Semantic Web & Ontologies

Pablo Mollá Chárlez

September 24, 2024

Contents

1 Definition of concepts **1**
1.1 Principles of Linked Data . . . 2
1.2 Semantic Web Standards . . . 2
1.2.1 RDF: Resource Description Framework . . . 2
1.2.1.1 Knowledge Base . . . 3
1.2.1.2 Namespace . . . 3
1.2.1.3 QName: Qualified Name . . . 4
1.2.1.4 Base URI . . . 4
1.2.1.5 Turtle: Syntax for writing RDF . . . 4
1.2.1.6 Blank Node . . . 5
1.2.1.7 Multi-Valued Relationships . . . 6
1.2.1.8 RDF Schema . . . 6
1.2.1.9 Short Task . . . 8
1.3 SPARQL . . . 9
1.3.1 Types of SPARQL Queries . . . 9
1.3.2 Structure of a SPARQL Query . . . 10

1 Definition of concepts

The **Semantic Web** is an extension of the traditional web where data is structured in such a way that it can be understood, shared, and reused across different applications, platforms, and communities. Its primary goal is to transform the web from a collection of linked documents (like HTML pages) into a web of linked data. This enables machines to process, interpret, and reason about the data, facilitating more intelligent and automated services. The Semantic Web uses standards such as RDF (Resource Description Framework), OWL (Web Ontology Language), and SPARQL to represent and query data.

Key characteristics of the Semantic Web:

- Data is structured and described in a way that allows machines to understand its meaning (semantics).
- It supports reasoning, inference, and knowledge discovery by connecting related information.
- Data across different sources can be seamlessly linked and queried, enabling more powerful data integration.

Ontologies in the context of the Semantic Web are formal models that define the relationships between different concepts and entities within a specific domain. An ontology provides a shared vocabulary and a set of rules (or axioms) that describe how concepts are related to one another, allowing machines to understand and process data consistently.

Key components of Ontologies include:

- **Classes**: Categories or types of things in a domain (e.g., "Person", "Book", etc).
- **Properties**: Attributes or relationships between things (e.g., a person has a name, or a person writes a book).
- **Instances**: Specific examples of classes (e.g., "John" is an instance of the class "Person").
- **Axioms**: Rules that define how classes and properties intereact, ensuring logical consistency within the ontology.

Ontologies are essential for the Semantic Web because they provide the structure and logic needed to make sense of data, enabling interoperability and automated reasoning across diverse datasets.

1.1 Principles of Linked Data

According to Tim Berners-Lee (inventer of the World Wide Web, while at CERN, the European Particle Physics Laboratory, in 1989), there are **4 essential principles of linked data**:

- Use URIs (Uniform Resource Identifiers) to name (identify) things: Every "thing" (whether it's a physical object, a concept, or a digital resource) should have a unique identifier, called a URI. This ensures that each entity can be unambiguously referenced across the web. They have to be globally unique.
  - A URI is a string that follows the syntax:

    <scheme name> :<hierarchical part>[<query>][&<fragment>]

    All URLs are URIs but not all URIs are URLs. There are 2 main types of URIs: URLs (Uniform Resource Locators, these not only identify a resource but also provide a way to locate it on the web, think of usual websites) and URNs (Uniform Resource Names, these name a resource but do not describe how to locate it, think of ISBNs for books). Moreover, a URI always refers to one entity (video, picture...), never to more entities, although one entity can be referred by multiple URIs.
- Use HTTP URIs (Hypertext Transfer Protocol) so that these things can be looked up: The URI should be an HTTP URI, meaning it can be accessed via the web. This allows users and applications to "dereference" it, meaning they can look it up to retrieve useful information about what it represents
- Provide useful information about what a name identifies when it's looked up, using open standards such as RDF, SPARQL, etc: It should return meaningful information.
- Refer to other things using their HTTP URI-based names when publishing data on the Web: This creates a web of data where entities are interconnected, making it easier to discover related resources and build a richer, more navigable web of information.

1.2 Semantic Web Standards

We will define and give examples of the following standards: RDF, RDFS, SPARQL and OWL.

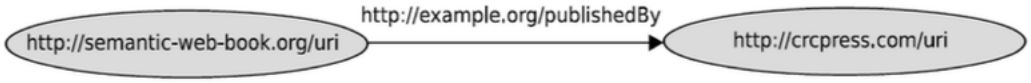
1.2.1 RDF: Resource Description Framework

RDF is a standard for representing structured information about reousrces on the web. It uses a directed graph model to express relationships between entities in the form of subject-predicate-object triples. Each triple is like a sentence:

- Subject: The thing being described.
- The property or relationship of the subject.

- The value of another resources related to the subject.

For instance, the subject is the URI **https://semantic-web-book.org/uri** (perhaps a book), the predicate is **http://example.org/publishedBy** and the object is the actual publisher **http://crcpress.com/uri**.



This structure forms a graph where **URIs** are nodes, and predicates are directed edges between nodes.

1.2.1.1 Knowledge Base

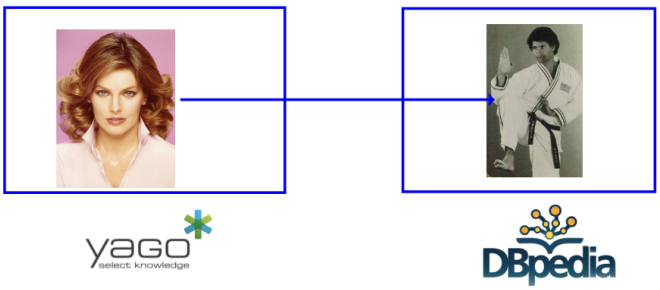
RDF graphs can be combined to form a Knowledge Base (KB) - a collection of RDF triples or graphs that represent a dataset. Each KB is assigned a name (a URI) and 2 KBs never should have the same name as it would lead to ambiguity and conflicts in identifying and referencing the data. URIs are meant to uniquely identify a resource on the web. To visualize things easier: URIs $\subset$ RDF (triples) $\subset$ KB (Knowledge Base).

One interesting property of KB is Cross-Referencing, meaning that a KB can make statements about entities defined in other KBs. For instance, let's consider the following example:

```
# Define a namespace prefix 'y'
@prefix y: <http://yago-knowledge.org/> .

# Another prefix 'd'
@prefix d:<http://dbpedia.org/>

# A triple that use the prefix "y" and "d"
y:Priscilla y:loves d:MikeStone .
```



1.2.1.2 Namespace

A namespace is a way to organize and disambiguate URIs within a RDF. It defines a common prefix for a group of URIs to prevent naming conflicts between different vocabularies or datasets. For example, **http://example.org/** is a namespace that might be used to define various properties and classes, such as:

$$\underbrace{\textbf{http://example.org/}}_{namespace/}\underbrace{\textbf{publishedBy}}_{property}$$

A namespace prefix is an abbreviation for the first part of a URI. For instance;

@prefix dbp: <http://dbpedia.org/>.
dbp:Elvis = <http://dbpedia.org/Elvis>

1.2.1.3 QName: Qualified Name

A qualified name is a shorthand notation for referring to URIs by using a common prefix and a local name. It combines a namespace with a specific property or class. For instance, if **http://example.org/** is the namespace and **publishedBy** is a property, the QName might be written as **ex:publishedBy**, where **ex** is a defined prefix for the namespace. The concept QName is very similar to CURIE, however, this last one can be used in more general contexts.

Another example, would be considering as **KB1** and **KB2** namespaces which they contain Elvis, Priscilla, Lisa and Elvis, Michael as local names, respectively. Therefore, valid QNames would be **KB1:Elvis**, **KB1:Priscilla** or **KB2:Elvis**.

1.2.1.4 Base URI

A base URI is a reference URI for URIs within a document of dataset for simplification and consistency. For example, by defining:

@base <http://yago-knowledge.org/>.
<Elvis> = <http://yago-knowledge.org/Elvis>

In this scenario, <Elvis> is a relative URI from the previously defined base URI.

1.2.1.5 Turtle: Syntax for writing RDF

Turtle (Terse RDF Triple Language) is a textual syntax for expressing RDF data. It is a compact and human-readable way to write RDF triples, which consist of subject, predicate and object. It helps simplify RDF statements by allowing the use of prefixes, Base URIs and QNames.

- @prefix P:<URI>.
- @base <URI>.
- Any t riple: subject predicate object.

For instance, let's consider:

```
# Define the base URI for relative references
@base <http://yago-knowledge.org/> .

# Define a namespace prefix 'ex'
@prefix ex: <http://example.org/> .

# A triple that uses the base URI and a full URI
<Elvis> ex:publishedBy <http://crcpress.com/uri> .
```

Explanation of this Turtle document:

- **Base URI**: The line **@base <http://yago-knowledge.org/>.** defines a base URI. This means that any relative URI (like <Elvis>) will be automatically expanded to **http://yago-knowledge.org/Elvis**.
- **QName**: The line **@prefix ex: <http://example.org/>.** defines a prefix which is further used in the triple by including the QName **ex:publishedBy** which in fact represents **http://example.org/publishedBy**.
- **Triple RDF**: Finally, the line **<Elvis> ex:publishedBy <http://crcpress.com/uri> .** is an RDF triple that links the subject **<Elvis>** (resolved as "http://yago-knowledge.org/Elvis") to the considered object **http://crcpress.com/uri** via the predicate **ex:publishedBy** which this last one expands to "http://example.org/publishedBy".

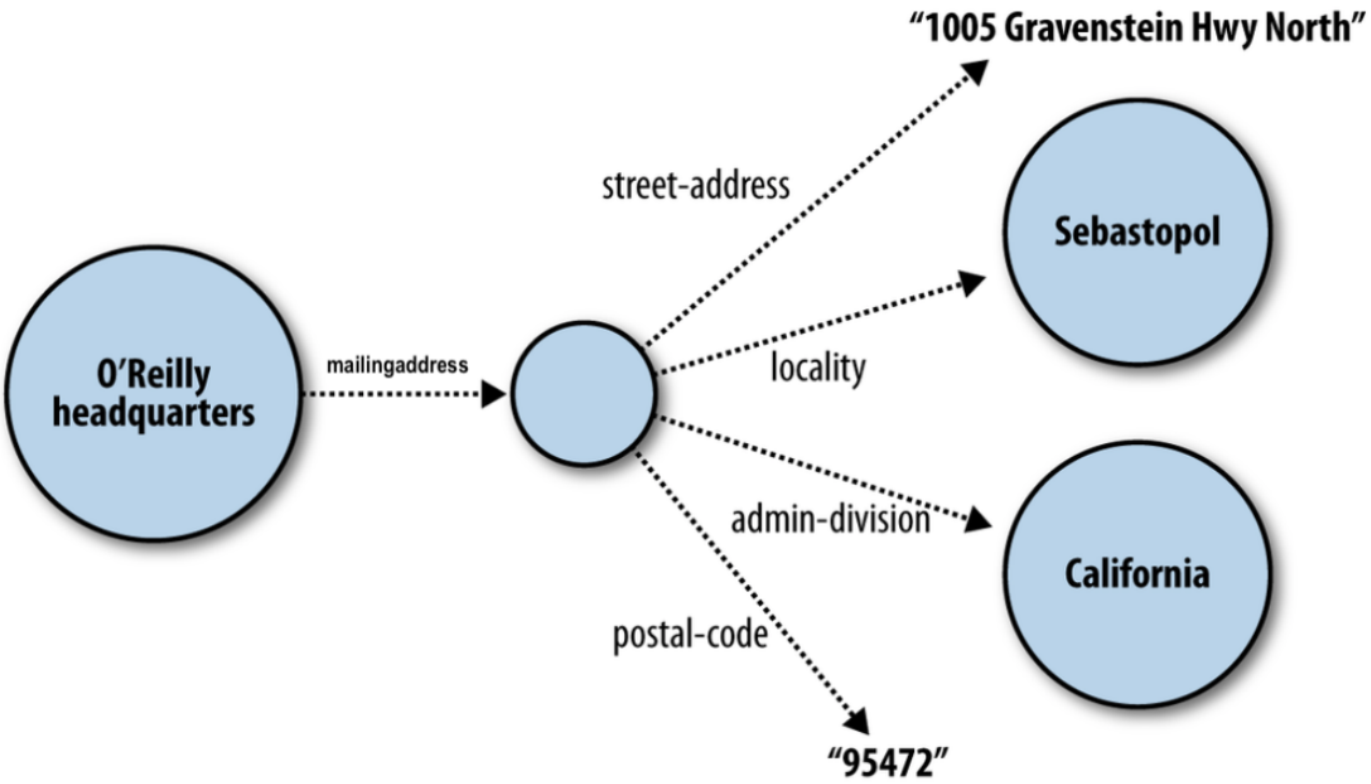
1.2.1.6 Blank Node

A blank node is a concept in RDF, representing an entity in the RDF data model that does not have a URI (Uniform Resource Identifier). Instead, blank nodes are used locally in an RDF graph for internal references without being explicitly identified outside the graph.

Characteristics of Blank Nodes:

- **Local Identifiers**: Blank nodes are identified by a node ID, but this ID is used only locally within a specific RDF graph.
- **Representation in RDF Syntax**: In Turtle, blank nodes can be represented in 2 ways:
  1. Using a node identifier like _**:entity**.
  2. As a concise syntax using square brackets [].
- **Purpose**: Blank nodes often arise when you need to describe complex relationships or attribtes of entities without providing full URIs for every node in the RDF graph.
- **Graph Disambiguation**: When combining multiple RDF graphs, blank nodes must be disambiguated. Since their names can change across graphs (i.e., the local identifiers might conflict), it's important to handle them carefully during merging.
- **Common Use Cases**: Representing intermediate/auxiliary data part of a larger structure that doesn't need a global identifier.

Let's consider the following example and its Turtle syntax code:



The triples always follow the same structure: subject + predicate + object

```
# Define a namespace prefix 'ex'
@prefix ex: <http ://www.lri.fr/sw/example> .

# A triple that use the prefix "ex" and a blank node
ex: OreillyHeadquarters ex: mailingaddress _:entity .

# Blank node description properties of mailingaddress
_:entity ex:street-address "1005 Gravenstein Hwy North" ;
         ex:locality ex:Sebastopol ;
         ex:admin-division ex:California ;
         ex:postal-code "95472" .
```

1.2.1.7 Multi-Valued Relationships

In RDF, a multi-valued relationship refers to a case where an entity (subject) has multiple values for the same property (predicate). For example, in the context of a recipe, the **ex:Pasta** has multiple **ex:hasSauce** properties, such as pesto, carbonara, tomato and etc.

A proper way to represent such data, and most importantly, make it reusable for other users, is to apply the previously defined blank nodes. For instance, if we want to describe exactly how a given sauce for pasta is made, you would need to define it as follows:

```
# Define a namespace prefix 'ex'
@prefix ex: <http ://www.recipes.es/european/italian/> .

# A triple that use the prefix "ex" and a blank node
ex: Pasta ex: hasSauce _:pesto, _:carbonara .

# Blank node description properties of pesto
_:pesto ex:sauceName "Pesto" ;
        ex:hasIngredient _:pestoIngredient1, _:pestoIngredient2, _:pestoIngredient3 .

_:pestoIngredient1 ex:name "Basil";
                   ex:amout "50 g" .
_:pestoIngredient2 ex:name "Garlic";
                   ex:amount "2 Gloves" .
_:pestoIngredient3 ex:name "Olive Oil" ;
                   ex:amount "2 tbsp" .

# Blank node description properties of carbonara
_:carbonara ex:sauceName "Carbonara" ;
            ex:hasIngredient _:carbonaraIngredient1, _:carbonaraIngredient2, _:carbonaraIngredient3 .

_:carbonaraIngredient1 ex:name "Eggs" ;
                       ex:amount "2" .

_:carbonaraIngredient2 ex:name "Pecorino Cheese" ;
                       ex:amount "50 g" .

_:carbonaraIngredient3 ex:name "Guanciale" ;
                       ex:amount "100 g" .
```

1.2.1.8 RDF Schema

RDF Schema (RDFS) vocabulary, which is an extension of RDF, provides a basic set of constructs to describe relationships and properties of resources in a semantic web environment. Let's highlight some of the RDFs Vocabulary concepts:

- It is a vocabulary (a set of predefined terms) used for describing **class hierarchies** and **properties** of resources in RDF data. It includes constructs such as ‘rdfs:Class‘, ‘rdfs:subClassOf‘, ‘rdfs:domain‘, ‘rdfs:range‘, and ‘rdfs:label‘.
- **Basic RDFS Terms**:
  1. rdfs:Class: Represents a class or category. For example, ‘y:Person‘ is a class representing people.
  2. rdfs:subClassOf: Defines class hierarchies. For instance, ‘y:Singer rdfs:subClassOf y:Person‘ means that all singers are a subclass of people, implying that every singer is a person.
  3. rdfs:domain: Specifies the class to which a property applies. If ‘ex:creator rdfs:domain ex:Document‘, it means that the ‘creator‘ property is applicable to instances of the class ‘Document‘.
  4. rdfs:range: Defines the expected type of values for a property. For example, ‘ex:creator rdfs:range ex:Person‘ indicates that the value of the ‘creator‘ property must be a person.
  5. rdfs:label: Provides a human-readable name or description for a resource.

Let’s consider an example.

```
# Define a namespace prefix 'ex'
@prefix ex: <http://www.lri.fr/sw/example> .

# Triple using the previous prefix
ex:LesMiserables ex:creator ex:Hugo .

# Vocabulary
ex:creator rdf:type rdf:Property .
ex:creator rdfs:domain ex:Document .
ex:creator rdfs:range ex:Person .
ex:creator rdfs:label "author/creator" .
```

This example models the relationship between the document ”Les Miserables” and its creator ”Hugo”. The ”ex:creator” property connects the resource ”ex:LesMiserables” (a document) to the resource ”ex:Hugo” (a person). The property ”rdfs:domain ex:Document” means that ”ex:creator” is used for documents, and ”rdfs:range ex:Person” means the value for ”ex:creator” must be a person. Finally, the ”rdfs:label” provides the human-readable name ”author/creator” for the ‘ex:creator‘ property.

To not confuse with the common ”rdf” prefix values.

- **rdf:type**: Specifies the type of a resource.
- **rdf:Property**: Declares a resource as a property.
- **rdf:Class**: Declares a resource as a class.
- **rdf:subject**, **rdf:predicate**, **rdf:object**: Refer to parts of a triple.
- **rdf:Resource**: The most general class (everything is a resource).

Some examples include:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: <http://www.lri.fr/sw/example> .


# Define classes
ex:Person rdf:type rdf:Class .
ex:Document rdf:type rdf:Class .
```

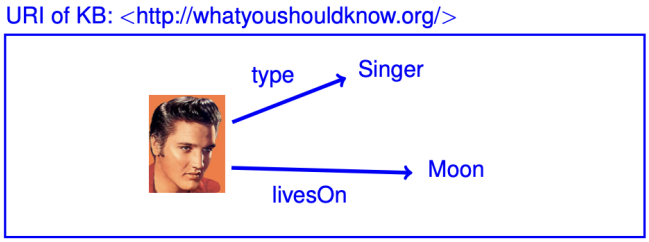
```
# Define properties
ex:creator rdf:type rdf:Property .
ex:publishedIn rdf:type rdf:Property .

# Describe resources using rdf:type and other properties
ex:Hugo rdf:type ex:Person .
ex:LesMiserables rdf:type ex:Document .
ex:LesMiserables ex:creator ex:Hugo .
```

In summary, "rdf": can be followed by a wide variety of terms as well (e.g., type, Class, Property, subject, etc.), which are part of the RDF vocabulary used to express relationships, types, and data about resources.

1.2.1.9 Short Task

Write the following facts in Turtle, using RDF vocabulary where possible. Start with:
" @prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns".




```
# Predefined Knowledge base to use
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .

# Custom Properties induced from the graph defined in my own vocabulary
@prefix ex: <http://whatyoushouldknow.org> .

rdf:Elvis_Presley rdf:type ex:Singer;
                  ex:livesOn ex:Moon .
```

1.3 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is a standard query language used to retrieve and manipulate data stored in the form of RDF triples. It allows querying RDF data efficiently. It is one of the core standards of the Semantic Web, along with RDF and OWL (web Ontology Language).

1.3.1 Types of SPARQL Queries

Let's consider the main four SPARQL type of queries.

1. **SELECT**: Returns a table of results that match the query pattern (like SQL). It is used to select specific variables from the dataset.

```
# Asking to return a table containing the "name" variable
SELECT ?name WHERE {

    # Find all resources "person" of type "Person"
    ?person rdf:type ex:Person .
    # Once "person" is found, return the "name" associated
    # with that person through the property "hasName"
    ?person ex:hasName ?name .
}
```

   This query return a list of names of all resources of type "Person".

2. **CONSTRUCT**: Returns an RDF graph (a set of triples) by constructing newRDF statements from the query pattern.

```
# Constructing a new RDF which will return triples based on
# the data matched in WHERE clause
# The new created triples where each "person" will have a full name
CONSTRUCT {
    ?person ex:hasFullName ?fullName .
}

WHERE {
   # Find all resources with a first name
   ?person ex:firstName ?firstName .

   # Find all resources with a last name
   ?person ex:lastName ?lastName .

   # Combines (concat) first and last name (separated by space)
   # and binds it to a new variable "fullName"
   BIND(CONCAT(?firstName, " ", ?lastName) AS ?fullName)
}
```

   This query generates a new graph with full names of people based on their first and last names.

3. **DESCRIBE**: Returns all RDF statements related to a resource. The exact data returned depends on the specific implementation of the SPARQL endpoint.

```
# Askin for a detailed description of the "person" found
# in the dataset
DESCRIBE ?person WHERE {
    # Look for a resource of type "Person"
    ?person rdf:type ex:Person .

    # Filter to find the specific person's name
    ?person ex:hasName "Elvis Presley" .
}
```

This query returns all the data about the resource describing "Elvis Presley".

4. **ASK**: Returns a boolean value, indicating whether a specific condition or pattern exists in the RDF dataset.

```
# Asking whether the following pattern exists or not (boolean)
ASK WHERE {
    # Look for a resource of type "Person"
    ?person rdf:type ex:Person .

    # Check whether any of those people are
    # are calle "Elvis Presl"
    ?person ex:hasName "Elvis Presley" .
}
```

This query checks if there is a person named "Elvis Presley" in the dataset.

1.3.2 Structure of a SPARQL Query

A basic SPARQL query consists of the following parts:

1. PREFIX: Declares the namespaces (similar to prefixes in RDF).
2. SELECT/CONSTRUCT/ASK/DESCRIBE: Specifies the type of query.
3. WHERE: Defines the graph pattern to match.
4. FILTER: Optional, addsconditions to the filter results.

Let's say we have RDF data about people (DBpedia RDF Datatset) and their birthplaces (Custom Vocabulary), and we want to find out the birthplace of Elvis Presley.

```
PREFIX ex: <http://example.org/vocab#>
PREFIX dbpedia: <http://dbpedia.org/resource/>

SELECT ?birthplace WHERE {
    dbpedia:Elvis_Presley ex:birthPlace ?birthplace .
}
```

This query:

- Uses PREFIX to declare the ex: and dbpedia: namespaces.
- Performs a SELECT query to retrieve the birthplace of Elvis Presley (dbpedia:Elvis_Presley).
- The WHERE clause defines the pattern to match: find the birthplace related to Elvis Presley using the ex:birthPlace property/predicate. We need the ex: namespace because ex:birthPlace is the property being used to extract the birthplace of Elvis. The dbpedia: namespace provides the resource (Elvis Presley), but the property connecting Elvis to his birthplace is defined in a different schema (ex:).