



RDF and RDF Schema

Oscar Corcho, Raúl García Castro, Oscar Muñoz-García
{ocorcho,rgarcia}@fi.upm.es, omunoz@delicias.dia.fi.upm.es
<http://www.oeg-upm.net/>

Ontological Engineering Group
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn,
28660 Boadilla del Monte, Madrid, Spain

Work distributed under the license Creative Commons Attribution-Noncommercial-Share Alike 3.0

Main References



Gómez-Pérez, A.; Fernández-López, M.; Corcho, O. **Ontological Engineering**. Springer Verlag. 2003

Capítulo 4: Ontology languages



Brickley D, Guha RV (2004) *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation.

<http://www.w3.org/TR/PR-rdf-schema>

Lassila O, Swick R (1999) *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation.

<http://www.w3.org/TR/REC-rdf-syntax/>

Prud'hommeaux E, Seaborne A (2008) *SPARQL Query Language for RDF*. W3C Recommendation.

<http://www.w3.org/TR/rdf-sparql-query/>



Jena web site: <http://jena.sourceforge.net/>

Jena API: http://jena.sourceforge.net/tutorial/RDF_API/

Jena tutorials: <http://www.ibm.com/developerworks/xml/library/j-jena/index.html>

<http://www.xml.com/pub/a/2001/05/23/jena.html>



SPARQL validator: <http://www.sparql.org/validator.html>

SPARQL implementations: <http://esw.w3.org/topic/SparqlImplementations>

SPARQL tutorials: <http://jena.sourceforge.net/ARQ/Tutorial/>

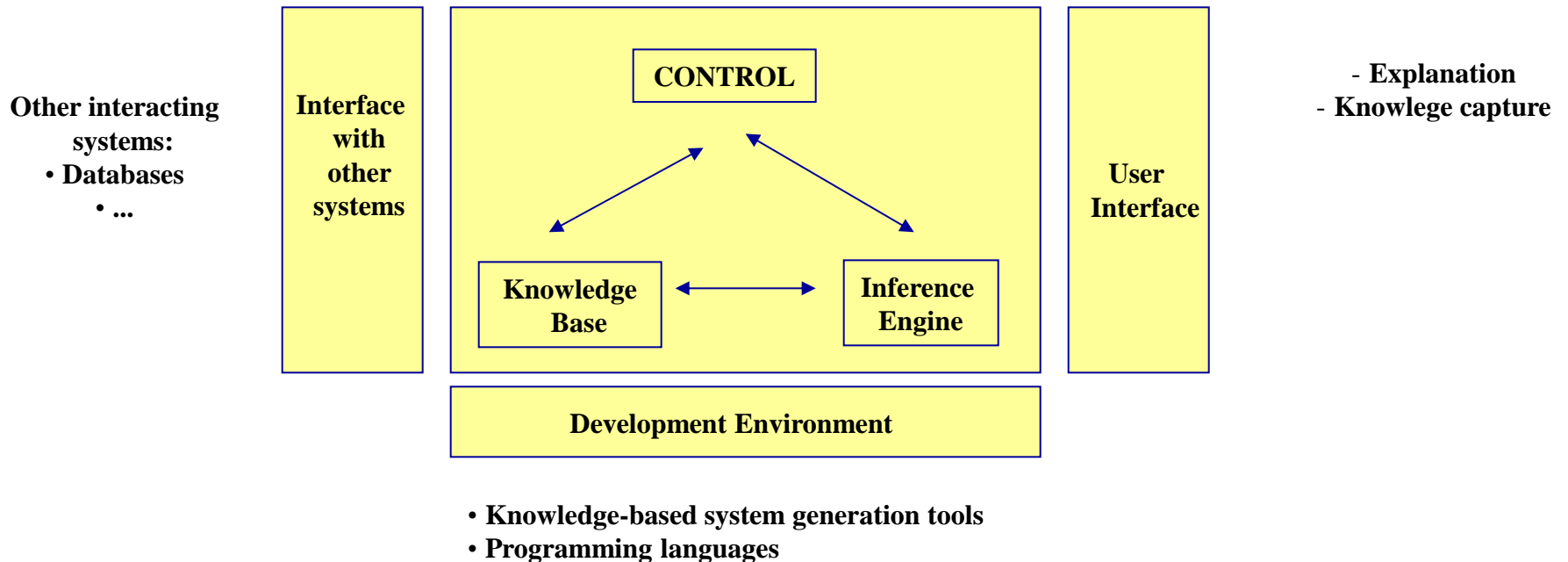
<http://www.w3.org/2004/Talks/17Dec-sparql/intro/all.html>

<http://www.cs.man.ac.uk/~bparsia/2006/row-tutorial/>

Table of Contents

1.	An introduction to knowledge representation formalisms	30'
2.	Resource Description Framework (RDF)	30'
	2.1. RDF primitives	
	2.2. Reasoning with RDF	
3.	RDF Schema	30'
	3.1 RDF Schema primitives	
	3.2 Reasoning with RDFS	
4.	RDF(S) management APIs	60'
5.	RDF(S) query languages: SPARQL	45'
6.	An example of an RDF(S)-based application	25'

Common Architecture of a Knowledge-based System



Knowledge Representation Formalisms. A Summary

- **Knowledge representation**

- To store knowledge so that programs can process it and achieve the verisimilitude of human intelligence

- **Knowledge representation formalisms/techniques**

- Originated from theories of human information processing.
- Since knowledge is used to achieve intelligent behavior, the fundamental goal of knowledge representation is to represent knowledge in a manner as to facilitate inferencing i.e. drawing conclusions from knowledge.
- Some examples are:

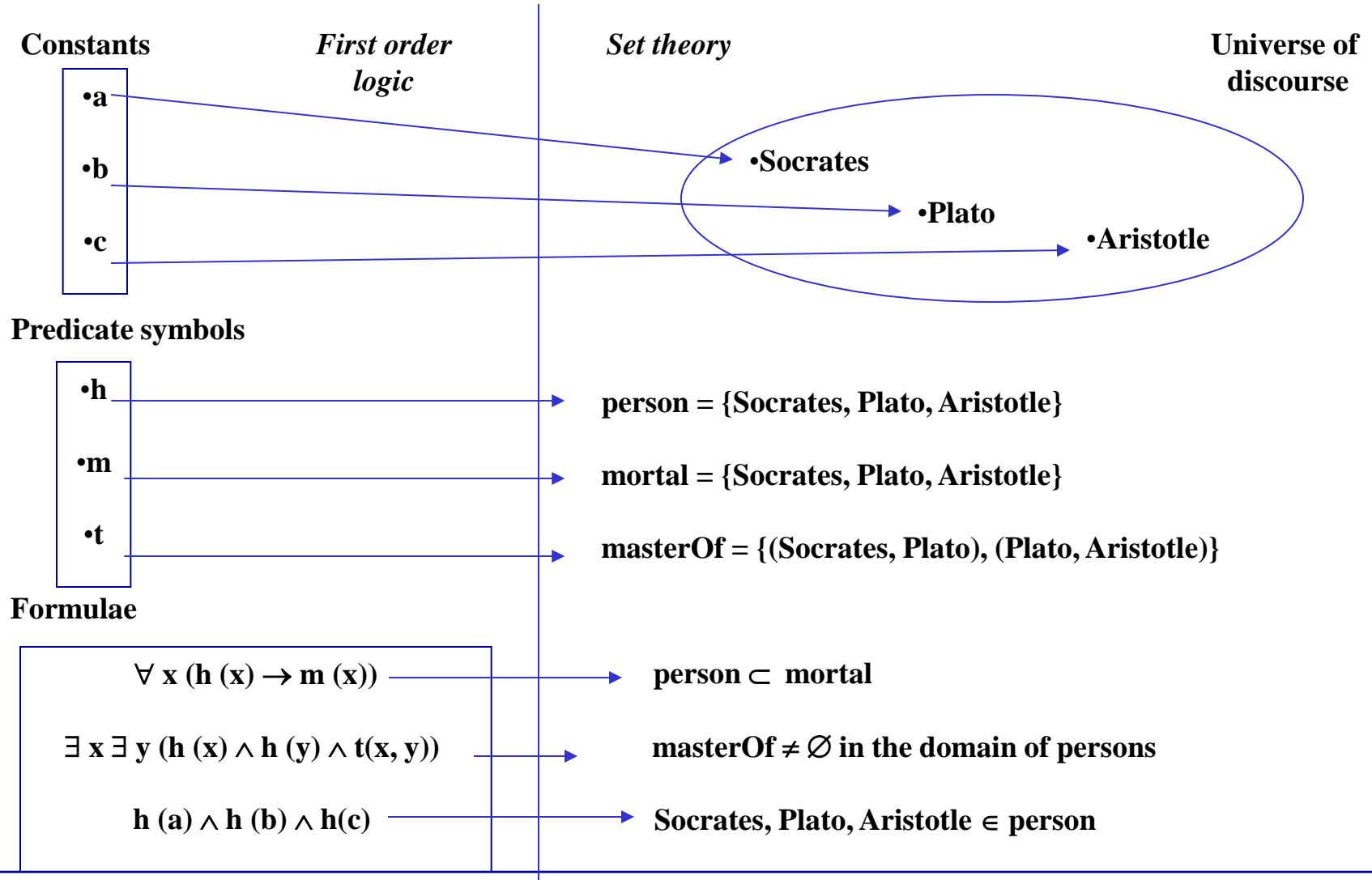
- First order logic
- Semantic networks and conceptual maps
- Frames
- Description logic
- Production rules
- Fuzzy logic
- Bayesian networks
- Etc.



These are the ones that we will analyse

First order logic. Basic elements

We can establish mappings between logical symbols and domain objects (universe of discourse)



First order logic. Formalisation

- **Se tiene un robot que distribuye paquetes en oficinas. Se sabe que:**
 - Los paquetes de la habitación 27 son más pequeños que los de la habitación 28.
 - Todos los paquetes de la misma habitación son del mismo tamaño.
 - En un instante concreto el robot sabe que:
 - i) El paquete A está en la habitación 27 ó 28 (pero no sabe en cuál).
 - ii) El paquete B está en la habitación 27.
 - iii) El paquete B no es más pequeño que el A.
 - El robot quiere probar que el paquete A está en la habitación 27.

First order logic. Formalisation. Solution

- **Se tiene un robot que distribuye paquetes en oficinas. Se sabe que:**
 - Los paquetes de la habitación 27 son más pequeños que los de la habitación 28.
 $\forall x \forall y (\text{paquete}(x) \wedge \text{situadoEn}(x, h27) \wedge \text{paquete}(y) \wedge \text{situadoEn}(y, h28) \rightarrow \text{menor}(x, y))$
 - Todos los paquetes de la misma habitación son del mismo tamaño.
 $\forall x \forall y \forall h (\text{paquete}(x) \wedge \text{paquete}(y) \wedge \text{habitacion}(h) \wedge \text{situadoEn}(x, h) \wedge \text{situadoEn}(y, h) \rightarrow \text{igual}(x, y))$
 - En un instante concreto el robot sabe que:
 - i) El paquete A está en la habitación 27 ó 28 (pero no sabe en cuál).
 $\text{paquete}(a) \wedge \text{habitacion}(h27) \wedge \text{habitacion}(h28) \wedge (\text{situadoEn}(a, h27) \vee \text{situadoEn}(a, h28))$
 - ii) El paquete B está en la habitación 27.
 $\text{paquete}(b) \wedge \text{situadoEn}(b, h27)$
 - iii) El paquete B no es más pequeño que el A.
 $\neg \text{menor}(b, a)$
 - El robot quiere probar que el paquete A está en la habitación 27.
 $\text{¿situadoEn}(a, h27)?$

Semantic Network. Basic elements

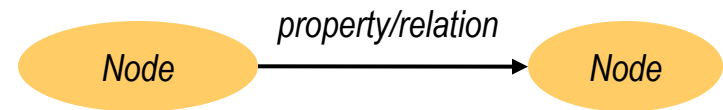
- **Nodes**

- They represent entities or concepts, or values



- **Edges**

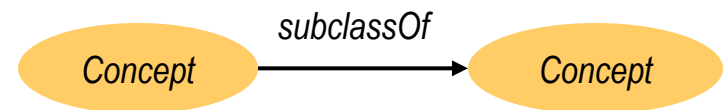
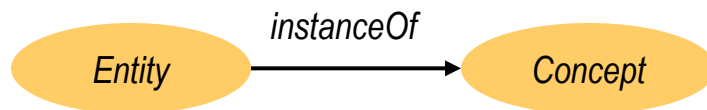
- They represent properties or relations



- **The semantics (mapping to the real world) depends on the tags used for nodes and edges**

- **There is no predefined KR vocabulary**

- Although sometimes there are *structural* edges

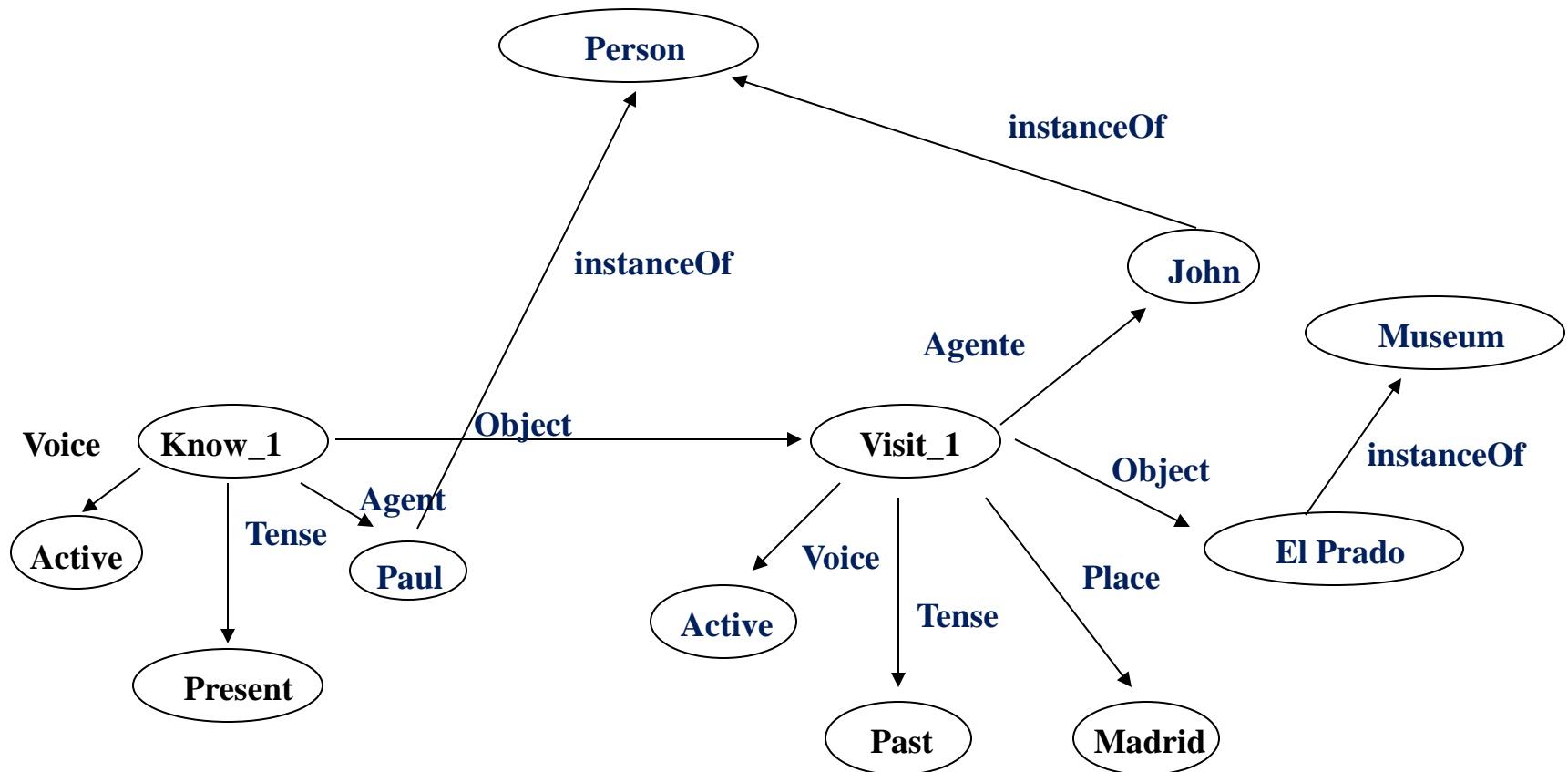


Semantic networks. Example

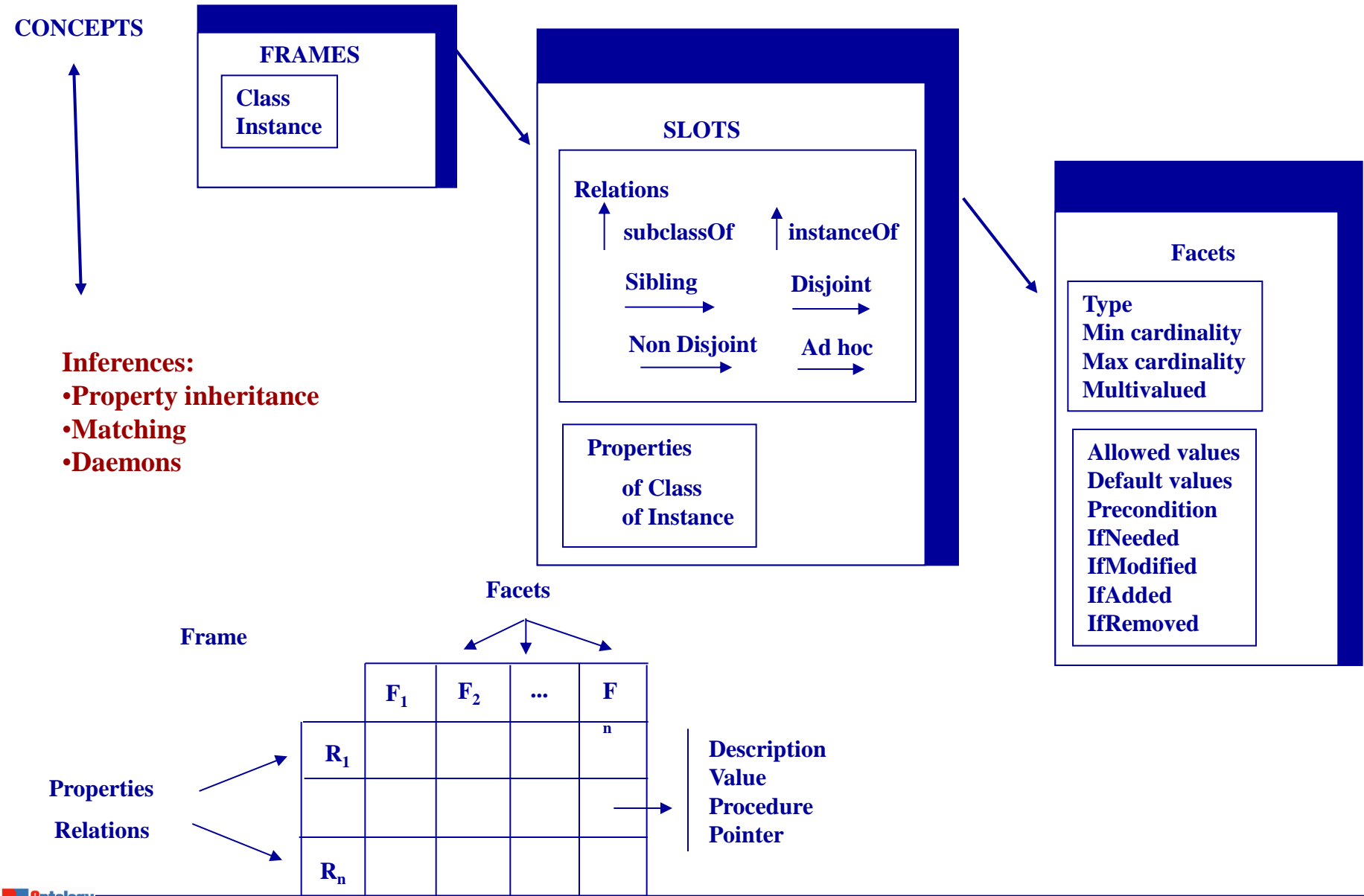
- **Paul and John are persons**
- **El Prado is a museum**
- **Paul knows that John visited El Prado in Madrid**

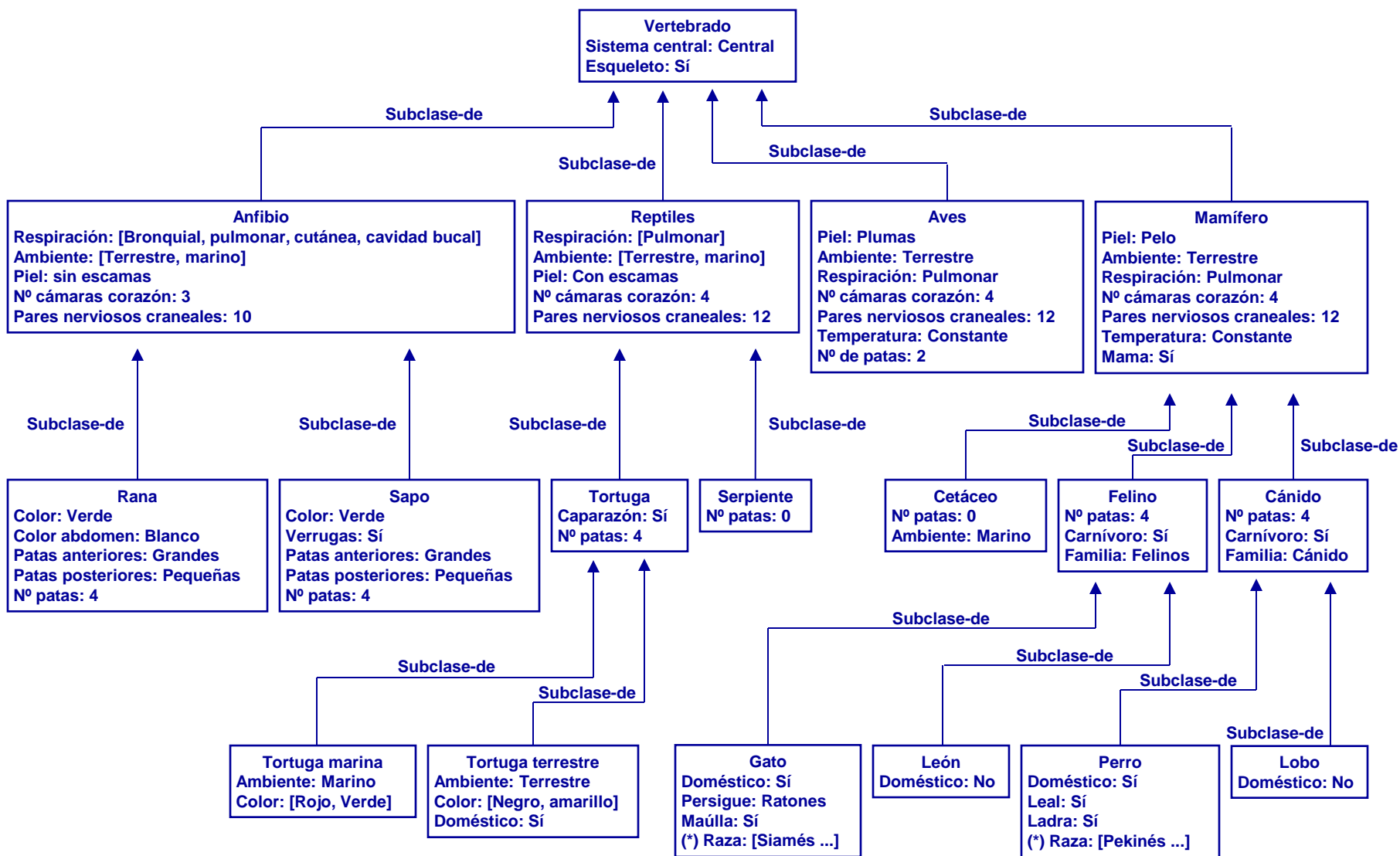
Semantic networks. Example. Solution

- Paul and John are persons
- El Prado is a museum
- Paul knows that John visited El Prado in Madrid



Frames. Basic elements



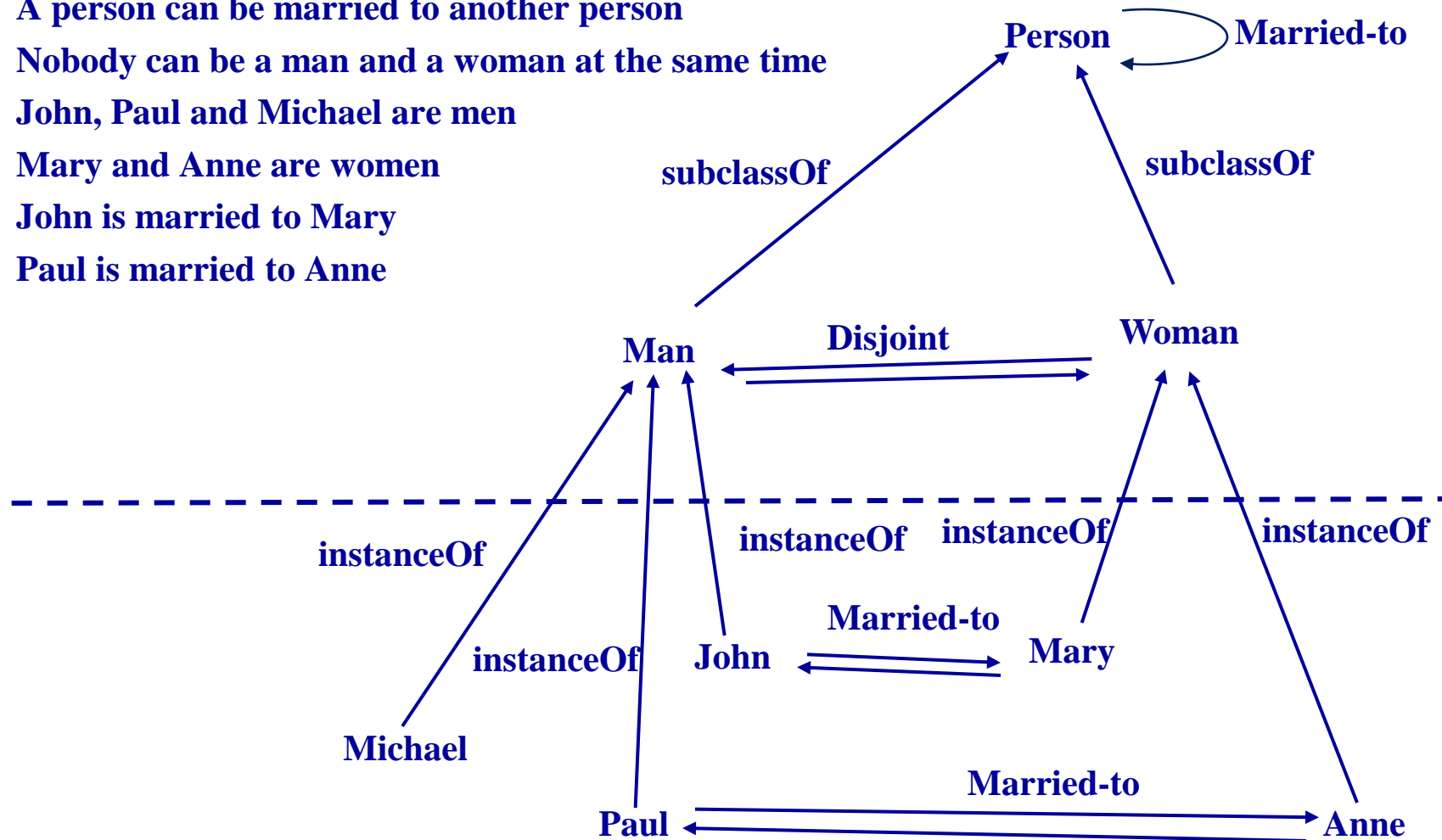


Frames. Example

- **Men and women are persons**
- **A person can be married to another person**
- **Nobody can be a man and a woman at the same time**
- **John, Paul and Michael are men**
- **Mary and Anne are women**
- **John is married to Mary**
- **Paul is married to Anne**

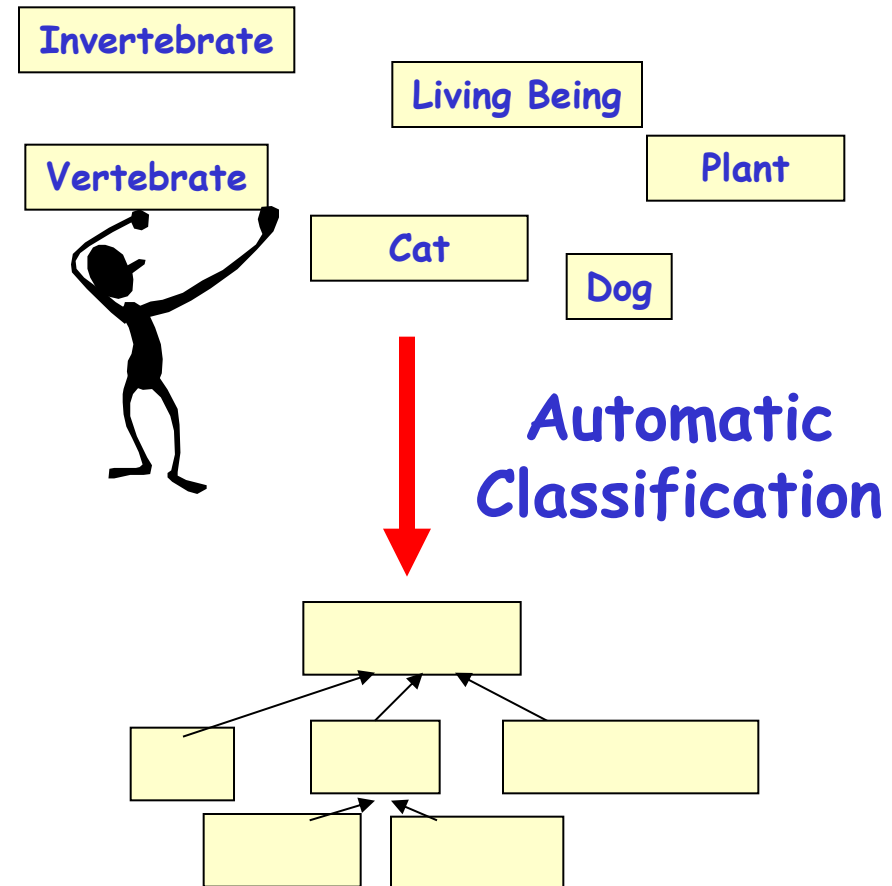
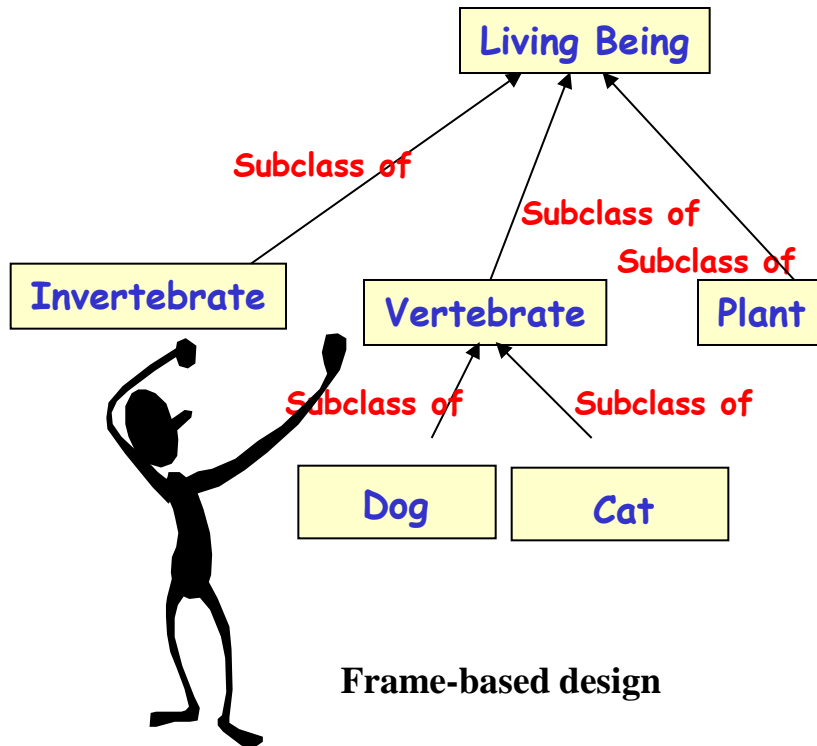
Frames. Example

- Men and women are persons
- A person can be married to another person
- Nobody can be a man and a woman at the same time
- John, Paul and Michael are men
- Mary and Anne are women
- John is married to Mary
- Paul is married to Anne



Description Logics. Basic elements

- A subset of first order logic with good reasoning properties
- **Automatic classification**



KR Formalisms

Language

Ontolingua/KIF

OKBC

OCML

LOOM

FLogic

SHOE

XOL

OIL

DAML+OIL

RDF(S)

OWL

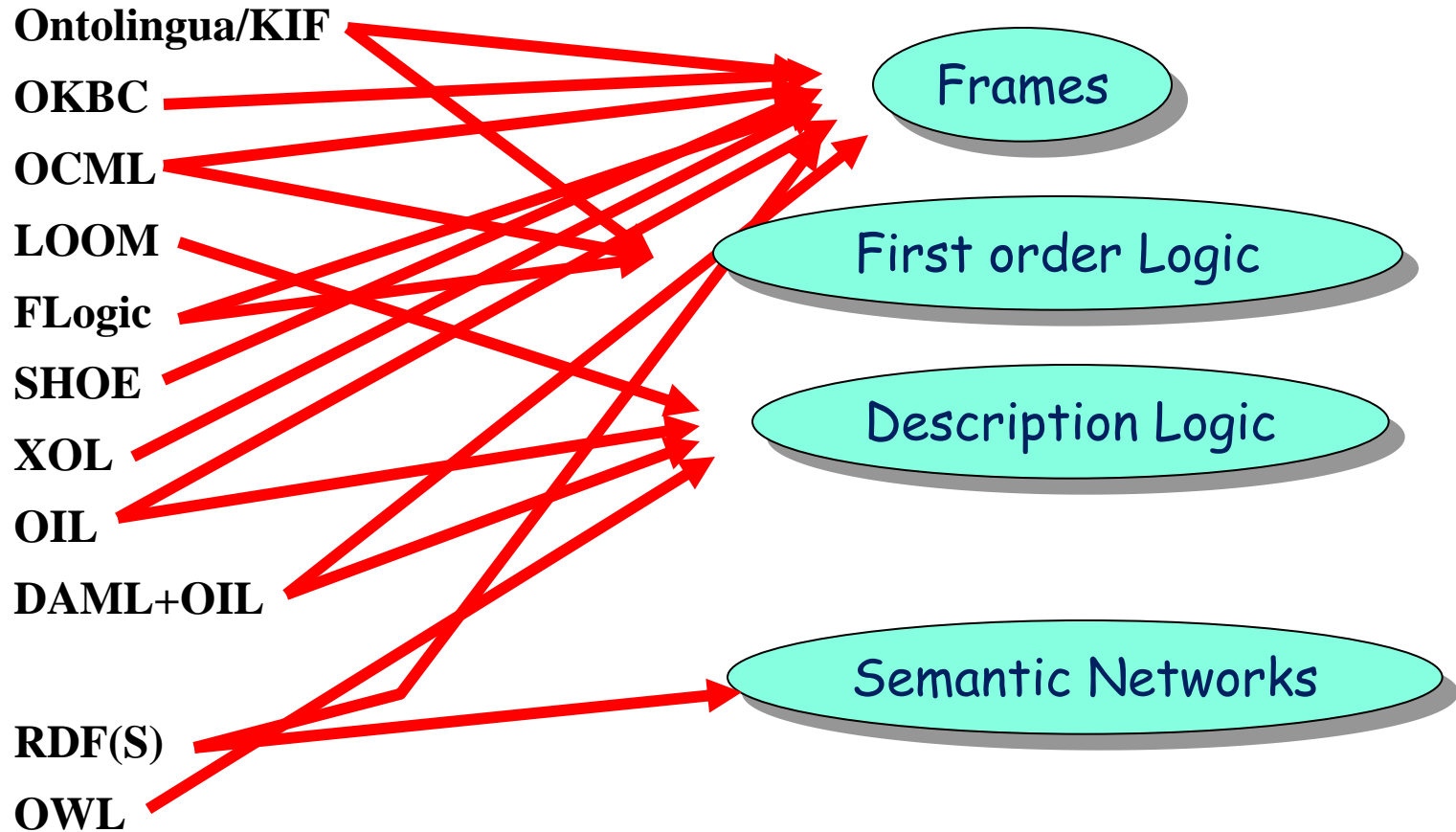
Formalism

Frames

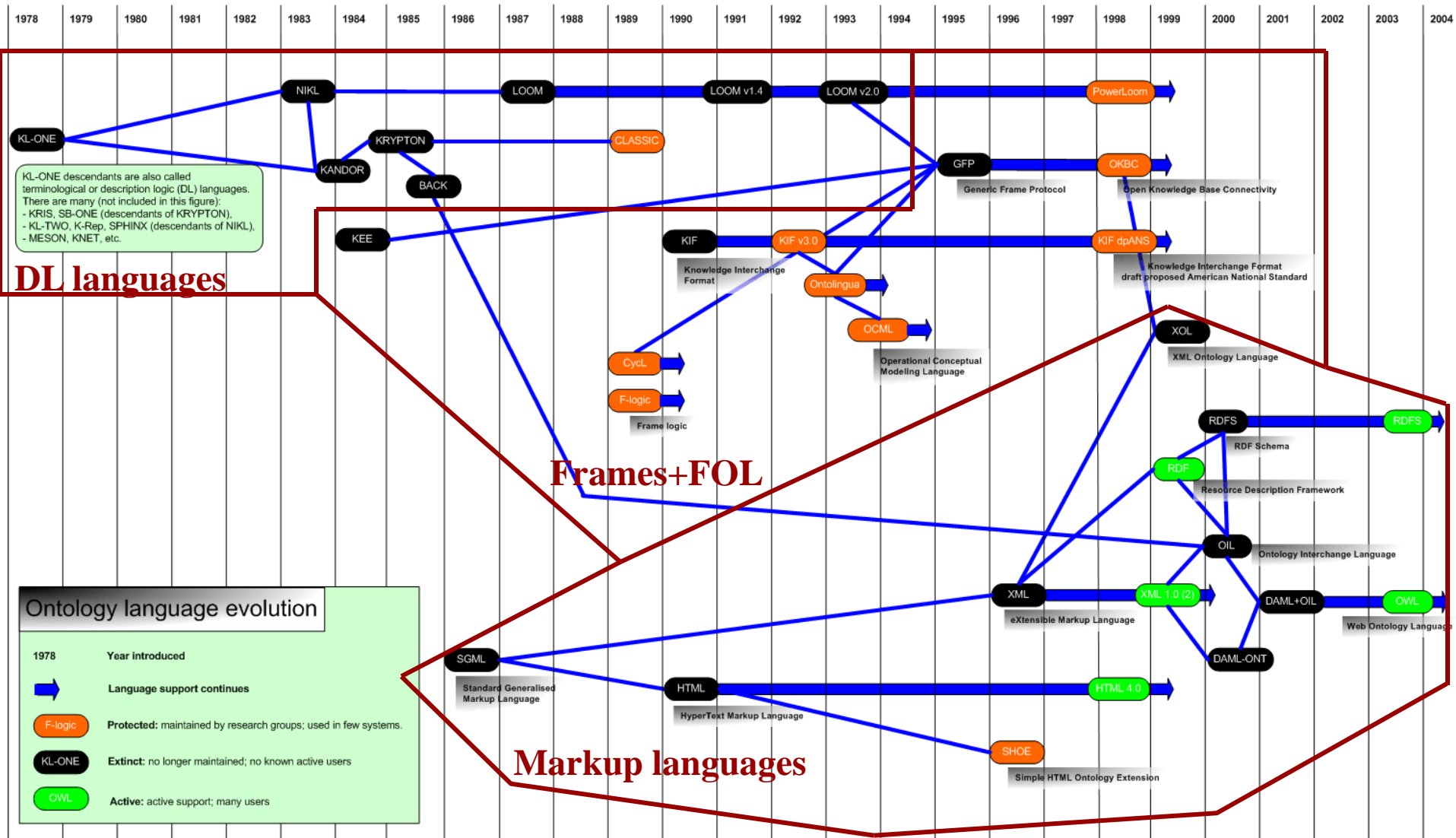
First order Logic

Description Logic

Semantic Networks



Ontology language evolution



Ontology Languages (I)

Traditional ontology languages

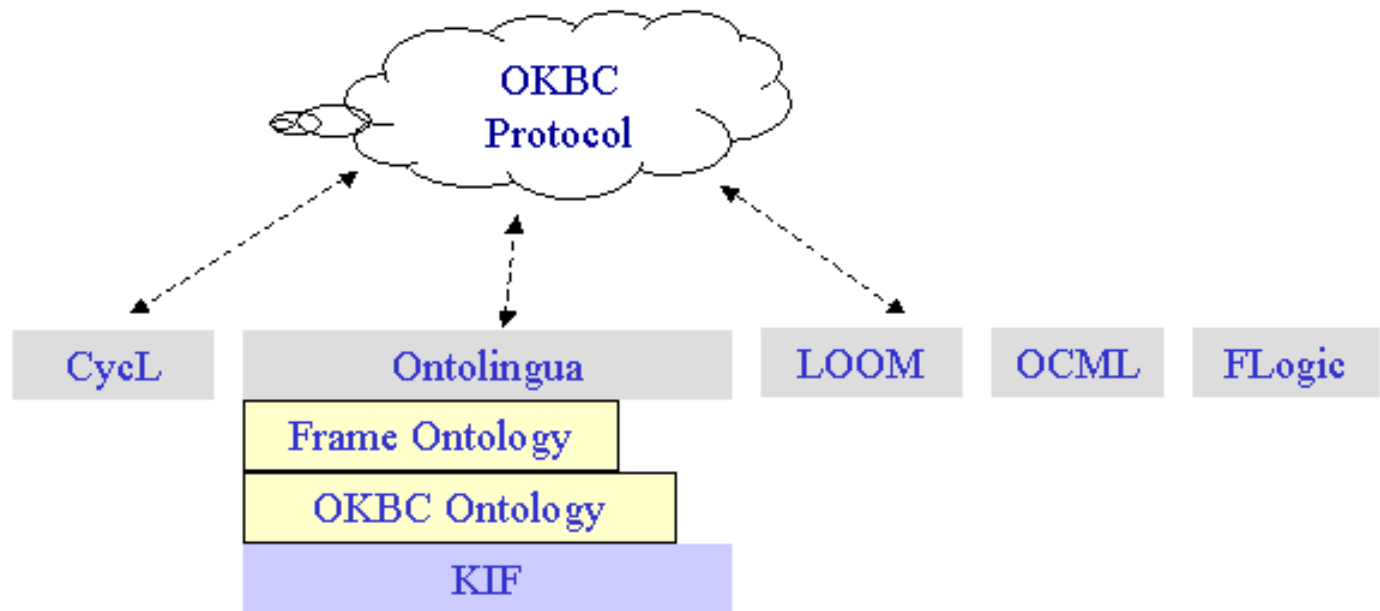
Ontolingua/KIF

OKBC

OCML

LOOM

FLogic



Ontology Languages (II)

Ontology markup languages

Standards & Recommendations of W3C

XML

RDF(S)

Ontology specification languages

SHOE

XOL

OIL

DAML+OIL

OWL

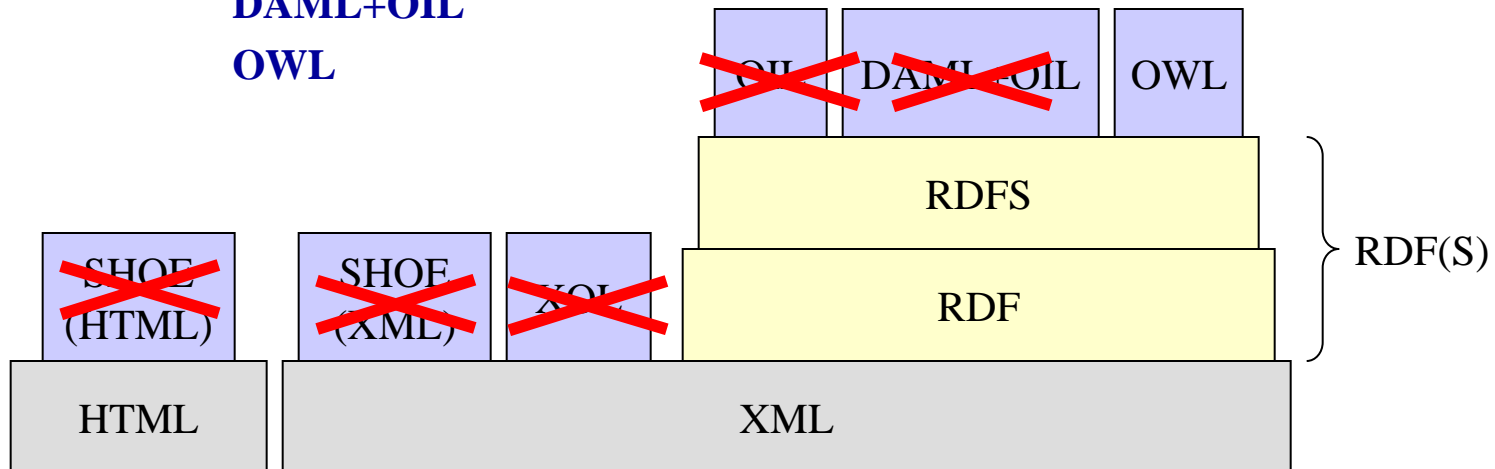
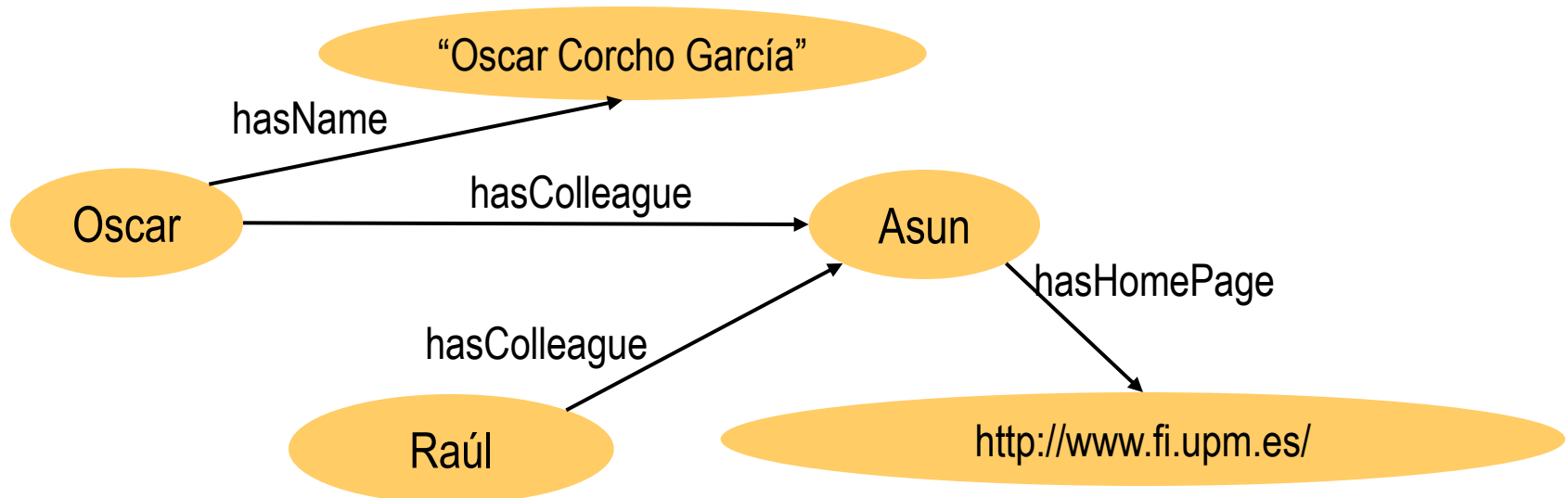
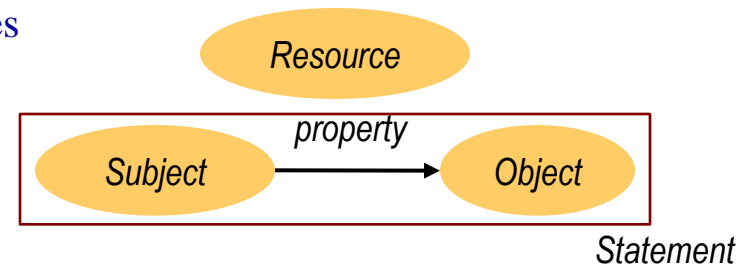


Table of Contents

1. An introduction to knowledge representation formalisms
2. **Resource Description Framework (RDF)**
 - 2.1. RDF primitives
 - 2.2. Reasoning with RDF
3. RDF Schema
 - 3.1 RDF Schema primitives
 - 3.2 Reasoning with RDFS
4. RDF(S) management APIs
5. RDF(S) query languages: SPARQL
6. An example of an RDF(S)-based application

RDF: Resource Description Framework

- W3C recommendation
- RDF is a basic KR language, based on **semantic networks**
 - Useful to represent metadata and describe any type of information in a machine-accessible way (aka data model)
 - Resources are described in terms of properties and property values using RDF statements.
 - Statements are represented as triples, consisting of a subject, predicate and object. [S, P, O]



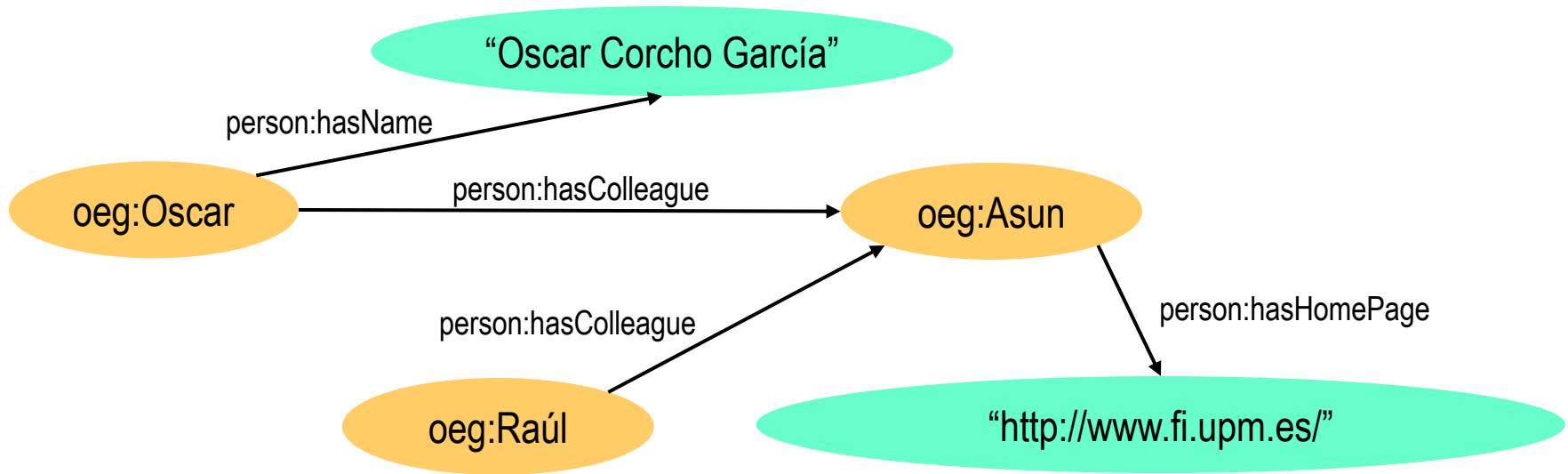
RDF (and other W3C Recommendations) and URIs

- A URI (Unique Resource Identifiers) is a Web identifier
 - e.g. `http://www.oeg-upm.net/ontologies/people#Oscar`
 - URI \neq URL
 - If we open a Web browser and point to that URI, the corresponding object will not be downloaded or shown
 - If URLs work for **locating** uniquely (with no collisions) a Web page/resource, why not using the same approach for **identifying** Web resources?
 - Other valid URIs could be
 - `ftp://www.oeg-upm.net/ontologies/people#Oscar`
 - `persons://www.oeg-upm.net/ontologies/people#Oscar`
 - ...
- URIs allow identifying
 - Individuals: `http://www.oeg-upm.net/ontologies/people#Oscar`
 - Kinds of things: `http://www.ontologies.org/ontologies/people#Person`
 - Properties of those things:

`http://www.ontologies.org/ontologies/people#hasColleague`

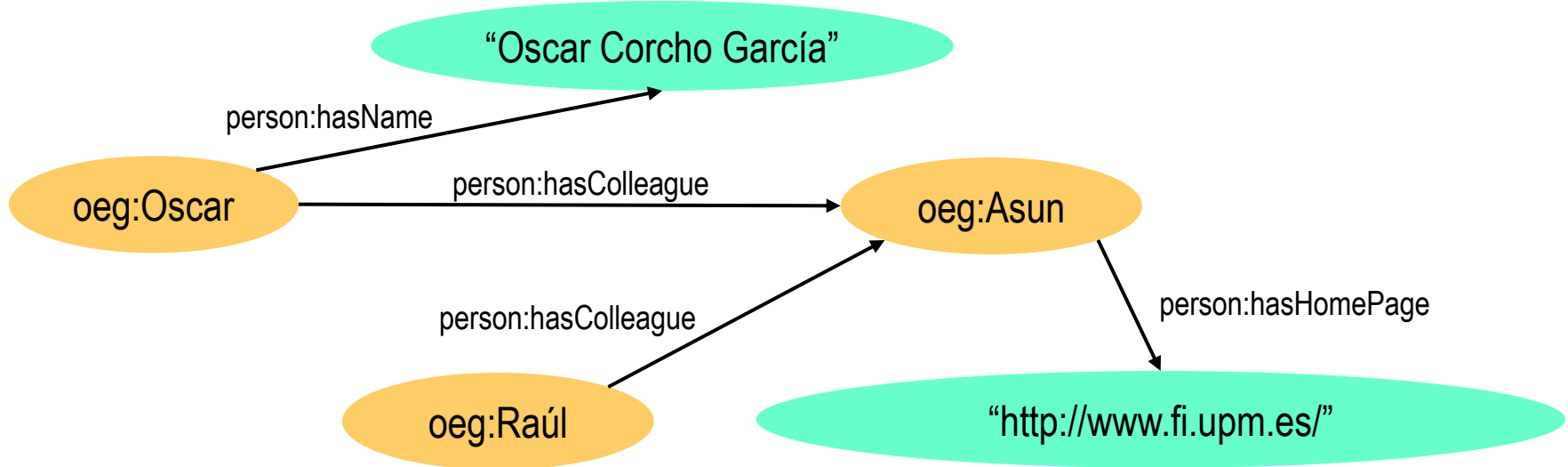
RDF (and other W3C Recommendations) and URIs

- For practical purposes, especially if handwritten, URIs are shortened using XML namespaces
 - `xmlns:oeg="http://www.oeg-upm.net/ontologies/people#"`
 - `oeg:Oscar` is equivalent to `http://www.oeg-upm.net/ontologies/people#Oscar`



RDF (and other W3C Recommendations) and URIs

- A set of URIs is sometimes known as a vocabulary
 - The RDF Vocabulary
 - The set of URIs used to describe the RDF concepts: `rdf:Property`, `rdf:Resource`, `rdf:type`, etc.
 - The RDFS Vocabulary
 - The set of URIs used in describing RDF Schema: `rdfs:Class`, `rdfs:domain`, etc.
 - The ‘Person’ Vocabulary
 - `person:hasColleague`, `person:Person`, `person:Employee`, etc.



RDF Serialisations

- **Normative**
 - RDF/XML (www.w3.org/TR/rdf-syntax-grammar/)
- **Alternative (for human consumption)**
 - N3 (<http://www.w3.org/DesignIssues/Notation3.html>)
 - Turtle (<http://www.dajobe.org/2004/01/turtle/>)
 - TriX (<http://www.w3.org/2004/03/trix/>)
 - ...

Important note: the order of RDF statements in a serialisation does not affect the behaviour of a parser/application

RDF Serialisations. RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:person="http://www.ontologies.org/ontologies/people#"
  xmlns="http://www.oeg-upm.net/ontologies/people#"
  xml:base="http://www.oeg-upm.net/ontologies/people">

  <rdf:Property rdf:about="http://www.ontologies.org/ontologies/people#hasHomePage"/>
  <rdf:Property rdf:about="http://www.ontologies.org/ontologies/people#hasColleague"/>
  <rdf:Property rdf:about="http://www.ontologies.org/ontologies/people#hasName"/>

  <rdf:Description rdf:about="#Raúl"/>
  <rdf:Description rdf:about="#Asun">
    <person:hasColleague rdf:resource="#Raúl"/>
    <person:hasHomePage>http://www.fi.upm.es</person:hasHomePage>
  </rdf:Description>
  <rdf:Description rdf:about="#Oscar">
    <person:hasColleague rdf:resource="#Asun"/>
    <person:hasName>Oscar Corcho García</person:hasName>
  </rdf:Description>

</rdf:RDF>
```

RDF Serialisations. N3

```
@base <http://www.oeg-upm.net/ontologies/people >  
@prefix person: <http://www.ontologies.org/ontologies/people#>  
:Asun  person:hasColleague :Raúl ;  
        person:hasHomePage "http://www.fi.upm.es/".  
:Oscar  person:hasColleague :Asun ;  
        person:hasName "Oscar Corcho García".
```

Exercise



•Objective

- Get used to the different syntaxes of RDF

•Tasks

- Take the text of an RDF file and create its corresponding graph
- Take an RDF graph and create its corresponding RDF/XML and N3 files



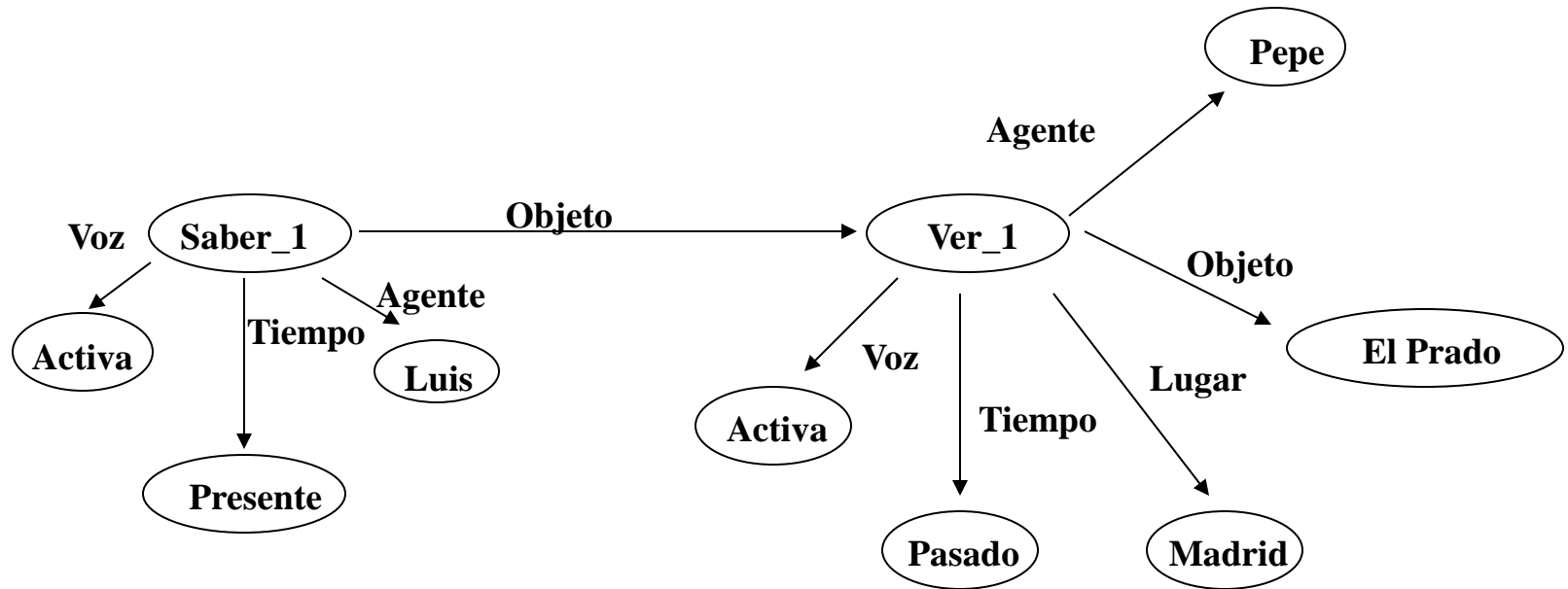
Exercise 1.a. Create a graph from a text file

- **Open the file StickyNote_PureRDF.rdf**
- **Create the corresponding graph from it**
- **Compare your graph with those of your colleagues**



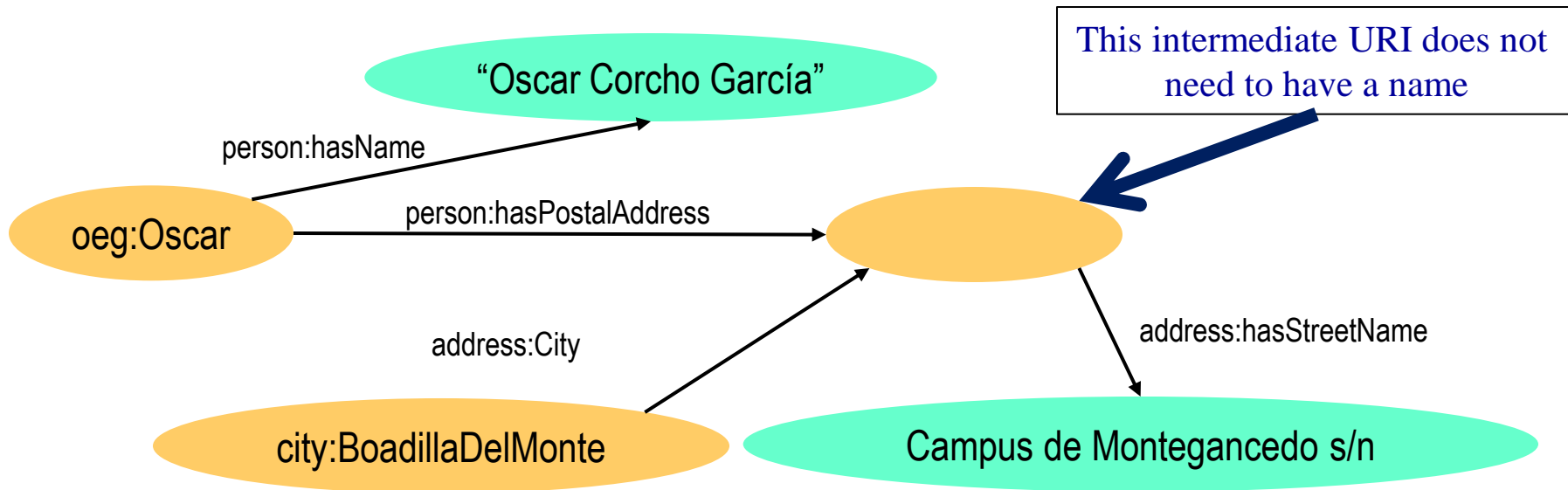
Exercise 1.b. Create RDF/XML and N3 text files from an RDF graph

- Transform the following graph into RDF/XML and N3 syntaxes



Blank nodes: structured property values

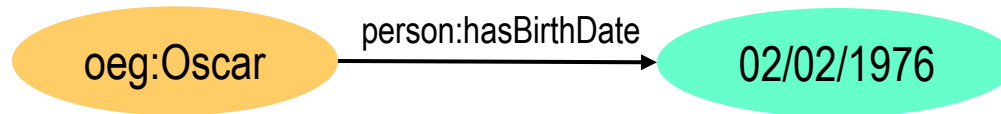
- Most real-world data involves structures that are more complicated than sets of RDF triple statements



- In RDF/XML, it is an `<rdf:Description>` node with no `rdf:about`
- In N3, it is an `_:oscarAddress`

Typed literals

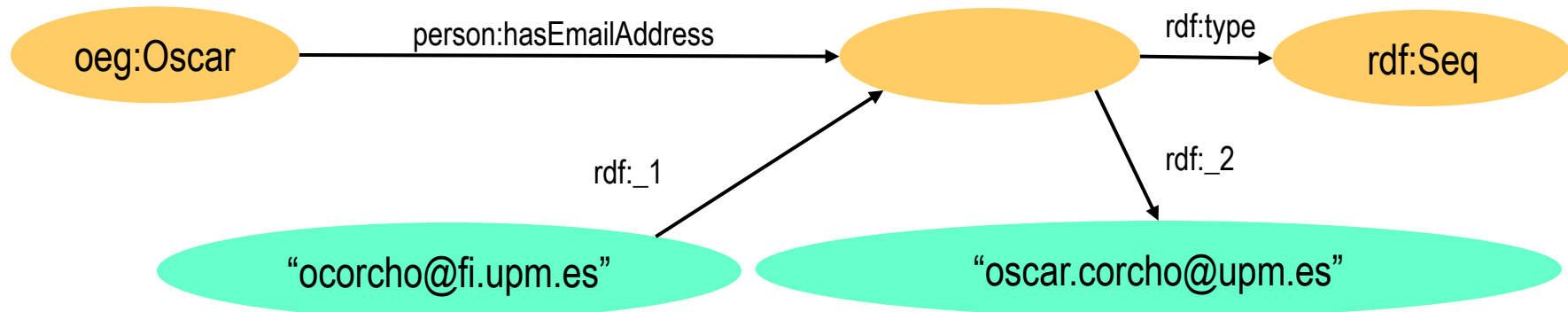
- So far, all values have been presented as strings
- XML Schema datatypes can be used to specify values (objects in some RDF triple statements)



- In RDF/XML, this is expressed as:
 - `<rdf:Description rdf:about="#Oscar">`
 `<person:hasBirthDate`
 `rdf:datatype="http://www.w3.org/2001/XMLSchema#date">02/02/1976`
 `</person:hasBirthDate>`
 `</rdf:Description>`
- In N3, this is expressed as:
 - `oeg:Oscar person:hasBirthDate "02/02/1976"^^xsd:date .`

RDF Containers

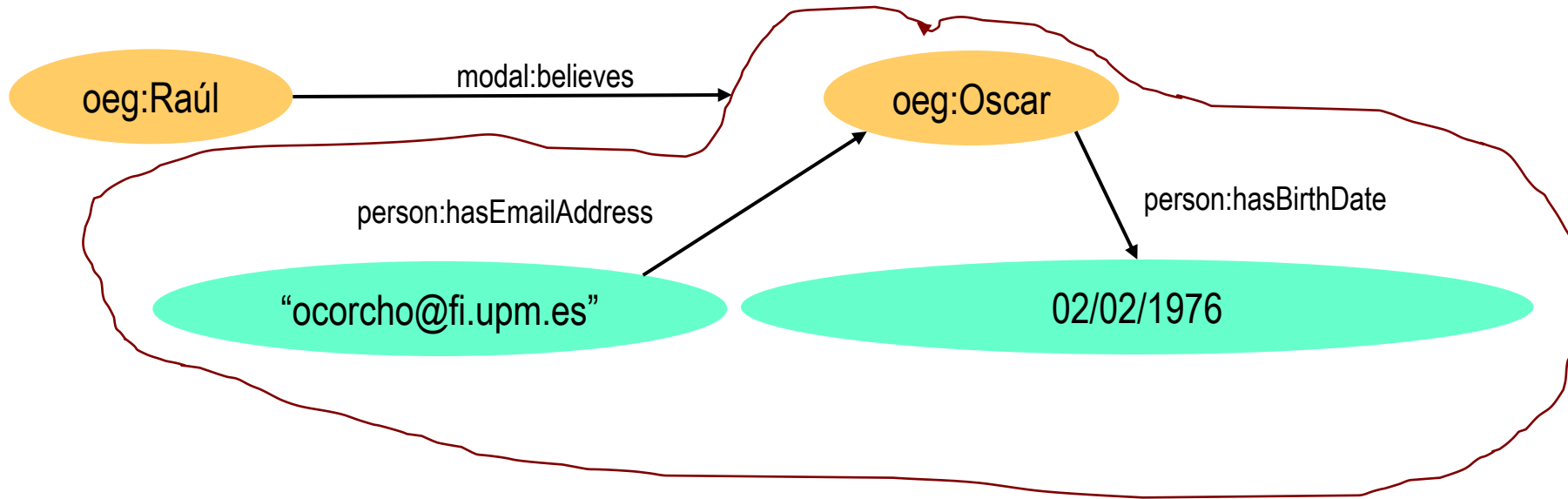
- **There is often the need to describe groups of things**
 - A book was created by several authors
 - A lesson is taught by several persons
 - etc.
- **RDF provides a container vocabulary**
 - `rdf:Bag` → A group of resources or literals, possibly including duplicate members, where the order of members is not significant.
 - `rdf:Seq` → A group of resources or literals, possibly including duplicate members, where the order of members is significant.
 - `rdf:Alt` → A group of resources or literals that are alternatives (typically for a single value of a property).



RDF Reification

- **RDF statements about other RDF statements**

- “Raúl believes that Oscar’s birthdate is on Feb 2nd, 1976 and that his e-mail address is ocorcho@fi.upm.es”



- **RDF Reification**

- Allows expressing beliefs (and other modalities)
- Allows expressing trust models, digital signatures, etc.
- Allows expressing metadata about metadata

Main value of a structured value (scarcely used)

- **Sometimes one of the values of a structured value is the main one**
 - The weight of an item is 2.4 kilograms.
 - The most important value is 2.4, which is expressed with `rdf:value`

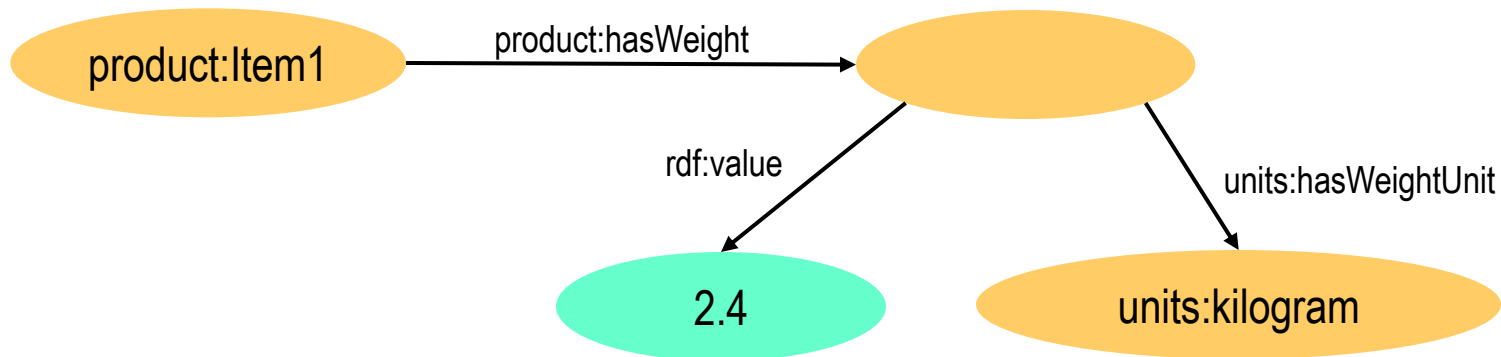


Table of Contents

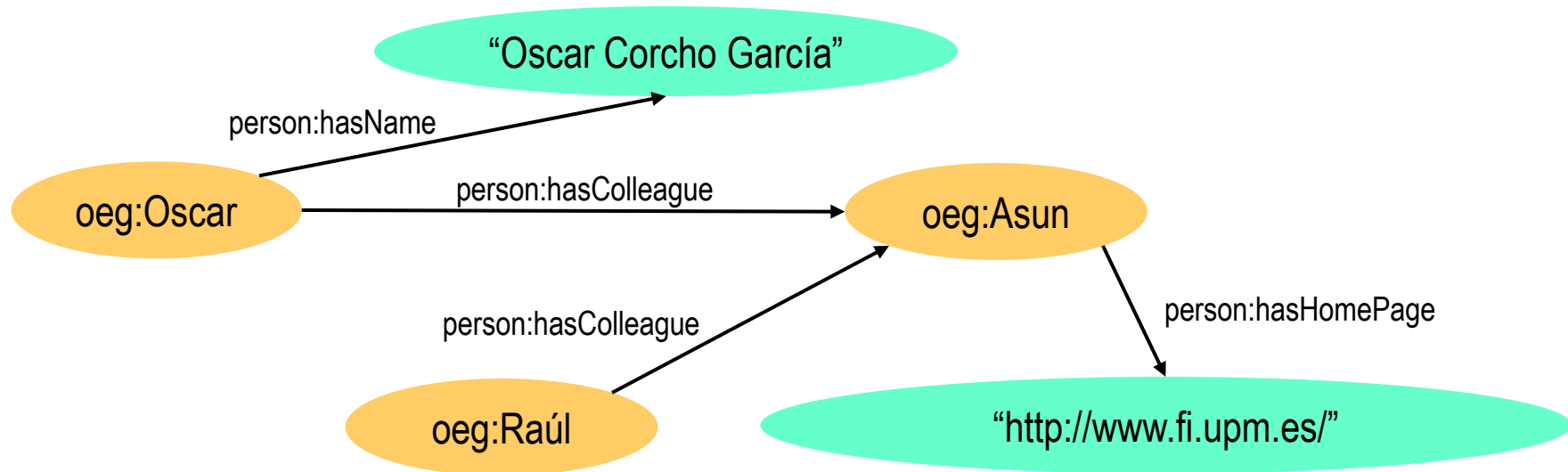
1. An introduction to knowledge representation formalisms
2. **Resource Description Framework (RDF)**
 - 2.1. RDF primitives
 - 2.2. Reasoning with RDF
3. RDF Schema
 - 3.1 RDF Schema primitives
 - 3.2 Reasoning with RDFS
4. RDF(S) management APIs
5. RDF(S) query languages: SPARQL
6. An example of an RDF(S)-based application

RDF inference. Graph matching techniques

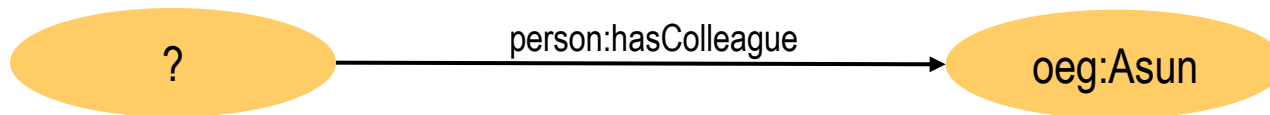
- **RDF inference is based on graph matching techniques**
- **Basically, the RDF inference process consists of the following steps:**
 - Transform an RDF query into a template graph that has to be matched against the RDF graph
 - It contains constant and variable nodes, and constant and variable edges between nodes.
 - Match against the RDF graph, taking into account constant nodes and edges.
 - Provide a solution for variable nodes and edges.

RDF inference. Examples (I)

- **Sample RDF graph**



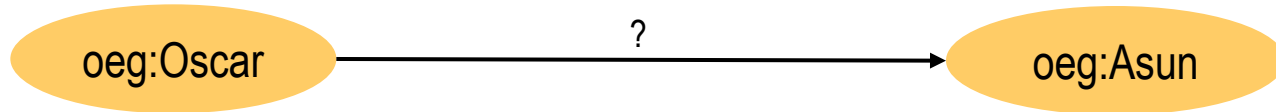
- **Query: “Tell me who are the persons who have Asun as a colleague”**



– Result: `oeg:Oscar` and `oeg:Raúl`

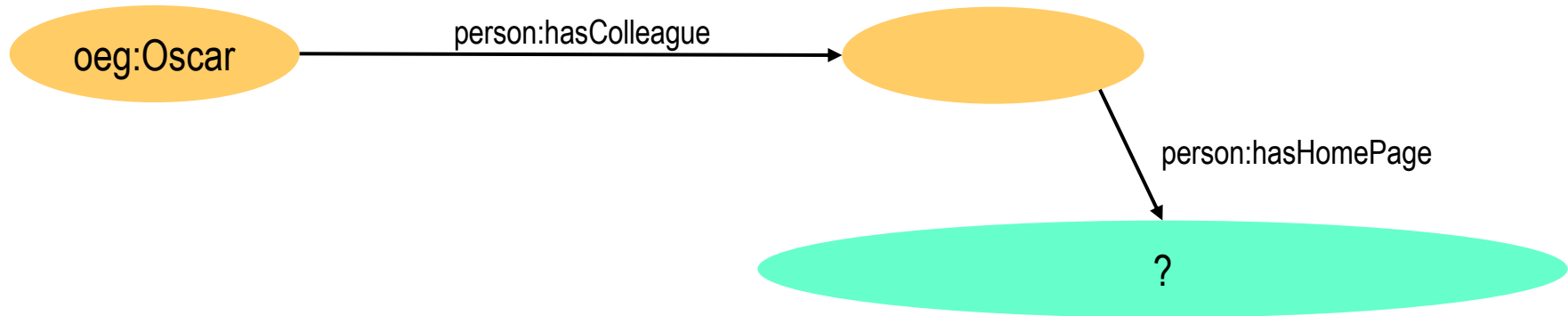
RDF inference. Examples (II)

- Query: “Tell me which are the relationships between Oscar and Asun”



- Result: oeg:hasColleague

- Query: “Tell me the homepage of Oscar colleagues”



- Result: “http://www.fi.upm.es/”

RDF inference. Entailment rules

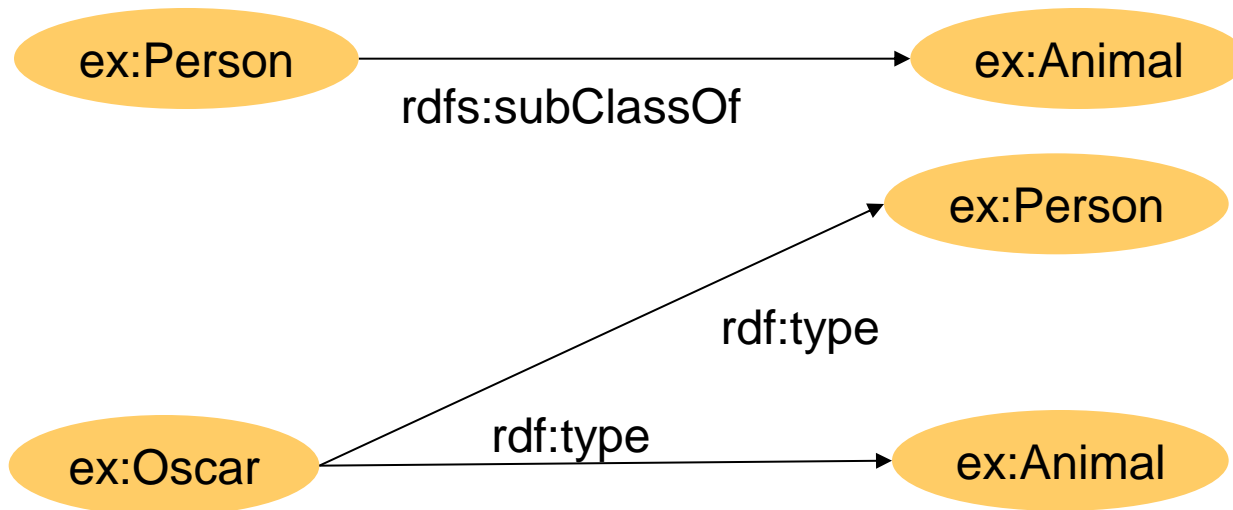
Rule Name	if E contains	then add
rdf1	uuu aaa yyy .	aaa rdf:type rdf:Property .
rdf2	uuu aaa lll . where lll is a well-typed XML literal .	_:nnn rdf:type rdf:XMLLiteral . where _:nnn identifies a blank node allocated to lll by rule lg.

Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
 - 2.1. RDF primitives
 - 2.2. Reasoning with RDF
3. **RDF Schema**
 - 3.1 RDF Schema primitives
 - 3.2 Reasoning with RDFS
4. RDF(S) management APIs
5. RDF(S) query languages: SPARQL
6. An example of an RDF(S)-based application

RDFS: RDF Schema

- **W3C Recommendation**
- **RDF Schema extends RDF to enable talking about classes of resources, and the properties to be used with them.**
 - Class definition: `rdfs:Class`, `rdfs:subClassOf`
 - Property definition: `rdfs:subPropertyOf`, `rdfs:range`, `rdfs:domain`
 - Other primitives: `rdfs:comment`, `rdfs:label`, `rdfs:seeAlso`, `rdfs:isDefinedBy`
- **RDFS vocabulary adds constraints on models, e.g.:**
 - $\forall x,y,z \text{ type}(x,y) \text{ and } \text{subClassOf}(y,z) \rightarrow \text{type}(x,z)$



RDF(S) = RDF + RDF Schema. RDF/XML syntax

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:person="http://www.ontologies.org/ontologies/people#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns="http://www.oeg-upm.net/ontologies/people#"
  xml:base="http://www.oeg-upm.net/ontologies/people">
  <rdfs:Class rdf:about="http://www.ontologies.org/ontologies/people#Professor">
    <rdfs:subClassOf>
      <rdfs:Class rdf:about="http://www.ontologies.org/ontologies/people#Person"/>
    </rdfs:subClassOf>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://www.ontologies.org/ontologies/people#Lecturer">
    <rdfs:subClassOf rdf:resource="http://www.ontologies.org/ontologies/people#Person"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://www.ontologies.org/ontologies/people#PhDStudent">
    <rdfs:subClassOf rdf:resource="http://www.ontologies.org/ontologies/people#Person"/>
  </rdfs:Class>

  ...
```

RDF(S) = RDF + RDF Schema. RDF/XML syntax

...

```
<rdf:Property rdf:about="http://www.ontologies.org/ontologies/people#hasHomePage"/>
<rdf:Property rdf:about="http://www.ontologies.org/ontologies/people#hasColleague">
  <rdfs:domain rdf:resource=" http://www.ontologies.org/ontologies/people#Person"/>
  <rdfs:range rdf:resource=" http://www.ontologies.org/ontologies/people#Person"/>
</rdf:Property>
<rdf:Property rdf:about="http://www.ontologies.org/ontologies/people#hasName">
  <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</rdf:Property>

<person:PhDStudent rdf:ID="Raúl"/>
<person:Professor rdf:ID="Asun">
  <person:hasColleague rdf:resource="#Raúl"/>
  <person:hasHomePage>http://www.fi.upm.es</person:hasHomePage>
</person:Professor>
<person:Lecturer rdf:ID="Oscar">
  <person:hasColleague rdf:resource="#Asun"/>
  <person:hasName>Oscar Corcho García</person:hasName>
</person:Lecturer>
</rdf:RDF>
```

RDF(S) Serialisations. N3

@base <<http://www.oeg-upm.net/ontologies/people> >

@prefix person: <<http://www.ontologies.org/ontologies/people#>>

person:hasColleague **a** **rdf:Property**;
 rdfs:domain **person:Person**;
 rdfs:range **person:Person**.

person:Professor **rdfs:subClassOf** **person:Person**.

person:Lecturer **rdfs:subClassOf** **person:Person**.

person:PhDStudent **rdfs:subClassOf** **person:Person**.

:Asun **a** **person:Professor**;
 person:hasColleague **:Raúl** ;
 person:hasHomePage “<http://www.fi.upm.es/>”.

:Oscar **a** **person:Lecturer**;
 person:hasColleague **:Asun** ;
 person:hasName “**Oscar Corcho García**”.

:Raúl **a** **person:PhDStudent**.

a is equivalent to **rdf:type**

Exercise



•Objective

- Get used to the different syntaxes of RDF(S)

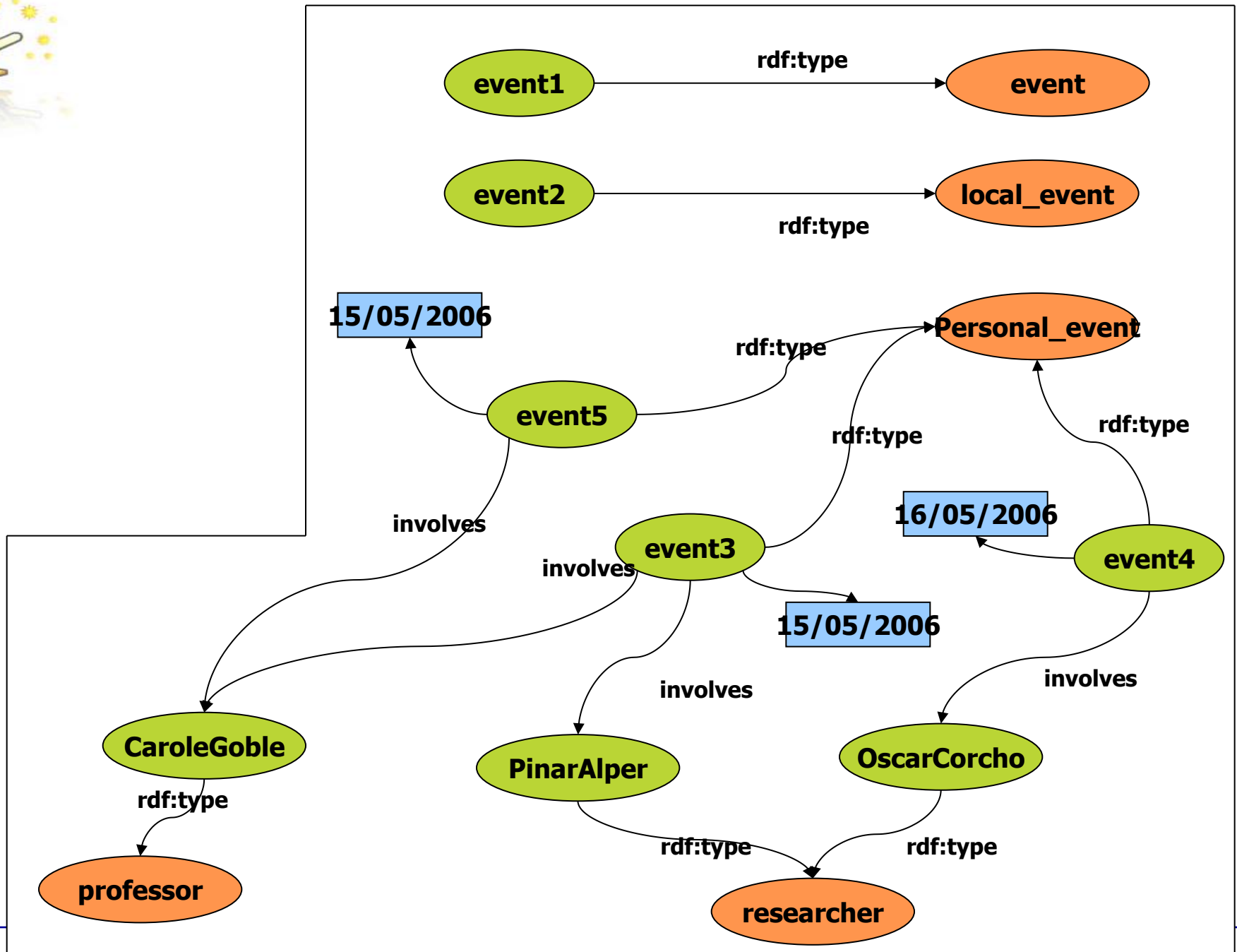
•Tasks

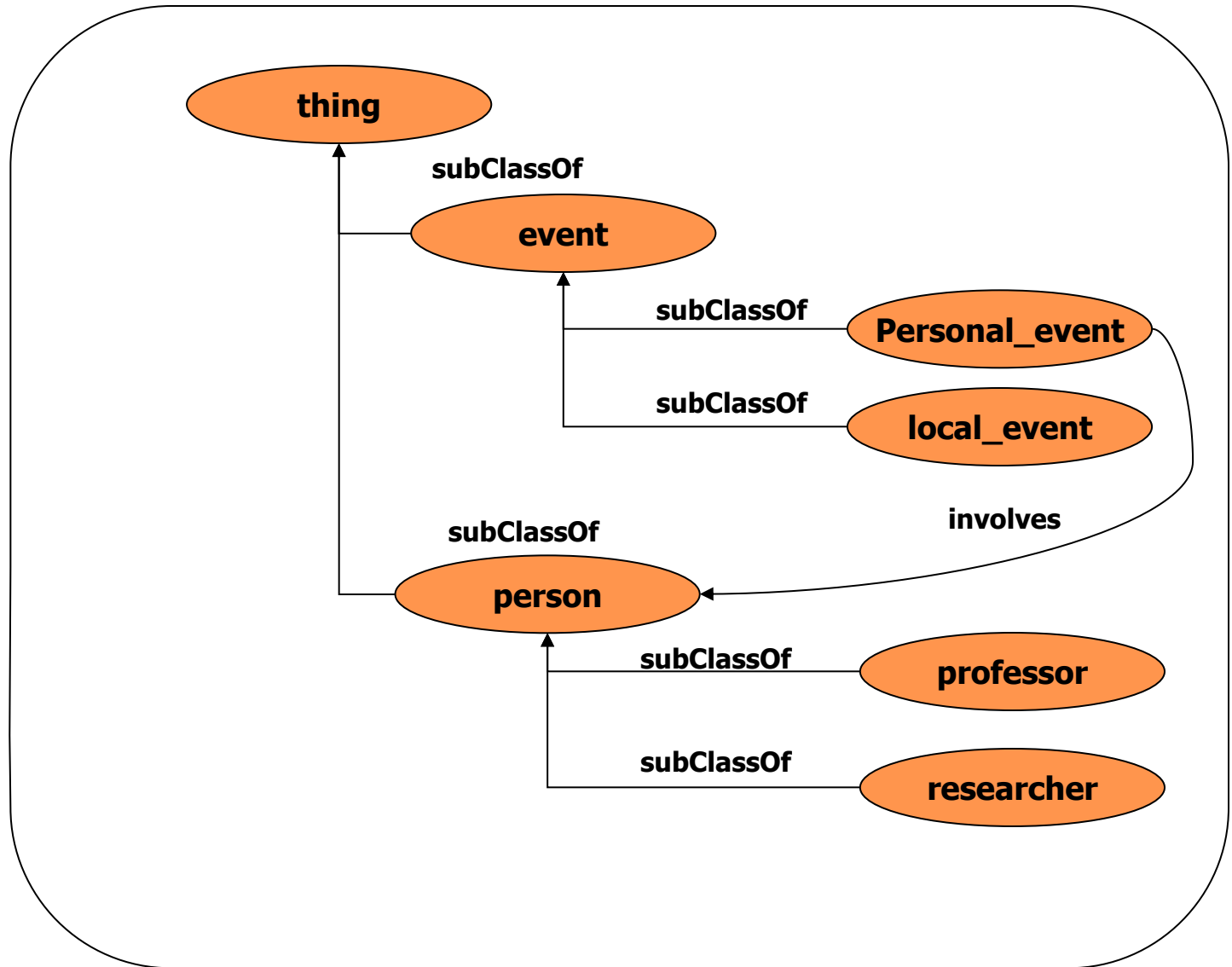
- Take the text of an RDF(S) file and create its corresponding graph
- Take an RDF(S) graph and create its corresponding RDF/XML and N3 files



Exercise 2.a. Create a graph from a text file

- **Open the files StickyNote.rdf and StickyNote.rdfs**
- **Create the corresponding graph from them**
- **Compare your graph with those of your colleagues**







Exercise 2.b. Create RDF/XML and N3 text files from an RDF(S) graph

- Transform the following graph into RDF/XML and N3 syntaxes

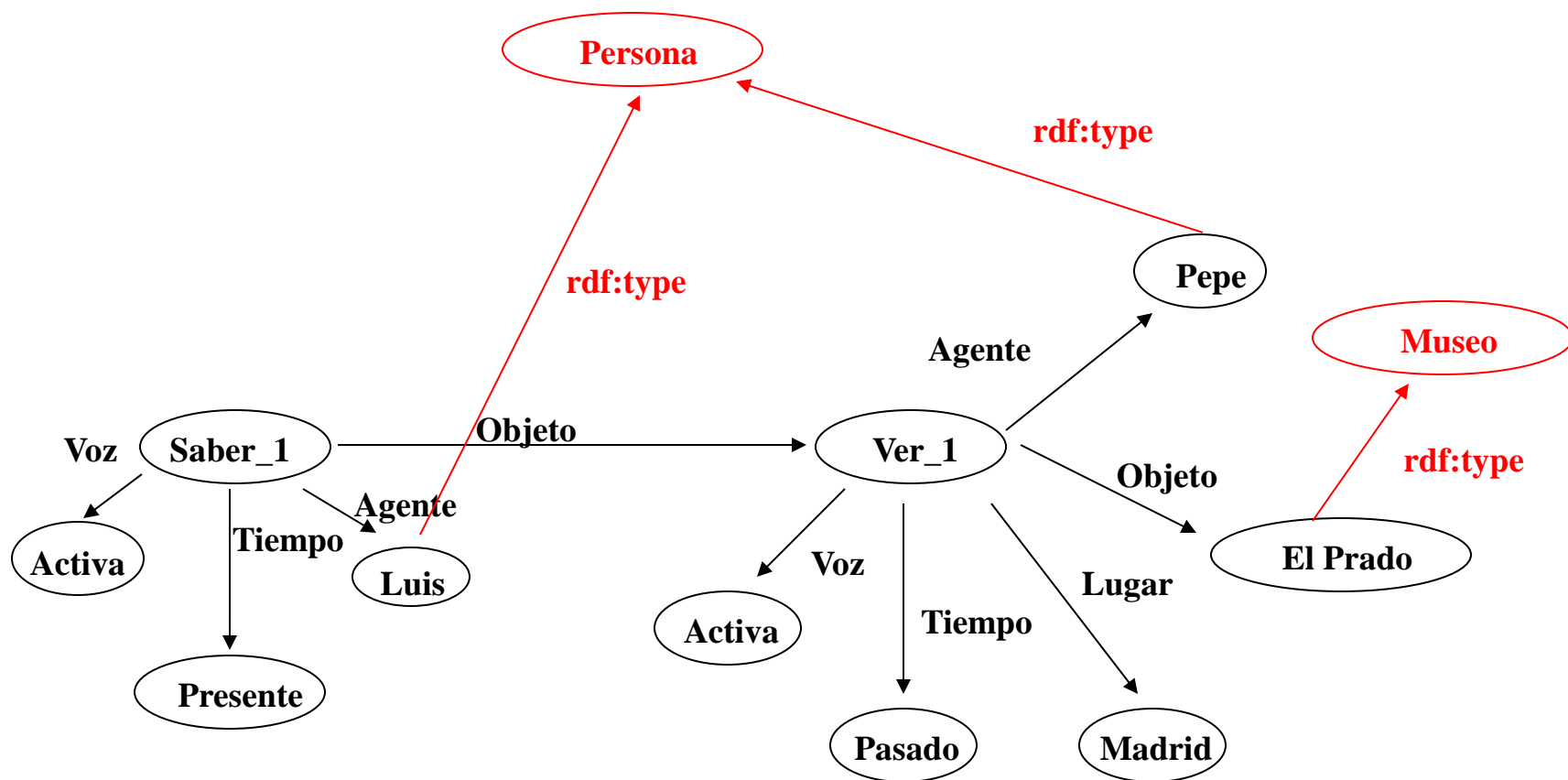


Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
 - 2.1. RDF primitives
 - 2.2. Reasoning with RDF
3. **RDF Schema**
 - 3.1 RDF Schema primitives
 - 3.2 **Reasoning with RDFS**
4. RDF(S) management APIs
5. RDF(S) query languages: SPARQL
6. An example of an RDF(S)-based application

RDF(S) inference. Entailment rules

Rule Name	If E contains:	then add:
rdfs1	uuu aaa lll. where lll is a plain literal (with or without a language tag).	_ :nnn rdf:type rdfs:Literal . where _ :nnn identifies a blank node allocated to lll by rule lg.
rdfs2	aaa rdfs:domain XXX . uuu aaa yyy .	UUU rdf:type XXX .
rdfs3	aaa rdfs:range XXX . uuu aaa vv .	VV rdf:type XXX .
rdfs4a	uuu aaa xxx .	UUU rdf:type rdfs:Resource .
rdfs4b	uuu aaa vv .	VV rdf:type rdfs:Resource .
rdfs5	UUU rdfs:subPropertyOf vv . vv rdfs:subPropertyOf xxx .	UUU rdfs:subPropertyOf xxx .
rdfs6	UUU rdf:type rdf:Property .	UUU rdfs:subPropertyOf UUU .
rdfs7	aaa rdfs:subPropertyOf bbb . uuu aaa yyy .	uuu bbb yyy .
rdfs8	UUU rdf:type rdfs:Class .	UUU rdfs:subClassOf rdfs:Resource .
rdfs9	UUU rdfs:subClassOf xxx . vv rdf:type UUU .	VV rdf:type xxx .
rdfs10	UUU rdf:type rdfs:Class .	UUU rdfs:subClassOf UUU .
rdfs11	UUU rdfs:subClassOf vv . vv rdfs:subClassOf xxx .	UUU rdfs:subClassOf xxx .
rdfs12	UUU rdf:type rdfs:ContainerMembershipProperty .	UUU rdfs:subPropertyOf rdfs:member .
rdfs13	UUU rdf:type rdfs:Datatype .	UUU rdfs:subClassOf rdfs:Literal .

RDF(S) inference. Additional inferences

ext1	UUU rdfs:domain WW . WW rdfs:subClassOf ZZZ .	UUU rdfs:domain ZZZ .
ext2	UUU rdfs:range WW . WW rdfs:subClassOf ZZZ .	UUU rdfs:range ZZZ .
ext3	UUU rdfs:domain WW . WWW rdfs:subPropertyOf UUU .	WWW rdfs:domain WW .
ext4	UUU rdfs:range WW . WWW rdfs:subPropertyOf UUU .	WWW rdfs:range WW .
ext5	rdf:type rdfs:subPropertyOf WWW . WWW rdfs:domain WW .	rdfs:Resource rdfs:subClassOf WW .
ext6	rdfs:subClassOf rdfs:subPropertyOf WWW . WWW rdfs:domain WW .	rdfs:Class rdfs:subClassOf WW .
ext7	rdfs:subPropertyOf rdfs:subPropertyOf WWW . WWW rdfs:domain WW .	rdf:Property rdfs:subClassOf WW .
ext8	rdfs:subClassOf rdfs:subPropertyOf WWW . WWW rdfs:range WW .	rdfs:Class rdfs:subClassOf WW .
ext9	rdfs:subPropertyOf rdfs:subPropertyOf WWW . WWW rdfs:range WW .	rdf:Property rdfs:subClassOf WW .

Exercise

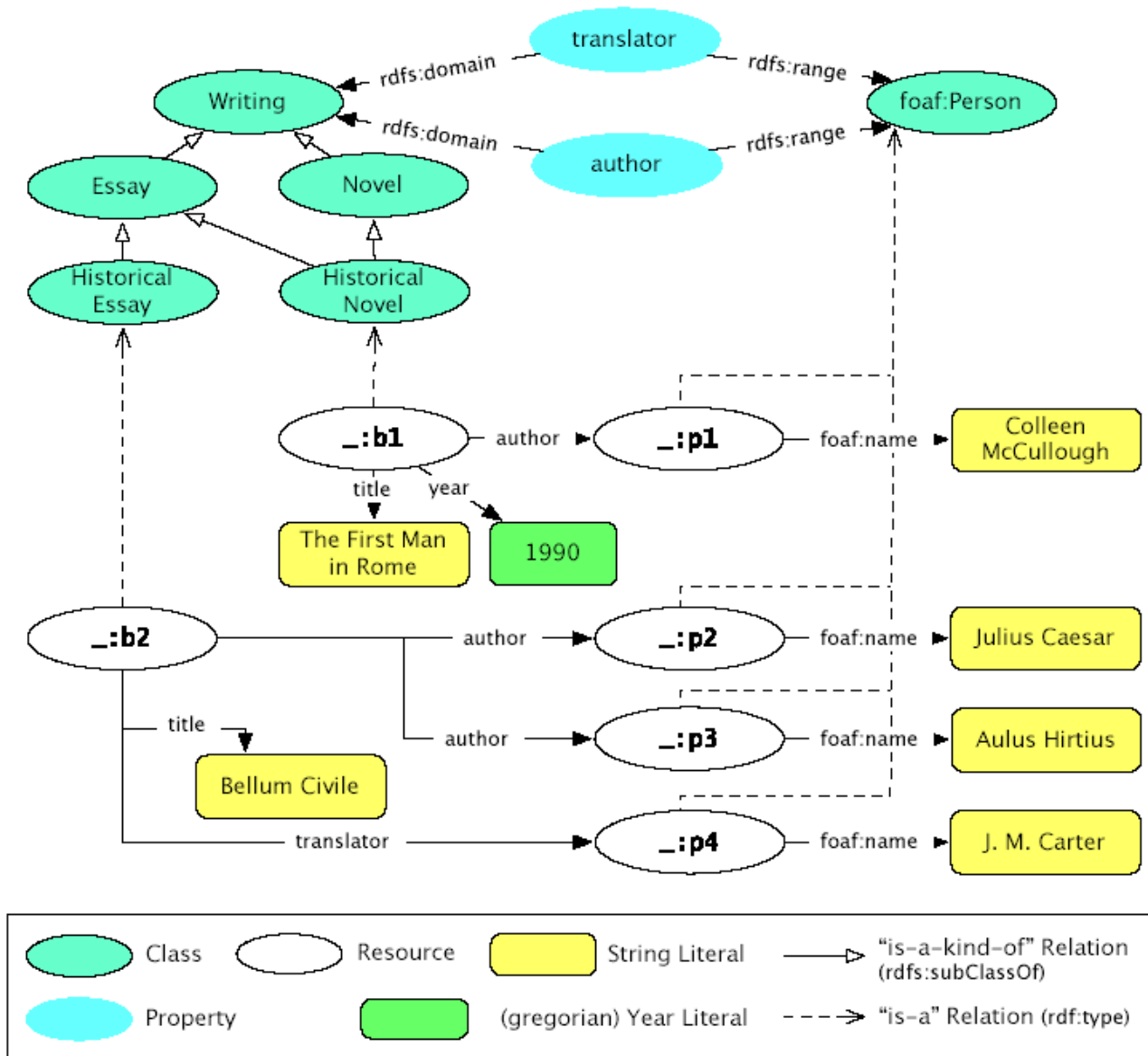


- **Objective**

- Understand how to RDF(S) reasoning works

- **Tasks**

- Generate all possible RDF triples that can be inferred from an RDF(S) graph



Exercise



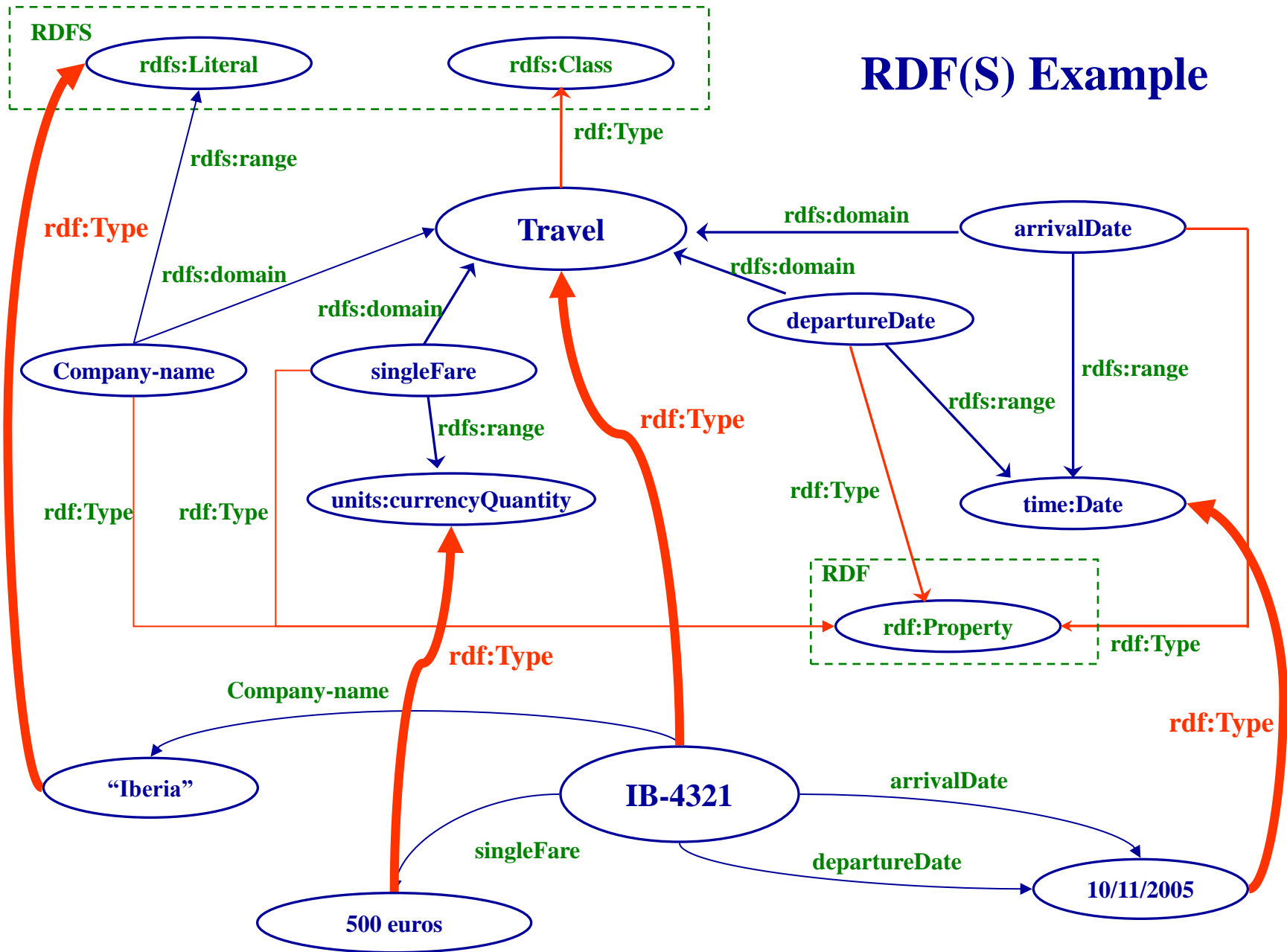
- **Objective**

- Understand how to use an RDF(S) development tool to create RDF(S) ontologies

- **Tasks**

- Create the previous ontologies in an RDF(S) development tool

RDF(S) Example



Exercise



- **Objective**

- Understand the features of RDF(S) for implementing ontologies, including its limitations

- **Tasks**

- Take the ontologies previously defined and create their graphs
 - First only include the vocabulary from the domain
 - Then include references to the RDF and RDFS vocabularies

Domain description

- Un lugar puede ser un lugar de interés.
 - Los lugares de interés pueden ser lugares turísticos o establecimientos, pero no las dos cosas a la vez.
 - Los lugares turísticos pueden ser palacios, iglesias, ermitas y catedrales.
 - Los establecimientos pueden ser hoteles, hostales o albergues.
 - Un lugar está situado en una localidad, la cual a su vez puede ser una villa, un pueblo o una ciudad.
 - Un lugar de interés tiene una dirección postal que incluye su calle y su número.
 - Las localidades tienen un número de habitantes.
 - Las localidades se encuentran situadas en provincias.
-
- Covarrubias es un pueblo con 634 habitantes de la provincia de Burgos.
 - El restaurante “El Galo” está situado en Covarrubias, en la calle Mayor, número 5.
 - Una de las iglesias de Covarrubias está en la calle de Santo Tomás.



Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. **RDF(S) management APIs**
 - 4.1 RDF(S) management APIs
 - 4.2 The Jena API, with a hands-on activity
5. RDF(S) query languages: SPARQL
6. An example of an RDF(S)-based application

Sample RDF APIs

RDF libraries for different languages:

- Java, Python, C, C++, C#, .Net, Javascript, Tcl/Tk, PHP, Lisp, Obj-C, Prolog, Perl, Ruby, Haskell
- List in <http://esw.w3.org/topic/SemanticWebTools>

Usually related to a RDF repository

- **Multilanguage:**
 - Redland RDF Application Framework (C, Perl, PHP, Python and Ruby): <http://www.redland.opensource.ac.uk/>
- **Java:**
 - Jena: <http://jena.sourceforge.net/>
 - Sesame: <http://www.openrdf.org/>
- **PHP:**
 - RAP - RDF API for PHP: <http://www4.wiwiiss.fu-berlin.de/bizer/rdfapi/>
- **Python:**
 - RDFLib: <http://rdflib.net/>
 - Pyrpale: <http://infomesh.net/pyrpale/>

Jena

- **Java framework for building Semantic Web applications**
- **Open source software from HP Labs:**
- **The Jena framework includes:**
 - A RDF API
 - An OWL API
 - Reading and writing RDF in RDF/XML, N3 and N-Triples
 - In-memory and persistent storage
 - A rule based inference engine
 - SPARQL query engine

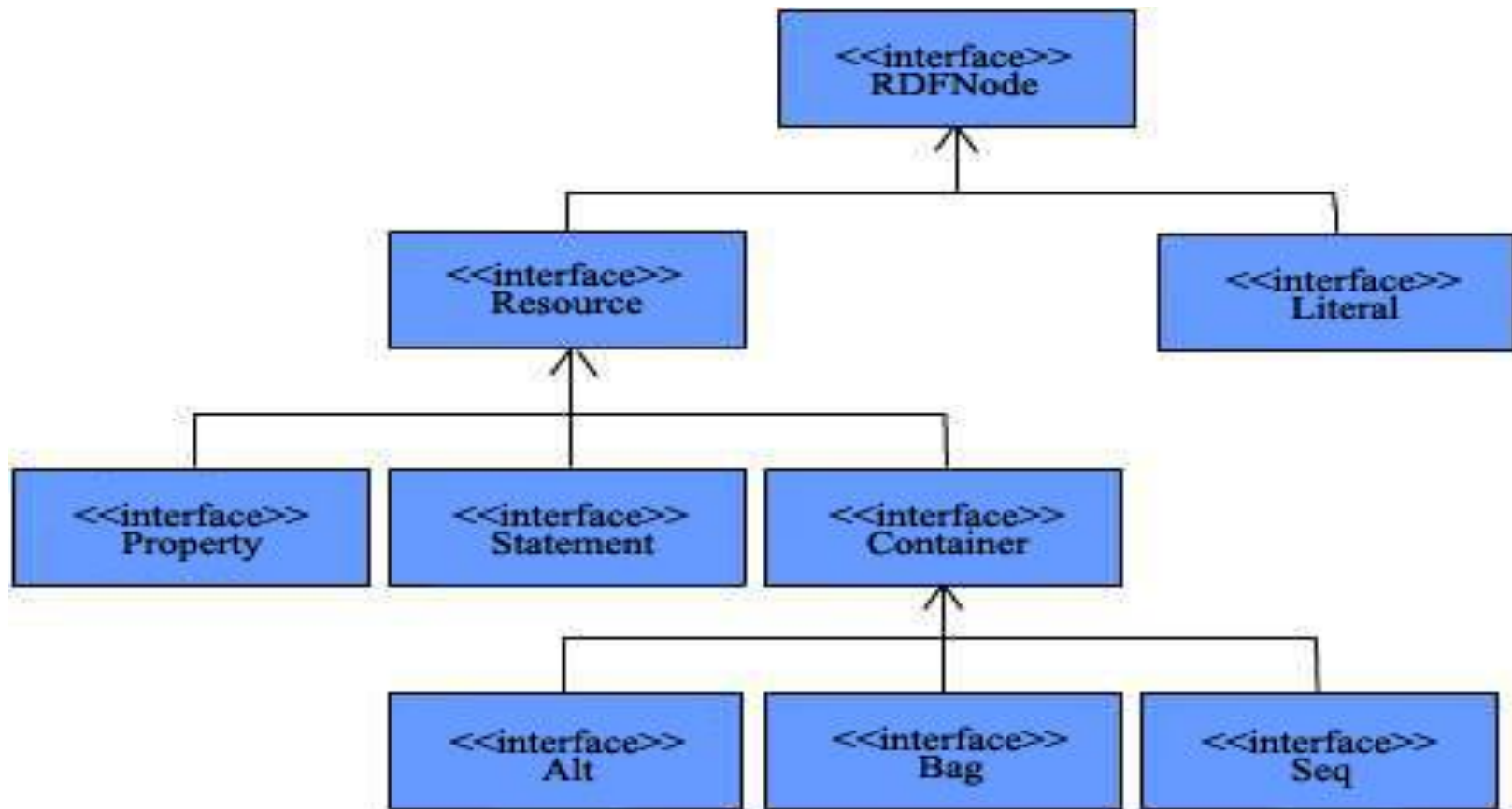
Sesame

- **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**
- **Highly expressive query and transformation languages**
 - SeRQL, SPARQL
- **Various backends**
 - Native Store
 - RDBMS (MySQL, Oracle 10, DB2, PostgreSQL)
 - main memory
- **Reasoning support**
 - RDF Schema reasoner
 - OWL DLP (OWLIM)
 - domain reasoning (custom rule engine)

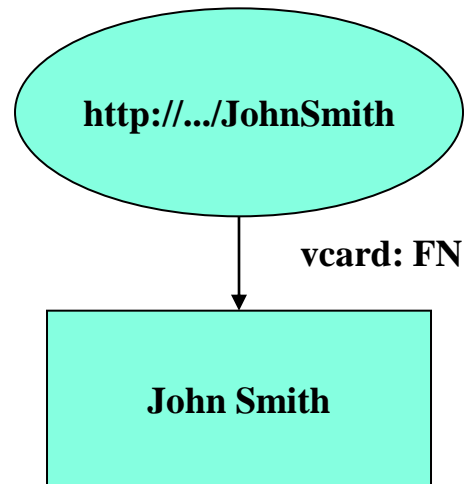
Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. **RDF(S) management APIs**
 - 4.1 RDF(S) management APIs
 - 4.2 **The Jena API, with a hands-on activity**
5. RDF(S) query languages: SPARQL
6. An example of an RDF(S)-based application

Jena API Structure

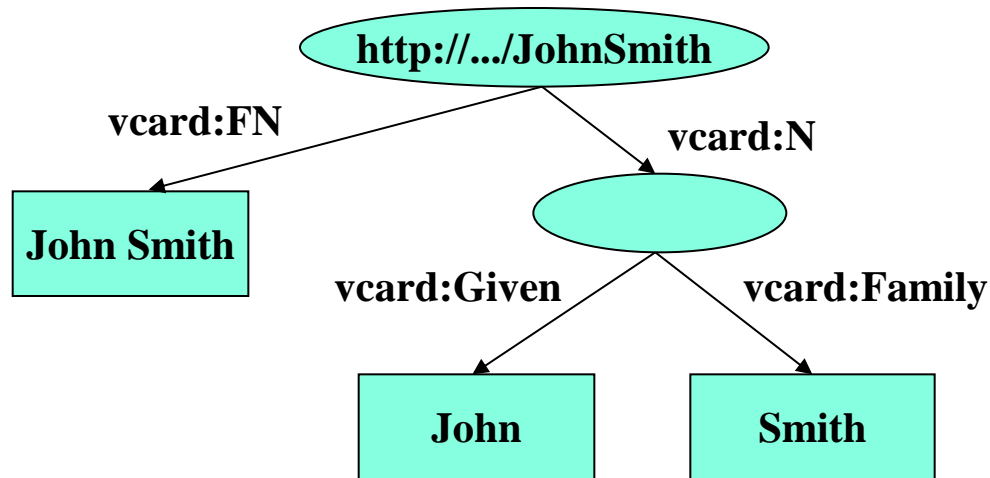


Data Model



```
// some definitions
static String personURI = "http://somewhere/JohnSmith";
static String fullName = "John Smith";
// create an empty Model
Model model = ModelFactory.createDefaultModel();
// create the resource
Resource johnSmith = model.createResource(personURI);
// add the property
johnSmith.addProperty(VCARD.FN, fullName);
```

Another data model



```
// some definitions
String personURI = "http://somewhere/JohnSmith"; String givenName = "John";
String familyName = "Smith";
String fullName = givenName + " " + familyName;
// create an empty
Model model = ModelFactory.createDefaultModel();
// create the resource
// and add the properties cascading style
Resource johnSmith = model.createResource(personURI)
    .addProperty(VCARD.FN, fullName)
    .addProperty(VCARD.N, model.createResource())
    .addProperty(VCARD.Given, givenName)
    .addProperty(VCARD.Family, familyName));
```

Statements

```
// list the statements in the Model
StmtIterator iter = model.listStatements();
// print out the predicate, subject and object of each statement
while (iter.hasNext())
{
    Statement stmt = iter.nextStatement(); // get next statement
    Resource subject = stmt.getSubject(); // get the subject
    Property predicate = stmt.getPredicate(); // get the predicate
    RDFNode object = stmt.getObject(); // get the object
    System.out.print(subject.toString());
    System.out.print(" " + predicate.toString() + " ");
    if (object instanceof Resource) {
        System.out.print(object.toString());
    }
    else { // object is a literal
        System.out.print(" \"" + object.toString() + "\"");
    }
    System.out.println(" .");
} // end of while
```

```
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#N anon:14df86:ecc3dee17b:-7fff
anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Family "Smith"
anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Given "John"

http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#FN
"John Smith" .
```

Writing RDF

```
Model model = ModelFactory.createDefaultModel();
Resource jsmith =
model.createResource("http://somewhere/johnsmith")
    .addProperty(VCARD.FN, "John Smith")
    .addProperty(VCARD.N, model.createResource())
    .addProperty(VCARD.Given, "John")
    .addProperty(VCARD.Family, "Smith"));
model.write(new PrintWriter(System.out));
```

```
model.write(System.out, "RDF/XML-ABBREV");
```

```
model.write(System.out, "N-TRIPLE");
```

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
>
  <rdf:Description rdf:nodeID='A0'>
    <vcard:Given>John</vcard:Given>
    <vcard:Family>Smith</vcard:Family>
  </rdf:Description>
  <rdf:Description rdf:about='http://somewhere/johnsmith'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID='A0' />
  </rdf:Description>
</rdf:RDF>
```

Reading RDF

```
// create an empty model
Model model = ModelFactory.createDefaultModel();

// use the FileManager to find the input file
InputStream in = FileManager.get().open( inputFileName );
if (in == null) {
    throw new IllegalArgumentException("File not found");
}

// read the RDF/XML file
model.read(in, "");

// write it to standard out
model.write(System.out);
```

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
>
  <rdf:Description rdf:nodeID="A0">
    <vcard:Family>Smith</vcard:Family>
    <vcard:Given>John</vcard:Given>
  </rdf:Description>
  <rdf:Description rdf:about='http://somewhere/JohnSmith/'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID="A0"/>
  </rdf:Description>
  ...
</rdf:RDF>
```


Navigating a model

```
// retrieve the John Smith vcard resource from the model  
Resource vcard = model.getResource(johnSmithURI);
```

Three ways of retrieving property values:

```
// retrieve the value of the N property  
Resource name = (Resource) vcard.getProperty(VCARD.N).getObject();
```

```
// retrieve the value of the N property  
Resource name = vcard.getProperty(VCARD.N).getResource();
```

```
// retrieve the given name property  
String fullName = vcard.getProperty(VCARD.N).getString();
```

Multiple values in properties

```
// add two nickname properties to vcard
vcard.addProperty(VCARD.NICKNAME, "Smithy")
    .addProperty(VCARD.NICKNAME, "Adman");

// set up the output
System.out.println("The nicknames of \"" + fullName + "\" are:");

// list the nicknames
StmtIterator iter = vcard.listProperties(VCARD.NICKNAME);
while (iter.hasNext()) {
    System.out.println(" " + iter.nextStatement().getObject().toString());
}
```

Querying a model

```
// select all the resources with a VCARD.FN property
ResIterator iter = model.listSubjectsWithProperty(VCARD.FN);
if (iter.hasNext()) {
    System.out.println("The database contains vcards for:");
    while (iter.hasNext()) {
        System.out.println(" " + iter.nextStatement()
                           .getProperty(VCARD.FN).getString());
    }
} else {
    System.out.println("No vcards were found in the database");
}
```

```
The database contains vcards for:
Sarah Jones
John Smith
Matt Jones
Becky Smith
```

Create resources

```
// URI declarations
String familyUri = "http://family/";
String relationshipUri = "http://purl.org/vocab/relationship/";

// Create an empty Model
Model model = ModelFactory.createDefaultModel();

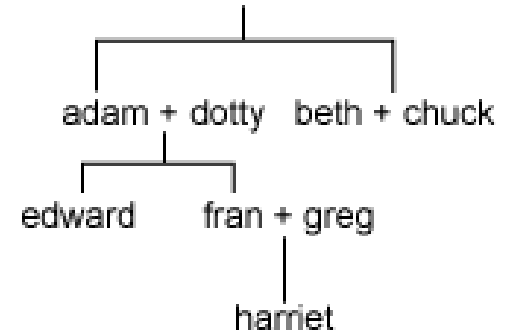
// Create a Resource for each family member, identified by their URI
Resource adam = model.createResource(familyUri+"adam");
Resource beth = model.createResource(familyUri+"beth");
Resource dotty = model.createResource(familyUri+"dotty");
// and so on for other family members

// Create properties for the different types of relationship to represent
Property childOf = model.createProperty(relationshipUri,"childOf");
Property parentOf = model.createProperty(relationshipUri,"parentOf");
Property siblingOf = model.createProperty(relationshipUri,"siblingOf");
Property spouseOf = model.createProperty(relationshipUri,"spouseOf");

// Add properties to adam describing relationships to other family members
adam.addProperty(siblingOf,beth);
adam.addProperty(spouseOf,dotty);
adam.addProperty(parentOf,edward);

// Can also create statements directly . . .
Statement statement = model.createStatement(adam,parentOf,fran);

// but remember to add the created statement to the model
model.add(statement);
```



Querying a model

```
// List everyone in the model who has a child:
ResIterator parents = model.listSubjectsWithProperty(parentOf);

// Because subjects of statements are Resources, the method returned a ResIterator
while (parents.hasNext()) {

    // ResIterator has a typed nextResource() method
    Resource person = parents.nextResource();

    // Print the URI of the resource
    System.out.println(person.getURI());
}

// Can also find all the parents by getting the objects of all "childOf" statements
// Objects of statements could be Resources or literals, so the Iterator returned
// contains RDFNodes
NodeIterator moreParents = model.listObjectsOfProperty(childOf);

// To find all the siblings of a specific person, the model itself can be queried
NodeIterator siblings = model.listObjectsOfProperty(edward, siblingOf);

// But it's more elegant to ask the Resource directly
// This method yields an iterator over Statements
StmtIterator moreSiblings = edward.listProperties(siblingOf);
```

Using selectors to query a model

```
// Find the exact statement "adam is a spouse of dotty"
```

```
model.listStatements(adam, spouseOf, dotty);
```

```
// Find all statements with adam as the subject and dotty as the object
```

```
model.listStatements(adam, null, dotty);
```

```
// Find any statements made about adam
```

```
model.listStatements(adam, null, null);
```

```
// Find any statement with the siblingOf property
```

```
model.listStatements(null, siblingOf, null);
```

Exercise



•Objective

- Understand how to use an RDF(S) management API

•Tasks

- Read an ontology in RDF(S) from two files:
 - GP_Santiago.rdf (conceptualization)
 - GP_Santiago.rdfs (instances)
- Write the class hierarchy of the ontology, including the instances of each class



Hands-on

- **To read an ontology in RDF(S) from two files:**
 - GP_Santiago.rdf (conceptualization)
 - GP_Santiago.rdfs (instances)
- **To write the class hierarchy of the ontology, including the instances of each class:**

```
Class Practica2:MedioTransporte
  Class Practica2:Tren
  Class Practica2:Bicicleta
    Instance Practica2:GP_Santiago_Instance_70
  Class Practica2:Automovil
  Class Practica2:AutoBus
  Class Practica2:APie
Class Practica2:InfraEstructuraTransporte
  Class Practica2:ViaFerreia
  Class Practica2:Sendero
  Class Practica2:Carretera
    Instance Practica2:A6
...
```




Set up

- **Requirements:**
 - Java JDK 5
 - Eclipse (optional)
 - **Create a directory for your project**
 - **Install Jena from the USB:**
 - Unzip *Jena-2.5.5.zip/lib* in the project directory
 - **Copy the ontologies from the USB:**
 - Copy *ontologies/rdf* in the project directory
 - **In Eclipse:**
 - Create a new Java project (from existing source)
 - Append the Jena libraries to your classpath if needed (check JDK libs)
 - Write Java code using the Jena API
 - <http://jena.sourceforge.net/javadoc/index.html>
 - Compile
 - Run
- Or copy the JenaProjectTemplate directory in your computer



Hints

- **Create ontology model:**

```
public static OntModel  
    createOntologyModel(OntModelSpec spec)
```

- **Read the ontology in the file**

```
Model read(java.lang.String url)
```

- **Add all the statements in another model to this model**

```
Model add(Model m)
```



More hints

- **List root classes**

```
ExtendedIterator listHierarchyRootClasses()
```

- **List subclasses of a class**

```
ExtendedIterator listSubClasses(boolean  
    direct)
```

- **List instances of a class**

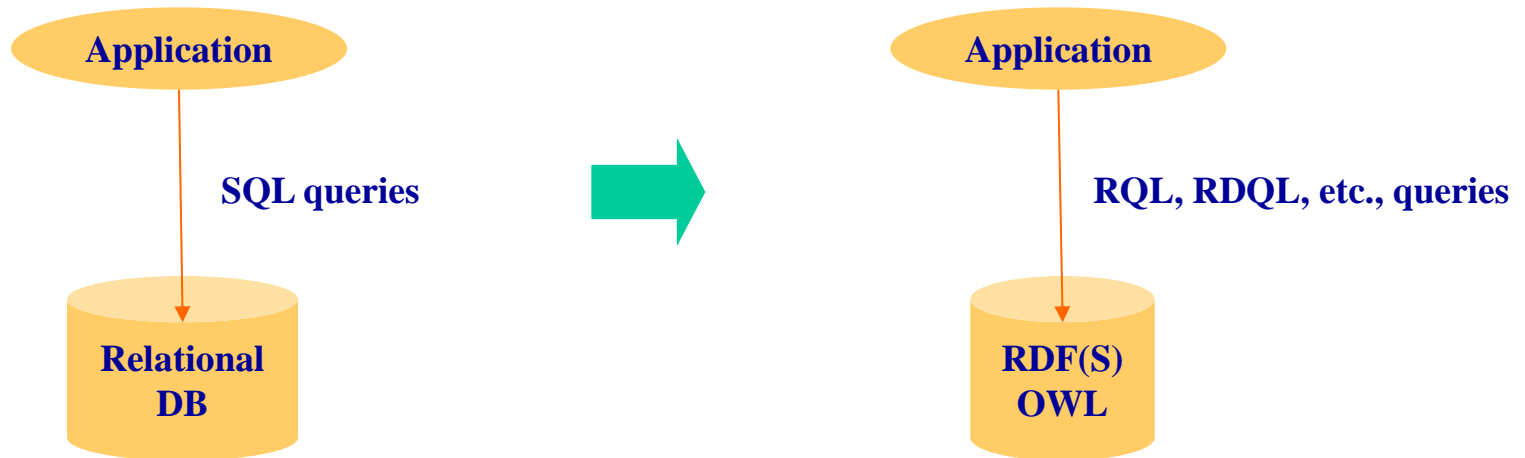
```
ExtendedIterator listInstances(boolean direct)
```

Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. RDF(S) management APIs
5. **RDF(S) query languages: SPARQL**
 - 5.1 RDF(S) query languages and SPARQL
 - 5.2 Turtle RDF syntax
 - 5.3 Graph patterns
 - 5.4 Restricting values and solutions
 - 5.5 SPARQL query forms
 - 5.6 Hands-on activity
6. An example of an RDF(S)-based application

RDF(S) query languages

- Languages developed to allow accessing datasets expressed in RDF(S) (and in some cases OWL)

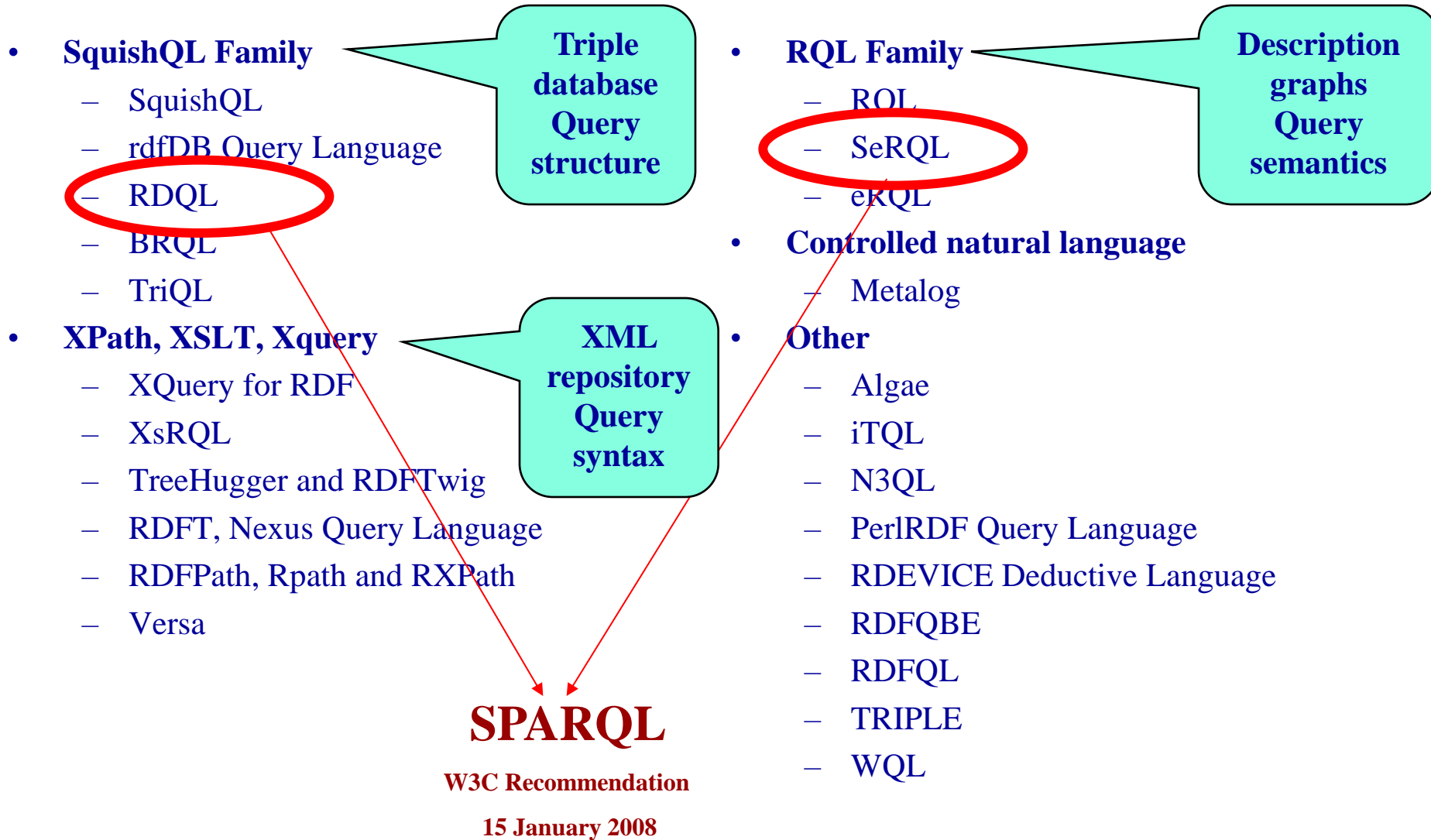


- Supported by the most important language APIs
 - Jena (HP labs)
 - Sesame (Aduna)
 - Boca (IBM)
 - ...
- There are some differences wrt languages like SQL, such as
 - Combination of different sources
 - Trust management
 - Open World Assumption

Query types

- **Selection and extraction**
 - “Select all the essays, together with their authors and their authors’ names”.
 - “Select everything that is related to the book ‘Bellum Civile’”
- **Reduction**: we specify what it should not be returned
 - “Select everything except for the ontological information and the book translators”
- **Restructuring**: the original structure is changed in the final result
 - “Invert the relationship ‘author’ by ‘is author of’”
- **Aggregation**
 - “Return all the essays together with the mean number of authors per essay”
- **Combination and inferences**
 - “Combine the information of a book called ‘La guerra civil’ and whose author is Julius Caesar with the book whose identifier is ‘Bellum Civile’”
 - “Select all the essays, together with its authors and author names”, *including also the instances of the subclasses of Essay*.
 - “Obtain the relationship ‘coauthor’ among persons who have written the same book”.

RDF(S) query language families



SPARQL

- **SPARQL Protocol and RDF Query Language**
- **Supported by:** Jena, Sesame, IBM Boca, etc.
- **Features**
 - It supports most of the aforementioned queries
 - It supports **datatype reasoning** (datatypes can be requested instead of actual values)
 - **The domain vocabulary and the knowledge representation vocabulary are** treated differently by the query interpreters.
 - It allows making queries over properties with multiple values, over multiple properties of a resource and over **reifications**
 - Queries can contain **optional statements**
 - Some implementations support **aggregation queries**
- **Limitations**
 - Neither **set operations** nor **existential or universal quantifiers** can be included in the queries
 - It does not support **recursive queries**

SPARQL is also a protocol

- SPARQL is a Query Language ...:

Find names and websites of contributors to PlanetRDF:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?website
FROM <http://planetrdf.com/bloggers.rdf>
WHERE {
    ?person foaf:weblog ?website .
    ?person foaf:name ?name .
    ?website a foaf:Document }
```

- ... and a Protocol.

```
http://.../qps?query-lang=http://www.w3.org/TR/rdf-sparql-query/
&graph-id=http://planetrdf.com/bloggers.rdf&query=PREFIXfoaf:
<http://xmlns.com/foaf/0.1/...
```

- Services running SPARQL queries over a set of graphs
- A transport protocol for invoking the service
- Based on ideas from earlier protocol work such as Joseki
- Describing the service with Web Service technologies

A simple SPARQL query

Data:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .  
@prefix : <http://example.org/book/> .  
:book1 dc:title "SPARQL Tutorial" .
```

Query:

```
SELECT ?title  
WHERE  
{  
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .  
}
```

Query result:

title
"SPARQL Tutorial"

- A pattern is *matched* against the RDF data
- Each way a pattern can be matched yields a solution
- The sequence of solutions is filtered by: Project, distinct, order, limit/offset
- One of the result forms is applied: SELECT, CONSTRUCT, DESCRIBE, ASK

Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. RDF(S) management APIs
5. **RDF(S) query languages: SPARQL**
 - 5.1 RDF(S) query languages and SPARQL
 - 5.2 **Turtle RDF syntax**
 - 5.3 Graph patterns
 - 5.4 Restricting values and solutions
 - 5.5 SPARQL query forms
 - 5.6 Hands-on activity
6. An example of an RDF(S)-based application

Turtle: URIs, blank nodes, literals

- **URIs**

Enclosed in `<>`
`<URI>`

or

`@prefix prefix <http://....>`
`prefix:name`

- **Blank Nodes**

`:name`

or

`[]` for a Blank Node used once

- **Literals**

`"Literal"`

`"Literal"@language`

`"""Long literal with
newlines"""`

- **Datatypes Literals**

`"lexical form"^^datatype URI`

`"10"^^xsd:integer`

`"2006-09-04"^^xsd:date`

Turtle: Triples and abbreviations

- **Triples separated by .**

:a :b :c . :d :e :f .

- **Common triple predicate and subject:**

:a :b :c, :d .

which is the same as :a :b :c . :a :b :d .

- **Common triple subject:**

:a :b :c; :d :e .

which is the same as: :a :b :c . :a :d :e .

- **Blank node as a subject**

:a :b [:c :d]

which is the same as: :a :b _:x . _:x :c :d .

for blank node _:x

- **RDF Collections**

– :a :b (:c :d :e :f)

which is short for many triples

Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. RDF(S) management APIs
5. **RDF(S) query languages: SPARQL**
 - 5.1 RDF(S) query languages and SPARQL
 - 5.2 Turtle RDF syntax
 - 5.3 **Graph patterns**
 - 5.4 Restricting values and solutions
 - 5.5 SPARQL query forms
 - 5.6 Hands-on activity
6. An example of an RDF(S)-based application

Graph patterns

- **Basic Graph Patterns**, where a set of triple patterns must match
- **Group Graph Pattern**, where a set of graph patterns must all match
- **Optional Graph patterns**, where additional patterns may extend the solution
- **Alternative Graph Pattern**, where two or more possible patterns are tried
- **Patterns on Named Graphs**, where patterns are matched against named graphs

Basic graph patterns: Multiple matches

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Johnny Lee Outlaw" .
_:a foaf:mbox <mailto:jlow@example.com> .
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
{
  ?x foaf:name ?name .
  ?x foaf:mbox ?mbox }

```

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Basic graph patterns: Matching RDF literals

```
@prefix dt:    <http://example.org/datatype#> .
@prefix ns:    <http://example.org/ns#> .
@prefix :      <http://example.org/ns#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .

:x    ns:p      "cat"@en .
:y    ns:p      "42"^^xsd:integer .
:z    ns:p      "abc"^^dt:specialDatatype .
```

SELECT ?v WHERE { ?v ?p "cat" }

v

SELECT ?v WHERE { ?v ?p "cat"@en }

v

<http://example.org/ns#x>

SELECT ?v WHERE { ?v ?p 42 }

v

<http://example.org/ns#y>

SELECT ?v WHERE { ?v ?p "abc"^^<http://example.org/datatype#specialDatatype> }

v

<http://example.org/ns#z>

Basic graph patterns: Blank node labels in query results

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

```
_:a foaf:name "Alice" .
```

```
_:b foaf:name "Bob" .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?x ?name
```

```
WHERE { ?x foaf:name ?name }
```

x	name
_:c	"Alice"
_:d	"Bob"

=

x	name
_:r	"Alice"
_:s	"Bob"

Group graph pattern

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { { ?x foaf:name ?name . }
        { ?x foaf:mbox ?mbox . }
      }
```

```
SELECT ?x
WHERE {}
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { { ?x foaf:name ?name . }
        { ?x foaf:mbox ?mbox . FILTER regex(?name, "Smith") }
      }
```

Optional graph patterns

```
@prefix foaf:      <http://xmlns.com/foaf/0.1/> .
@prefix rdf:       <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

_:a  rdf:type      foaf:Person .
_:a  foaf:name     "Alice" .
_:a  foaf:mbox     <mailto:alice@example.com> .
_:a  foaf:mbox     <mailto:alice@work.example> .

_:b  rdf:type      foaf:Person .
_:b  foaf:name     "Bob" .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
       OPTIONAL { ?x foaf:mbox ?mbox }
}
```

name	mbox
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

Multiple optional graph patterns

```
@prefix foaf:      <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name      "Alice" .
_:a  foaf:homepage  <http://work.example.org/alice/> .

_:b  foaf:name      "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
WHERE { ?x foaf:name ?name .
        OPTIONAL { ?x foaf:mbox ?mbox } .
        OPTIONAL { ?x foaf:homepage ?hpage }
}
```

name	mbox	hpage
"Alice"		<http://work.example.org/alice/>
"Bob"	<mailto:bob@work.example>	

Alternative graph patterns

```
@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .

_:a dc10:title      "SPARQL Query Language Tutorial" .
_:a dc10:creator    "Alice" .
_:b dc11:title      "SPARQL Protocol Tutorial" .
_:b dc11:creator    "Bob" .
_:c dc10:title      "SPARQL" .
_:c dc11:title      "SPARQL (updated)" .
```

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { { ?book dc10:title ?title } UNION
        { ?book dc11:title ?title } }
```

title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"
"SPARQL Query Language Tutorial"

```
SELECT ?x ?y
WHERE { { ?book dc10:title ?x } UNION
        { ?book dc11:title ?y } }
```

x	y
	"SPARQL (updated)"
	"SPARQL Protocol Tutorial"
"SPARQL"	
"SPARQL Query Language Tutorial"	

```
SELECT ?title ?author
WHERE
{ { ?book dc10:title ?title . ?book dc10:creator ?author }
  UNION
  { ?book dc11:title ?title . ?book dc11:creator ?author } }
```

author	title
"Alice"	"SPARQL Protocol Tutorial"
"Bob"	"SPARQL Query Language Tutorial"

Patterns on named graphs

```
# Named graph: http://example.org/foaf/aliceFoaf
@prefix foaf:<http://.../foaf/0.1/> .
@prefix rdf:<http://.../1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:<http://.../2000/01/rdf-schema#> .

_:a foaf:name      "Alice" .
_:a foaf:mbox      <mailto:alice@work.example> .
_:a foaf:knows     _:b .

_:b foaf:name      "Bob" .
_:b foaf:mbox      <mailto:bob@work.example> .
_:b foaf:nick      "Bobby" .
_:b rdfs:seeAlso   <http://example.org/foaf/bobFoaf> .

<http://example.org/foaf/bobFoaf>
  rdf:type         foaf:PersonalProfileDocument .
```

```
# Named graph: http://example.org/foaf/bobFoaf
@prefix foaf:<http://.../foaf/0.1/> .
@prefix rdf:<http://.../1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:<http://.../2000/01/rdf-schema#> .

_:z foaf:mbox      <mailto:bob@work.example> .
_:z rdfs:seeAlso   <http://example.org/foaf/bobFoaf> .
_:z foaf:nick      "Robert" .

<http://example.org/foaf/bobFoaf>
  rdf:type         foaf:PersonalProfileDocument .
```

Patterns on named graphs II

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?src ?bobNick
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE
{
  GRAPH ?src
  { ?x foaf:mbox <mailto:bob@work.example> .
    ?x foaf:nick ?bobNick
  }
}
```

src	bobNick
<http://example.org/foaf/aliceFoaf>	"Bobby"
<http://example.org/foaf/bobFoaf>	"Robert"

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX data: <http://example.org/foaf/>
```

```
SELECT ?nick
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE
{
  GRAPH data: bobFoaf {
    ?x foaf:mbox <mailto:bob@work.example> .
    ?x foaf:nick ?nick
  }
}
```

nick
"Robert"

Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. RDF(S) management APIs
5. **RDF(S) query languages: SPARQL**
 - 5.1 RDF(S) query languages and SPARQL
 - 5.2 Turtle RDF syntax
 - 5.3 Graph patterns
 - 5.4 **Restricting values and solutions**
 - 5.5 SPARQL query forms
 - 5.6 Hands-on activity
6. An example of an RDF(S)-based application

Restricting values

```
@prefix dc:    <http://purl.org/dc/elements/1.1/> .
@prefix :      <http://example.org/book/> .
@prefix ns:    <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
       FILTER regex(?title, "^SPARQL")
}
```

title

"SPARQL Tutorial"

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
       FILTER regex(?title, "web", "i" )
}
```

title

"The Semantic Web"

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x ns:price ?price .
       FILTER (?price < 30.5)
       ?x dc:title ?title . }
```

title	price
-------	-------

"The Semantic Web"	23
--------------------	----

Value tests

- Based on XQuery 1.0 and XPath 2.0 Function and Operators
- XSD boolean, string, integer, decimal, float, double, dateTime
- Notation <, >, =, <=, >= and != for value comparison
Apply to any type
- BOUND, isURI, isBLANK, isLITERAL
- REGEX, LANG, DATATYPE, STR (lexical form)
- Function call for casting and extensions functions

Solution sequences and modifiers

- **Order modifier:** put the solutions in order
- **Projection modifier:** choose certain variables
- **Distinct modifier:** ensure solutions in the sequence are unique
- **Reduced modifier:** permit elimination of some non-unique solutions
- **Offset modifier:** control where the solutions start from in the overall sequence of solutions
- **Limit modifier:** restrict the number of solutions

```
SELECT ?name
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY ?name DESC(?emp)
```

```
SELECT ?name
WHERE
  { ?x foaf:name ?name }
```

```
SELECT DISTINCT ?name
WHERE { ?x foaf:name ?name }
```

```
SELECT REDUCED ?name
WHERE { ?x foaf:name ?name }
```

```
SELECT ?name WHERE { ?x foaf:name ?name }
ORDER BY ?name
LIMIT 5
OFFSET 10
```

```
SELECT ?name
WHERE { ?x foaf:name ?name }
LIMIT 20
```

Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. RDF(S) management APIs
5. **RDF(S) query languages: SPARQL**
 - 5.1 RDF(S) query languages and SPARQL
 - 5.2 Turtle RDF syntax
 - 5.3 Graph patterns
 - 5.4 Restricting values and solutions
 - 5.5 **SPARQL query forms**
 - 5.6 Hands-on activity
6. An example of an RDF(S)-based application

SPARQL query forms

- **SELECT**
 - Returns all, or a subset of, the variables bound in a query pattern match.
- **CONSTRUCT**
 - Returns an RDF graph constructed by substituting variables in a set of triple templates.
- **ASK**
 - Returns a boolean indicating whether a query pattern matches or not.
- **DESCRIBE**
 - Returns an RDF graph that describes the resources found.

SPARQL query forms: SELECT

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

```
_:a foaf:name "Alice" .
```

```
_:a foaf:knows _:b .
```

```
_:a foaf:knows _:c .
```

```
_:b foaf:name "Bob" .
```

```
_:c foaf:name "Clare" .
```

```
_:c foaf:nick "CT" .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?nameX ?nameY ?nickY
```

```
WHERE
```

```
{ ?x foaf:knows ?y ;  
  foaf:name ?nameX .  
  ?y foaf:name ?nameY .  
  OPTIONAL { ?y foaf:nick ?nickY }  
}
```

nameX	nameY	nickY
"Alice"	"Bob"	
"Alice"	"Clare"	"CT"

SPARQL query forms: CONSTRUCT

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
  
_:a foaf:name "Alice" .  
_:a foaf:mbox <mailto:alice@example.org> .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>  
  
CONSTRUCT { <http://example.org/person#Alice> vcard:FN ?name }  
  
WHERE { ?x foaf:name ?name }
```

Query result:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .  
  
<http://example.org/person#Alice> vcard:FN "Alice" .
```


SPARQL query forms: ASK

```
@prefix foaf:      <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name      "Alice" .
_:a  foaf:homepage  <http://work.example.org/alice/> .

_:b  foaf:name      "Bob" .
_:b  foaf:mbox      <mailto:bob@work.example> .
```

```
PREFIX foaf:      <http://xmlns.com/foaf/0.1/>
ASK { ?x foaf:name "Alice" }
```

Query result:

yes

SPARQL query forms: DESCRIBE

```
PREFIX ent: <http://org.example.com/employees#>
DESCRIBE ?x WHERE { ?x ent:employeeId "1234" }
```

Query result:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0> .
@prefix exOrg: <http://org.example.com/employees#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>

_:a      exOrg:employeeId      "1234" ;

         foaf:mbox_shalsum      "ABCD1234" ;
         vcard:N
           [ vcard:Family        "Smith" ;
             vcard:Given         "John" ] .

foaf:mbox_shalsum  rdf:type  owl:InverseFunctionalProperty .
```

Exercise



•Objective

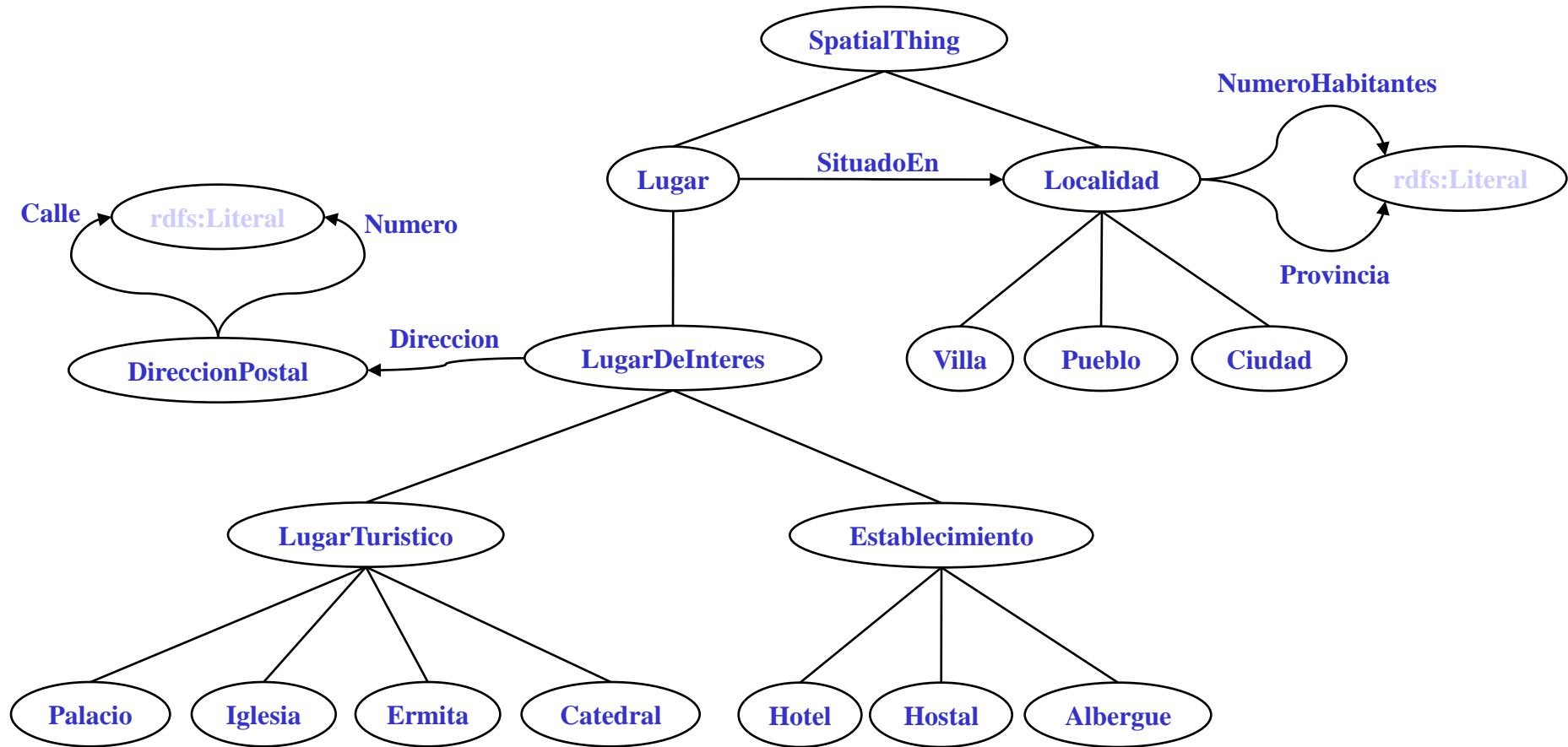
- Understand how to perform SPARQL queries

•Tasks

- Perform a set of SPARQL queries over the sample ontology.
 - Browse to:
 - <http://my.computer.ip:8080/openrdf-workbench>
 - Select repository GP-native-rdfs
 - Select the Query option from the left menu



Sample ontology





Queries on the model

1) Get all the classes

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x WHERE { ?x a rdfs:Class. }
```

2) Get the subclasses of the class *Establecimiento*

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { ?x rdfs:subClassOf pr:Establecimiento. }
```

3) Get the instances of the class *Ciudad*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { ?x a pr:Ciudad. }
```



Queries on the instances

4) Get the number of inhabitants of *Santiago de Compostela*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { pr:Santiago_de_Compostela pr:NumeroHabitantes ?x. }
```

5) Get the number of inhabitants of *Santiago de Compostela* and of *Arzua*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE {
    {pr:Santiago_de_Compostela pr:NumeroHabitantes ?x.}
    UNION
    {pr:Arzua pr:NumeroHabitantes ?x.}
}
```

6) Get different places with the inhabitants number, ordering the results by the name of the place (ascending)

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x ?y WHERE { $sitio pr:NumeroHabitantes ?y;
                      rdfs:label ?x.}

ORDER BY ASC(?x)
```



Queries on the instances II

7) Get all the instances of *Localidad* with their inhabitant number (if it exists)

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x ?y WHERE { $sitio a pr:Localidad;
                        rdfs:label ?x.
                        OPTIONAL {$sitio pr:NumeroHabitantes ?y.} }
```

8) Get all the places with more than 200.000 inhabitants

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x ?y WHERE { $sitio pr:NumeroHabitantes ?y;
                        rdfs:label ?x.
                        FILTER(?y > 200000) }
```

9) Get postal data of *Pazo de Breogan* (calle, número, localidad, provincia)

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?calle ?numero ?poblacion ?provincia
WHERE { pr:Pazo_Breogan pr:SituadoEn $pob;
        pr:Direccion $dir.
        $pob rdfs:label ?poblacion;
        pr:Provincia ?provincia.
        $dir pr:Calle ?calle;
        pr:Numero ?numero.}
```



Queries with inference

10) Get the subclasses of class *Lugar*

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { ?x rdfs:subClassOf pr:Lugar. }
```

11) Get the instances of class *Localidad*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { ?x a pr:Localidad. }
```

Special query (SELECT *)

12) Get the values of all the variables in the query

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT * WHERE { ?x pr:NumeroHabitantes ?y. }
```




Different query forms

13) Describe the resource with *rdfs:label* "Madrid"

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
DESCRIBE ?x WHERE { ?x rdfs:label "Madrid". }
```

14) Construct the RDF(S) graph that directly relates all the touristic places with their respective provinces, using a new property called "estaEn".

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
CONSTRUCT {?x pr:estaEn ?y}
WHERE {
    ?x a pr:LugarTuristico;
        pr:SituadoEn $pob.
    $pob pr:Provincia ?y. }
```

15) Ask if there is some instance of *Pueblo*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
ASK WHERE {?a a pr:Pueblo}
```

16) Ask if there is some instance of *Ermita*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
ASK WHERE {?a a pr:Ermita}
```

Table of Contents

- 1. An introduction to knowledge representation formalisms**
- 2. Resource Description Framework (RDF)**
 - 2.1. RDF primitives**
 - 2.2. Reasoning with RDF**
- 3. RDF Schema**
 - 3.1 RDF Schema primitives**
 - 3.2 Reasoning with RDFS**
- 4. RDF(S) management APIs**
- 5. RDF(S) query languages: SPARQL**
- 6. An example of an RDF(S)-based application**