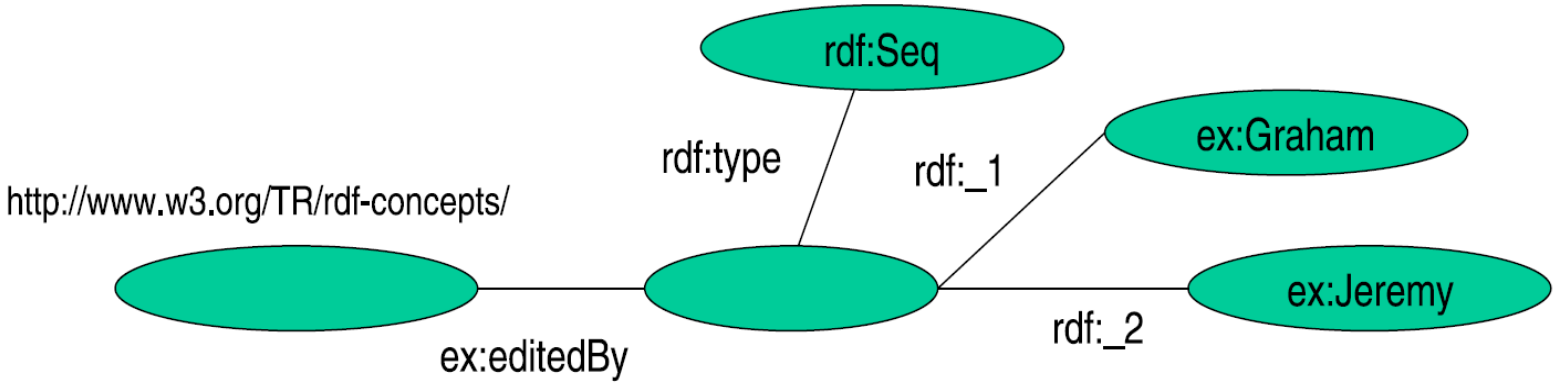
RDF Containers Graph Representation: Bag

"The lecture is attended by John, Mary and Chris"



RDF Containers Graph Representation: Seq

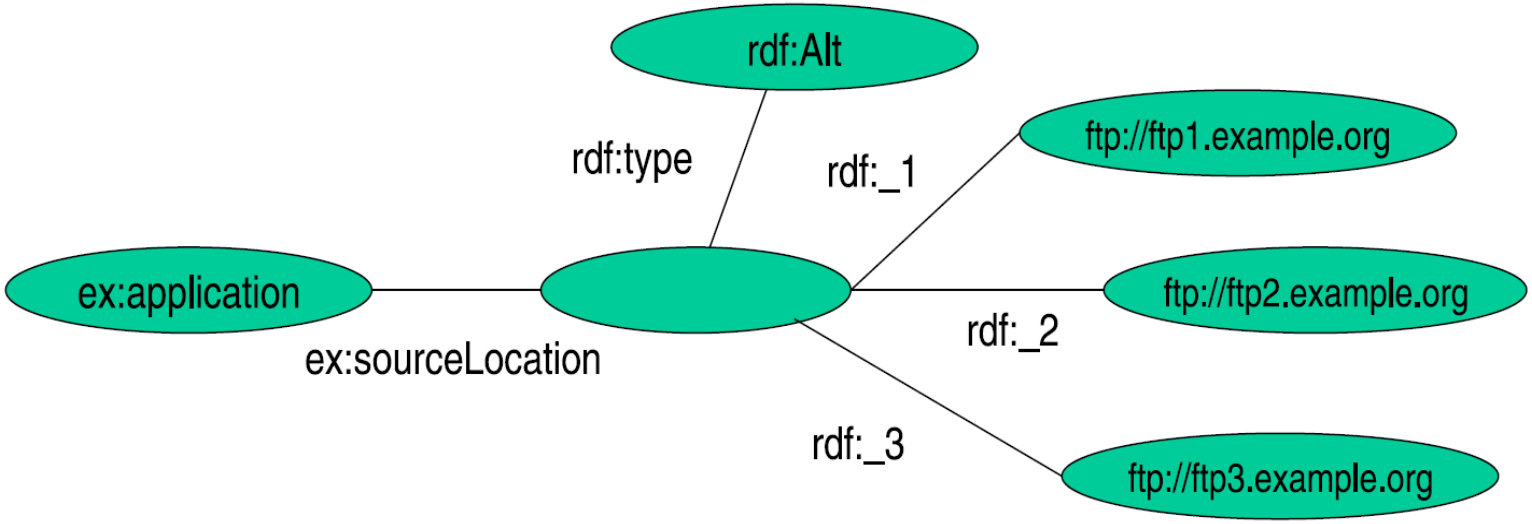
"[RDF-Concepts] is edited by Graham and Jeremy (in that order)"



RDF Containers Graph Representation: Alt

"The source code for the application may be found at

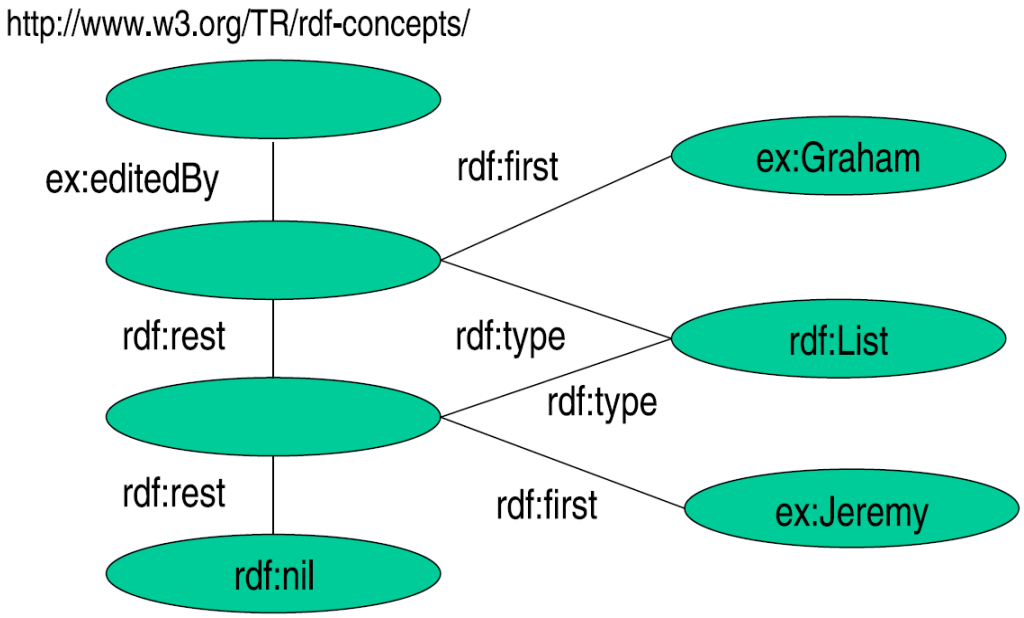
***ftp1.example.org**, **ftp2.example.org**, **ftp3.example.org**"*



RDF Collections

"[RDF-Concepts] is edited by Graham and Jeremy

(in that order) and nobody else"



RDF provides support for describing groups containing only the specified members, in the form of RDF collections.

Reification I

- Reification: statements about statements

Mary claims that John's name is "John Smith".

```
⟨<#myStatement>, rdf:type, rdf:Statement⟩
⟨<#myStatement>, rdf:subject, <#john>⟩
⟨<#myStatement>, rdf:predicate, <#hasName>⟩
⟨<#myStatement>, rdf:object, "John Smith"⟩
```

This kind of statement can be used to describe belief or trust in other statements, which is important in some kinds of applications

Necessary because there are only triples in RDF: we cannot add an identifier directly to a triple (then it would be a quadruple)

Reification II

- Reification: statements about statements

Mary claims that John's name is "John Smith".

```
⟨<#myStatement>, rdf:type, rdf:Statement⟩
⟨<#myStatement>, rdf:subject, <#john>⟩
⟨<#myStatement>, rdf:predicate, <#hasName>⟩
⟨<#myStatement>, rdf:object, "John Smith"⟩
```

```
⟨<#john>, <#hasName>, "John Smith"⟩
```

In such a way we attached a label to the statement.

Reification III

- Reification: statements about statements

Mary claims that John's name is "John Smith".

```
⟨<#myStatement>, rdf:type, rdf:Statement⟩
⟨<#myStatement>, rdf:subject, <#john>⟩
⟨<#myStatement>, rdf:predicate, <#hasName>⟩
⟨<#myStatement>, rdf:object, "John Smith"⟩

⟨<#mary>, <#claims>, <#myStatement>⟩
```

RDF uses only binary properties. This restriction seems quite serious because often we use predicates with more than two arguments. Luckily, such predicates can be simulated by a number of binary predicates.

RDF Vocabulary

- RDF defines a number of resources and properties
- We have already seen: `rdf:XMLLiteral`, `rdf:type`, . . .
- RDF vocabulary is defined in the namespace: `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

- Classes:
  - `rdf:Property`, `rdf:Statement`, `rdf:XMLLiteral`
  - `rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdf:List`
- Properties:
  - `rdf:type`, `rdf:subject`, `rdf:predicate`, `rdf:object`,
  - `rdf:first`, `rdf:rest`, `rdf:_n`
  - `rdf:value`
- Resources:
  - `rdf:nil`

RDF Vocabulary

- Typing using **`rdf:type`**:

  `<A, rdf:type, B>`

  "A belongs to class B"

- All properties belong to class **`rdf:Property`**:

  `<P, rdf:type, rdf:Property>`

  "P is a property"

  `<rdf:type, rdf:type, rdf:Property>`

  "rdf:type is a property"

The RDF Schema (RDFS)

How to represent the semantics of data models

RDF Vocabulary Description Language 1

- Types in RDF:

```
<#john, rdf:type, #Student>
```

- What is a "`#Student`"?

- RFD is not defining a vocabulary about the statements, but only to express statements

- We know that "`#Student`" identifies a category (a concept or a class), but this is only implicitly defined in RDF

RDF Vocabulary Description Language 2

- We need a language for defining RDF types:
  - Define classes:
    - *"`#Student` is a class"*
  - Relationships between classes:
    - *"`#Student` is a sub-class of `#Person`"*
  - Properties of classes:
    - *"`#Person` has a property `hasName`"*
- RDF Schema is such a language

RDF Vocabulary Description Language 3

- Classes:
  `<#Student, rdf:type, #rdfs:Class>`
- Class hierarchies:
  `<#Student, rdfs:subClassOf, #Person>`

- Properties:
  `<#hasName, rdf:type, rdf:Property>`
- Property hierarchies:
  `<#hasMother, rdfs:subPropertyOf, #hasParent>`

- Associating properties with classes (a):
  - "The property `#hasName` only applies to `#Person`"
    `<#hasName, rdfs:domain, #Person>`

- Associating properties with classes (b):
  - "The type of the property `#hasName` is `#xsd:string`"
    `<#hasName, rdfs:range, xsd:string>`

RDFS Vocabulary

- RDFS Extends the RDF Vocabulary
- RDFS vocabulary is defined in the namespace:

  `http://www.w3.org/2000/01/rdf-schema#`

RDFS Classes

- `rdfs:Resource`
- `rdfs:Class`
- `rdfs:Literal`
- `rdfs:Datatype`
- `rdfs:Container`
- `rdfs:ContainerMembershipProperty`

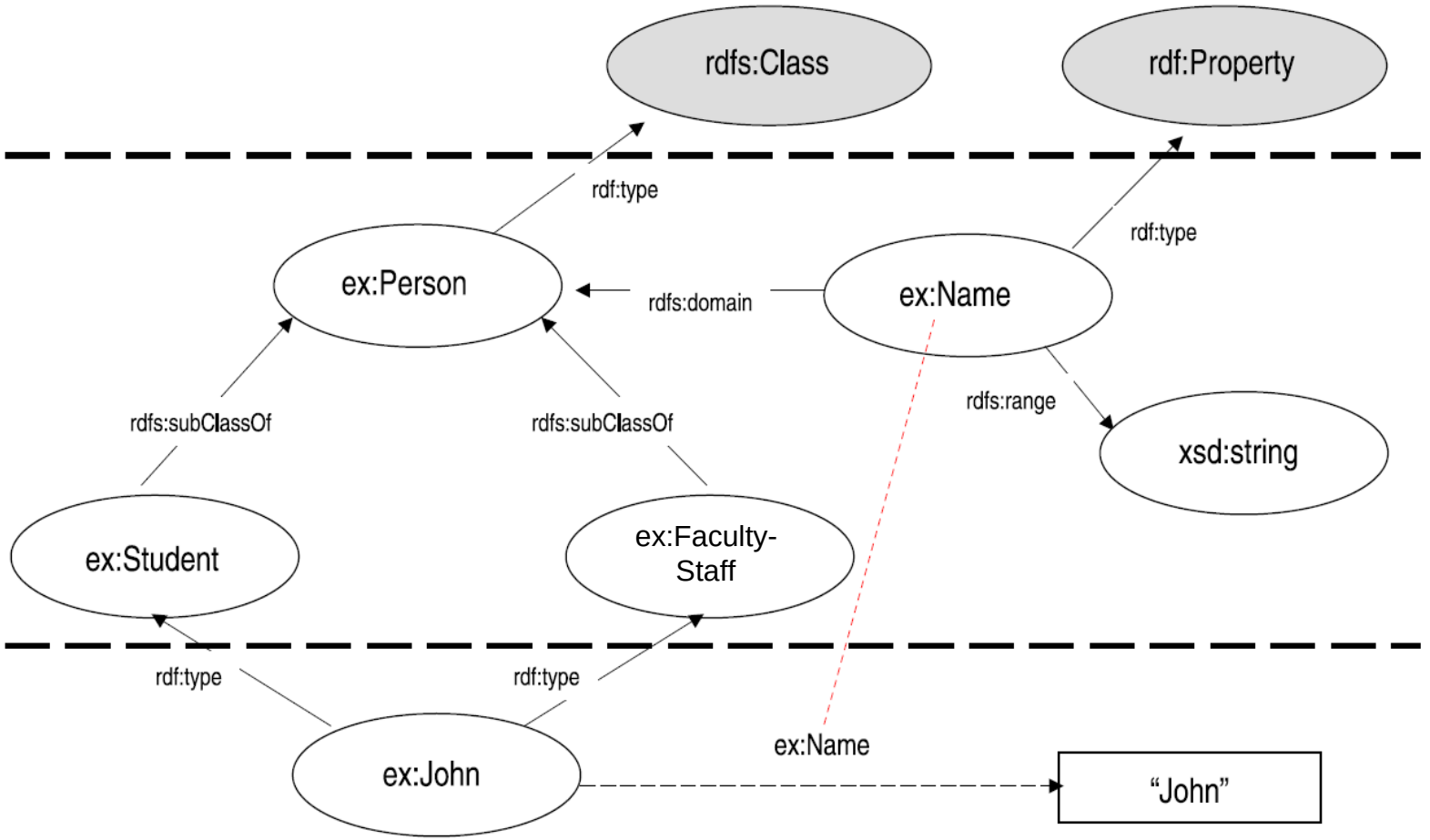
RDFS Properties

- `rdfs:domain`
- `rdfs:range`
- `rdfs:subPropertyOf`
- `rdfs:subClassOf`
- `rdfs:member`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`
- `rdfs:comment`
- `rdfs:label`

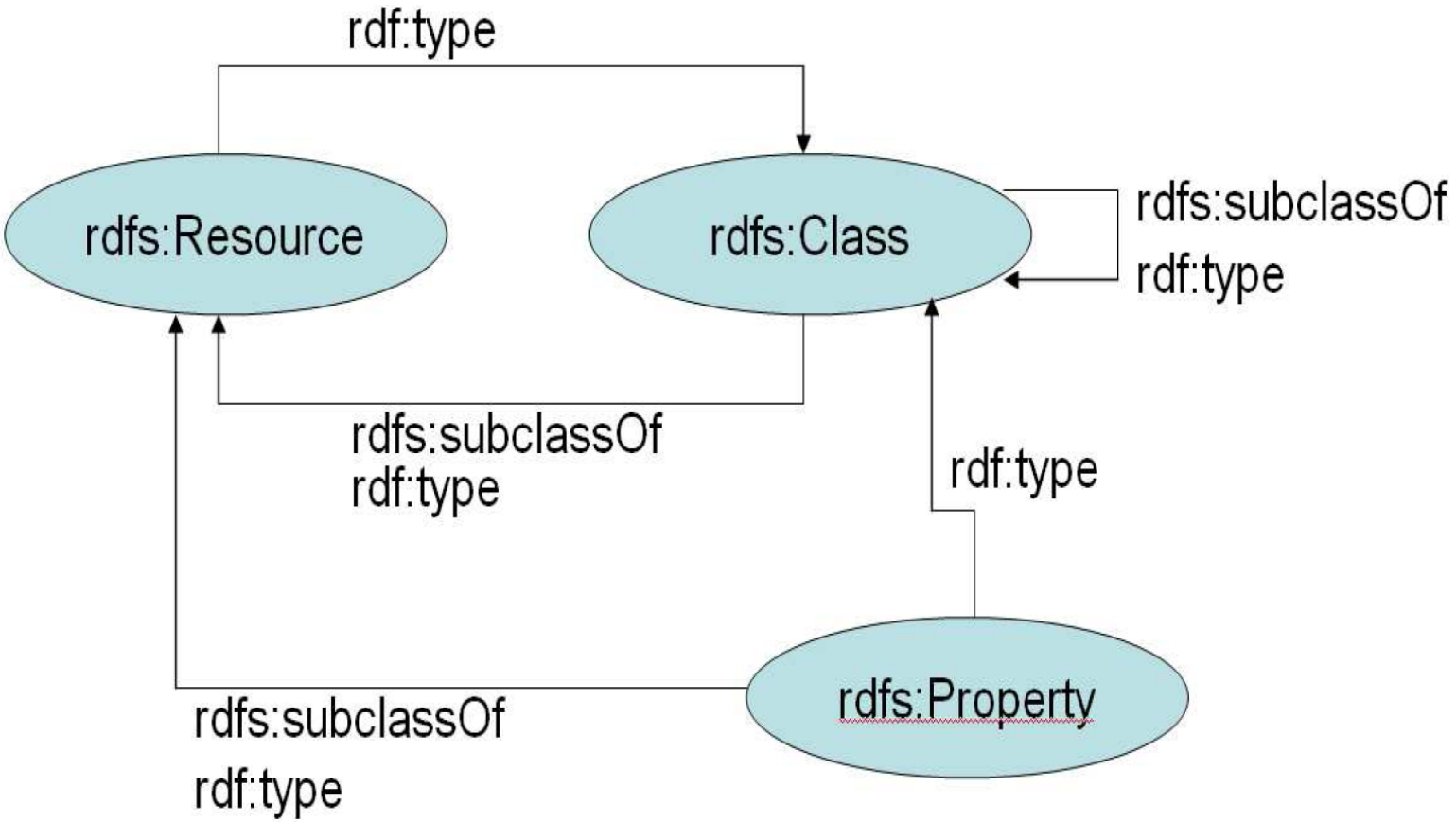
RDFS Principles

- **Resource**
  - All resources are implicitly instances of `rdfs:Resource`
- **Class**
  - Describe sets of resources
  - Classes are resources themselves - e.g. Webpages, people, document types
    - Class hierarchy can be defined through `rdfs:subClassOf`
    - Every class is a member of `rdfs:Class`
- **Property**
  - Subset of RDFS Resources that are properties
    - **Domain**: class associated with property: `rdfs:domain`
    - **Range**: type of the property values: `rdfs:range`
    - Property hierarchy defined through: `rdfs:subPropertyOf`

RDFS Example



RDFS Vocabulary Example



RDFS Metadata Properties

- Metadata is “data about data”
- Any meta-data can be attached to a resource, using:
  - **`rdfs:comment`**
    - Human-readable description of the resource, e.g.
      - ⟨`<ex:Person>, rdfs:comment, "A person is any human being"`⟩
  - **`rdfs:label`**
    - Human-readable version of the resource name, e.g.
      - ⟨`<ex:Person>, rdfs:label, "Human being"`⟩
  - **`rdfs:seeAlso`**
    - Indicate additional information about the resource, e.g.
      - ⟨`<ex:Person>, rdfs:seeAlso, <http://xmlns.com/wordnet/1.6/Human>`⟩
  - **`rdfs:isDefinedBy`**
    - A special kind of rdfs:seeAlso, e.g.
      - ⟨`<ex:Person>,rdfs:isDefinedBy,<http://xmlns.com/wordnet/1.6/Human>`⟩

RDF(S) SEMANTICS

Semantics

- RDF(S) vocabulary has built-in “meaning”
- RDF(S) Semantics
  - Makes meaning explicit
  - Defines what follows from an RDF graph
- Semantic notions
  - Subgraph
  - Instance
  - Entailment

Subgraph

- *E* is a subgraph of *S* if and only if *E* predicates are a subset of *S* predicates
  - ⟨<#john>, <#hasName>, _:johnsname⟩
  - ⟨<_:johnsname, <#firstName>, "John"^^xsd:string⟩
  - ⟨<_:johnsname, <#lastName>, "Smith"^^xsd:string⟩
- Subgraphs:
  - ⟨<#john>,<#hasName>, :johnsname⟩
  - ⟨_:johnsname, <#firstName>, "John"^^xsd:string⟩
  - ⟨_:johnsname, <#firstName>, "John"^^xsd:string⟩
  - ⟨_:johnsname, <#lastName>, "Smith"^^xsd:string⟩
  - ⟨<#john>, <#hasName>, _:johnsname⟩

Instance

- *S'* is an instance of *S* if and only if some blank nodes in *S* are replaced with blank nodes, literals or URIs
  - ⟨<#john>, <#hasName>, _:johnsname⟩
  - ⟨_:johnsname, <#firstName>, "John"^^xsd:string⟩
  - ⟨_:johnsname, <#lastName>, "Smith"^^xsd:string⟩
- Instances:
  - ⟨<#john>, <#hasName>, <#abc>⟩
  - ⟨<#abc>, <#firstName>, "John"^^xsd:string⟩
  - ⟨<#abc>, <#lastName>, "Smith"^^xsd:string⟩
  - ⟨<#john>, <#hasName>, _:X⟩
  - ⟨_:X, <#firstName>, "John"^^xsd:string⟩
  - ⟨_:X, <#lastName>, "Smith"^^xsd:string⟩
  - ⟨<#john>, <#hasName>, _:johnsname⟩
  - ⟨_:johnsname, <#firstName>, "John"^^xsd:string⟩
  - ⟨_:johnsname, <#lastName>, "Smith"^^xsd:string⟩
- Every graph is an instance of itself!

Entailment

- **Entailment** or logical implication is a relation between sentences of a formal language

- *S* entails *E* if *E* logically follows from *S*
  - Written: `S |= E`

- A graph entails all it subgraphs
  - If *S'* is a subgraph of *S*: `S |= S'`

- All instances of a graph *S* entail *S*
  - If *S''* is an instance of *S*: `S'' |= S`

RDFS Entailment

<http://example.org/#john> rdf:type <http://example.org/#Student>
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>

entails

<http://example.org/#john> rdf:type <http://example.org/#Person>

<http://example.org/#hasName> rdfs:domain <http://example.org/#Student>
<http://example.org/#mary> <http://example.org/#hasName> “Mary”

entails

<http://example.org/#mary> rdf:type <http://example.org/#Student>

<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>

entails

<http://example.org/#john> <http://example.org/#hasParent <http://example.org/#mary>

RDFS Entailment

<http://example.org/#john> rdf:type <http://example.org/#Student>
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>

entails

<http://example.org/#john> rdf:type <http://example.org/#Person>

<http://example.org/#hasName> rdfs:domain <http://example.org/#Student>
<http://example.org/#mary> <http://example.org/#hasName> “Mary”

entails

<http://example.org/#mary> rdf:type <http://example.org/#Student>

<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>

entails

<http://example.org/#john> <http://example.org/#hasParent <http://example.org/#mary>

RDFS Entailment

<http://example.org/#john> rdf:type <http://example.org/#Student>
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>

entails

<http://example.org/#john> rdf:type <http://example.org/#Person>

<http://example.org/#hasName> rdfs:domain <http://example.org/#Student>
<http://example.org/#mary> <http://example.org/#hasName> “Mary”

entails

<http://example.org/#mary> rdf:type <http://example.org/#Student>

<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>

entails

<http://example.org/#john> <http://example.org/#hasParent <http://example.org/#mary>

Entailment Rules

- Semantics defined through *entailment rules*
- Rule:
  - If *S* contains `<triple pattern>` then add `<triple>`
- Executing all entailment rules yields *realization* of S
- S entails E if E is a subgraph of the realization of S
- Axiomatic triple are always added

RDF Entailment

- if *E* contains `<A, B, C>` then add

  `<B, rdf:type, rdf:Property>`

- if *E* contains `<A, B, l>` (l is a valid XML literal) then add

  `<_:X, rdf:type, rdf:XMLLiteral>`

  where `_:X` identifies to blank node allocated to `l`

RDFS Entailment 1

- everything in the subject is a resource
  - if *E* contains `<A,B,C>` then add `<A, rdf:type, rdfs:Resource>`
- every non-literal in the object is a resource
  - if *E* contains `<A,B,C>` (C is not a literal) then add `<C, rdf:type, rdfs:Resource>`
- every class is subclass of `rdfs:Resource`
  - if *E* contains `<A, rdf:type, rdfs:Class>` then add `<A, rdfs:subClassOf, rdfs:Resource>`
- inheritance:
  - if *E* contains `<A, rdf:type, B>`, `<B, rdfs:subClassOf, C>` then add `<A, rdf:type, C>`
- `rdfs:subClassOf` is transitive
  - if *E* contains `<A, rdfs:subClassOf, B>`, `<B, rdfs:subClassOf, C>` then add `<A, rdfs:subClassOf, C>`

RDFS Entailment 2

- **`rdfs:subClassOf`** is reflexive
  - if *E* contains **`<A, rdf:type, rdfs:Class>`** then add **`<A, rdfs:subClassOf, A>`**
- **`rdfs:subPropertyOf`** is transitive
  - if *E* contains **`<A, rdfs:subPropertyOf, B>`**, **`<B, rdfs:subPropertyOf, C>`** then add **`<A, rdfs:subPropertyOf, C>`**
- **`rdfs:subPropertyOf`** is reflexive
  - if *E* contains **`<P, rdf:type, rdf:Property>`** then add **`<P, rdfs:subPropertyOf, P>`**
- domain of properties
  - if *E* contains **`<P, rdfs:domain, C>`**, **`<A, P, B>`** then add **`<A, rdf:type, C>`**
- range of properties
  - if *E* contains **`<P, rdfs:range, C>`**, **`<A, P, B>`** then add **`<B, rdf:type, C>`**

RDFS Entailment 3

- every literal is a member of **`rdfs:Literal`**
  - if *E* contains **`<A, B, l>`** (l is a plain literal) then add **`<_:X, rdf:type, rdfs:Literal>`**

- every datatype is subclass of **`rdfs:Literal`**
  - if *E* contains **`<A, rdf:type, rdfs:Datatype>`** then add **`<A, rdfs:subClassOf, rdfs:Literal>`**

RDF(S) SERIALIZATION

RDF Serialization Formats

There are several machine readable serialization formats for RDF

- RDF/XML
- Turtle
- N3

RDF/XML 1

- Serializing RDF for the Web
  - XML as standardized interchange format:
    - Namespaces (e.g. rdf:type, xsd:integer, ex:john)
    - Encoding (e.g. UTF8, iso-8859-1)
    - XML Schema (e.g. datatypes)
- Reuse of existing XML tools:
  - Syntax checking (i.e. schema validation)
  - Transformation (via XSLT)
    - Different RDF representation
    - Layout (XHTML)
    - Different XML-based formats
- Parsing and in-memory representation/manipulation (DOM/SAX)
- . . .

RDF/XML 2

```
<#john, #hasName, "John">
<#john, #marriedTo, #mary>
```

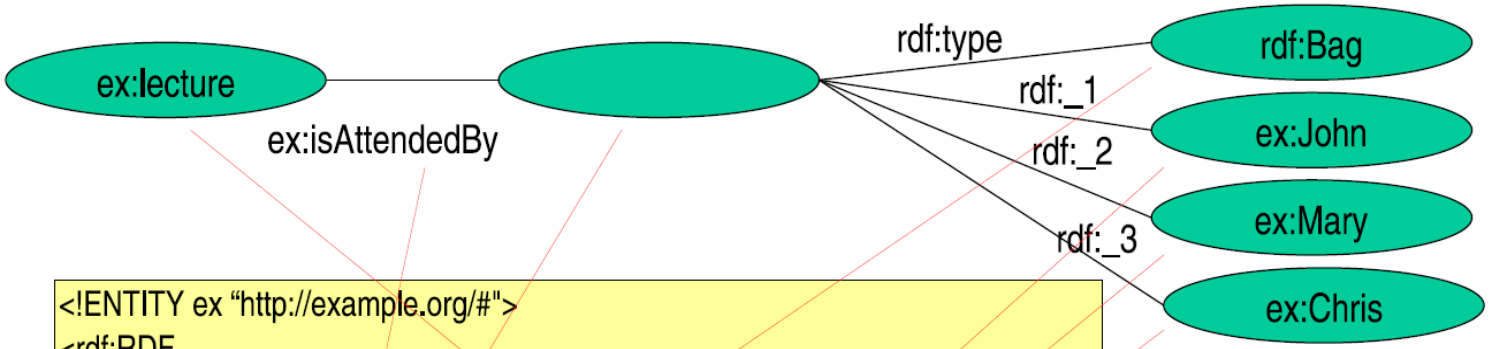
```
<!ENTITY ex "http://example.org/#">

<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:ex="http://example.org#">

    <rdf:Description rdf:about="http://example.org/#john">
      <ex:hasName>John</ex:hasName>
      <ex:marriedTo rdf:resource="&ex;mary"/>
    </rdf:Description>

</rdf:RDF>
```

Head

Body

Foot

RDF/XML 3




```
<!ENTITY ex "http://example.org/#">
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:ex="http://example.org#">
    <rdf:Description rdf:about="&ex;lecture">
        <ex:isAttendedBy>
            <rdf:Bag>
                <rdf:li rdf:resource="&ex;John"/>
                <rdf:li rdf:resource="&ex;Mary"/>
                <rdf:li rdf:resource="&ex;Chris"/>
            </rdf:Bag>
        </ex:isAttendedBy>
    </rdf:Description>
```

Turtle

- Turtle stands for Terse RDF Triple Language
- An RDF serialization
- Triple representation of <Subject, Predicate, Object>
- Example:

```
@prefix person: <http://example/person/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
person: A foaf:name “Jek" .
person: A foaf:mbox <mailto:jek@example.net> .
person: B foaf:name “Yuan" .
_:b foaf:name “Jeff" .
_:b foaf:mbox <mailto:jeff@example.org> .
```

RDF N3 syntax

- Notation3, or N3
- A shorthand non-XML serialization of RDF models
- Designed for human-readability
  - much more compact and readable than XML RDF notation
- Example

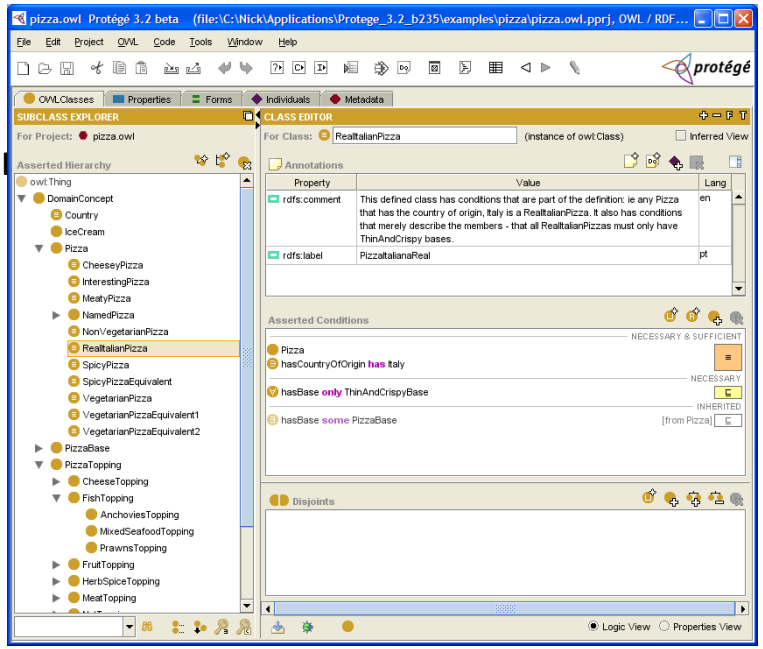
```
{:John :Loves :Mary} :accordingTo :Bill
```

TOOLS

Tool Support for RDF/RDFS

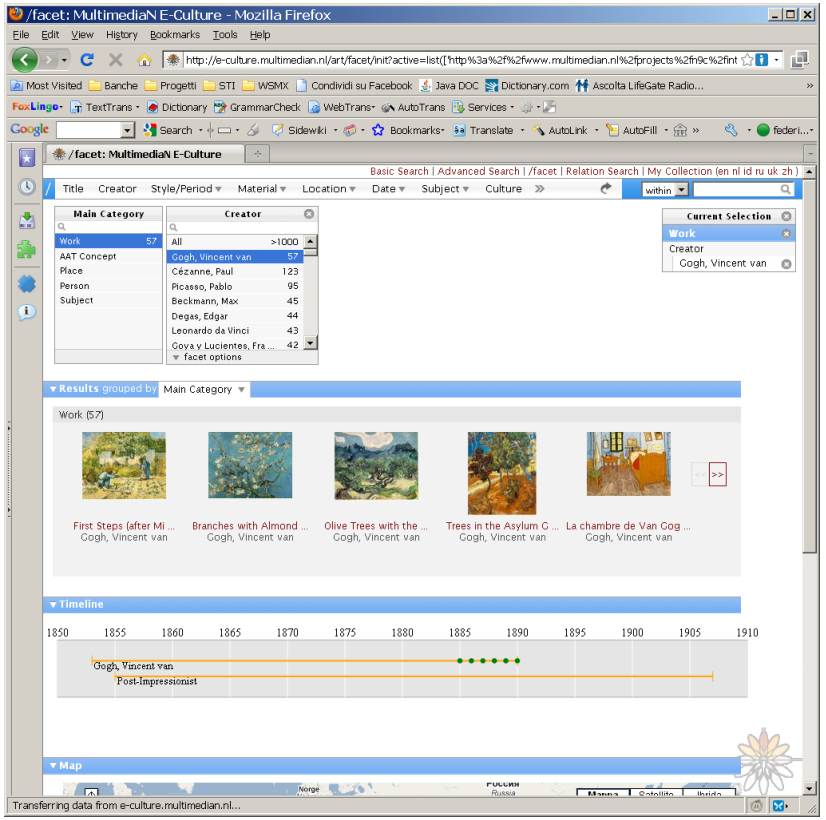
- Ontology editors
  - Protégé (http://protege.stanford.edu/)
- Browser
  - /facet (http://slashfacet.semanticweb.org/)
- RFD repositories
  - Sesame (http://www.openrdf.org/)
- APIs
  - RDF2Go – Java (http://semanticweb.org/wiki/RDF2Go)
  - Jena – Java (http://jena.sourceforge.net/)
- Validator
  - W3C Validator (http://www.w3.org/RDF/Validator/)

- Developed by Stanford Medical Informatics
- Has a large user community (approx 30k)
- Support
  - Graph view, consistency check, web, mergi
- No support
  - Addition of new basic types
  - Limited multi-user support



/facet

- /facet is a generic browser for heterogeneous semantic web repositories
- Works on any RDFS dataset without any additional configuration
- Select and navigate facets of resources of any type
- Make selections based on properties of other, semantically related, types
- Allows the inclusion of facet-specific display options



Sesame

- A framework for storage, querying and inferencing of RDF and RDF Schema
- A Java Library for handling RDF
- A Database Server for (remote) access to repositories of RDF data
- Features:
  - Light-weight yet powerful Java API
  - SeRQL, SPARQL
  - High scalability (O(10^7) triples on desktop hardware)
  - Various backends (Native Store, RDBMS, main memory)
  - Reasoning support
  - Transactional support
  - Context support
  - RDF/XML, Turtle, N3, N-Triples

Jena

- A Java framework for building Semantic Web applications
- Initiated by Hewlett Packard (HP) Labs Semantic Web Programme.
- Includes:
  - A RDF API
  - Reading and writing RDF in RDF/XML, N3 and N-Triples
  - An OWL API
  - In-memory and persistent storage
  - SPARQL query engine

RDF2Go

- **RDF2Go** is an abstraction over triple (and quad) stores. It allows developers to program against rdf2go interfaces and choose or change the implementation later easily

- It can be extended: you can create an adapter from any RDF Object Model to RDF2Go object model

- Directly supported implementations:
  - Jena 2.4
  - Jena 2.6
  - Sesame 2

W3C Validator

- RDF Validator
- Parse RDF documents and detects errors w.r.t. the current RDF specification
- Available online service
- Downloadable code
- Based on ARP parser (the one also adopted in Jena)

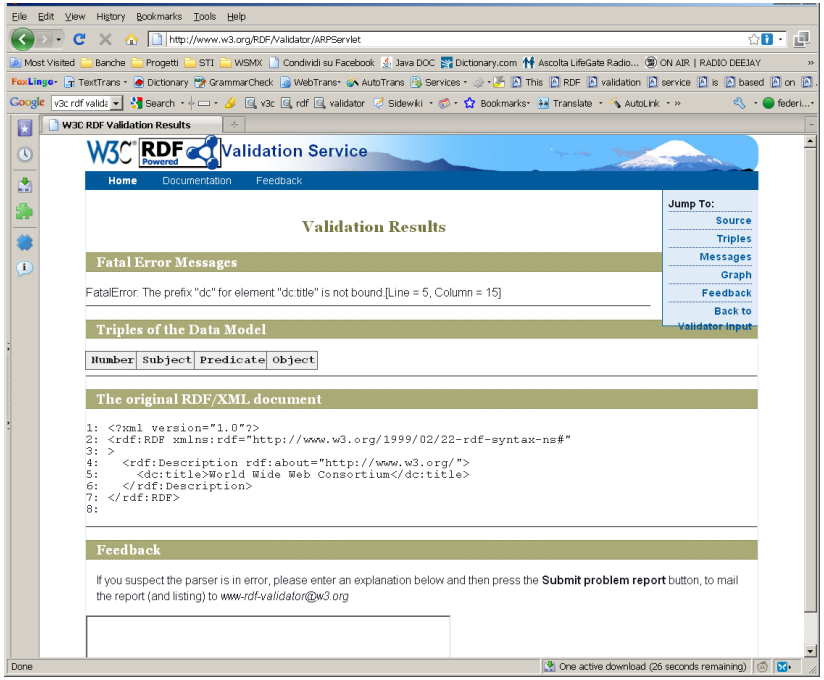


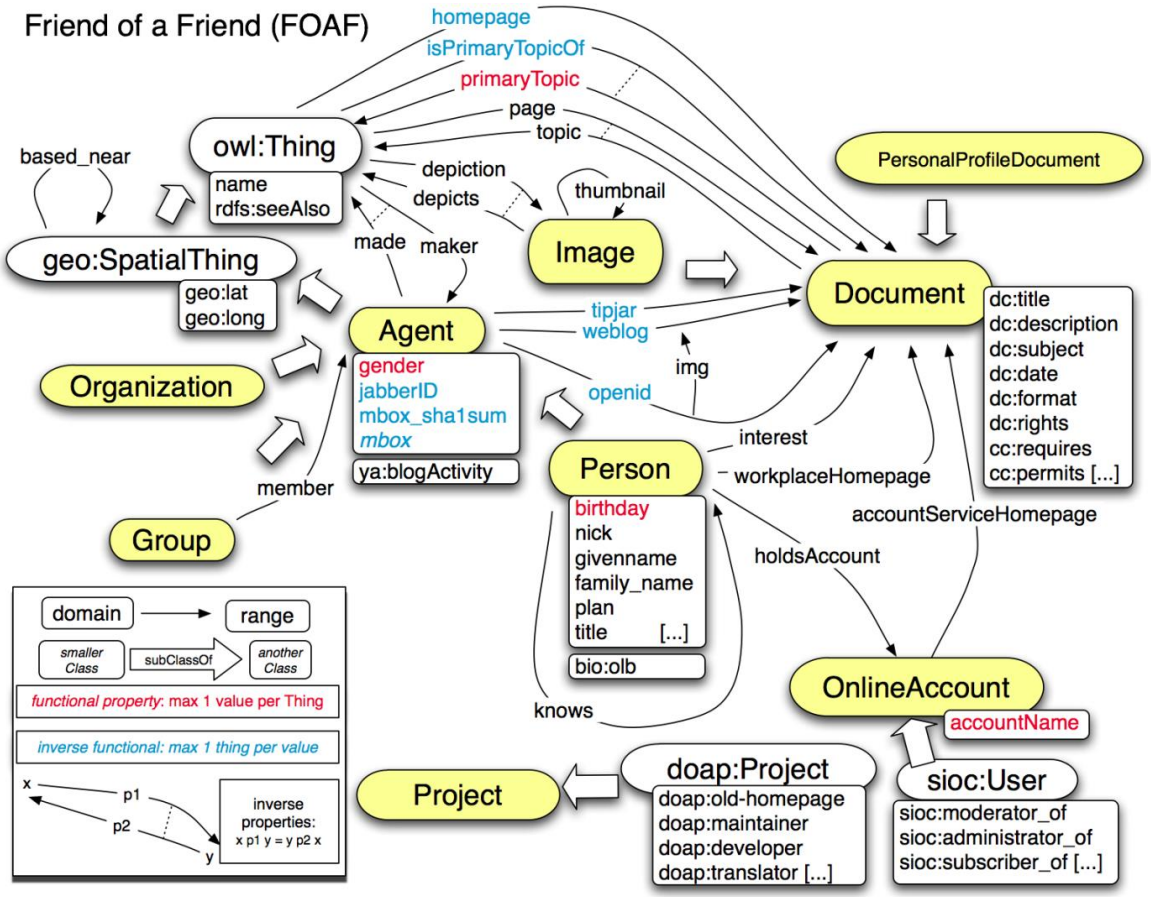
ILLUSTRATION BY A LARGER EXAMPLE

An example of usage of RDF and RDF(S)

Friend of a Friend (FOAF)

- Friend of a Friend is a project that aims at providing simple ways to describe people and relations among them
- FOAF adopts RDF and RDFS
- Full specification available on: http://xmlns.com/foaf/spec/
- Tools based on FOAF:
  - FOAF search (http://foaf.qdos.com/)
  - FOAF builder (http://foafbuilder.qdos.com/)
  - FOAF-a-matic (http://www.ldodds.com/foaf/foaf-a-matic)
  - FOAF.vix (http://foaf-visualizer.org/)

FOAF Schema



[http://www.foaf-project.org/]

Questions?