

## **Chapter 5**

# **RDF Schema (and a bit of OWL)**

Schema Information and Reasoning in an Open World

# ONTOLOGIES

Schema languages, metadata languages, modeling languages, ontologies ...

## Classical Data Models: seen as Specification and Constraints

- every schema description defines a (more or less complete) ontology:
- ER Model (1976, entity types, attributes, relationships with cardinalities),
- UML (1997, classes with subclasses, associations with cardinalities, OCL assertions to schema components etc.).

## Knowledge Representation

Metadata provides additional information about resources of a type, or about a property.

- F-Logic signatures (1989),
- ... RDFS and OWL (Web Ontology Language)

## SCHEMA INFORMATION IN AN OPEN WORLD

- schema describes
  - allowed properties for an object,
  - datatype constraints for literal properties [Here: XSD literal types],
  - allowed types/classes for reference properties,
  - cardinality constraints.

### Closed World: Schema as Constraints

- a database must satisfy the constraints. It must be a *model* of the formulas – *the given data alone must be a model*.

### Open World: potentially incomplete knowledge

- schema information as *additional information*
- since the world must be a model of the schema, some information can be *derived* from the schema.
- complain only if information is *contradictory* to the schema.

## METADATA INFORMATION: TYPES, PROPERTIES, AND ONTOLOGIES

- Types and properties (i.e., everything that is used in a namespace ) are not only “names”, but are resources “somewhere in the Web”, identified by a URI (used in RDF or in XML via namespaces).

⇒ a *domain ontology* describes the notions used in a namespace.

### Schema and Ontology Information

- what types/classes are there,
- subclass information,
- what properties objects of a given type must/can have,
- to what types some property is applicable and what range it has,
- cardinalities of properties,
- default values,
- that some properties are transitive, symmetric, subproperties of another or excluding each other etc.

## REASONING WITH RDF, RDF SCHEMA AND OWL

- theoretical details will be discussed later. The underlying thing is *Description Logic (DL) Reasoning*
- there are DL reasoners available for the Jena Framework:
  - an internal one:  
`jena -q -inf -qf sparql-file`  
for invoking SPARQL with its internal reasoner
  - an external one:  
(integrated into the semweb.jar used in the lecture as plug-in)  
`jena -q -pellet -qf sparql-file`  
for invoking SPARQL with the Pellet reasoner class
  - external ones as Web Services ...

## USE OF THE JENA TOOL

- option “-t”: transform (between N3 and RDF/XML)  
jena -t -pellet -if *rdf-file* .
- option “-q”: query  
jena -q -pellet [-if *rdf-input-file*] -qf *query-file* .
- option “-e”: export the class tree (available only when the pellet reasoner is activated).  
Input is an RDF or OWL file:  
jena -e -pellet -if *rdf-file*.
- [note: since Jan. 2008, the former [-il RDF/XML] for indicating RDF/XML vs N3 input can be omitted in most cases]

## ASIDE: DIG INTERFACE - DESCRIPTION LOGIC IMPLEMENTATION GROUP

- Web page: <http://dl.kr.org/dig/>
- agreed “tell-and-ask-interface” of DL Reasoners as Web Service:
- tell them the facts and ask them queries, or for the whole inferred model
- e.g. supported by “Pellet”
- URL for download see Lecture Web page

```
may@dbis01:~/SemWeb-Tools/pellet-1.3$ ./pellet-dig.sh &  
PelletDIGServer Version 1.3 (April 17 2006)  
Port: 8081
```

- invoke the SPARQL Jena interface by  

```
jena -q -qf sparql-file -inf -r reasoner-url
```

  
(e.g.: <http://localhost:8081>)
- note: the tell-functionality seems to transfer only part of the knowledge → incomplete reasoning → currently not recommended.

## 5.1 RDF Schema Notions

- RDF is the instance level
- XML: DTDs and XML Schema for describing the structure/schema of the instance
- RDF Schema: stronger than DTD/XML – “semantic-level”
  - describe the structure of the RDF instance (i.e. the “schema” of the RDF graph, not of the RDF/XML file):
  - describes the schema *semantically* in terms of an (lightweight) ontology (OWL provides then much more features):
    - \* class/subclass
    - \* property/subproperty, domains and ranges



## PREDEFINED RDFS CLASSES

The obvious ones

**rdfs:Resource** is “everything”. All things described by RDF are called resources, and are instances of the class `rdfs:Resource`. This is the class of everything. All other classes are subclasses of this class. `rdfs:Resource` is an instance of `rdfs:Class`.

**rdfs:Class** : all things (resources and literals) are of `rdf:type` of some `rdfs:Class`.

`rdf:Properties` have an `rdfs:Class` as domain and another `rdfs:Class` or `rdfs:Datatype` as range.

`mon:Country` `rdf:type` `rdfs:Class`.

An `rdfs:Class` is simply a resource  $X$  that is of ( $X$  `rdf:type` `rdfs:Class`). Usually, class names start with a capital letter.

Later, **owl:Class** will provide more interesting concepts of *intensionally defined* classes – like “the class `father` is the class of things that are male and have children”.

**rdf:Property** is a subset of `rdfs:Resource` that contains all properties.

`mon:capital` `rdf:type` `rdf:Property`.

Usually, property names start with a non-capital letter.

[note: it's `rdf:Property`, not `rdfs:Property`!]

## PREDEFINED RDFS CLASSES

**rdfs:Datatype** is the class of datatypes.

**rdfs:Literal** is the subclass of `rdfs:Resource` that contains all literals (i.e., values of `rdfs:Datatypes`).

Literals do (usually) not have a URI, but a literal representation (as already discussed for integers and strings).

E.g. the following holds

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
xsd:int rdf:type rdfs:Datatype .
```

- There is another `rdfs:Datatype`: `rdf:XMLLiteral` which will be discussed later for RDF/XML.
- Note that *reification* takes place here: `rdfs:Datatype` is both an instance of and a subclass of `rdfs:Class`! Each instance of `rdfs:Datatype` is a subclass of `rdfs:Literal`.

## SEMANTICS OF SUBCLASSES AND SUBPROPERTIES

**rdfs:subClassOf** specifies that one rdfs:Class is an rdfs:subClassOf another:

for any model  $\mathcal{M}$  of the RDFS model theory,

$$\mathcal{M} \models \forall C_1, C_2 : ( \text{holds}(C_1, \text{rdfs:subClassOf}, C_2) \rightarrow (\forall x : (\text{holds}(x, \text{rdf:type}, C_1) \rightarrow \text{holds}(x, \text{rdf:type}, C_2))))$$

**rdfs:subPropertyOf** specifies that one rdf:Property is an rdfs:subPropertyOf another:

$$\mathcal{M} \models \forall P_1, P_2 : ( \text{holds}(P_1, \text{rdfs:subPropertyOf}, P_2) \rightarrow (\forall x, y : (\text{holds}(x, P_1, y) \rightarrow \text{holds}(x, P_2, y))))$$

## SEMANTICS OF DOMAIN AND RANGE

**rdfs:domain** specifies that the domain of an rdf:Property is a certain rdfs:Class:

$$\mathcal{M} \models \forall C, P : ( \text{holds}(P, \text{rdfs:domain}, C) \rightarrow \\ (\forall x : (\exists y : \text{holds}(x, P, y)) \rightarrow \text{holds}(x, \text{rdf:type}, C)))$$

**rdfs:range** specifies that the range of an rdf:Property is a certain rdfs:Class:

$$\mathcal{M} \models \forall C, P : ( \text{holds}(P, \text{rdfs:range}, C) \rightarrow \\ (\forall y : (\exists x : \text{holds}(x, P, y)) \rightarrow \text{holds}(y, \text{rdf:type}, C)))$$

## INFERENCE RULES

- The above are *built-in inference rules* of the RDFS Model Theory
- until now, the SPARQL query language was applied to pure RDF facts (*extensional knowledge*)
- for the *inference rules* (= *intensional knowledge*), a *reasoner* is required.
- Queries are then not evaluated against the *fact base*, but against the *model* of the factbase and the rules.

## SUBCLASS, DOMAIN, RANGE: EXAMPLE

```
@prefix : <foo://bla/names#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
    :has_cat rdfs:domain :Person .
    :has_cat rdfs:range :Cat .
    :Person rdfs:subClassOf :LivingBeing .
    :Cat rdfs:subClassOf :LivingBeing .
    <foo://bla/persons/john> :has_cat <foo://bla/cats/garfield>.
    <foo://bla/persons/mary> rdf:type :Person.
```

[Filename: RDF/subclass.n3]

```
prefix : <foo://bla/names#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X ?T
from <file:subclass.n3>
where {?X rdf:type ?T}
```

[Filename: RDF/subclass.sparql]

- activate the (internal) reasoner when invoking Jena.

## SUBCLASS, DOMAIN, RANGE: EXAMPLE (CONT'D)

Recall the previous example. Given the following facts:

```
:has_cat rdfs:domain :Person .  
:has_cat rdfs:range :Cat .  
:Person rdfs:subClassOf :LivingBeing .  
:Cat rdfs:subClassOf :LivingBeing .  
<foo://bla/persons/john> :has_cat <foo://bla/cats/garfield> .  
<foo://bla/persons/mary> rdf:type :Person .
```

The domain/range information does not act as a constraint, but as information. From that knowledge, the following facts can be *inferred*:

- :has\_cat implies that the subject (John) is a Person, and the object (Garfield) is a cat,
- both are thus LivingBeings.

## SUBPROPERTIES

- outlook: combine it with owl:TransitiveProperty.

```
@prefix : <foo://bla/names#> .
@prefix person: <foo://bla/persons/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
  person:john :child person:alice, person:bob.
  person:kate :child person:john.
  :child rdfs:subPropertyOf :descendant.
  :descendant rdf:type owl:TransitiveProperty.
```

[Filename: RDF/descendants.n3]

```
prefix : <foo://bla/names#>
select ?X ?Y
from <file:descendants.n3>
where {?X :descendant ?Y}
```

[Filename: RDF/descendants.sparql]



## COMPARISON

### SQL

- queries only against the database (no intensional knowledge),
  - equivalent to tree expressions in relational algebra, based on set theory,
  - formal semantics can be given purely syntactically with the algebra,
- ⇒ in the DB lecture, we did not need logic.
- equivalent to the relational calculus, semantics of queries can be given by the calculus. Equivalent to *nonrecursive Datalog* (cf. Slide 100) with “negation as failure” (top-down) stratification (bottom-up).

### SPARQL + RDFS

- only restricted negation
- RDFS: built-in rules (positive, recursive Datalog)
- requires fixpoint computation (recursion by subclasses and subproperties)
- SPARQL: positive, nonrecursive Datalog
- intuitive bottom-up semantics

## RDFS AXIOMATIC TRIPLES

See RDF Semantics and Model Theory, <http://www.w3.org/TR/rdf-mt>.

Axioms: expected to hold in any RDFS model:

```
rdf:type rdfs:domain rdfs:Resource .  
rdfs:domain rdfs:domain rdf:Property .  
rdfs:range rdfs:domain rdf:Property .  
rdfs:subPropertyOf rdfs:domain rdf:Property .  
rdfs:subClassOf rdfs:domain rdfs:Class .
```

```
rdf:type rdfs:range rdfs:Class .  
rdfs:domain rdfs:range rdfs:Class .  
rdfs:range rdfs:range rdfs:Class .  
rdfs:subPropertyOf rdfs:range rdf:Property .  
rdfs:subClassOf rdfs:range rdfs:Class .
```

```
rdfs:Datatype rdfs:subClassOf rdfs:Class .
```

... and some more.

## USING RDF IN THE WORLD WIDE WEB

- The (Semantic) Web is not seen as a collection of documents, but as a collection of correlated information (described via documents)
- using RDF, everybody can make statements about any resource (cf. link-bases in XLink)
  - incremental, world wide data and meta-data
  - distributed RDFS,
  - distributed RDF,
  - often using only virtual resources (URIs).
- not assumed that complete information about any resource is available.
- Open world, no notion of (implicit) negation.

## REASONING BASED ON RDFS

- RDF/RDFS *model theory* as above.
- incomplete knowledge when reasoning: “open world assumption”
- potentially even inconsistent information;
- statements can be equipped with probabilities or labeled as opinions;  
fuzzy reasoning, belief revision ...

... lots of artificial intelligence applications ...

... but there is even more.

## EXAMPLE/EXERCISE

Consider again the employee-manages-departments example (Slide 22).

- Give the RDF Graph.
- give the N3 triples and feed them into the Jena tool.

## ADDITIONAL RDF/RDFS VOCABULARY

The rdf/rdfs namespaces provide some more vocabulary:

Like most data models, RDF provides a representation for *Collections*:

- Collections: `rdf:Alt`, `rdf:Bag`, `rdf:Seq`, `rdf>List` are collections. Lists have properties `rdf:first` (a resource) and `rdf:rest` (a list). Others have properties `_1`, `_2`, ... that refer to their members.
- (`rdfs:Container`, `rdfs:member`, `rdfs:ContainerMembershipProperty`)

... these are partially used implicitly (e.g., collections in `owl:intersectionOf`, `owl:OneOf`), but often not supported by OWL reasoners if used explicitly (see Slides 383 ff.).

## 5.2 Some simple OWL Notions (about modeling and properties)

- so far: RDFS allows for specification of subclasses, subproperties, domain and range
- simple, intuitive, nevertheless problematic (paradoxes).
- development of RDFS and OWL (Web Ontology Language) was not well-defined.
- OWL does *not* build upon RDFS
  - some OWL notions extend RDFS notions,
  - some RDFS notions are not contained in OWL,
  - OWL itself comes in three (incremental) variants.

... this will be analyzed later.

Let's continue with some more intuitive and pragmatic notions contributed by OWL.

- OWL Namespace: `<http://www.w3.org/2002/07/owl#>`

## SUBCLASSES OF PROPERTIES

Triple syntax: *some property rdf:type a specific type of property*

### According to their ranges

- owl:ObjectProperty – subclass of rdf:Property; object-valued (i.e. rdfs:range must be an Object class)
- owl:DatatypeProperty – subclass of rdf:Property; datatype-valued (i.e. its rdfs:range must be an rdfs:Datatype)

... both are not really interesting to derive new things, but useful as constraints.

### According to their Properties

- owl:TransitiveProperty, owl:SymmetricProperty

### According to their Cardinality

- specifying n:1 or 1:n cardinality:  
owl:FunctionalProperty, owl:InverseFunctionalProperty



## TRANSITIVE AND SYMMETRIC PROPERTIES

- transitive: ancestors (cf. Slide 230), train connections etc.
- symmetric: married

```
@prefix : <foo://bla/names#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
  [ :name "John"; :married [ :name "Mary" ] ] .
  :married rdf:type owl:SymmetricProperty.
```

[Filename: RDF/symmetric-married.n3]

```
prefix : <foo://bla/names#>
select ?X ?Y
from <file:symmetric-married.n3>
where { [ :name ?X ; :married [ :name ?Y] ] }
```

[Filename: RDF/symmetric-married.sparql]

## FUNCTIONAL CARDINALITY SPECIFICATION

*a property* rdf:type owl:FunctionalProperty

- not a constraint, but
- if such a property results in two things ... these things are inferred to be the same.

```
@prefix : <foo://bla/names#>.
@prefix person: <foo://bla/persons/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>.
:world :has_pope person:josephratzinger .
:world :has_pope [ :name "Benedikt XVI" ] .
:has_pope rdf:type owl:FunctionalProperty.
```

[Filename: RDF/pop.es.n3]

```
prefix : <foo://bla/names#>
prefix person: <foo://bla/persons/>
select ?N from <file:pop.es.n3>
where { person:josephratzinger :name ?N }
```

[Filename: RDF/pope.sparql]

## INVERSE PROPERTIES

- *some owl:ObjectProperty owl:inverseOf some owl:ObjectProperty*
- owl:DatatypeProperties cannot have an inverse  
(this would define properties of objects, cf. next slide)

```
@prefix : <foo://bla/names#> .
@prefix person: <foo://bla/persons/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
  person:john :child person:alice, person:bob.
  person:john :parent person:kate .
  :ancestor rdf:type owl:TransitiveProperty.
  :child rdfs:subPropertyOf :ancestor.
  :child owl:inverseOf :parent.
```

[Filename: RDF/inverse.n3]

```
prefix : <foo://bla/names#>
select ?X ?Y
from <file:inverse.n3>
where {?X :ancestor ?Y}
```

[Filename: RDF/inverse.sparql]

## No Inverses of owl:DatatypeProperties!

- an owl:DatatypeProperty must not have an inverse:
- “:john :age 35” would imply “35 :ageOf :john” which would mean that a literal has a property, which is not allowed.

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <foo://bla/names#> .
# :john :name "John"; :age 35; :child [:name "Alice"], [:name "Bob"; :age 8].
:age a owl:DatatypeProperty.
:child a rdf:Property.
:childOf owl:inverseOf :child.
:ageOf owl:inverseOf :age.
```

[Filename: RDF/inverseDTProp.n3]

```
jena -t -pellet -if inverseDTProp.n3
```

```
WARN [main] (OWLLoader.java:352) - Unsupported axiom:
```

```
Ignoring inverseOf axiom between foo://bla/names#ageOf (ObjectProperty)
and foo://bla/names#age (DatatypeProperty)
```

## SPECIFICATION OF INVERSE FUNCTIONAL PROPERTIES

- Mathematics: a mapping  $m$  is inverse-functional if the inverse of  $m$  is functional:  
 $x p y$  is inverse-functional, if for every  $y$ , there is at most one  $x$  such that  $xpy$  holds.
- Example:
  - hasCarCode is functional: every country has one car code,
  - hasCarCode is also inverse functional: every car code uniquely identifies a country.
- OWL:  
:m-inverse owl:inverseOf :m .  
:m-inverse a owl:FunctionalProperty .  
not allowed for e.g. mon:carCode a owl:DatatypeProperty:

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

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
```

```
@prefix : <foo:bla#>.
```

```
:carCode a owl:DatatypeProperty; rdfs:domain :Country;  
  owl:inverseOf :isCarCodeOf.
```

```
# :Germany :carCode "D".
```

[Filename: RDF/noinverse.n3]

- the specification is detected to be invalid.

## OWL:INVERSEFUNCTIONALPROPERTY

- such cases are described with owl:InverseFunctionalProperty
- a property  $P$  is an owl:InverseFunctionalProperty if  $\forall x, y_1, y_2 : P(y_1, x) \wedge P(y_2, x) \rightarrow y_1 = y_2$  holds

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:carCode rdfs:domain :Country; a owl:DatatypeProperty;
  a owl:FunctionalProperty; a owl:InverseFunctionalProperty.
:name a owl:DatatypeProperty; a owl:FunctionalProperty.
:Germany :carCode "D"; :name "Germany".
:DominicanRepublic :carCode "D"; :name "Dominican Republic".
```

[Filename: RDF/invfunctional.n3]

- the fragment is detected to be inconsistent.

## OWL:HASKEY (OWL 2)

- description of key attributes  $(k_1, \dots, k_n)$  is a relevant issue in data modeling.  
Note that InverseFunctionalProperty covers the simple case that  $n = 1$ .
- pellet2: not yet supported (27.10.08, tracker ticket #174).

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <foo:bla#>.
:name a owl:DatatypeProperty; a owl:FunctionalProperty.
:Country owl:hasKey (:carCode).
:Germany :carCode "D"; :name "Germany".
:DominicanRepublic :carCode "D"; :name "Dominican Republic".
:Duesseldorf :carCode "D"; :name "Duesseldorf".
```

[Filename: RDF/haskey.n3]

- the fragment is inconsistent.

## Wrap-up

- So far, a reasonable expressiveness for data+schema is provided by RDF, RDFS and simple OWL constructs.
- The rest of OWL will not allow for new concepts, but for a more expressive description of the ones already described by RDFS.
- Graph data model, expressed by triples (the canonical way to express a graph with labeled edges)
  - nodes are individuals, classes, and properties
  - edge labels are properties (and thus also nodes)
- N3 normal form, several abbreviated/nested forms allowed.
- RDFS and OWL semantics tailored to “Open World”, as (inconsistency)-tolerant as possible,
- mapping to first-order logic preferable (decidable Description Logic fragments),
- no negative information? This must be given very explicit as *knowledge*.
- note: in model theory, from “false”, everything follows. Thus, do not derive “false” as long as possible. Be tolerant.



## EXAMPLE: THE MONDIAL ONTOLOGY

See `mondial.n3`, `mondial-europe.n3` and `mondial-meta.n3` on the Web page.

Note that it is highly redundant: defining just `rdfs:domain` and `rdfs:range` of properties implies most of the classes (and also most of the `rdfs:type` relationships in `mondial.n3`).

```
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?X
from <file:mondial.n3>
from <file:mondial-meta.n3>
where {?X rdf:type mon:Country}
```

[Filename: RDF/mondial-meta-query.sparql]

- activate Jena with reasoner (if `mondial.n3` is too big, use `mondial-europe.n3` instead)

Mondial is not an interesting example for RDFS (and OWL):

- it's mainly data, no intensional knowledge, no complex ontology
- for that reason it is a good example for SQL and XML.
- RDFS and OWL is interesting when information is *combined* and additional knowledge can be derived.

## Developing Ontologies

- have an idea of the required concepts and relationships (ER, UML, ...),
- generate a (draft) n3 or RDF/XML instance,
- write a separate file for the metadata,
- load it into Jena with activating a reasoner.
- If the reasoner complains about an inconsistent ontology, check the metadata file alone. If this is consistent, and it complains only when also data is loaded:
  - it may be due to populating a class whose definition is inconsistent and that thus must be empty.
  - often it is due to wrong datatypes. Recall that datatype specification is not interpreted as a constraint (that is violated for a given value), but as additional knowledge.