Biomedical semantics, information retrieval and knowledge discovery - (9)

RDF Schema

Dr. Dietrich Rebholz-Schuhmann
Dr. Leyla Jael Garcia Castro

December 19, 2020

Outline

Objectives and learning outcomes

Objectives

- We will continue our journey on RDF, introducing more elements from RDFS and deepening into the possibilities it offers

Learning Outcomes

- Naming RDF-Schema elements such as classes, sequences or collections.
- Using RDF-Schema to represent knowledge.
- Contrasting the advantages and limitations of RDF-Schema.
- Comparing RDF vs RDF-Schema.

RDF Schema

Introduction to RDF Schema

- RDF Schema is part of RDF (the standard model) and it complements RDF (the vocabulary)
- RDF is useful to distribute data
  BUT we cannot necessarily validate the data.
- RDF is useful to interlink data (via statements)
  BUT we cannot derive new classes.
- RDF is suitable for data storage
  BUT we would like to draw conclusions from the data.

So, RDF Schema is meant to introduce the flavor of object-oriented programming into RDF and to use the RDF data for more purposes, for example to use the graph/tree structure and the semantics in the graph to draw conclusions from the data (some reasoning).

Principles for the triples construction (with classes)

The following features guide the triple formation:

- Classes commonly are used for S and O to form statements about the classes
- Properties are special classes that have been defined to be used for P, i.e. properties are classes that relate the S to the O in a well-defined manner
- For the properties, we can specify (or limit) the "domain" of the property, i.e. restrict the selection of S, and in the same way can specify the "range" of the property, i.e. restrict the selection of O
- For classes and properties, we can
  - assign a type and assign a label
  - define subclasses and subproperties, respectively
- In total, we can define a class (or property) itself and specify the meta-data to describe the class (or property).

Triples construction (with classes): examples

Let any be a name space where we ourselves have registered our own namespace for URIs.

- any:Mammal rdf:type rdf:Class .
  any:Cat rdf:type rdf:Class .
  defines mammal and cat as a classes
- any:Cat rdf:label "cat" .
  gives the class any:Cat the literal label "cat".
- any:Cat rdf:type any:Mammal .
  assigns also the class any:Mammal to the class any:Cat
- any:Cat rdf:subClassOf rdf:Mammal .
  restricts the class any:cat to the class any:Mammal and assigns the same properties

RDF/RDFS usage in vocabularies

- "Fundamental Concepts": URIs for entities/concepts, where the entities/concepts will be used on a global scale (e.g. definition of rdfs:Class, rdfs:Property and other in rdf and rdfs vocabs that will be used to define other vocabs)
- "Concepts for the definition of the schema", i.e. concepts that enable a unified way of representing the data, e.g. rdfs:comment, rdfs:label
- "Concepts for general/public use", i.e. lot of other concepts that help to shape the data, e.g. the concept of a Protein or a Gene in specialized vocabs

Different RDF notations

- RDF encoded in XML:
  regular tree structure as cascaded tags
- Triple notation:
  each element of the triple is represented as an XML element
- N3 notation:
  Namespace and data entry
- Turtle:
  Similar to N3 but less verbose (it uses namespaces)
- JSON-LD:
  Similar to JSON, with additional elements supporting Linked Data

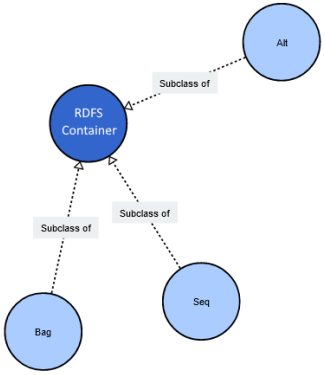
All notations are equivalent, i.e. we can transform forth and back between them: they look different and the shorter notations may require more attention to detail.

Pizza example in RDF/RDFS

```
@prefix pizza: <http://www.example.org/pizza#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
pizza:Teig pizza:hatZutat pizza:ZutatMilch .
pizza:ZutatMilch rdf:value pizza:Milch ; pizza:hatMenge "500 ml" .
pizza:Teig pizza:hatZutat pizza:ZutatSalz .
pizza:ZutatSalz rdf:value pizza:Salz ; pizza:hatMenge "1 Prise"
```

The first two lines form the header and make use of standard references. Each line is a single statement, i.e. *subject predicate object*, or *node-1 edge node-2*.

Collections



RDF offers 3 defined collections and one way to list a generic collection of elements

- `rdf:Seq`, `rdf:Bag` and `rdf:Alt` are all subclasses of `rdfs:Container`
- `rdf:Bag` is intended to be unordered (e.g. list of regular pizza toppings, you can add as many as you want in any order)
- `rdf:Seq` is intended to be ordered and the numerical order is meaningful (e.g. author list)
- `rdf:Alt` is intended to only one of the options should be choose (e.g. list of pizza special toppings, only one can be used at a time)
- `rdf:parseType` similar to an array, with no special intention other than list elements

RDF:Seq for – now the XML expanded form

- In RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF xmlns:pizza="http://www.example.org/pizza#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="pizza:Teig">
    <pizza:Zutat>
      <rdf:Seq>
        <rdf:li rdf:resource="pizza:Milch" />
        <rdf:li rdf:resource="pizza:Mehl" />
        ...
      </rdf:Seq>
</pizza:Zutat>
  </rdf:Description>
</rdf:RDF>
```

- In RDF/XML

```
pizza:Teig pizza:Zutat
  ( pizza:Milch pizza:Mehl pizza:Salz ) .
```

The construct Seq has been introduced to generate lists, i.e. reducing the overheads on repeating syntactical constructs for further items to be added to a data entry.

Generic lists/collections

- In RDF/XML

```
<rdf:Description rdf:about="pizza:Teig">
  <pizza:Zutat rdf:parseType="Collection">
    <rdf:Description rdf:about="pizza:Milch" />
    <rdf:Description rdf:about="pizza:Mehl" />
    <rdf:Description rdf:about="pizza:Salz" />
  </pizza:Zutat>
</rdf:Description>
```

- In Turtle

```
pizza:Teig pizza:Zutat
  ( pizza:Milch pizza:Mehl pizza:Salz ) .
```

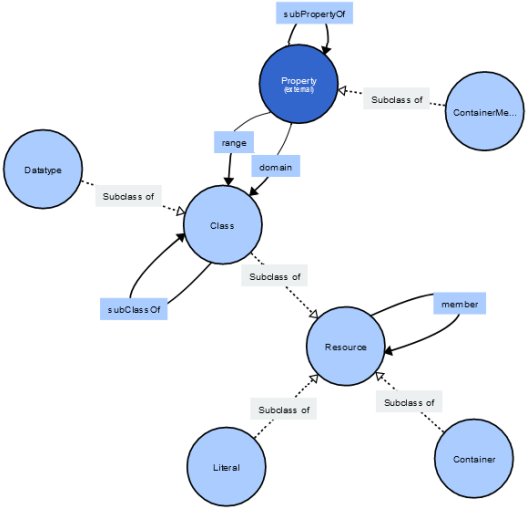
The construct `Collection` denotes a specified set of elements that form the whole set, i.e. the set is specified by only the mentioned elements and nothing more. This can be compared to enumerations in other programming languages (no intended meaning as in rdf:Seq, rdf:Bag or rdf:Alt)

Use of a collection in the Pizza Example

```
@prefix pizza: <http://www.example.org/pizza#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
pizza:Teig pizza:Zutat
( pizza:Milch pizza:Salz pizza:Mehl
  pizza:Backpulver pizza:Butter ) .
```

The ingredients are here kept in a collection (by the use of parentheses).
Collections can be applied to the subject and to the object forming a powerful way to represent larger sets of triples.
Collections can also be nested and involve blank nodes.

RDFS at a glance



Combined elements in RDF and RDFS

The following classes represent:

- `rdfs:Class`: all classes
- `rdfs:Resource`: resources, i.e. definition of an object.
- `rdf:Property`: resources that denote relations.
- `rdfs:Datatype`: datatypes and is a superclass of `rdf:XMLLiteral`.
- `rdf:XMLLiteral`: datatypes in RDF(S) and the values of these datatypes.
- `rdfs:Literal`: literals;
  `rdf:XMLLiteral` is a subclass of `rdfs:Literal`.
- `rdf:List`, `rdf:Bag`, `rdf:Alt`, `rdf:Seq`, `rdfs:Container`: different types of collections
- `rdf:Statement`: all reified triple, i.e. the statements.

RDF-Type: Classes and instances

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix pizza: <http://www.example.org/pizza#> .

pizza:Essen rdf:type rdfs:Class .
pizza:Pizza rdf:type rdf:Essen .
pizza:Salat rdf:type rdf:Essen .
```

RDFS allows to define new classes and then use the classes to generate instances of the classes. In this example, `Pizza` and `Salad` are both instances of the class `Essen`.

RDF type: classes and instances

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix pizza: <http://www.example.org/pizza#> .

pizza:Essen rdf:type rdfs:Class .
pizza:Pizza rdf:subClassOf rdf:Essen .
pizza:Salat rdf:subClassOf rdf:Essen .

pizza:Mehl rdf:subClassOf rdf:Essen .
pizza:Mehl rdf:partOf rdf:Pizza.
```

The definition of classes and properties enables the efficient design and development of knowledge representation. In this example *Mehl* is a `SubClassOf` *Essen* but also `partOf` *Pizza*.

Hierarchies of Properties

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix pizza: <http://www.example.org/pizza#> .

pizza:Pizza rdf:subClassOf rdf:Essen .
pizza:Pizza pizza:hatZutat rdf:Mehl .
pizza:Pizza pizza:hatZutat rdf:Milch .
```

Here is again the standard example, how *Pizza* is composed.

Hierarchies of Properties (2)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix pizza: <http://www.example.org/pizza#> .

pizza:hatFesteZutat rdf:subPropertyOf pizza:hatZutat .
pizza:hatFluessigkeit rdf:subPropertyOf pizza:hatZutat .

pizza:Pizza rdf:subClassOf rdf:Essen .
pizza:Pizza pizza:hatFesteZutat rdf:Mehl .
pizza:Pizza pizza:hatFluessigkeit rdf:Milch .
```

In this example (in comparison to the previous example) we further specify the property `hatZutat` into two alternative Properties.

Dependencies between classes and properties

- There are dependencies between classes and properties.
- A property expects in its argument (or object) a specific type or a number or a label (String).
- From a different perspective: to achieve the correct representation and the validation it is paramount, that the types and properties fit together, i.e. they are corresponding.
- The validation is similar to the XML schema.

Fitting classes and properties

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix pizza: <http://www.example.org/pizza#> .

pizza:hatFesteZutat rdf:subPropertyOf pizza:hatZutat .
pizza:hatFluessigkeit rdf:subPropertyOf pizza:hatZutat .

pizza:Milch rdf:type pizza:Fluessigkeit .
pizza:Mehl rdf:type pizza:FesteZutat .
pizza:hatFesteZutat rdfs:domain pizza:Essen .
pizza:hatFesteZutat rdfs:range pizza:FesteZutat .
pizza:hatFluessigkeit rdfs:domain pizza:Essen .
pizza:hatFluessigkeit rdfs:range pizza:Fluessigkeit .

pizza:Pizza rdf:subClassOf rdf:Essen .
pizza:Pizza pizza:hatFesteZutat rdf:Mehl .
pizza:Pizza pizza:hatFluessigkeit rdf:Milch .
```

The specification of the correct class against the correct property is essential to find the right setup between both of them.

Blank nodes in RDF/RDFS

Not all entities/concepts have to be specified with a URI. The use of blank nodes (_:Z1, _:Z2 in Turle, rdf:nodeID="Z1" in RDF/XML) forms a reference to concepts/entities that are used only locally.

```
@prefix pizza: <http://www.example.org/pizza#> .
pizza:Teig pizza:hatZutat _:Z1, _:Z2 .
_:Z1 pizza:Zutat pizza:Milch; pizza:hatMenge "500 ml" .
_:Z2 pizza:Zutat pizza:Salz; pizza:hatMenge "1 Prise" .
```

A blank node does not have a URI, i.e. it has been introduced without being backed by an online resource. This construct resembles the use of variables in regular programming.
In the example above, two blank notes for a *Zutat* have been introduced and the properties are specified in the following lines of data code.

Summary

Summary

- RDF is meant to collect data in a standardized form, RDF-Schema is the solution to enable expansion of the design without losing quality in the data.
- RDFS introduces design principles in RDF, which relates well, i.e. consistently, to the solutions known from object-oriented programming.
- Sharing of RDFS definitions of classes and properties forms a key step forward to develop a shared environment of knowledge.

Summary

With all the elements we have covered so far —semantics, terminologies, namespaces, URIs, RDF and RDFS, we are ready to have 5-star resources on the Semantic Web as specified by Sir Tim Berners Lee

$\star$ make your stuff available on the Web (whatever format) under an open license (e.g. PDF).

$\star\star$ make it available as structured data (e.g., Excel instead of image scan of a table).

$\star\star\star$ make it available in a non-proprietary open format (e.g., CSV instead of Excel).

$\star\star\star\star$ use URIs to denote things, so that people can point at your stuff (remember identification is important).

$\star\star\star\star\star$ link your data to other data to provide context (i.e. go for structured and semantic data).

So where are we now?

- RDFS introduces new and rich features to the standard formats, which enable better and more efficient design and development of data standards according to the needs of the data provider.
- The Triple representation enables the generation of graph-based data sets by statements. In addition, extensions enable the efficient listing of statements but also the design of new classes and properties, which themselves are derived from classes and properties.
- With regards to biomedical semantics, we now enter the field, where the data is not only produced and kept according to the predefined data standards, but can also be enriched with new data standards that fully comply with the pre-existing data standards.
- With regards to information retrieval and knowledge discovery, the new data can be richer than the preexisting one and the new classes and properties can be exploited to retrieve very specific information and to identify knowledge that diverges from pre-existing knowledge.