Knowledge representation and semantic technologies - RDF

Sveva Pepe

Thursday 17th December, 2020

1 Introduction to 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"

First, the problem was the difficulty in supporting the **interoperability** between systems by the original world wide web technologies. *Interoperability* means that the data produce by one application can be read and correctly interpret by another application, of course there is **no** a priori knowledge of one application to the other.
HTML was was **not** consider very good for the interoperability. The *main lack* was a lack of **semantic** that was behind the WWB. It was **not** very easy for the machine to derive semantic of web pages.
Then, there was an *idea* of creating a kind of **semantic web project**.

The idea was, through *tagging of document*, to make the semantic of the information content of web page explicit. Make it explicit means that a machine **should be able** to automatically classify this information with the *right semantic*, avoid *natural language processing technique*. We are **not** going to use algorithm that interpret the natural language statement to understand the semantic of the content, but we are going to use **the tagging**, artificial structure created by HTML language to understand this semantic.

- 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

Alternative of HTML is XML that solve **only** one of the two problem related to automatic interoperability between applications.
Problem 1: somehow there is a better separation between **tagging** that it is done for the web pages due to the *formatting reason* and **tagging** that it is *annotation with markups* that is done for really *explain the content*. This is **still** big problem of HTML.
Problem 2: the adoption of XML **would not** solved the fundamental missing aspect of HTML. We are **not** really able to automatically extract the information because the same kind of meaning of similar pages can be represent in very different ways even in XML, **not only** in HTML. This lead to problem in *creating intelligent search engine*, intelligent classification semantically aware.

HTML:

```
<H1>Seminari di Ingegneria del Software</H1>
  <UL>
    <LI>Teacher: Giuseppe De Giacomo
    <LI>Room: 7
    <LI>Prerequisites: none
  </UL>
```

XML:

```
<course>
  <title>Seminari di Ingegneria del Software
  </title>
  <teacher>Giuseppe De Giacomo</teacher>
  <room>1AI, 1I</room>
  <prereq>none</prereq>
</course>
```

HTML:
We are humans so we can derive the meaning of **black part**, without looking at the **tag part**. The *tagging* part **does not** help us so much to understand the content. If we are able to extract this information it could be a record in a *database*. But it very difficult to extract this information in a *fully automatic way* in multiple web pages with different structure.
XML:
XML *would provide* tagging that it is more **semantic**, *apparently*. Because the tag name are really *suggesting* us the **meaning** of the text that it is tagged by this things. But **how you can understand this tagging?** There is *ambiguity* in natural language, you could have multiple meanings. And then you **do not** know which type of course is, if it is a Master course or other. Different documents can represent **same information** or **information with same meaning** with different tag and different structure.

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

How can we solve this problem?
The **only** way out to solve this problem is to have a some kind of syncronization a priori agreement on structure that should be follow.
The *idea* of the semantic web was to produce a **standard** that could make this easier but it is very complicated.

viewpoint:

the Web = a web of data

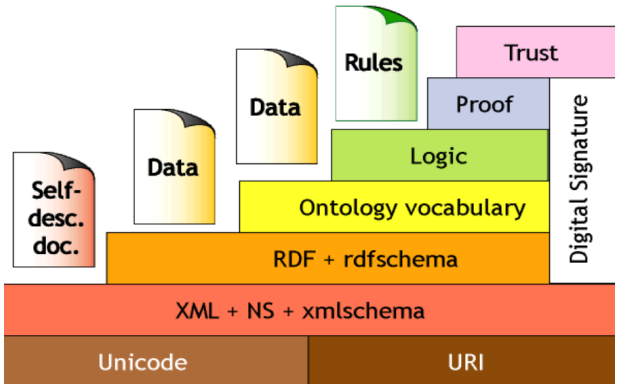
goal:

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

main ideas:

- ontology
- standards
- "layers"

This idea of providing *technologies* that give interoperability between systems, program that are abl to understand information produce by other program. This idea was *reproposed* by different flavour.
Try to provide technology that allow looking at the web, **not as web of pages of document** but as a web of *data*. Allow for extracting data from the web site just like if website is **database**. The ultimate idea is to use web pages as a tables. It is **not** so easy to do.



Semantic web proposal: original presented this architecture, it is called *semantic web cache*. *Every layer* should provide a kind of technology. The original WWB technology is contained in Unicode and URI layer. Original was consider XML as a markup language.
The real **semantic web layers** are: *RDF + schema, OWL, Logic, Proof* and *Trust*. The fact that one layer is on top of the other essentiallly means that the **upper layer** uses the *language or the protocol* of the **lower layer** to make the standard and protocol of this layer working.
The ultimate **goal** was to provide, Trust, a very sofisticated semantic services for user of the web.

In the **original proposal** the idea of the semantic support was to make program understand the content automatically. Instead, on the **web data** approach the idea is that this additional sofisticated semantic services should *enable* for viewing every web resources as a *potential data source*.

XML layers

- 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

In XML the markups are *user-definable*. We can use personalize tagging, while in HTML we have predefined tags.
Resources on the be are *identified*, **URI**. URI identifies web resources.

RDF + RDFS layers

RDF = a simple conceptual data model
W3C standard (1999)

RDF model = set of RDF triples

triple = expression (statement)

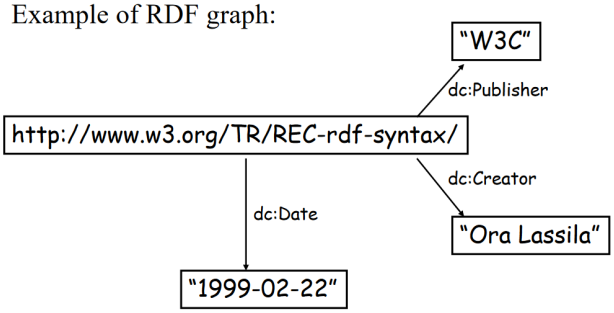
(subject, predicate, object)

- subject = resource
- predicate = property (of the resource)
- object = value (of the property)

=> an RDF model is a **graph**

RDF = *resource description framework*, this is a simple conceptual data model. It is also called **graph language** because what RDF language allow us to do is to create *graphs* that are labeled on the *nodes* and on the *edges*. **Edges** are represented by a *triple*: **(subject, predicate, object)**. These names are, by *default*, **URIs**. In particular, the *subject* is a **resource** (there is an identifier), then *predicate* is **property of this resource**. The *object* is the **value of the resource**.

Example of RDF graph:

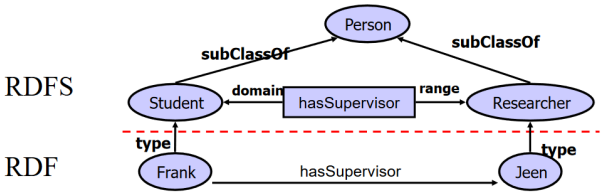


We can represent the *statement* of a triple as an **edge** between two *nodes*. For example, the triple is (http://www.w3.org/TR/REC-rdf-syntax/, dc:Date, "1999-02-22"). Predicate is the **label of the edge**. In the figure, we have a **graph** that represent *three* triples. In the end, RDF is just a language to build a *graph* using **URI**. *Graph language* specializes in **web resources**. This is a *metadata* about a document and we can *annotate* web resources by creating association with values of properties.

Together with RDF, the RDF layer proposed another language **RDFS** = RDF schema. With RDF schema we use the RDF language to create conceptual model, so something that **doesn't** talk about to single URI resource but to a *classes* of resources. And also **binary properties** that create associations between such resources.
In the figure, RDF schema represent 3 classes: Student, Person and Reseacher. *SubClassOf* is a relation between classes, it is a *property* between classes. Then we also have a *binary property*, HasSupervisor, that create association between single resources belonging to the two classes. Then, with RDF we can specifies single instances of the classes of our schema. The pair (Frank, Jean) is an *instance* of HasSupervisor property. So we can create instances of classes/properties, *semantic relations* between classes and also relations between classes and properties.
What is on *top on the red line* is a **TBox**, we see *concept* that are **classes** and *roles* that are **binary relation**. We have also *semantic axioms* like subClassOf.
What we see *below the red line* id the **ABox**. When we talk about *single instances/individuals*, URIs. For these *individuals* we provide **concept assetions**, like Jean that belongs to the class Reseacher.
We can see that this is very similar to an ABox and TBox but the **only difference** is that we are talking about *URIs*, we are **not** talking about abstract symbols. I can create **datasets** using URIs.

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

example:



The problem is that URI, resource, is **not** really dereferenceable, which means that if I want to click on in and see something open on the browser I **don't** see anynthing because the web server **cannot** find any file corresponding to this URI. That's the **only** problem. This is called *non dereferanceable* URI.
The URI that really can be clicked is called *dereferanceable one.*

How to move from annotating resources to writing an arbitrarily knowledge base?
You can provided to use a name space for which you have form of control. *No one* else is going to use this names that you are going to use in you RDF graph.
In the end, it is **not** a matter of existing thing but it is a matter of **potentially existing thing**, potentially existing identified.
RDF schemas, together with RDF, provide a kind of *language* that creating a **knowledge base**, with TBox and ABox.

Ontology layer

ontology = shared conceptualization
⇒ conceptual model
(more expressive than RDF + RDFS)
⇒ expressed in a true knowledge representation language

OWL (Web Ontology Language) = standard language for ontologies

This layer is based on **description logics**. Real place where DL coming to place.
The *ontology layer* is based on the notion of ontology[a], **conceptual model**[b]. It means that it try to provide a very high level view of the domain of interest. A view that is possibly *shared* by all the application, all the potential applications that could use this domain of interest.
Essentially, ontology is a **a description logic knowledge base** because it is expressed in knowledge representation language that it is a DL.
There is a standard of writing ontology in the *semantic web*, it is called **OWL**. A language corresponding to a description logic.

[a] shared conceptualization of the domain of interest
[b] model of high level of abstraction

The **ontology layer** uses the RDF language to *express* KB in DL. RDF is the *concrete language* in which we write the statement in the language. It is a kind of **low-level** language for ontology. Then the ontology itself has *another language* that **encodes the axioms** of the description logic KB. This language is actually translated into RDF, it becomes a set of triples that is a graph that encode the axioms in a simple language. Then the RDF itself is translate into XML/HTML web page. We create a kind of **hierarchical** languages. Everytime we write something we have a translation to the lower level.

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 layer on top of Ontology has been modified several times during years. Because it was introduct a *rule layer* that sometime replace the proof layer or coupling the proof layer.
At some point, rules were proposed to *perform inference over the ontology*. Upper layer were we do some sofisticated processing of information. This layer is called **proof** because we should able to derive *new information* running sofisticated reasoning algorithm on ontology that are description logic KB.
But someone proposed the fact that in this layer **logic rules**, like datalog rules, should be use to perform sofisticated form of inference over ontology.
Why some researchers propose rule approach?
Because *description logics* are **complementary** to rules from the knowledge representation viewpoint. This complementary role of rules w.r.t. DL could be very useful, so creating an architecture in which there is an *ontology layer* and a *rule layer* could allow to benefit from **both approaches** knowledge representation. For instance, the rule layer could capture pieces of knowledge that **cannot** be expressed in OWL (information base on closed world assumption).

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 **trust layer** had more importance, *at least* in principle, w.r.t to proof/rule layer because Trust layer try to address a question that becomes increasingly important in the real world. The problem of **trusting information** that you can collect on the web, problem of fake news.
We **cannot** use some standard techniques to solve this issue, but *at least* some standard should provide some basic statement to try to mitigate the problem of trust. One aspect of problem of trust is **provenance**, we are tryng to trust the provenance of this information. For example, where the information come from.

- ontology = shared conceptualization of a domain of interest
- shared vocabulary => simple (shallow) ontology
- (complex) relationships between "terms" => deep ontology
- AI view:
  - ontology = logical theory (knowledge base)
- DB view:
  - ontology = conceptual model

When we create an *ontology* we create a **knowledge base** in a description logic language because OWL is a description logic in the end. In *databases* this idea of **shared conceptualization** is **not** new because when we build the database according to the classical methodology we should produce a *conceptual model of the domain* that it should try to give us a very abstract *high-level view* of the domain made of entities and the relations. This is something that it is *well-known* in computer science.

Link 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."

Link data are the most important outcoming in the real world of the semantic web. *Link data* is the name under which a metodology for publishing data on the web has been proposed. It is just a metodology **not** a language. Raccomandation of a metodology for publishing data on the web. This is important for **open data**. **Open data** are published by public organization. So there are rules in europe that force many/all public organizations to publish their all data. The format of these data should be *open*, which means that there is **no** "proprietary".
On top of these regulations for data publishing there would be a proposal of link data, which is also classify the quality of the data publishing from the public organizations. The *link data* metodology is base of the **RDF** format.

1.1 RDF layer

This layer is on top of the two layer of standard technology: *lessical layer with use URI and Unicode character* and *HTML/XML language layer*. **RDF** contributes to the Semantic Web in the sense that it provide a language for *annotating web resources*.

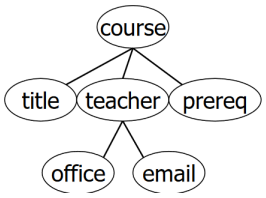
- XML: eXtensible Mark-up Language
- XML documents are written through a user-defined set of tags
- tags are used to express the "semantics" of the various pieces of information

Historically, HTML has problem in separation between *formatting tags* and *structuring tags*. Annotations made using this language were **almost not** understood as semantic structuring. That was the *problem*. There are *formatting tags* and *non formatting tags*, but also the **non formatting tags** are mostly used for formatting purposing. They are **not** used driven by an idea of providing semantic structure to the information represented.
The **idea** was to provide a new generation language, XML, which it should have been able to solve this problem, so separate the formatting aspect from the real high-level structuring. This *high-level structuring* could be more easily used to help deriving the **real content** of the information.

- XML: document = labelled tree
- node = label + attributes/values + contents

```
<course date="...">
  <title>...</title>
  <teacher>
    <office>...</office>
    <email>...</email>
  </teacher>
<prereq>...</prereq>
</course>
```

=



In the end, the XML language (also with HTML) create a page that it is nothing more that a **tree** with labels.
In the case of HTML the *labels* are just the tags of the predefined language, instead in XML the *labels* are defined by user. But *remain* the problem of **ambiguity** of labels.
In the end, XML **does not** solve the problem of provide some *semantic markups* for semantic structure for the content of the web page.

- 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

On the other hand, RDF create a language for *metadata* **annotations** for web pages, web resources. This is an approach that it is **not aiming** to replace HTML. But RDF can also be use in combinatin with HTML to provide *metadata annotation* inside web pages and so **not only** on top of the HTML/XML layer.
RDF is translate into a *lower level language*, translated into XML, so in the end, RDF specification is written in the hypertext language, it is consider a web document. But this is just a matter of representation, actually RDF is **another format** in which only translated in XML.

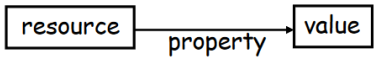
RDF model = set of RDF triples

triple = expression (statement)

(subject, predicate, object)

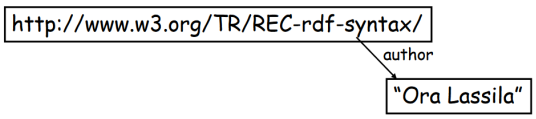
- subject = resource
- predicate = property (of the resource)
- object = value (of the property)

An *RDF resource* is called **RDF model**.
An RDF model is a *set of RDF triples*, and an RDF triple is an expression with 3 components: **subject, property** and **object**.
Subject is a node, *property* is the label of the edge and *object* is another node.



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 "OraLassila"`



⇒ RDF model = **graph**

The *graphical* representation of the statement is show in the picture. If we write a set of statements we create a **graph**, who's node can be *either* **URI** *or* **strings/literals**.

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
- remark: URIs are used as predicates, i.e., graph nodes can be used as edge labels (RDF has meta-modeling abilities)

There are 3 kinds of informations of basical elements of the RDF language: **URI, literal** and **blank node**.
Literal can **only** appear in **object position** of triple.
Blank node can appear **only** in the **subject** or **object position**.
URI can appear in **all positions**.
URI are used as predicates, so URI are used in two roled: subject and predicate of the triple. So, there is a kind of strange use of URI. This give to RDF an aspect called **meta-modelling**, in the sense that the same identifier can be used as the argument of the predicate or the predicate itself. This give to RDF special property w.r.t *standard approaches to knowledge representation* based on classical logic. In classical logic, *also in DL*, we **cannot** have that individual names are also used as concept or role because *concept names, role names* and *individuals* are **mutually disjoint** alphabet.

- **blank node** (bnode) = RDF graph node 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
```

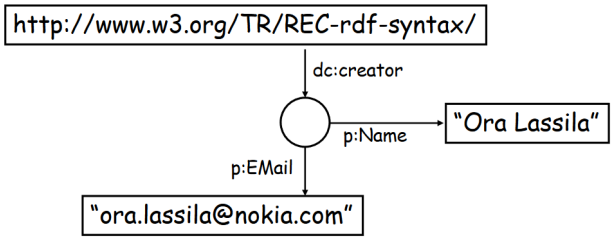
  - `_:p1` is the blank node (b-node)
- bnodes can be used both as subjects and objects

Blank node from the syntactic viewpoint is a *sequence of characters* starting with _. The *blank node* **doesn't** mean a *URI* because blank node is kind of anonymous, empty node.
If we use the graphical representation, we would represent blank node by a node **without** a label. With blank node we are representing some missing informations.
Blank node looks similar to *null values* in SQL tables. But here you can use different blank nodes for different things.
Furthermore, blank nodes canbe use in *subject* or in *object* position.

- values of properties need not be simple strings
- the value of a property can also be a graph node (corresponding to a resource)
- using blank nodes, arbitrarily complex tree and graph structures are possible

Actually, **blank nodes** allow us to creating *complex* structure in RDF graph because without "null value" we have just subject − > predicate − > object.

example:

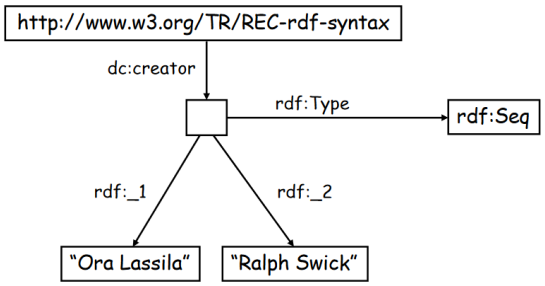


This URI, http://www.w3.org/TR/REC-rde-syntax/, has a *creator*, but we **do not know** exactly who is, so it is **blank node**. This blank node has a *name* and and *Email*. We create a kind of **container**.

- Containers are collections
  - they allow grouping of resources (or literal values)
- It is possible to make statements about the container (as a whole) or about its members individually
- Different types of containers exist
  - **bag** - unordered collection
  - **seq** - ordered collection (= "sequence")
  - **alt** - represents alternatives
- It is also possible to create collections based on URI patterns
- Duplicate values are permitted (no mechanism to enforce unique value constraints)

More significant example is the usage of **blank nodes** to encode *containers*, that are real data structures: **bag, seq** and **alt**.

example:



We have a *creator* of URI that it is a blank node, it is **square** because it has special *type*. We assign the type *sequence* to this blank node and then we use some predefined properties ($rdf : \_1$, $rdf : \_2$), the sequence of the names of the authors.
This blank node is kind of **auxiliary node** to create a complex value, so the value is **not** one creator but a sequence of creators.

- One can make RDF statements about other RDF statements
  - example: "Ralph believes that the web contains one billion documents"
- Higher-order statements
  - allow us to express beliefs (and other modalities)
  - are important for trust models, digital signatures,etc.
  - also: metadata about metadata
  - are represented by modeling RDF in RDF itself

⇒ reification

This *idea* that when I use the blank node I can make complex structure can be used also for **encode** *high-order statement*.
Suppose that we have **nasted statement**, like the example in the picture, where "Ralph" is the *subject*, "believes" is the *predicate* and "the web contains one bilione documents" is the *object*. The *object* is **another statement**. **Can I create a representation in RDF who's object is another statement?**
Yes, we have to technique called **reification**.
Reification means that I create a blank node that represent the whole statement.

- RDF provides a built-in predicate vocabulary for reification:
  - **rdf:subject**
  - **rdf:predicate**
  - **rdf:object**
  - **rdf:statement**
- Using this vocabulary (i.e., these URIs from the rdf: namespace) it is possible to represents a triple through a blank node

We need to use some **special predicates** to do this encoding. RDF has also its *own* vocabulary. Similar to the example that we were doing with the *sequence* but in this case the type is **not** "rdf:Seq" but it is "rdf:statement".
A *rdf:statetement* is an **rdf triple** so we are saying that this blank node is of type statement and it has 3 properties: subject, predicate, object. And we connect the *real* subject of the triple with rdf:subject and the same for other two.

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

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

- The blank node _:x is the **reification** of the statement (it is an anonymous URI that represents the whole triple)
- Now, "The statement "The technical report on RDF was written by Ora Lassila" was written by the Library of Congress" can be represented using the bnode _:x, by adding to the above four triples the following triple:

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

We are going to **reified** the statement that you see on the picture. We are going to use the *blank node* to represent the all statement, "_".
We able to *nest* statement inside other statements.

RDF model = edge-labeled graph = set of triples

- graphical notation (graph)
- (informal) triple-based notation

  e.g., **`(subject, predicate, object)`**
- formal syntaxes:
  - N3 notation
  - Turtle notation
  - concrete (serialized) syntax: RDF/XML syntax

A *triple* corresponds an **edge** between two nodes: *subject* and *object*. This **directed edge** from subject to object is **labelled** by the *predicate*, which is an identifier.
Now we can say that **RDF model** is consider as a *set of triples*. This set of triples is written according different possible *syntax*. There are different standardize syntaxes: **N3 notation, Turtle notation** and **serialize syntax** (translation of RDF in XML).

- N3 notation:

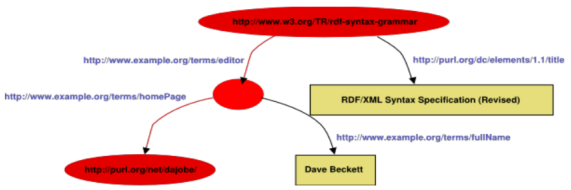
```
subject predicate object .
```

- Turtle notation: example:

```
@prefix
    rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
:mary rdf:type <http://www.ex.org/Gardener>.
:mary :worksFor :ElJardinHaus.
:mary :name "Dalileh Jones"@en.
_:john :worksFor :ElJardinHas.
_:john :idNumber "54321"^^xsd:integer.
```

- concrete (serialized) syntax: RDF/XML syntax

We *focus* on the **Turtle notation** because it is more human oriented syntax, for RDF. We can see from the picture that there are *triples* in this Turtle notation that are **sequence of tree elements**. The triples terminates by the *full stop*.
The important aspect is the *definition of prefix* because URI can appear in prefix notation. For instance, in the picture we can see prefix **rdf**, it is an *abbreviation* of URI. "54321" is a literal of type integer. xsd, prefix of xml schema datatyoe, is an abreviated URI.




```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
```

On top of the picture there is an **model of RDF graph** written in graphical notation. We can see that there are *four* triples, four edges and there is also one *blank node*. Then we can see the possible representation in XML syntax. You can see that there are predefined tag names and also the tag names themselves are **URI**. For instance, *rdf:Description* in which we define the *subject* of our fragment that we are representing, we are going to talk abou things/triples related the URI consider as root node of the graph.
In this XML fragment there is some nested things, for instance *ex:editor* which is the **predicate** of the *edge* in the left side of the root. And for the editor we have description *ex:homePage*, which is **subsequence** edge label ans so on and so forth.
We are going to use XML tags to represent the *structure* of the graph itself.

- RDFa = RDF in attributes
- Provides a simple way for embedding RDF semantic annotations into XHTML documents
- More precisely: RDFa specifies a set of attributes that can be used in XHTML elements
- Such attributes allow for expressing RDF triples inside the document
- W3C standard (2008)
- No compatibility problem (XHTML browsers correctly display documents with RDFa attributes)

There is a possibility of writing RDF inside HTML/XML document.
We can see another *standard*. The **idea** is very simple. This standard define a *set of attributes* that can be use in HTML elements. So, inside hypertext document we can use **special attributes** to specify triples talking about the document itself or elements of the document itself. It is a kind of possibility of writing *RDF triples* inside HTML document.

consider the following XHTML document:

```
...
<h2>The trouble with Bob</h2>
<h3>Alice</h3>
...
```

and suppose that the above h2 element specifies the title and the h3 element specifies the author of the document.

We can add RDFa attributes to the above code (and use the Dublin Core (DC) vocabulary) to specify this:

```
<div about="http://example.com/twb"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
  <h2 property="dc:title">The trouble with Bob</h2>
  <h3 property="dc:creator">Alice</h3> ...
</div>
```

XHTML is HTML but it must be written in a syntax that it is strictly compliant with XML. The different is that HTML **doesn't** enforce exactly the same syntattic rule of XML. We can see in the picture the example with "Bob" and "Alice" in XHTML and we know that tagging, as usual, **doesn't tell us** anything about the *meaning* of the sentences inside the tags.
We can use RDF to *annotate* these elements of HTML. *On the bottom* of the picture we have an *expansion* of HTML in which the **bold** strings is the usage of *RDF attributes* to create definition of triples talking about two HTML elements.
We create "div" container, we need to add one HTML element that becomes the container of the part of the hypertext that we want to annotate. It is used to declare some **attribute**, in this case "about", *subject* of the triples that you are going to talk about. Then we can say that we add to the element "h2" the *attribute property* and the **value of the property** is *title*, which means that we are creting a triple who's *subject* is **about value**, the *predicate* is the **property value** and the *object* the **string inside tag h2**. We use the *RDF language* to say that this string is the *title* of the resource, written in "about". The same for h3. This is a interesting things because we are doing this inside the *document itself*. We are **creating the document** and we are creating also the **metadata of the document**. In thi way we understand the annotation, semantic.


```
<div about="http://example.com/twb"
 xmlns:dc="http://purl.org/dc/elements/1.1/">
  <h2 property="dc:title">The trouble with Bob</h2>
  <h3 property="dc:creator">Alice</h3> ...
</div>
```


- The above use of the `about` and `property` attribute corresponds to defining the following RDF triples:

```
http://example.com/twb dc:title "The trouble with Bob".
http://example.com/twb dc:author "Alice".
```

- The `about` attribute specifies the URI that is the subject
- The `property` attribute specifies the predicate
- The objects are the literals corresponding to the content of the element

There is another attribute "xmlns", which is the **declaration** of the prefix *dc*. This is **not** RDF standard but xmlns standard.

we use RDFa and the Friend-of-a Friend (FOAF) vocabulary:

```
<div typeof="foaf:Person"
     xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <p property="foaf:name">Alice Birpemswick</p>
  <p>
    Email:
    <a rel="foaf:mbox"
       href="mailto:alice@example.com">alice@example.com</a>
  </p>
  <p>
    Phone:
    <a rel="foaf:phone"
       href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
</div>
```

The effect of **bold annotation** here is to create equivalent triples using *blank node* as a subject. This is done using **rel property**.

The above use of the `about` and `property` attribute corresponds to defining the following RDF triples:

```
_:x rdf:type foaf:person
_:x foaf:name "Alice Birpemswick"
_:x foaf:mbox mailto:alice@example.com
_:x foaf:phone tel:+1-617-555-7332
```

Here we can see triples having the *same* blank node in the subject position.

Notice: the subject of the above triples is a blank node because we did not assign the `about` attribute to the div element

RDF schema

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"

The **RDF schema** is the possibility of creating a sort of TBox or *intensional information* in RDF. **Not** by introduce a *new syntax* but just by **introduce a new vocabulary**. Vocabulary means a *set of URIs*, special interpreted URI. An *RDF schema* is **nothing else** that another RDF vocabulary and it is a *real standard* (recommended by w3c concortium).
Vocabulary contains *20/26* of special URI, and this URIs allows for defining **shema RDF**, in particular we can define *Classes*, new concept; then **relation of sub class between classes**, *subClassOf*; then **instance of classes**, *type* and we can define also **properties** and **relation between properties** and finally, the *domain* and *range* of properties.

- vocabulary for defining **classes** and **properties**
- vocabulary for 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)

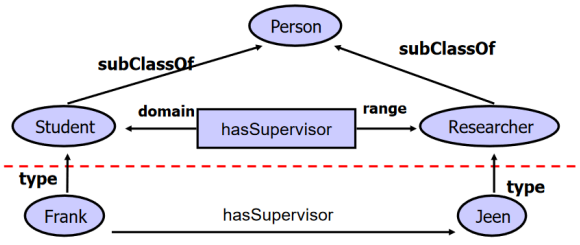
This vocabulary has a *predefined meaning*, which means that there is a *standard* URI. For instance, **rdfs:Class** is the URI that allows us to say that *resource* represent a *Class*. **What does it mean a Class?**
It means a *set of URIs having this characteristic*. It is just like Class or concept in DL. **rdfs** stands for the schema standard, instead **rdf** stands for the RDF vocabulary standard.

vocabulary for 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)

We have similar vocabulary for *properties*. We can say that a **resource identifier** is a *property*. We can see that property is *already* decleared in RDF vocabulary because we have prefix **rdf**.
Property is a **role**, it is a *binary relation*, just like a role in DL. Property associate *subject* and *object*, just like in a triple the URI appearing in predicate associate subject and object.

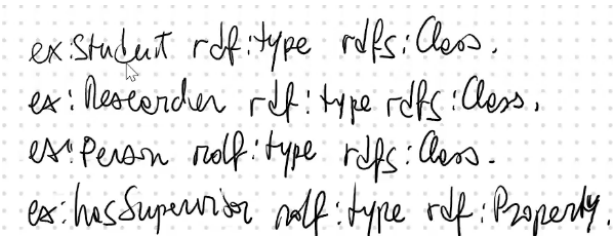
The triple is just a *declaration* of **instance of the property**. Then we can write *typing* assertions for the first component of the property, **domain**, and for the second component of the property, **range**.
Domain is *assigning a class* to the domain of the property, where the *domain of the property* is the **first element** in a property instantiation, it is the *subject*.
Range is *assigning a class* to the object of the property where the *object of the property* is the **second element** in a property instantiation, it is the *object*.
We are **typing** the domain of the range of the property.

example:

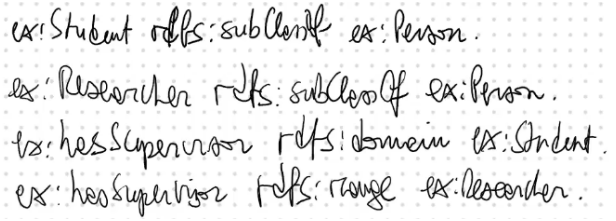


The example is written in the *graph*, and the graph **not showing** all the triples that we need to create the schema above the *red dashed line*.
This is **not fully** specifing the schema.

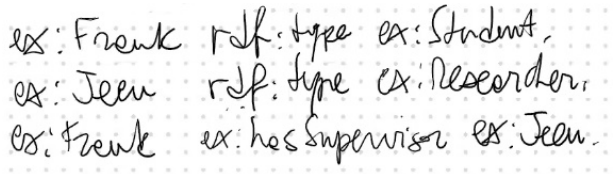
Exact triples of the previous example



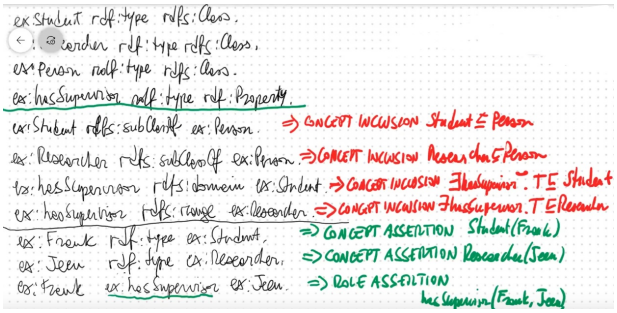
There are 3 classes: **Person, Student** and **Researcher** but we *have to define them*, it is **not** explicit written in the graph. The same also for the peroperty: **hasSupervisor**.
In the picture you can see the definition of the triples for these 3 classes and property.
With these 4 triples I essentially said that first three URIs are *classes* and the last one is a *property.*



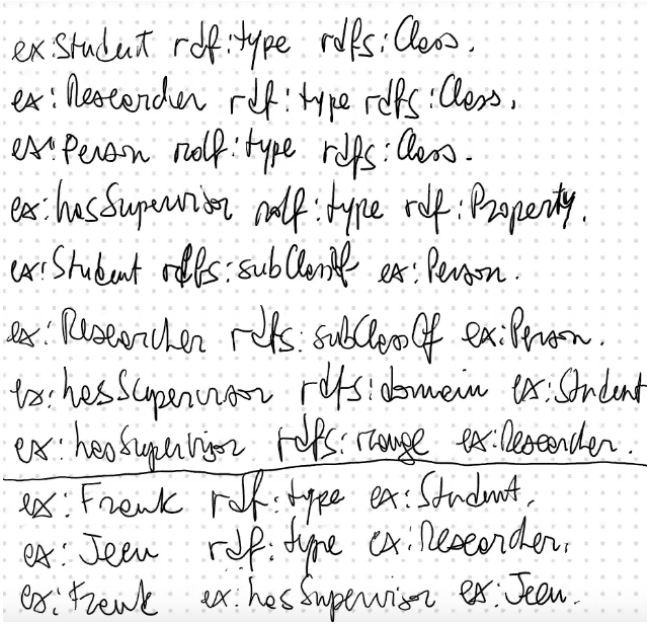
Now we are going to write the real **edges** of this schema, 4 edges: **2 subClassOf, domain** and **range**.
In the picture we have written the edges.



We have also defined some **instances**: Frank is a Student and Jean is a Researcher. These are just **ABox assertions**. The last line is a **property instance**.



Below the *horizzontal line* we are in a knid of **ABox** and above the line we see a **TBox assertions** because there are just class and properties definitions, and semantic relations between Classes and properties. Instead the *individuals/instances of Classes and properties* are below the line.



Parallel between RDFS, RDF schema, and DL. We write also the **syntax** in DL. These are the correspondences between description logic and RDF schema.
$hasSupervisor^-$ is the **inverse role** to represent domain restriction, this is **not** in ALC, we need inverse role operator to express the domain restriction.
We see something interesting, because an *RDF graph* with schema, so using RDF schema vocabulary, is very **similar** to declaration logic knowledge base.
We define *simple logic knowledge base*, it is **simple** because we **don't have concept constructor**, we **only** have concept names, we **do not** use *disjunction, conjucntion* and so on. There is **only** domain and range description.
We see that there is a **correspondence** between RDF schema and DL. Both of them talk about classes that are called *concept* in DL and properties that are called *roles* in DL.

- 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

⇒formal definition of entailment and query answering over RDF(S) graphs

Now, something about the **semantics**.
What is the semantic? How do we interpret real meaning of writing some triples?
This problem becomes important when we include also *RDFS* because we are defining a *small KB*.
There is a *standard semantic* for RDF schema.
How was it defined?
It is complicated by essentially 3 aspect:

- **blank nodes**
- **meta-modeling**
- **formal semantic for RDFS vocabulary**

The presence of **blank nodes** because the interpretation of blank node is **not** immediate because the introduction of this *special value* is **not** easy to define when you consider reasoning problem that you pose on you dataset.

Is there a different if I return blank node or if I return a real node?
Yes, there should be a different. It is **not** easy question. Expecially, address with the **formal semantics**. Now the *formal semantic* means mathematically precise semantics, *unambiguous semantic*.
In RDFS there was a *notion of model* of a graph, so interpretation structure that satisfies **all the triples** in a graph. There is kind of notion of interpretation of the graph, for RDF model and based on this, there was formal semantic, but it very complicated w.r.t DL.

Why?
Because of the second issue: **meta-modeling**. The fact, the you can use the **same symbol** in *subject, predicate* and *object position*. **Subject** and **object** position are *instance position* in a triple, while the **predicate** position is a position of predicate.
In *classical logic*, like as DL, predicate **cannot** appear in **instance position** and instance position **cannot** appear in predicate position. There is *strict typing* of the symbols. This is **not true** in RDF, every symbol can play both **role** of the *instance* and of the *predicate*. **No** strict separation between predicate symbols and instance symbols. However, given this **meta-modeling**, the fact that same symbol can play as a *predicate* and as a *instance* introduces a diffculty formal that give proper semantic to this polymorphic usage of the same URI.
The *third* problem is that we have an RDFS vocabulary so we need to give semantic to Classes, subClassOf etc. This is **not** a very difficult problem. The **real problem** is given by *meta-modeling* difference between standard DL and RDF.

- formal semantics for RDFS vocabulary
- RDFS statements = **constraints** over the RDF graph
- entailment in RDF + RDFS = reasoning (query answering) over an **incomplete database with constraints**

There is some correspondence with DL semantics. Some other aspects are *simpler* than DL and other, like *meta-modeling* and *blank node*, are present in RDFS and **not present** in DL, so for some aspect it is more difficult to give a semantic to RDF schema than DL.

example (meta-classes):

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

What does it mean meta-modeling?
In the picture, *MotorVehicle* is on **type Class**. And also *myClasses* is on **type Class**. Then I can say, *without any syntactic error*, that *MotorVehicle* is an **instance** of *myClasses*. The point here is that there is a Class, myClasses, which contains an *instance*, MotorVehicle, but *MotorVehicle* is **itself a Class**. So, there is a Class, who's element/instance is a Class. This is allow but we **must give** a *semantic* to it. **What does it mean?**
It means that you have Classes made of Classes, and this is why it is called *meta-modeling* because now *myClasses* is a **meta-Class** because it is a Class who's instances are Classes themselves.

problems with meta-data:

```
:a rdf:type :C .
:C rdf:type :R .
:R rdf:type :a .
```

or

```
:C rdf:type :C .
```

are correct (formally meaningful) RDF statements

$\Rightarrow$ but no intuitive semantics

Given this possibility of *meta-modelling* you can also create **paradoxical** examples. Since the *same* symbol can appear in both in instance position and in Class position, you can think to use it in the same statement, like the last example in the picture. This is *correct* in RDF.
How it is possible to interpret something that could be consider as a paradoxical statement?
The interpretation of the symbol C as an individual, single URI, is **separated** from the interpretation of the same symbol C as a Class/Property. We are not say "The Class C is included in the Class C" but we say that "The symbol C, interprets as an individual, belongs to the symbol C interprets as a Class". The interpretation of these two things is **not the same**. In the end, we **cannot** build a *contradictory* statement, which is good because we **cannot** generate *inconsistent graph.*

1.1.1 SPARQL

SPARQL is a *query* language for RDF models. It is standardize by w3c.

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. }
```

A simple form of *query* in SPARQL.
A **SPARQL query** is in form SELECT, WHERE statement.
In this example, there is a **PREFIX** definition, just like prefix that we have seen in Turtle notation. Here there is notation of "abc" as URI framgment. This is a definition of writing URI in abbreviation form. In **SELECT** there are sequence of identifier symbols started with ? or also with $. In **WHERE** condition we contain *triples.*
In SPARQL we are **not only** write stardard RDF triples, we can also use another lexical element which is called *variable.* For instance, ?capital is a variable symbols.
WHERE condition is a *sequence of triples* and all section is called **basic graph pattern**, (BGP). It is **not** really an RDF graph because there is something that you **cannot** write in RDF, variables.

When there are variables the RDF graph is called *graph pattern.* It means that there are *variables positions* in the triple and this variable positions match with different value, standard RDF lexical elements (URI, blank node, literals).
In **SELECT** we *select* sequence of variables, in the picture on top we have just 2 variables, this is called **target list**. Essentially, they are the valeus returned when the query is *evaluated.* So, we want to see the values of this variables such that the *graph pattern* is match in the real RDF model.
In **WHERE** there are *conditions* that *must be true* in the graph to return the result.

Are there some triple in graph, that we are going to evaluate this query, such that there are a triples that matched with triples patterns?
This is the question that we are going to answer, so we can return variables selected in SELECT section if there are triples that satisfies WHERE conditions. We are going to extract variables by this query.
For some aspect this SPARQL-query is very *similar* to query in SQL.
If context has **only** one graph for the execution we **don't** need to specofy the graph, otherwise if the system is handling at the same time multiple graphs, we *must specify* in the FROM section which graph we refer to and the graph name is another URI.

The **idea** is that the algorithm that must evaluate this query on the RDF model, RDF graph currently in the scope of this system, should find **all the hits matching the pattern** for the RDF triples. The *graph pattern* is evaluated on the graph, on the real dataset, on the real RDF triples and every set of triple that match the graph-pattern produced an answer and so the values of the variables that are specified in SELECT are returned.

What is returned?
Usually, *a list of variables*. In the example, a list of pairs. We have a kind of **table**. The result of the query is a table, **not an RDF graph**.

- **basic graph pattern** (BGP) query = RDF graph with possibly variables as labels
- SPARQL defines on top of a BGP additional operators (AND, FILTER, UNION, OPTIONAL)

Basic graph pattern is a RDF graph with *variables* that appears in the triples. The variables can appear in every position (subject, predicate, object).
We can do more in WHERE condition, navigation pattern adding some *operators* that allow us to compose in a logical way different BGP. There are **FILTER, UNION, OPTIONAL** and **AND** condition that are constructors that allow building more complex expressions.

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>

In the picture, we have the 5 triples in the RDF graph that are using *blank node*.
"mailto" is a *protocol* of URI and **not** a prefix. Both in *SPARQL-query* and in *RDF-graph* we have the *prefix*.
We want to return *(name,mail)* of the people given thi RDF graph.
How do we extract name and mail of the people?
First, we will see if there is a match between triples patters (in WHERE) and triples in the RDF graph. If it is true we return (name, mail). The result is the table in red.
The fifth triple, _:c, match the second triple in the pattern but there is **no** match for the first triple in the pattern, so we **do not** get any answer from this.

FILTER condition

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>

We just add in the query, wit respect to previous example, **FILTER** condition. FILTER imposes further condition on the matches of graph pattern that we find in the graph.
regex(?name,"^J") means that the value returned by the variable name starts with the letter uppercase J.
The FILTER condition says that for every match found for the basic graph pattern in the RDF graph, we return the result **only if** the match satisfies the condition in the FILTER.

OPTIONAL condition

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>

The query is a *little* similar to the previous case. In this example, means that one or multiple triple is inside the OPTIONAL condition the matches that we are looking for in the graph **could optionally** contain a match of the this part of the graph pattern that it is the name.
When we build query with **OPTIONAL condition** we are dividing the graph patter into a *mandatory* part, the one that it is **not** in the scope of the optional, and *optional* part, in which it is **not necessary** to match. Even you *just match* with mandatory part you have a result. Here we have third match in wich we **do not** have the name, so we have only the mandatory part.

The examples described above refer to the version 1.0 of SPARQL.

- SPARQL 1.1 (released in 2013) extends the first varsion 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

The version 1.1 of SPARQ presents some *extensions* of SPARQL in different directions: *presence of aggregate functions(COUNT, MAX, MIN, GROUP-BY)*, *entailment regimes*. **Entailment regimes**, semantics of SPARQL query over RDF model.
One of the main characteristic introduced by this new version of SPARQL is the possibility of expressing **property paths** in queries. A *property path* is a **regular expression** using properties, predicates, URIs. Using URIs meant that we are taling about the *edge label* of RDF graph, and since RDF edge label can **only** be URI it means that we write regular expression using URI.
Why is this important for expressivness of the languange?
Because *property paths* allow for expressing path of *arbitrary length* along the RDF graph, thus providing a fundamental **graph-oriented feature to SPARQL**. It means that in the first version, 1.0, SPARQL was **not able** to encode typical function that are evaluated on graph. For example, the *reacheability function* - give me the list of all the nodes that can be reach from this node.

We can consider *reachability* of an airport to a given airport is a typical *reachability problem* on the graph.
"Starting from fiumicino airport, give me the list of all airports I can reach"
This is typical *recursive query* and in SPARQL 1.0 we **didn't have** the possibility of writing this query. We **do not know** the maximum number of stops that lead you to new airport. We **cannot** write *finite graph pattern* that find all the airports that are reached from one airport. This problem is **strange** because in graph it is **not** a strange query, it is basic feature query in the graph because the fact the the node is/is **not** isolated from another node is very important. With SPARQL 1.1 we solve this problem using *property paths*.

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.

Table that contains syntax of such *regural expressions*. *Inverse path*: from object to subject. *Sequence path*: expression 1 follows expression 2 and so on and so forth.

Exercise

Express through a SPARQL query the following request:

- Return the names of all the ancestors and the descendants of John, assuming that the RDF graph uses the properties :hasFather and :hasMother to express kinship relations.

I want to knwo all the name of ancestors and the descendant of John. We have to write query using *hasFather*, *hasMother*. This is a query that need *recursion* because we **do not know** how many generations are represented in real graph. We **do not** know the graph when we write queries but we know the predicates to express relations.
We solve this exercise with **property paths**.

Solution

```
PREFIX ex: <http://example.org/example/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?z
WHERE { ex:John (ex:hasFather | ex:hasMother)+ ?x .
        ?x foaf:name ?z . }
UNION
      { ?x (ex:hasFather | ex:hasMother)+ ex:John .
        ?x foaf:name ?z . }
```

We are going to see matches that *span* over one or more arbitrary number of concatenation of either *hasFather* or *hasMother* property/predicate. There is a path made of *hasFather* or *hasMother* predicate from John to x.
Who is x?
x is either the *father* or *mother* or *grandfather* or *grandmather* ans so on. x is an **ancestor** of John. But we **do not** return x but z, the name of the ancestor because x should be URI.
For **descendant** of John we need to make a kind of dual query, so we use UNION constructor.
The **UNION constructor** is allow us to take the union of the result of graph patterns in the same query, logical or.

- RDF and SPARQL are W3C standards
- Widespread use for metadata representation, e.g.
  - Apple (MCF)
  - Adobe (XMP)
  - Mozilla/Firefox
- Oracle supports RDF, and provides an extension of SQL to query RDF data
- HP has a big lab (in Bristol) developing specialized data stores for RDF (Jena)

RDF was **quite popular** for *metadata representation* but **not** really in the semantic web sense. They use RDF as a formal language to write metadata in documentation, something disconnected from web application.
The **most** support for RDF was provided by Oracle, it took semantic web idea seriously.

- current main application of RDF: **linked data**
- linked data = using the Web to create **typed links** between data from different sources
- i.e.: create a **Web of data**
- DBpedia, Geonames, US Census, EuroStat, MusicBrainz, BBC Programmes, Flickr, DBLP, PubMed, UniProt, FOAF, SIOC, OpenCyc, UMBEL, Virtual Observatories, freebase,…
- each source: up to several million triples
- overall: over 30 billions triples (2012)

There is one **big are** where RDF has been really popular: *link data*. **Link data** means creting link (hyperlink) databases. Link data is real standard, **methodology**, **not** a language to use RDF and publishing data accordingly to some principles.
This *idea* is the closest thing that happen in the real world w.r.t the *goal of the semantic web/ goal of creating a web of data*. this idea of **web of data** was realized in a *subdomain* of context, that is the context of *open data*.
The **idea** of creating *open data repository* using RDF and using links, data also contain links - connection between the data themselves, give use the possibility to create relations between data or links between data and web resources, real HTML documents.

Linked Data: set of best practices for publishing and connecting structured data on the Web using URIs and RDF

Basic idea: apply the general architecture of the World Wide Web to the task of sharing structured data on global scale

•The Web is built on the idea of setting hyperlinks between documents that may reside on different Web servers.
•It is built on a small set of simple standards:
- Global identification mechanism: URIs, IRIs
- Access mechanism: HTTP
- Content format: HTML

Linked Data builds directly on Web architecture and applies this architecture to the task of sharing data on global scale

The methodology is a document that give a *set of best practices* for publishing and connecting structured data on the Web. The core is to use *RDF*, the other extremely important aspect is to use URIs, *dereferensable URIs*.

1. Use **URIs** as names for things.
2. Use HTTP URIs, so that people can look up those names (**dereferenceable URIs**).
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include **links** to other URIs, so that they can discover more things.

Dereferenceability = URIs are not just used for identifying entities: since they can be used in the same way as URLs, they also enable locating and retrieving resources describing and representing these entities on the Web.

Use **URIs** as identifier to the things that you want to talk about: individual, people, institutions and so on. If it is possible, use *HTTP URIs*, which means that there is an HTML page behind this or a framgment of HTML page.
Why?
Because this identifier plays just the role of any other identifier in a dataset, it is a symbol that represent something but it is *also a symbol that you can click* and look at the document that it is able to explain to human user the meaning of this.

Consider that we want to write Rome in database, but we don't know if we want to consider Roma, Rome or other form of this name. This is a huge problem, so we use different strings to represent the same thing in a dataset.
When you write a JOIN you must remember if that there is a chance that you are going to miss an answer because for example, in a table the city of Rome is written "Rome" and in the other table is written "Roma". The JOIN is not match and you are going to miss answer.
This problem, called **record link**, is *pervasive in the information* technology.
Link data have **not** solve the problem but *at least* there is a technical means. The *idea* is:

- Use URIs, but it **doesn't solve** the problem it is just a syntax for an identifier
- Use HTTP URIs, for the example of Rome, use the wikipedia italian page of Rome. This URI is stable. Use this identifier because it is popular, many people use it.

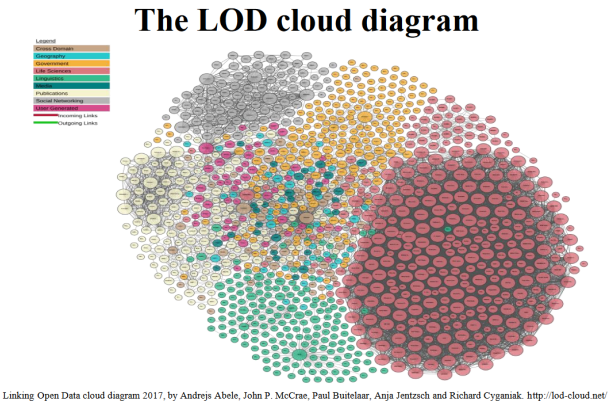
The idea is to exploit the power of the WWB in the data representation.
Then we should create *interconnected datasets*.
For instance, using geographical database URIs for expressing geospatial data. We can navigate through datasets with links.

Linked data may be **open** (publicly accessible and reusable) or **closed**

Linking Open Data (LOD): project which aims at creating an open Linked Data network

http://lod-cloud.net

Link data methodology can be use interchangeably for **both** *close* and *open* data, so that you really want to make publicly accessible and reusable or not.
The project called **LOD** that was trying to create a kind of official network of link open data project, this register/network is **not** completely updated to reality in the sense that there are a lot of open data project that are **not** register officially in the linking of LOD page.



Every node in this picture, LOD cloud, represent an RDF dataset published by an organization. Informally, this is the picture of **web of data** because every node represents RDF dataset. RDF data models that are more or less connected to each other.
Is there some software tool that it is able to really navigate in this huge amount of data?
No we **don't have tool** that can reasonably do some navigation in this huge graph. This is technical problem.

What it is possible at the moment is to create *local* SPARQL endpoint, that it is SPARQL web interface in which you can write SPARQL queries on one of this websites, and SPARQL query will be evaluated in local dataset. Even if there are links, there are **not** going to be followed in the execution of the query because it would make the query *extremely slow* in its execution.
There is still in some SPARQL endpoint the possibility of abilitating the **distributed execution**, so there are links go to another RDF graph go to this other website and pose subqueries to this website and get the answer and try to merge them with my local result in this dataset etc, but it is extremely **inefficient**. It is **not** used in reality.

What's the way out?
The way out is to **create copy that collect data**, local copy, so in one website you create big graph (off-line) and the you execute localling having you copy of the data. This is done, instead of running this as a separate processing on the different websites, we run it as a single dataset locally on one SPARQL endpoint and we get the answer.

- Crucial aspect of Linked Data (and of RDF usage in general): which URIs represent predicates (links)?
- Recommended practice in LOD: if possible, **use existing RDF vocabularies** (and preferably the most popular ones)
- In this way, a de-facto standard is created: all LOD sites use the same URI to represent the same property, and the semantics of such properties is shared (i.e., known by every application)
- This makes it possible for all applications to really understand the semantics of links

The *idea* of Semantic Web and RDF layer was try to provide a kind of semantic of the web resources in a way such that application are able to automatically understand the semantics of the web resources itself. The idea was to have a kind of *share semantic for the data*, for annotation data.
In which sense RDF is able to solve semnatic problem?
There is **one advantage**: the fact that *identifier* in RDD can be URI, so there is a standard identifier behind this approach, different from databases. In SQL you **do not** have any *reference standard*. In SQL you **don't have** to follow a syntax to write a representation of the thing that you want to talk about.
The standard URI says that you are going to use *unic identifier*.

If you have control of portion of the URI space you are going to use your own dataset. This solve *one* of the problem. The other problem is the fact **creating links to other dataset**. But the problem of creating links to other dataset is the *real semantic problem*. The fact that, an application that it is completely independent of the author of the dataset itself is able to understand the meaning of the dataset itself.

How does RDF solve this problem?
RDF **doesn't solve** this problem at all. The **only** real thing that it is done in this direction is the fact that, according to the methodology of RDF data production, you should use popular **vocabulary** for the properties that you want to talk about in many domains in RDF (faof, dc etc.).

- **Friend-of-a-Friend (FOAF)**, vocabulary for describing people
- **Dublin Core (DC)** defines general metadata attributes
- **Semantically-Interlinked Online Communities (SIOC)**, vocabulary for representing online communities
- **Description of a Project (DOAP)**, vocabulary for describing projects
- **Simple Knowledge Organization System (SKOS)**, vocabulary for representing taxonomies and loosely structured knowledge
- **Music Ontology** provides terms for describing artists, albums and tracks
- **Review Vocabulary**, vocabulary for representing reviews
- **Creative Commons (CC)**, vocabulary for describing license terms

Most popular and *old* vocabularies for RDF.
The most common aspect of general purpose domains of interest have been formalized by some vocabularies, *sets of URIs*, that have been defined to represent typical properties of some domains of interest.
For example, FOAF is set of URIs that represents properties of people. DC is general metadata attribute and so on.
If make the *effort* of sticking to the vocabulary when you taks about properties that are **already** almost standard vocabularies, then you are actually realizing the **shared semantic dream**. For instance if everybody, who wants to talk about the phone number of a person, should use the FOAF phone property URI. So, we agree on the identifier that we are going to use for our dataset, *at least* for most common property.

This is already an **advansement** respect to the previous situation in the world where everybody starts its own project creating its own identifier. But this create *big semantic problem* because **only** the organization that has produced these datasets **knows** the meaning of the identifier, it is **not shared**, it is **not** automatically understandable by another application because the other application has **no** idea of the meaning of this symbol in the another organization. This is the real *advantage*. It is **not** due to the *language*, is due to the **methodology** in the use of the language.
If you are going to write RDF, you must knwo first these vocabularies and then you can write and you have to consider these vocabularies as the facto standard. Before inventing new URIs to express some property/thing for your dataset try to reuse the popular URIs.