Semantic Web

Domenico Fabio Savo

Department of Management, Information, and Production Engineering

University of Bergamo

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Department of Management, Information and Production Engineering

Outline

What is the Semantic Web?

"The Semantic Web is a Web of actionable information - information derived from data through a **semantic theory** for interpreting the symbols".

"The semantic theory provides an account of meaning in which the logical connection of terms establishes **interoperability** between systems".

(Shadbot, Hall, Berners-Lee, The Semantic Web revisited, IEEE Intelligent Systems, May 2006)

The Semantic Web: Why?

search on the Web: problems...

...due to the way in which information is stored on the Web

Problem 1: web documents do not distinguish between information content and presentation (solved by XML)

Problem 2: different web documents may represent in different ways semantically related pieces of information

this leads to hard problems for intelligent information search on the Web

Separating content and presentation

Problem 1: web documents do not distinguish between information content and presentation

problem due to the HTML language

problem solved by current technology

- stylesheets (HTML, XML)
- XML

stylesheets allow for separating formatting attributes from the information presented

Separating content and presentation

- XML: eXtensible Mark-up Language
- XML documents are written through a user-defined set of tags
- tags are used to express the semantics of the various pieces of information

XML: example

HTML:

```
<H1>Database</H1>
        <UL>
                 <LI> Teacher: Stefano Paraboschi
                 <LI> Room: 7
                 <LI> Prerequisites: none
        </UL>
```

XML:

```
<course>
     <title> Database </title>
     <teacher> Stefano Paraboschi </teacher>
     <room> 7 </room>
     <prereq>none</prereq>
</course>
```

Limitations of XML

XML does not solve all the problems:

- legacy HTML documents
- different XML documents may express information with the same meaning using different tags

The need for a "Semantic" Web

Problem 2: different web documents may represent in different ways semantically related pieces of information

- different XML documents do not share the semantics of information
- idea: annotate (mark-up) pieces of information to express the meaning of such a piece of information
- the meaning of such tags is shared! $\Rightarrow$ shared semantics

The Semantic Web Iniziative

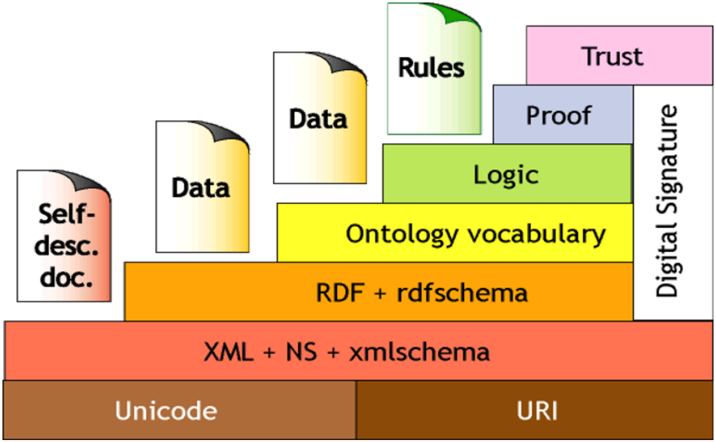
viewpoint: the Web = a web of data

goal: to provide a common framework to share data on the Web across application boundaries

main ideas:

- ontology
- standards
- "layers"

The Semantic Web Tower



The Semantic Web Layers

- XML layer
- RDF + RDFS layer
- Ontology layer
- Proof-rule layer
- Trust layer

The XML Layer

- **XML** (eXtensible Markup Language)

  user-definable and domain-specific markup

- **URI** (Uniform Resource Identifier)

  universal naming for Web resources

  same URI = same resource

  URIs are the ground terms of the SW

W3C standards

RDF + RDFS Layer

- RDF = a simple conceptual data model (W3C standard - 1999)
- RDF model = set of RDF triples
- triple = expression (statement)

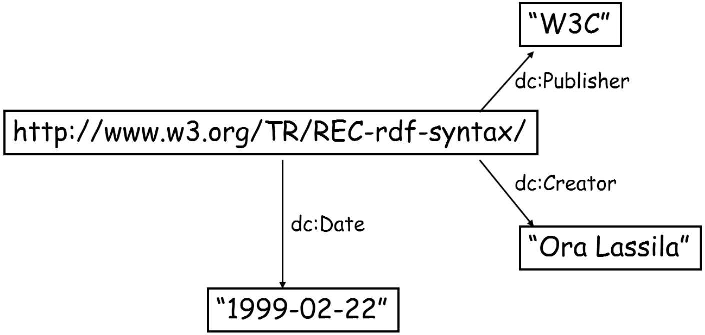
$$(subject, predicate, object)$$

where:
$subject$ = resource
$predicate$ = property (of the resource)
$object$ = value (of the property)

$\Rightarrow$ an RDF model is a **graph**

RDF + RDFS Layer

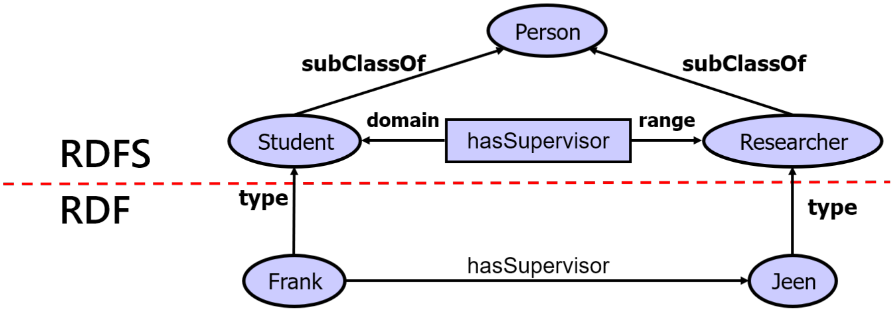
Example of RDF graph:



The RDF + RDFS Layer

- RDFS = RDF Schema
- vocabulary for RDF
- W3C standard (2004)

Example:



The Ontology Layer

- **Ontology** = shared conceptualization

  ⇒ conceptual model (more expressive than RDF + RDFS)

  ⇒ expressed in a true knowledge representation language

- **OWL** (Web Ontology Language) = W3C standard language for ontologies

The proof/rule layer

Beyond OWL:

- proof/rule layer
- rule: informal notion
- rules are used to perform inference over ontologies
- rules as a tool for capturing further knowledge (not expressible in OWL ontologies)

The Trust Layer

SW top layer:

- support for **provenance/trust**
- **provenance:**
  - where does the information come from?
  - how this information has been obtained?
  - can I trust this information?
- largely unexplored issue
- no standardization effort

The Semantic Web: the main ingredients

- underlying web layer (URI, XML)

  reusing and extending web technologies
- basic conceptual modeling language (RDF)
- ontology language (OWL)
- rules/proof
- reusing and extending AI technologies

  knowledge representation

  automated reasoning
- ...and database technologies

  data integration

The Notion of Ontology

- ontology = **shared conceptualization** of a domain of interest (Gruber, 1993)
- shared vocabulary ⇒ simple (shallow) ontology
- (complex) relationships between terms ⇒ deep ontology
- **AI view:**

  ontology = logical theory (knowledge base)
- **DB view:**

  ontology = conceptual model

Ontology: example

```
class-def animal                        % animals are a class
class-def plant                         % plants are a class
   subclass-of NOT animal               % that is disjoint from animals
class-def tree
   subclass-of plant                    % trees are a type of plants
class-def branch
   slot-constraint is-part-of           % branches are parts of some tree
       has-value tree
       max-cardinality 1
class-def defined carnivore             % carnivores are animals
   subclass-of animal
   slot-constraint eats                 % that eat any other animals
       value-type animal
class-def defined herbivore             % herbivores are animals
   subclass-of animal, NOT carnivore % that are not carnivores, and
   slot-constraint eats                   % they eat plants or parts of plants
       value-type plant OR (slot-constraint is-part-of has-value plant)
```

Ontologies: the role of logic

- **ontology** = **logical theory**
- **why?**

  declarative

  formal semantics

  reasoning (sound and complete inference techniques)
- well-established correspondence between conceptual modeling formalisms and logic

Ontologies and Description Logics

- OWL is based on a fragment of first-order predicate logic (FOL)
- Description Logics (DLs) = subclasses of FOL

  only unary and binary predicates

  function-free

  quantification allowed only in restricted form

  decidable reasoning
- DLs are one of the most prominent languages for Knowledge Representation
- expressive abilities of DLs have been widely explored reasoning in DLs has been extensively studied
- DL reasoners have been developed and optimized

$\Rightarrow$ **DLs as a central technology for the SW**

Linked Data

Linked Data: a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF.

Linking Open Data (LOD): "The goal of the W3C SWEO Linking Open Data community project is to **extend the Web with a data commons by publishing various open data sets as RDF on the Web** and by **setting RDF links between data items from different data sources.** RDF links enable you to navigate from a data item within one data source to related data items within other sources using a Semantic Web browser. As query results are structured data and not just links to HTML pages, they can be used within other applications"

Outline

Syntax and semantics

- **Syntax**: the structure of data
- **Semantics**: the meaning of data
- Two conditions necessary for interoperability:
  - **Adopt a common syntax**: this enables applications to parse the data.
  - **Adopt a means for understanding the semantics**: this enables applications to use the data.

XML

- XML: eXtensible Mark-up Language
- XML documents are written through a user-defined set of tags
- tags are used to express the semantics of the various pieces of information

Can we use XML to represent semantics?

Limitation for semantic markup

XML makes no commitment on:

(1.) Domain-specific ontological vocabulary

(2.) Ontological modeling primitives

Requires pre-arranged agreement on (1.) and (2.)

Only feasible for closed collaboration:

- agents in a small and stable community
- pages on a small and stable intranet

Not suited for sharing Web-resources

RDF

RDF is a data model

- the model is domain-neutral, application-neutral and ready for internationalization
- the model can be viewed as directed, labeled graphs or as an object-oriented model (object/attribute/value)

RDF data model is an abstract, conceptual layer independent of XML

- consequently, XML is a transfer syntax for RDF, not a component of RDF
- RDF data might never occur in XML form

The RDF Model

RDF Model = Set of **triples**

Triple = expression (statement)

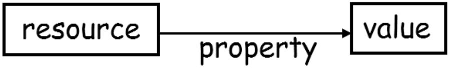
(subject, predicate, object)

where:

subject: resource

predicate: property (of the resource)

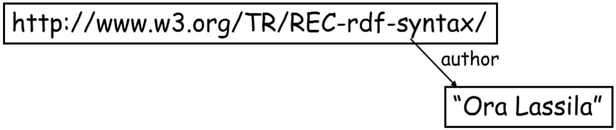
object: value (of the property)



RDF Triples

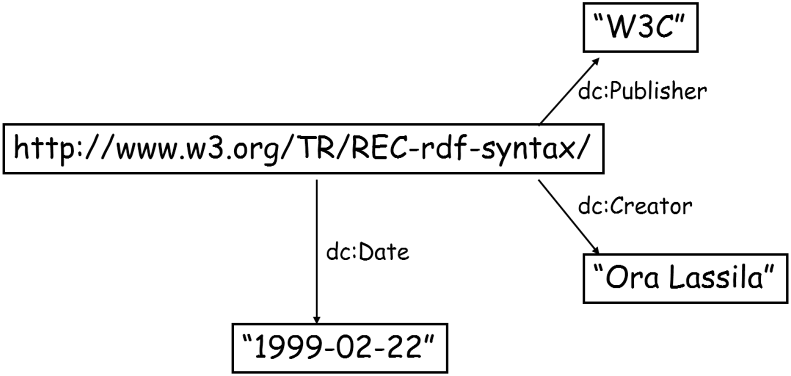
Example: '**the document at http://www.w3c.org/TR/REC-rdf-syntax has the author Ora Lassila**"

Triple: http://www.w3c.org/TR/REC-rdf-syntax author "Ora Lassila"



The RDF Model is a Graph

RDF graph: example



Node and edge labels in RDF graphs

Node and edge labels:

- URI
- literal (string)
- blank node (anonymous label)

but:

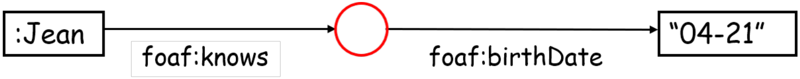
- a literal can only appear in object positions
- a blank node can only appear in subject or object positions

Note that since URIs can be used as predicates, then graph nodes can be used as edge labels $\Rightarrow$ **RDF has meta-modeling abilities**

Blank Nodes

Blank node (bnode) are RDF graph nodes with anonymous label (i.e., not associated with an URI)

example: "Jean has a friend born the 21st of April"



ex:Jean	foaf:knows	_:p1
_:p1	foaf:birthDate	04-21

_:p1 is the blank node (b-node)

bnodes can be used both as subjects and objects

Higher-order statements

In RDF, it is possible to specify statements that refer to other statements.

Example: "Ralph believes the Web contains one billion documents."

These types of statements are called Higher-order and allow expressing opinions

They are important for trust models, digital signatures, etc. and are represented through the modeling of RDF in RDF itself

Main tool: **Reification**, or the representation of an RDF statement as a resource

Reification I

Reification in RDF: the use of an RDF statement as the subject (or object) of another RDF statement

Examples of statements that need reification to be expressed in RDF:

- "The New York Times claims that **Joe is the author of the book ABC**";
- "The statement "**The technical report on RDF was written by Ora Lassila**" was written by the Library of Congress"

Reification II

RDF provides a vocabulary for **reification**: the use of an RDF statement as the subject (or object) of another RDF statement

- rdf:subject
- rdf:predicate
- rdf:object
- rdf:Statement

Through this vocabulary (i.e., these IRIs from the RDF namespace) it is possible to represent a **triple as a blank node**

Reification - an example I

The statement "**the technical report on RDF was written by Ora Lassila**" can be represented by the following 4 triples:

```
_:x rdf:type rdf:statement.
_:x rdf:predicate dc:creator.
_:x rdf:subject http://www.w3.org/REC-rdf-syntax/.
_:x rdf:object "Ora Lassila".
```

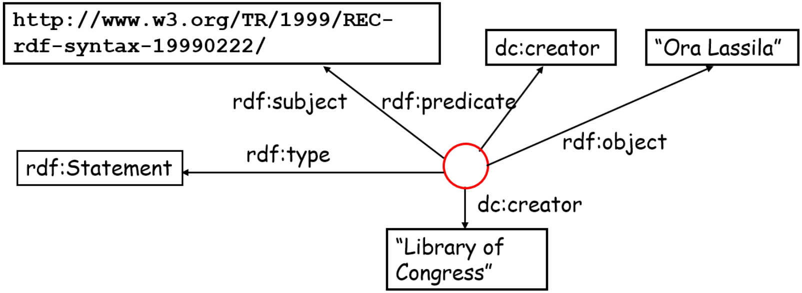
The blank node `_:x` is the **reification** of the statement (it is an anonymous URI that represents the entire triple).

In the same way, *"the statement* "**The technical report on RDF was written by Ora Lassila**" *was written by the Library of Congress*" can be represented by using `_:x` and adding this triple:

```
_:x dc:creator "Library of Congress".
```

Reification - an example II

*"the statement "***The technical report on RDF was written by Ora Lassila***" was written by the Library of Congress"*



RDF syntaxes

RDF model = edge-labeled graph = set of triples

Informally an RDF model can be represented graphically (as a graph), or as a set of triple (subject, predicate, object).

Formal syntaxes:

- N3 notation
- Turtle notation
- concrete (serialized) syntax: RDF/XML syntax

RDFS: RDF Schema

Defines small vocabulary for RDF:

- Class, subClassOf, type
- Property, subPropertyOf
- domain, range

Correspond to a set of RDF predicates:

- meta-level
- special (predefined) **meaning**

RDFS: vocabulary for defining classes and properties

Vocabulary for defining classes:

- **rdfs:Class** (a resource is a class)
- **rdf:type** (a resource is an instance of a class)
- **rdfs:subClassOf** (a resource is a subclass of another resource)

Vocabulary for defining properties:

- **rdf:Property** (a resource is a property)
- **rdfs:domain** (denotes the first component of a property)
- **rdfs:range** (denotes the second component of a property)
- **rdfs:subPropertyOf** (expresses ISA between properties)

RDFS – Example

Example (classes)

(ex:MotorVehicle, rdf:type, rdfs:Class)

(ex:PassengerVehicle, rdf:type, rdfs:Class)

(ex:Van, rdf:type, rdfs:Class)

(ex:Truck, rdf:type, rdfs:Class)

(ex:MiniVan, rdf:type, rdfs:Class)

(ex:PassengerVehicle, rdfs:subClassOf, ex:MotorVehicle)

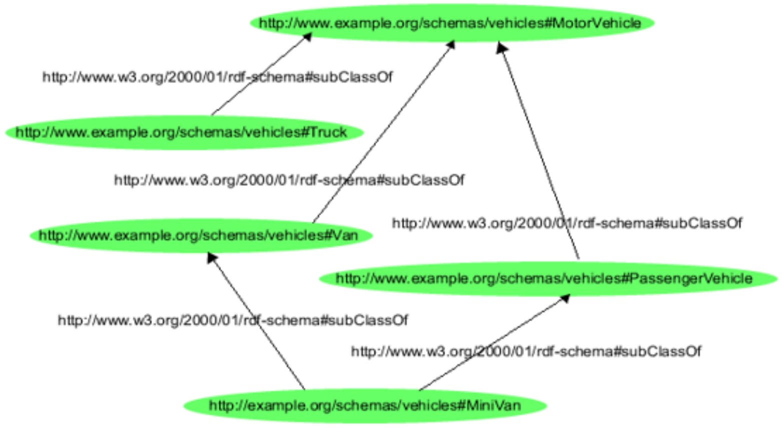
(ex:Van, rdfs:subClassOf, ex:MotorVehicle)

(ex:Truck, rdfs:subClassOf, ex:MotorVehicle)

(ex:MiniVan, rdfs:subClassOf, ex:Van)

(ex:MiniVan, rdfs:subClassOf, ex:PassengerVehicle)

RDFS — Example


http://www.example.org/schemas/vehicles#MotorVehicle
http://www.w3.org/2000/01/rdf-schema#subClassOf
http://www.example.org/schemas/vehicles#Truck
http://www.w3.org/2000/01/rdf-schema#subClassOf
http://www.example.org/schemas/vehicles#Van
http://www.w3.org/2000/01/rdf-schema#subClassOf
http://www.example.org/schemas/vehicles#PassengerVehicle
http://www.w3.org/2000/01/rdf-schema#subClassOf
http://www.w3.org/2000/01/rdf-schema#subClassOf
http://example.org/schemas/vehicles#MiniVan

RDFS – Example

Example (properties)

– –

(ex:weight, rdf:type, rdf:Property)

(ex:weight, rdfs:domain, ex:MotorVehicle)

(ex:weight, rdfs:range, Integer)

RDFS: meta-modeling abilities

Example (meta-classes)

(ex:MotorVehicle, rdf:type, rdfs:Class)

(ex:myClasses, rdf:type, rdfs:Class)

(ex:MotorVehicle, rdf:type, ex:myClasses)

RDF + REDF: Semantics

What is the exact meaning of an RDF(S) graph?

Initially, a formal semantics was not defined!

Main problems:

- bnodes
- meta-modeling
- formal semantics for RDFS vocabulary

In 2004, a model-theoretic semantics was provided
$\Rightarrow$ formal definition of entailment and query answering over RDF(S) graphs

Incomplete information in RDF graphs

bnodes are considered as existential values (null values)

This Introduces incomplete information in RDF graphs.

An RDF graph can be seen as an incomplete database represented in the form of a naive table

$\Rightarrow$ An RDF graph is represented by a unique table T, with values being constants or named existential variables.

$\Rightarrow$ An RDF graph can be seen as a boolean conjunctive query over the unique relation T.

Formal semantics for RDF + RDFS

Formal semantics for RDFS vocabulary.

RDFS statements = constraints over the RDF graph

$\Rightarrow$ entailment in RDF + RDFS = reasoning (query answering) over an incomplete database with constraints

Outline

Querying RDF: SPARQL

SPARQL: Simple Protocol And RDF Query Language

W3C standardization effort similar to the XQuery query language for XML data

Suitable for remote use (remote access protocol)

Example:

− − − − − − − − − − − − − − − − −

```
PREFIX abc: <http://mynamespace.com/exampleOntology#>
SELECT ?capital ?country
WHERE { ?x abc:cityname ?capital.
        ?y abc:countryname ?country.
        ?x abc:isCapitalOf ?y.
        ?y abc:isInContinent abc:africa. }
```

− − − − − − − − − − − − − − − − −

− Variables are outlined through the "?" prefix ("$" is also possible).

− The **?capital** and the **?country** will be returned.

− The SPARQL query processor returns all hits matching the pattern of the four RDF-triples.

SPARQL: query structure I

A SPARQL query includes, in the following order:

- prefix declaration, to shorten the URI (optional)
- definition of datasets, to specify the graph you want to query (potentially more than one)
- **SELECT** clause, to specify the information to be returned
- **WHERE** clause, to specify the query (or graph) pattern, i.e., the conditions that must be satisfied by the triples in the dataset
- any additional operators, to reorganize the query result (optional)

SPARQL: query structure II

```
 #prefix definitions
PREFIX es:  <...> ...
 #dataset definition
FROM <...>
 #data to be returned
SELECT ...
 #graph pattern specification
WHERE { ...}
 #operators
ORDER BY ...
```

SPARQL query forms

In addition to the SELECT query form SPARQL provides also:

- The **CONSTRUCT** query form that returns an RDF graph specified by a graph template.
- The **ASK** query form which can be used to test whether or not a graph pattern has a solution (Boolean query). In this case, no information is returned, but only whether or not a solution exists

SPARQL: example

Example:

```
PREFIX abc: <http://mynamespace.com/exampleOntology#>
SELECT ?capital ?country
WHERE { ?x abc:cityname ?capital.
               ?y abc:countryname ?country.
               ?x abc:isCapitalOf ?y.
               ?y. abc:isInContinent abc:africa.
               }
```

- the variables are identified by the prefix "?" (or "$")
- the query will return ?capital and ?country
- the SPARQL query engine returns the objects that satisfies the pattern of the four triple RDFs in the query

SPARQL BGP

Basic graph pattern (BGP) are sets of triple patterns (RDF graph with possibly variables as labels).

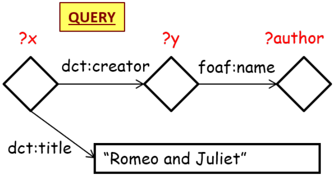
```
PREFIX abc:<http://mynamespace.com/exampleOntology# >
SELECT ?capital ?country
WHERE { ?x abc:cityname ?capital.
        ?y abc:countryname ?country.
        ?x abc:isCapitalOf ?y.
        ?y.  abc:isInContinent abc:africa.  }
```

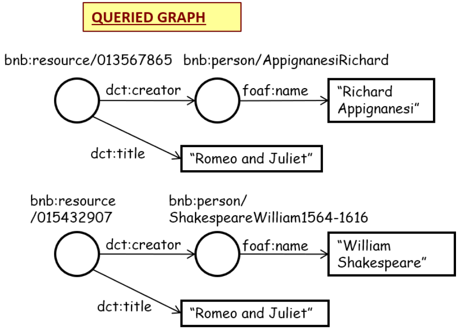
SPARQL defines on top of a BGP additional operators:

- AND
- FILTER
- UNION
- OPTIONAL

SPARQL: query evaluation

The query will return all the resources R for which there exist resources X and Y such that by replacing the variables ?authors, ?x, and ?y, respectively, we obtain the triples in the queried graph.





RESULT

author
"Richard Appignanesi"
"William Shakespeare"

The result of a query is a set of tuples (analogous to SQL), whose structure (labels and cardinality) reflects the SELECT clause of the query

SPARQL Query – Example

RDF graph:

```
@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> .
```

Query:

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

Result:

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

Constraints on Solutions

SPARQL allows for restricting variable bindings to those that satisfy some filter conditions by using the FILTER keyword.

Filter expression are Boolean expressions and only those results where the expression returns `true` are used.

The operator that can be used for specifying the filter expressions can be:

- **Logical**: !, $\&\&$, $||$
- **Mathematical**: $+$, $-$, $*$,$/$
- **Comparison**: $=$, !$=$, $>$, $<$, IN, $NOT\ IN$...
- **SPARQL tests**: `isIRI, isURI, isBlank, isLiteral, ...`
- **Regular expressions**
- and more...

Use of filters: example

RDF graph:

```
@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> .
```

Query:

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

Result:

"Johnny Lee Outlaw"	<mailto:jlow@example.com>

OPTIONAL Graph Patterns

Since one can not assume to always have complete structures in all RDF graphs it is useful to specify queries that allow information to be added to the answer where the information is available, but do not reject the answer because some parts of the query pattern do not match.

Optional graph pattern provides this facility: if the optional part does not match, no binding is created but the solution is not eliminated (similar to SQL outer JOINs).

The Optional part of a basic graph pattern is specified by using the **OPTIONAL** keyword:

```
{P1 OPTIONAL { P2 } }
```

The BGP following a keyword OPTIONAL can of be as a normal BGP.

Optional patterns: example

RDF graph:

```
@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> .
```

Query:

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

Result:

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

SPARQL 1.1: property paths

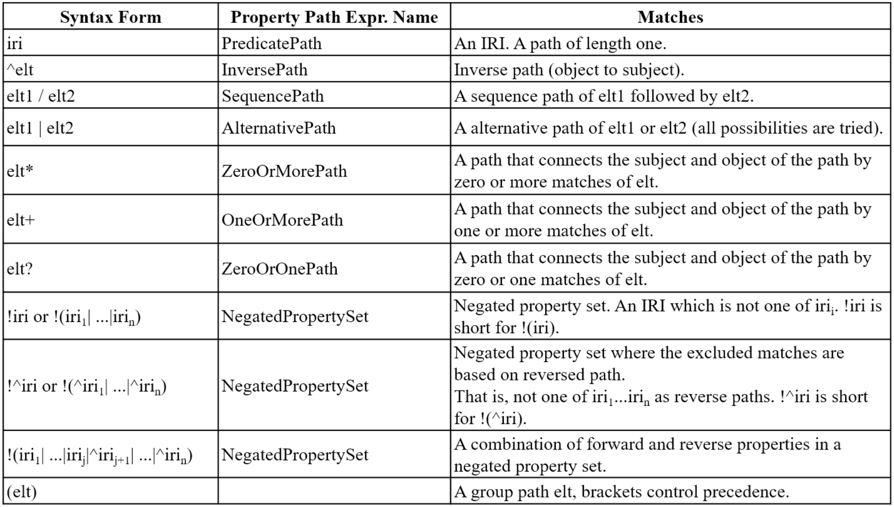
SPARQL 1.1 (released in 2013) extends the first version of SPARQL in different ways (e.g., aggregate functions, entailment regimes)

In particular, it allows for expressing property paths in queries.

A property path is essentially a regular expression using properties (URIs)

Property paths allow for expressing paths of arbitrary length along the RDF graph, thus providing a fundamental graph-oriented feature to SPARQL

Property paths in SPARQL



Syntax Form	Property Path Expr. Name	Matches
iri	PredicatePath	An IRI. A path of length one.
^elt	InversePath	Inverse path (object to subject).
elt1 / elt2	SequencePath	A sequence path of elt1 followed by elt2.
elt1 | elt2	AlternativePath	A alternative path of elt1 or elt2 (all possibilities are tried).
elt*	ZeroOrMorePath	A path that connects the subject and object of the path by zero or more matches of elt.
elt+	OneOrMorePath	A path that connects the subject and object of the path by one or more matches of elt.
elt?	ZeroOrOnePath	A path that connects the subject and object of the path by zero or one matches of elt.
!iri or !(iri$_1$| ...|iri$_n$)	NegatedPropertySet	Negated property set. An IRI which is not one of iri$_i$. !iri is short for !(iri).
!^iri or !(^iri$_1$| ...|^iri$_n$)	NegatedPropertySet	Negated property set where the excluded matches are based on reversed path. That is, not one of iri$_1$...iri$_n$ as reverse paths. !^iri is short for !(^iri).
!(iri$_1$| ...|iri$_j$|^iri$_{j+1}$| ...|^iri$_n$)	NegatedPropertySet	A combination of forward and reverse properties in a negated property set.
(elt)		A group path elt, brackets control precedence.

Negation in SPARQL

To express negation, SPARQL provides the **MINUS** binary operator and the **NOT EXISTS** clause.

- The **MINUS** operator calculates the difference between the binding of the main query and the one from the sub-query given by the operator.

```
PREFIX o: <http://opera.com/>
SELECT ?title
WHERE { ?opera o:hasCharacter ?ch.
        ?opera o:hasTitle ?title.
MINUS { ?opera o:hasCharacter 'Sigfredo'.
        ?opera o:hasTitle ?title.  }
}
```

- The **NOT EXISTS** expression defined within the FILTER clause filters out the results in binding of the graph pattern in the where clause for which it is not possible to calculate a binding with the graph pattern defined in the scope of the NOT EXISTS expression.

```
PREFIX o: <http://opera.com/>
SELECT ?title
WHERE { ?opera o:hasTitle ?title.
        FILTER (NOT EXISTS  ?opera o:hasCharacter 'Sigfredo')
}
```

RDF/SPARQL tools

- Jena = Java framework for handling RDF models and SPARQL queries (http://jena.sourceforge.net/)
- ARC = PHP implementation of a RDF/SPARQL engine (http://arc.semsol.org/)
- Virtuoso = database system able to natively deal with RDF data and SPARQL queries (http://virtuoso.openlinksw.com/)
- ...and many more (see http://esw.w3.org/topic/SparqlImplementations)

Outline

Back to the "semantic" issue

Does RDF solve the "semantic problem" that XML does not solve?

- no, unless the vocabulary used is shared

  $\Rightarrow$ the "meaning" of a predicate (edge) in an RDF graph must be known (shared) by every application.

- Can a program reason about some terms?

  if "A" is left of "B" and "B" is left of "C", is "A" left of "C"?
  obviously true for humans, not obvious for a program . . .
  . . . programs should be able to deduce such statements.

Limitations of RDFS

RDFS too weak to describe resources in sufficient detail

- No localised range and domain constraints

  Cant say that the range of hasChild is person when applied to persons and elephant when applied to elephants

- No existence/cardinality constraints

  Cant say that all instances of person have a mother that is also a person, or that persons have exactly 2 parents

- No transitive, inverse or symmetrical properties

  Cant say that isPartOf is a transitive property, that hasPart is the inverse of isPartOf or that touches is symmetrical

- ...

Ontologies

The Semantic Web needs a support of ontologies.

“defines the concepts and relationships used to describe and represent an area of knowledge”

Need of a Web Ontologies Language to define:

- the terminology used in a specific context
- possible constraints on properties
- the logical characteristics of properties
- the equivalence of terms across ontologies
- etc.

Web Ontology language Requirements

Desirable features identified for Web Ontology Language:

- Extends existing Web standards (XML, RDF, RDFS)
- Easy to understand and use (should be based on familiar KR idioms)
- Formally specified
- "Adequate" expressive power
- Possible to provide automated reasoning support

Initially, two languages developed to satisfy above requirements: **DAML** and **OIL**

The **OWL** language (based on DAML+OIL) became a W3C recommendation in 2004

OWL – Web Ontology Language

- Semantic Web KR language based on description logics (DLs).
- KR for web resources, using URIs as identifiers

The first version of the OWL family (standardized in 2004) was constituted by 3 different languages (with different expressive power):

- **OWL Full:** union of OWL syntax and RDF
- **OWL-DL** is a variant of $\mathcal{SHOIN}(D)$, where:
  - $\mathcal{S}$ stands for $\mathcal{ALC}$ extended with transitive roles,
  - $\mathcal{H}$ stands for role hierarchies (i.e., role inclusion assertions),
  - $\mathcal{O}$ stands for nominals, which means the possibility of using individuals in the TBox (i.e., the intensional part of the ontology),
  - $\mathcal{I}$ stands for inverse roles,
  - $\mathcal{N}$ stands for (unqualified) number restrictions.
  - (D) stand for data types, which are necessary in any practical knowledge representation language.
- **OWL-Lite:** subset of OWL DL and variant of the DL $\mathcal{SHIF}(D)$, where:
  - $\mathcal{F}$ stands for functionality of roles,

OWL 2

A new version of OWL, **OWL 2**, has become a W3C standard in 2009.

The design aim of OWL2 was to address user requirements for more expressivity of the language, while still preserving decidability of reasoning.

OWL 2 is a variant of $\mathcal{SROIQ}(D)$, which adds to OWL(1) DL several features:

- qualified number restrictions ($\mathcal{Q}$)
- regular role hierarchies ($\mathcal{R}$)
- better treatment of datatypes

The **OWL 2 profiles** are sublanguages of OWL 2 that are restricted in ways that significantly simplify ontological reasoning

- **OWL 2 QL** based on the DL DL-Lite
- **OWL 2 EL** based on the DL EL
- **OWL 2 RL** based on the DL RL

OWL Class Constructors

OWL construct	**DL construct**	**Example**
`ObjectIntersectionOf`	$C_1 \sqcap \cdots \sqcap C_n$	Human $\sqcap$ Male
`ObjectUnionOf`	$C_1 \sqcup \cdots \sqcup C_n$	Doctor $\sqcup$ Lawyer
`ObjectComplementOf`	$\neg C$	$\neg$Male
`ObjectOneOf`	$\{a_1\} \sqcup \cdots \sqcup \{a_n\}$	{john} $\sqcup$ {mary}
`ObjectAllValuesFrom`	$\forall P.C$	$\forall$hasChild.Doctor
`ObjectSomeValuesFrom`	$\exists P.C$	$\exists$hasChild.Lawyer
`ObjectMaxCardinality`	$(\leq n\, P)$	($\leq$ 1 hasChild)
`ObjectMinCardinality`	$(\geq n\, P)$	($\geq$ 2 hasChild)

...

Note: all constructs come also in the `Data...` instead of `Object...` variant.

OWL RDFS Syntax

Example: class $Person \sqcup \forall hasChild.Doctor \sqcup \exists hasChild.Doctor$

```
<owl:Class>
  <owl:intersectionOf rdf:parseType=" collection">
    <owl:Class rdf:about="#Person"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:toClass>
        <owl:unionOf rdf:parseType=" collection">
          <owl:Class rdf:about="#Doctor"/>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild"/>
            <owl:hasClass rdf:resource="#Doctor"/>
          </owl:Restriction>
        </owl:unionOf>
      </owl:toClass>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Other OWL Syntaxes

Example: class $Person \sqcup \forall hasChild.Doctor \sqcup \exists hasChild.Doctor$

OWL Functional Style Syntax:

ObjectUnionOf(:Person ObjectSomeValuesFrom(:hasChild :Doctor)
ObjectAllValuesFrom(:hasChild :Doctor))

Manchester Syntax:

Person or (hasChild some Doctor) or (hasChild only Doctor)

OWL axioms

OWL axiom	**DL syntax**	**Example**
`SubClassOf`	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
`EquivalentClasses`	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
`DisjointClasses`	$C_1 \sqsubseteq \neg C_2$	Man $\sqsubseteq$ $\neg$Female
`SameIndividual`	$\{a_1\} \equiv \{a_2\}$	{presObama} $\equiv$ {B.Obama}
`DifferentIndividuals`	$\{a_1\} \sqsubseteq \neg\{a_2\}$	{john} $\sqsubseteq$ $\neg${peter}
`SubObjectPropertyOf`	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
`EquivalentObjectProperties`	$P_1 \equiv P_2$	hasCost $\equiv$ hasPrice
`InverseObjectProperties`	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
`TransitiveObjectProperty`	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor
`FunctionalObjectProperty`	(**functional** P)	(**functional** hasFather)

...

OWL Semantics

Mapping **OWL** to equivalent DL $\mathcal{SROIQ}(D)$:

- Facilitates provision of reasoning services (using DL systems)
- Provides well defined semantics

DL semantics defined by **interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where:

$\Delta^{\mathcal{I}}$ is the *domain* of $\mathcal{I}$ (a non-empty set)

$\cdot^{\mathcal{I}}$ is an *interpretation function*, which maps:

individual c to an element $c^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$

each atomic concept (class) A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$

each atomic role (property) P to a subset $P^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

Inference Tasks in OWL

same as DLs: OWL ontology = DL Ontology

main reasoning tasks over ontologies:

- **consistency of the ontology:** given an ontology $\mathcal{O}$ verify if the set of models of $\mathcal{O}$ is not empty;
- **class (and property) consistency:** given an ontology $\mathcal{O}$ and a class C verify if $\mathcal{O} \models C \sqsubseteq \bot$;
- **class (and property) subsumption:** given an ontology $\mathcal{O}$ and two classes (properties) C_1 and C_2 verify if $\mathcal{O} \models C_1 \sqsubseteq C_2$;
- **instance checking:** given an ontology $\mathcal{O}$, class (resp. a property R) C an instance c (resp. a pair $(c1, c2)$), if $\mathcal{O} \models C(c)$ (resp. $\mathcal{O} \models R(c1, c2)$);
- **instance retrieval:** given and ontology $\mathcal{O}$ and class C, compute all the individual c in $\mathcal{O}$ such that $\mathcal{O} \models C(c)$
- **query answering:** given an ontology $\mathcal{O}$ and a query q (expressed in some query language), compute all tuples of individuals $\vec{t}$ such that $q(\vec{t})$ is entailed by $\mathcal{O}$.

Computational aspects of reasoning

reasoning in OWL 2 is decidable (and the complexity is characterized)

however: high computational complexity (EXPTIME)

(optimized) reasoning algorithms developed

However, processing OWL is computationally **too expensive** (exponential) especially for data-intensive Semantic Web applications, where scalability (or at least tractability) of processing/reasoning is a crucial property.

The OWL profiles: a solution

How to overcome these limitations if we want to build data-intensive Semantic Web applications?

- **solution 1:** do not reason over ontologies;
- **solution 2:** limit the expressive power of the ontology language

  ⇒ **tractable fragments of OWL (OWL profiles)**
  - **OWL 2 QL** based on the DL DL-Lite
  - **OWL 2 EL** based on the DL EL
  - **OWL 2 RL** based on the DL RL

Resoning in OWL 2 QL (DL-Lite)

Deciding if an ontology expressed in OWL 2 QL is consistent is polynomial.

Query answering in OWL 2 QL is in LOGSPACE with respect to data (ABox) complexity (first-order rewritability).

All main reasoning tasks in OWL 2 QL can be reduced to either ontology consistency or query answering.

$\Rightarrow$ **all main reasoning tasks in DL-Lite are tractable**

Resoning in OWL 2 EL (EL)

Complexity of reasoning in OWL 2 EL:

Intensional (TBox) reasoning is PTIME-complete (i.e., tractable).

Instance checking is PTIME-complete.

Conjunctive query answering is PTIME-complete with respect to data complexity (no first-order rewritability).

Resoning in OWL 2 RL (RL)

Intensional (TBox) reasoning is PTIME-complete (i.e., tractable).

Instance checking is PTIME-complete.

Conjunctive query answering is PTIME-complete with respect to data complexity (no first-order rewritability).

Reasoning in RL can be reduced to reasoning in positive Datalog.

OWL editors

- allow for visualizing/browsing/editing OWL ontologies
- able to connect to an external OWL reasoner
- sever OWL Ontology editors.
- the main current too is Protégé

OWL 2 Reasoner

Two categories:

OWL-DL reasoners, e.g.:

- Hermit
- Pellet
- Konclude
- Racer, RacerPro
- Fact++

Reasoners for ”tractable fragments” of OWL 2, e.g.:

- ELK (OWL 2 EL)
- Mastro, Ontop (OWL 2 QL)
- RDFox (OWL 2 RL)