Introduction to the Semantic Web Technologies

Konstantin Schekotihin

Alpen-Adria-Universität
Klagenfurt, Austria

Agenda

RDF summary

- Triple is a fundamental data structure of RDF
- A set of triples can be represented as a graph, where subjects and objects correspond to nodes and predicates to edges
- Two or more RDF graphs can be merged to a one graph
- Namespaces are sets of names used to simplify encoding and prevent collisions of abbreviations
- Existentially quantified variables are expressed as blank nodes

```
uni:student ex:likes [skos:prefLabel "Guinness"]
```

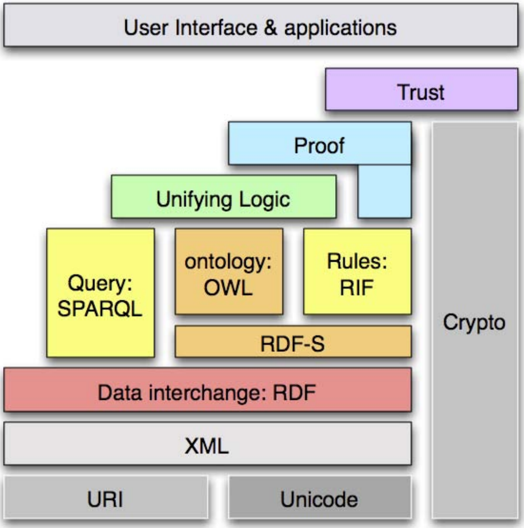
- rdf:type specifies relations between instances and classes

```
beer:Guinness rdf:type beer:Stout
```

- rdf:Property is the root property type indicating an identifier as a property rather than a subject or an object

```
foaf:name rdf:type rdf:Property
```

Semantic Web Technology Stack



RDF Schema

RDF allows to encode facts in form of triples

`<subject, predicate, object>`

Problem it is required to define a relation between two descriptions

- Students have an integer matriculation number
- Parent is a person that has a child

Solution create a language allowing to define classes (subject and object), properties (predicates) and relations between them

- RDF Schema is a W3C RDF Recommendation
- Allows to define “lightweight” hierarchies of classes as well as domain and range restrictions of properties
- RDF Schema is an RDF Document that uses RDF vocabulary
- RDFS Namespace:

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

RDFS Classes

Classes describe sets of instances, that are defined with URIs in RDF

ex:Nikon_D3 **rdf:type** ex:DigiCam .
ex:Nikon_D3 **rdf:type** ex:Nikon_Product .

Classes can form hierarchies X **rdfs:subClassOf** Y, i.e. everything of type X is also of type Y

ex:DigiCam **rdfs:subClassOf** ex:Camera .
ex:Camera **rdfs:subClassOf** ex:Photography_Equipment .

By the semantics of **rdfs:subClassOf** one can obtain the following entailments

ex:Nikon_D3 **rdf:type** ex:Camera .
ex:Nikon_D3 **rdf:type** ex:Photography_Equipment .
ex:DigiCam **rdf:subClassOf** ex:Photography_Equipment .

RDFS Properties

- Using `X` **rdfs:subPropertyOf** `Y` properties can also be organized in hierarchies just as classes

  ```
  ex:Nikon_D3 ex:madeBy           ex:Nikon .
  ex:madeBy   rdfs:subPropertyOf  ex:producedBy .
  ```

  These definitions allows us to conclude that

  ```
  ex:Nikon_D3 ex:producedBy       ex:Nikon .
  ```

- `X` **rdfs:domain** `Y` defines that
  - `X` is in instance of the class **rdf:Property** and `Y` is an instance of **rdfs:Class** where `rdf:Property` **rdfs:subClassOf** `rdfs:Class`
  - all subjects of triples whose predicate is `X` are of the type `Y`
- `X` **rdfs:range** `Y` defines that
  - all objects of triples whose predicate is `X` are of the type `Y`

Example: RDFS Properties

For example, given

$$
\begin{array}{lll}
\texttt{ex:madeBy} & \textbf{rdf:type} & \textbf{rdf:Property} \\
\texttt{ex:madeBy} & \textbf{rdfs:domain} & \texttt{ex:Product} \ . \\
\texttt{ex:madeBy} & \textbf{rdfs:range} & \texttt{ex:Company} \ .
\end{array}
$$

```
ex:Nikon_D3 ex:madeBy ex:Nikon .
```

we infer that

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

Example: RDFS Properties

For example, given

`ex:madeBy`	**rdf:type**	**rdf:Property**	
`ex:madeBy`	**rdfs:domain**	`ex:Product`	.
`ex:madeBy`	**rdfs:range**	`ex:Company`	.

```
ex:Nikon_D3 ex:madeBy ex:Nikon .
```

we infer that

`ex:Nikon_D3`	**rdf:type**	`ex:Product`	.
`ex:Nikon`	**rdf:type**	`ex:Company`	.

Domains and Ranges

- Multiple domain/range properties for one property `X` can be used to complex domain/range definitions

```
ex:madeBy  rdfs:range ex:Farmer .
ex:madeBy  rdfs:range ex:Company .
```

  means that every object in a triple with predicate `ex:madeBy` is both `ex:Farmer` and `ex:Company`

- Ranges can also include datatype restrictions **rdfs:Datatype**, which is instance and subclass of **rdfs:Class**

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

ex:shutterSpeed rdfs:range xsd:nonNegativeInteger .
```

Other features

- **rdfs:Literal** class of literal values, e.g. strings, integers, etc. Literals can be plain and typed
- **rdfs:Datatype** class of datatypes. Each instance of this class is a subclass of **rdfs:Literal**
- **rdf:XMLLiteral** class of XML values
- **rdfs:label** is used to provide human-readable version of the resource name and is an instance of **rdf:Property**
- RDFS extends handling of containers with some additional properties

FOL representation

- Each RDF document can be translated into a FOL formula

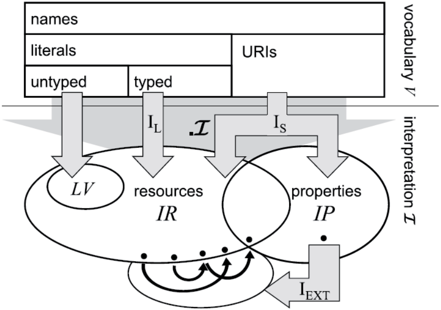
```
sec:horster rdf:type foaf:Person;
                   foaf:knows sec:rass,
                              [ a foaf:Person;
                                foaf:name "Peter" ] .
```

$$
\begin{aligned}
\exists bn_1 (& triple(sec{:}horster, rdf{:}type, foaf{:}Person) \wedge \\
& triple(sec{:}horster, foaf{:}knows, sec{:}rass) \wedge \\
& triple(sec{:}horster, rdf{:}knows, bn_1) \wedge \\
& triple(bn_1, rdf{:}type, foaf{:}Person) \wedge \\
& triple(bn_1, foaf{:}name, \text{‘}Peter\text{’}))
\end{aligned}
$$

- Problem: URIs might be used as predicates and as objects

RDF Simple interpretation

- RDF has a lot of reserved words in its vocabulary
- Reasoning requires their formalization
- Solution: map all URIs to sets of abstract resources *IR* and properties *IP*

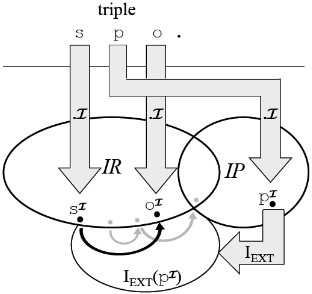


P. Hitzler, S. Rudolph, M. Kroetzsch: Foundations of Semantic Web Technologies. CRC Press, 2009

See http://www.w3.org/TR/2004/REC-rdf-mt-20040210 for a complete specification

RDF Simple model I

- A grounded triple `s p o .` is *true* in an interpretation $\texttt{s p o .}^{\mathcal{I}}$ iff `s`, `p` and `o` are elements of the vocabulary V and $\langle \texttt{s}^{\mathcal{I}}, \texttt{o}^{\mathcal{I}} \rangle \in I_{EXT}(\texttt{p}^{\mathcal{I}})$



P. Hitzler, S. Rudolph, M. Kroetzsch: Foundations of Semantic Web Technologies. CRC Press, 2009

RDF Simple model II

- An interpretation $\mathcal{I}$ is a model of a grounded graph G iff every $t \in G$ is true in $\mathcal{I}$
- I.e. $\mathcal{I}$ *satisfies* G, denoted by $\mathcal{I} \models G$

Graphs with blank nodes (not grounded)

- Blank nodes correspond to variables in first-order logic (Skolemization)
- Define an assignment function $\mathcal{A} : B \rightarrow IR$
- Extended interpretation $[\mathcal{I} + \mathcal{A}]$ is defined just as $\mathcal{I}$, which uses the function $\mathcal{A}$ to ground blank nodes
- Let G be a non-grounded graph. $\mathcal{I}(G) = \mathit{true}$ iff there exists an assignment $\mathcal{A}$ such that $[\mathcal{I} + \mathcal{A}](G) = \mathit{true}$

- A corrected version of the deductive system given in W3C Recomendation[1]
- Existential quantification for a map $G' \to G$

$$\frac{G}{G'} \tag{1}$$

- Subproperty (sp)

$$\frac{(A, sp, B)(B, sp, C)}{(A, sp, C)} \tag{2}$$

$$\frac{(A, sp, B)(X, A, Y)}{(X, B, Y)} \tag{3}$$

- Subclass (sc)

$$\frac{(A, sc, B)(B, sc, C)}{(A, sc, C)} \tag{4}$$

- Typing (type, domain, range)

$$\frac{(A, sc, B)(X, type, A)}{(X, type, B)} \tag{5}$$

$$\frac{(A, domain, B)(C, sp, A)(X, C, Y)}{(X, type, B)} \tag{6}$$

$$\frac{(A, range, B)(C, sp, A)(X, C, Y)}{(Y, type, B)} \tag{7}$$

Proof-theoretic semantics III

- Subproperty reflexivity (xRx, e.g. $\geq$ reflexive, $>$ not)

$$\frac{(X, A, Y)}{(A, sp, A)} \tag{8}$$

$$\frac{}{(p, sp, p)} \text{ for } p \in \{sp, sc, domain, range, type\} \tag{9}$$

$$\frac{(A, p, X)}{(A, sp, A)} \text{ for } p \in \{domain, range\} \tag{10}$$

$$\frac{(A, sp, B)}{(A, sp, A)(B, sp, B)} \tag{11}$$

- Subclass reflexivity

$$\frac{(X, p, A)}{(A, sc, A)} \text{ for } p \in \{domain, range, type\} \tag{12}$$

$$\frac{(A, sc, B)}{(A, sc, A)(B, sc, B)} \tag{13}$$

Proof-theoretic semantics IV

- Inference can be implemented with Datalog (no negation and functional symbols)
- Datalog is a family of knowledge representation languages developed for deductive databases

[1] Gutierrez, C., Hurtado, C. a., Mendelzon, A. O., & Pérez, J. (2011). Foundations of Semantic Web databases. Journal of Computer and System Sciences, 77(3), 520–541.

Parsers/Serializers

- Apache Jena (Java) jena.apache.org
  - Read/write RDF/XML, Turtle, N-Triples
  - rule-based inference engine
- NxParser (Java) nxparser.googlecode.com
  - Supports N-Quads <subject> <predicate> <object> <context> .
- Soprano (C++/Qt4) soprano.sf.net
  - Efficient RDF triple-store
  - Back-end of Nepomuk nepomuk.kde.org – KDE 4 Semantic Desktop " enriches and interconnects data from different desktop applications using semantic metadata stored as RDF" (Wikipedia)

RDF databases – Triple Stores

- Natural and RDB-based
- AllegroGraph www.franz.com natural triple store supporting SPARQL, RDFS and more
- Virtuoso virtuoso.openlinksw.com
- Jena and Sesame – in-memory triple stores jena.apache.org
- Oracle Spatial and Graph RDF store