

"TECHNO DU BIG-DATA »

Linked Open data

Stéphane Derrode, Dpt MI - stephane.derrode@ec-lyon.fr



LINKED DATA

1. Web des données, *Linked Data*
2. *Resource Description Framework (RDF)*
3. SparQL

1. LOD : Qu'est-ce que le Web ?

Un espace documentaire décentralisé, interconnecté, interopérable et *évolutif*.

- décentralisé → **HTTP 2014, HTTP 2.0**
- interconnecté → **URL, URI, IRI**
- interopérable → **HTML5, CSS, JS**

Vers un Web de données

Un espace **de données** décentralisé, interconnecté et interopérable.

- décentralisé → **HTTP**
- interconnecté → **URL**
- interopérable → ?

1. LOD : Web de ressources

Le web est constitué de **ressources**, par exemple :

- le bulletin météo du jour pour Lyon
- le bulletin météo du jour pour le lieu courant
- ma commande de café de jeudi dernier

Chaque ressource est identifiée par un IRI
(*Internationalized Resource Identifier*), e.g.:

- <http://meteo.example.com/lyon>
- <http://meteo.example.com/ici>
- <http://commerce.example.com/commande/192837>

1. LOD : Web de ressources

Une ressource n'est pas un simple fichier, dont on récupèrerait le contenu.

Elle est un objet *actif*, avec lequel on interagit.

- le bulletin météo du jour pour Lyon :
→ le contenu change régulièrement
- le bulletin météo du jour pour le lieu courant :
→ le contenu dépend de plus du contexte de l'utilisateur
- ma commande de café de jeudi dernier :
→ on peut agir dessus (par exemple pour l'annuler)

1. LOD : Ressources et représentations

- Une ressource n'est jamais manipulée directement, mais toujours à travers des **représentations** (pour la créer, la consulter, la modifier).
- Les représentations d'une ressource peuvent varier en fonction
 - de son état
 - de l'agent qui manipule la ressource (négociation de contenu, contexte)

représentation :	utilisable par :
texte (HTML...)	humains, moteurs de recherche
médias (image, son...)	<i>surtout</i> humains
données structurées	machines

1. LOD : Ressources et représentations

- Une ressource n'est jamais manipulée directement, mais toujours à travers des **représentations** (pour la créer, la consulter, la modifier).
- Les représentations d'une ressource peuvent varier en fonction
 - de son état
 - de l'agent qui manipule la ressource (négociation de contenu, contexte)

représentation :	utilisable par :
texte (HTML...)	humains, moteurs de recherche
médias (image, son...)	<i>surtout</i> humains
données structurées	machines

1. LOD : de HTML à XML

XML (*eXtensible Markup Language*) a été recommandé par le W3C en 1998. L'objectif était de pallier la sémantique « faible » de HTML.

```
<!-- HTML -->
<a href="http://champin.net/">
  Pierre-Antoine <strong>Champin</strong>
  (<em>Maître de conférences</em>)</a>
```

```
<!-- XML -->
<Person homepage="http://champin.net/">
  <givenName>Pierre-Antoine</givenName>
  <familyName>Champin</familyName>
  <job>Maître de conférences</job></Person>
```

1. LOD : de XML à RDF

- Le modèle sous-jacent de la syntaxe XML est un arbre (*XML Infoset*), ce qui n'est pas adapté à la structure décentralisée du Web.
- L'objectif du *Resource Description Framework* (RDF), recommandé par le W3C en 1999, vise à munir le Web d'un modèle de données plus adapté, ayant une structure de *graphe*.
- L'objectif est de construire le *Semantic Web* : un web dans lequel les machines ont (enfin) accès à la sémantique des données.
- Recommandation un peu hâtive, présentant quelques défauts importants (notamment l'absence de sémantique formelle).
→ faible adoption de RDF



1. LOD : de RDF à RDF

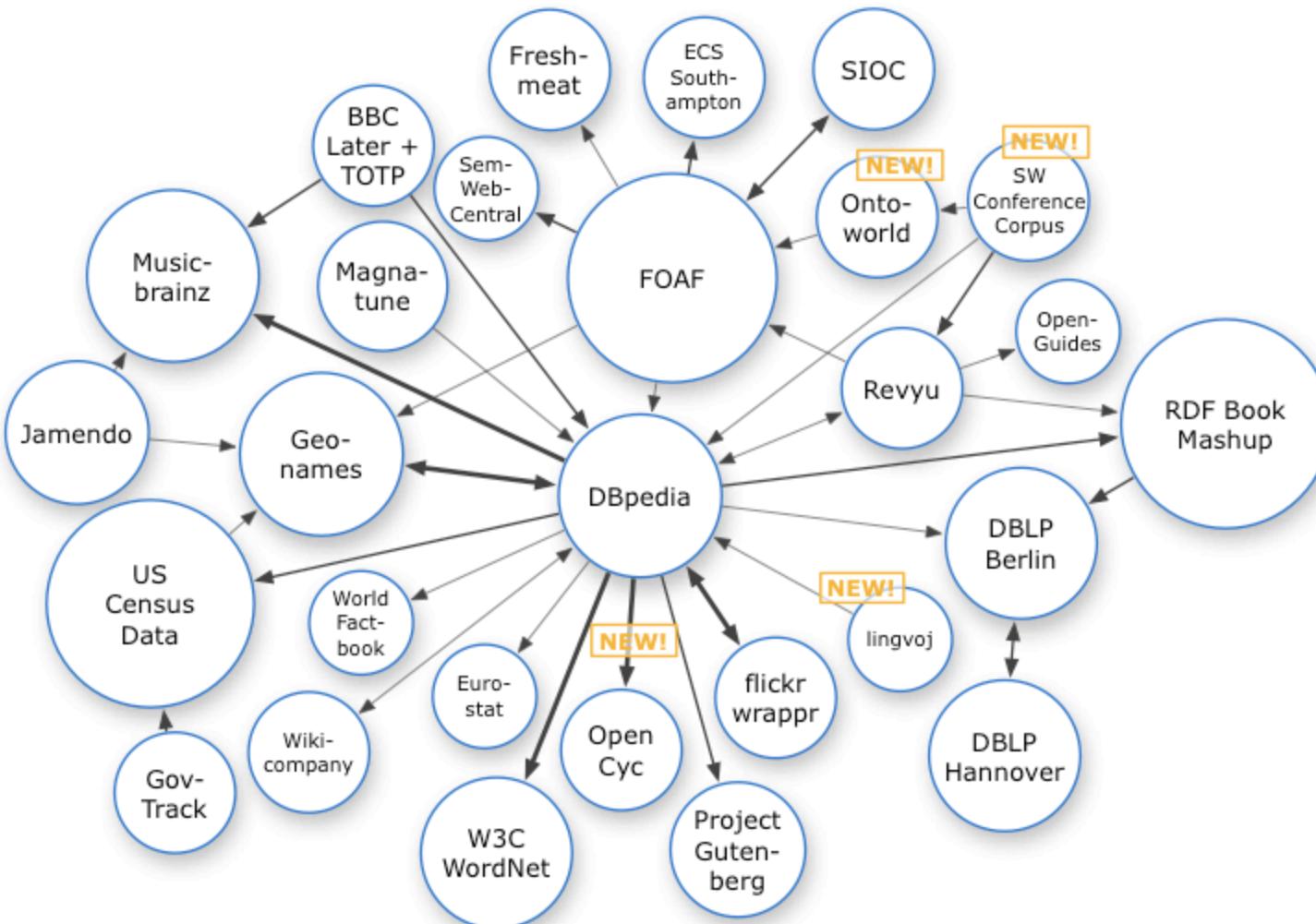
- En 2004, le W3C publie un nouvel ensemble de recommandations sur RDF pour remplacer celles de 1999.
- Pour des raisons de compatibilité avec l'existant, certains aspects sont conservés malgré les débats qu'ils suscitent, mais les défauts considérés comme majeurs sont corrigés.
- Après cet échec relatif, l'appellation *Semantic Web* tombe peu à peu en disgrâce. Certains défenseurs de RDF parlent plus modestement de *Data Web*, puis de *Web of Linked Data* (2006).
- En 2014, RDF est plus largement accepté. Le W3C publie une version révisée (RDF 1.1), endossant notamment plusieurs syntaxes concrètes.

1. LOD : Raw Data Now!

- The next Web,
https://www.ted.com/talks/tim_berners_lee_the_next_web



1. LOD : Linked open Data



Source image : Richard Cyganiak (<https://lod-cloud.net>)

1. LOD : Linked open Data

Les quatre principes de Linked Data

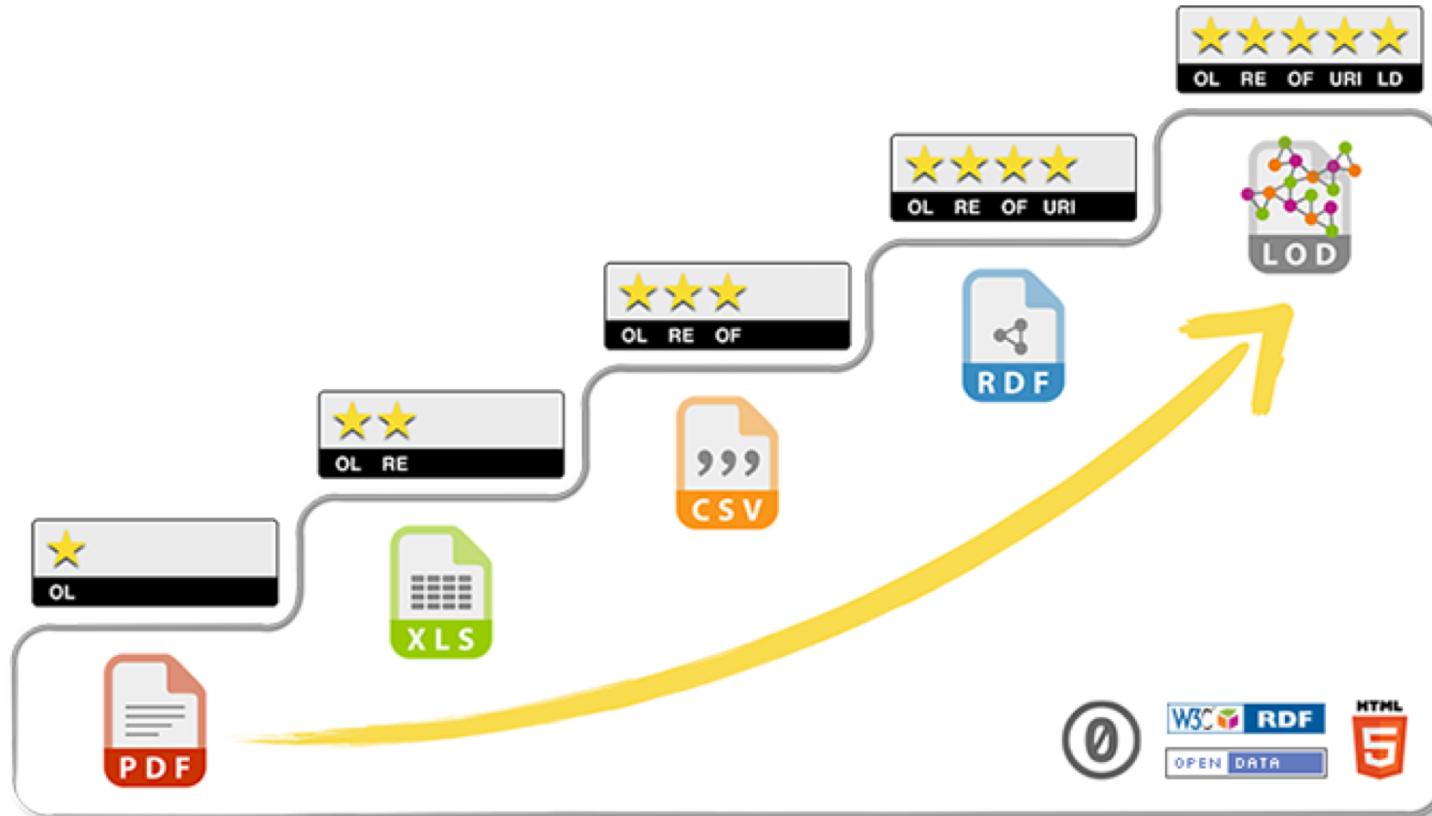
1. Utiliser des IRIs pour nommer les choses (= ressources).
2. Utiliser des IRIs HTTP pour pouvoir obtenir des *représentations* de ces ressources.
3. Fournir ces représentations en utilisant des langages et des protocoles standards ([RDF](#), [SPARQL](#)).
4. Inclure des liens pour permettre de découvrir de nouvelles ressources.

d'après Tim Berners Lee,

<http://www.w3.org/DesignIssues/LinkedData.html>

1. LOD : Linked open Data

Linked open data [star scheme \(<https://5stardata.info/en/>\)](https://5stardata.info/en/)



1. LOD : Projet emblématique, DBPédia

- Projet lancé par Chris Bizer en 2007.
- Objectif : extraire les informations structurées (*infobox*) présentes dans Wikipedia pour les exposer en RDF.
- En novembre 2015 :

The English version of the DBpedia knowledge base describes 4.58 million things, (...) including 1,445,000 persons, 735,000 places (including 478,000 populated places), 411,000 creative works (including 123,000 music albums, 87,000 films and 19,000 video games), 241,000 organizations (including 58,000 companies and 49,000 educational institutions), 251,000 species and 6,000 diseases.

1. Projet emblématique : DBPédia

Screenshot of the Wikipedia page for Lyon (French city) on the English Wikipedia.

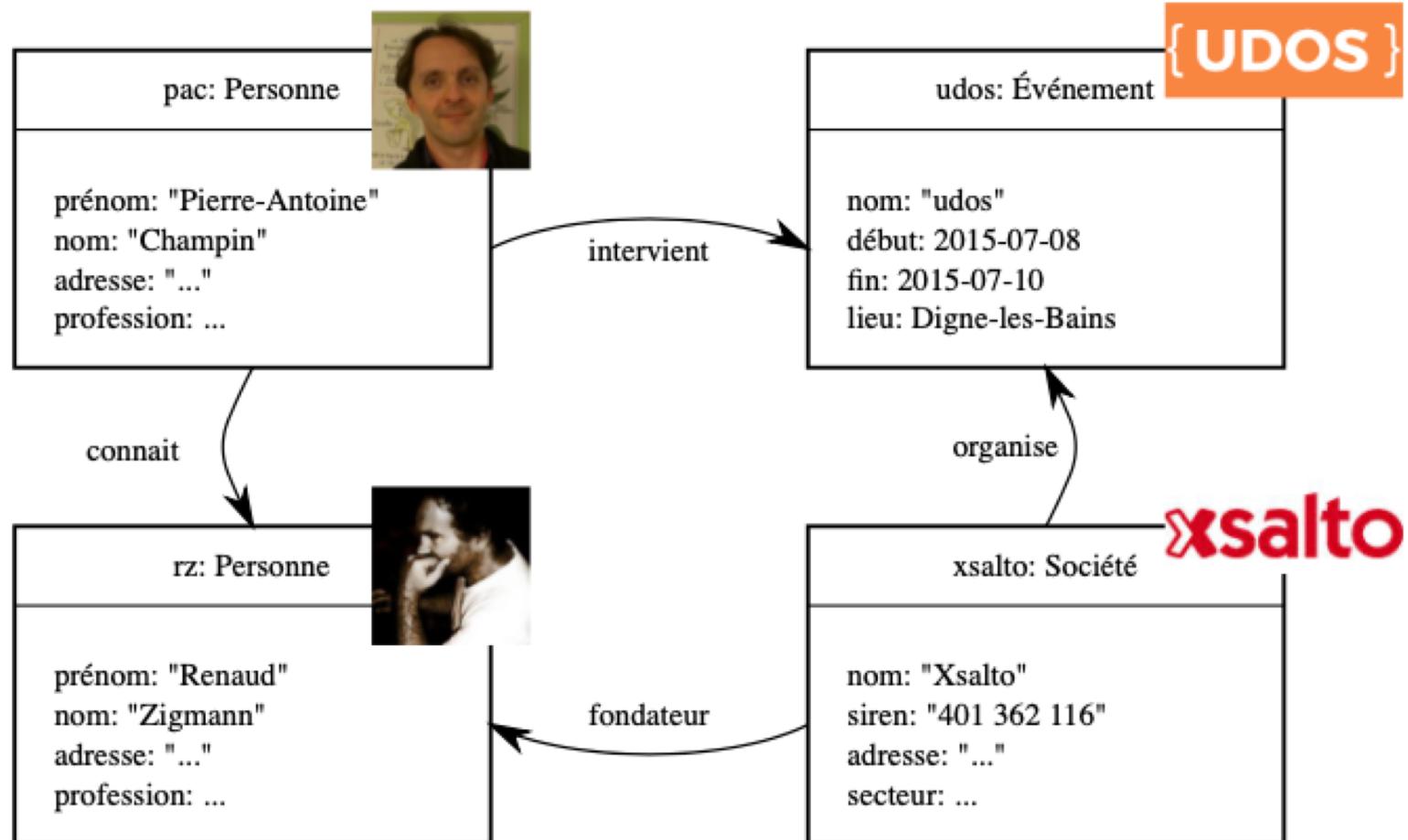
The page title is "Lyon". The main content starts with a brief history and then details Lyon's status as a major center of business, its gastronomy, and its role in cinema. It also highlights Lyon's industrial centers and software industry.

The right sidebar contains sections for the city flag (with a red field featuring three golden yellow stars), the city coat of arms (a heraldic shield with a lion and a cross), and a motto: "Avant, avant, Lyon le meilleur." (Arpitan: Forward, forward, Lyon the best). Below these are images of Lyon from Fourvière and a map showing Lyon's location in France.

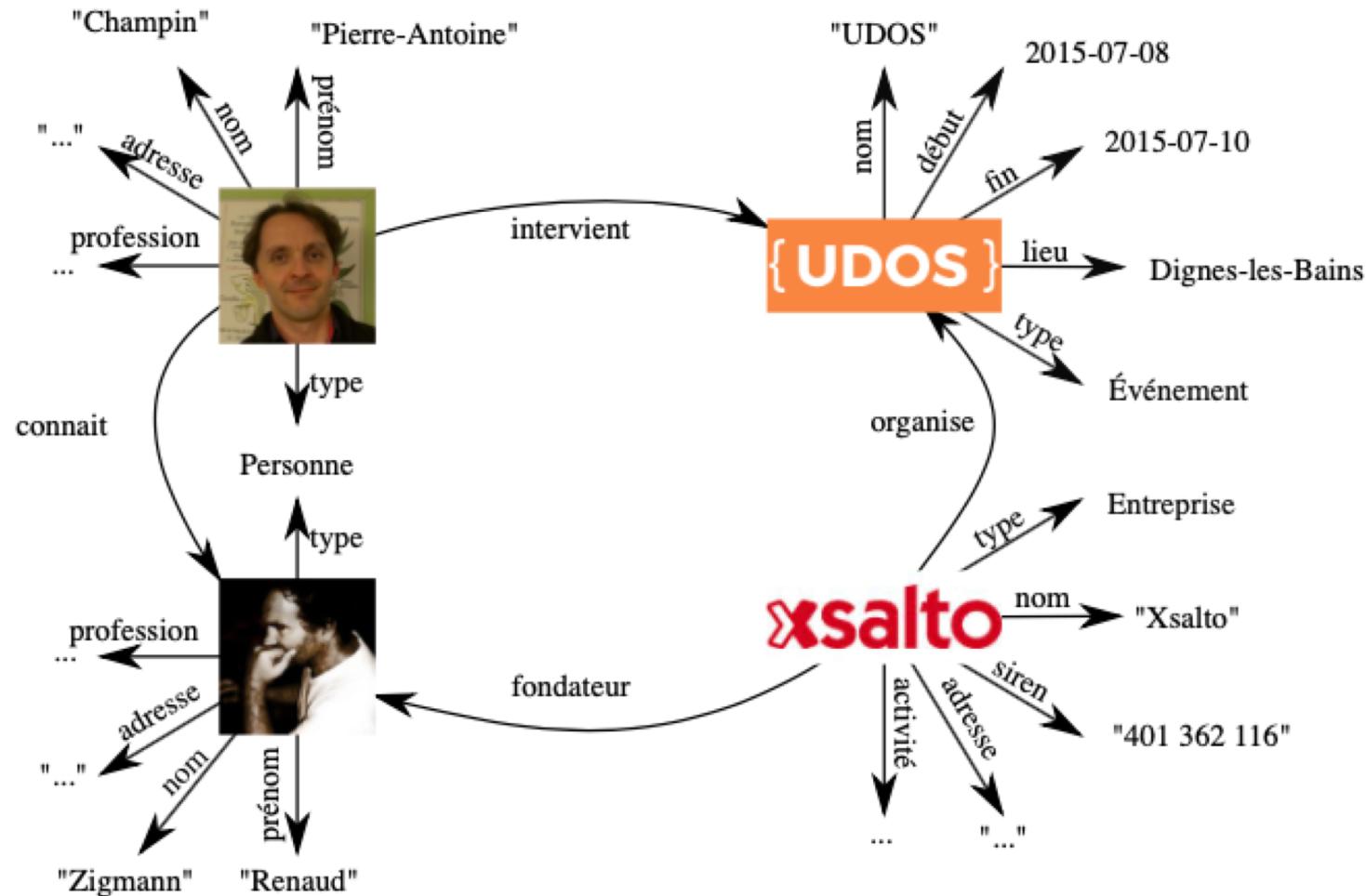
The "Administration" section lists the following details:

Time zone	CET (GMT+1)
Administration	
Country	France
Region	Rhône-Alpes
Department	Rhône (69)
Arrondissement	Lyon
Canton	chief town of 14 cantons
Subdivisions	9 arrondissements
Intercommunality	Urban Community of Lyon

2. RDF : des données aux données liées

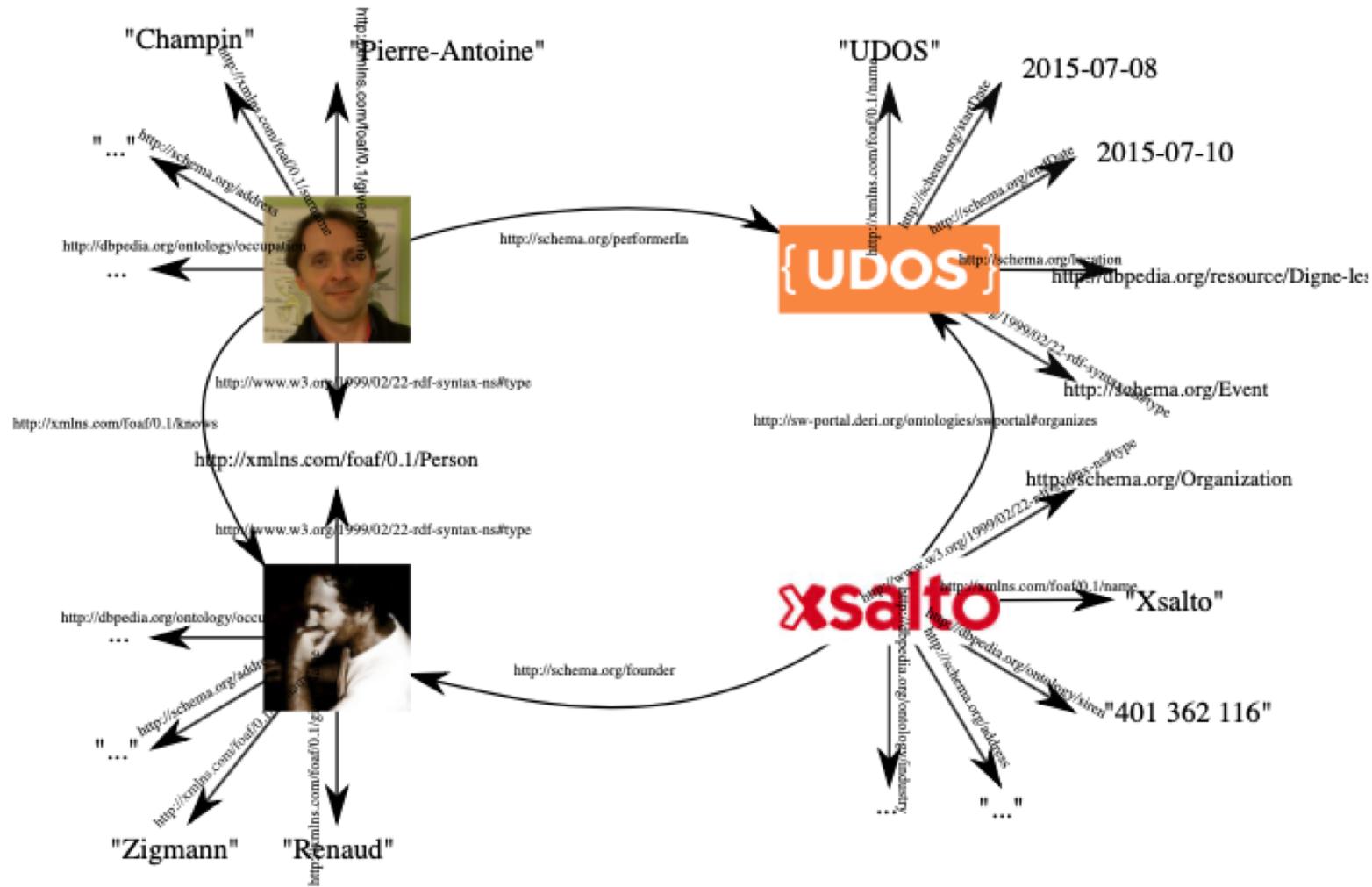


2. RDF : des données aux données liées



Données vues comme un graphe

2. RDF : Des données aux données liées



Données vues comme un graphe avec des IRIs

2. RDF : syntaxe abstraite

Toute information en RDF est représentée par un *triplet*, signifiant qu'une *chose* est en *relation* avec une autre.

Exemple :

Le laboratoire LIRIS (**sujet**)

a pour membre (**prédicat**)

Pierre-Antoine Champin (**objet**)

2. RDF : syntaxe abstraite

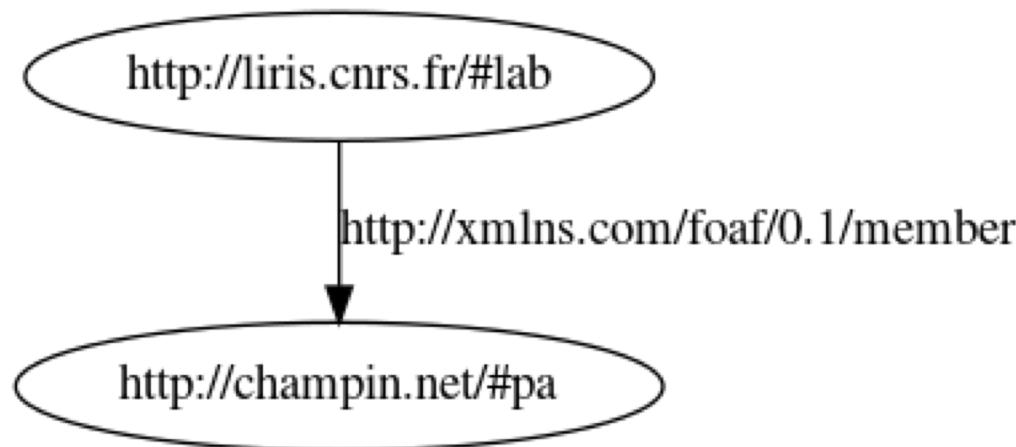
Les choses sont nommées par des IRIs :

<http://liris.cnrs.fr/#lab>

<http://xmlns.com/foaf/0.1/member>

<http://champin.net/#pa>

On peut représenter ceci graphiquement :



2. RDF : préfixes

Pour simplifier les **notations**, on définit des préfixes courts correspondant à des préfixes d'IRI :

liris: → http://liris.cnrs.fr/#

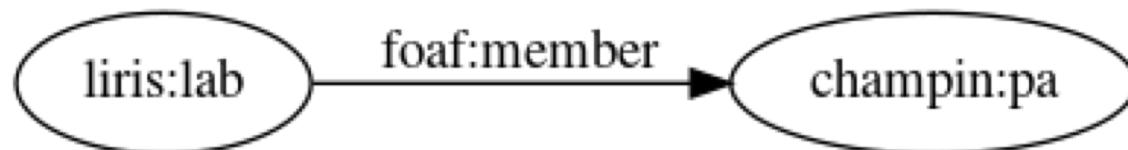
foaf: → http://xmlns.com/foaf/0.1/

champin: → http://champin.net/#

On utilise ensuite des *noms prefixés* :

liris:lab foaf:member champin:pa

et également sous forme graphique :

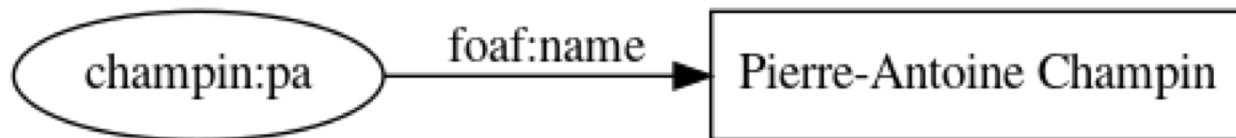


2. RDF : les littéraux

On peut également lier une ressource à une *donnée typée* (chaîne de caractères, entier, réel...), nommée un littéral.

champin:pa foaf:name "Pierre-Antoine Champin"

Traditionnellement, on représente les littéraux par des nœuds rectangulaires :

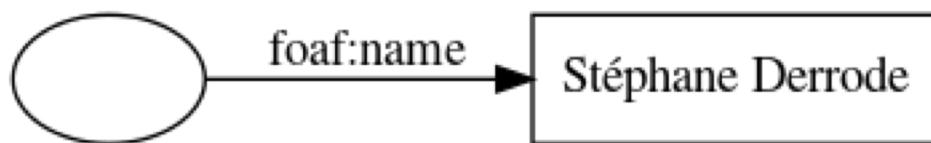


2. RDF : les nœuds muets

Enfin, RDF permet de parler d'une ressource sans connaître son IRI. Cela revient en logique à utiliser une variable quantifiée existentiellement.

(quelque chose) foaf:name "Stéphane Derrode"

On parle alors de nœud *muet* (par analogie aux variables muettes). Graphiquement, on représente cette ressource par un nœud vierge (*blank node*).

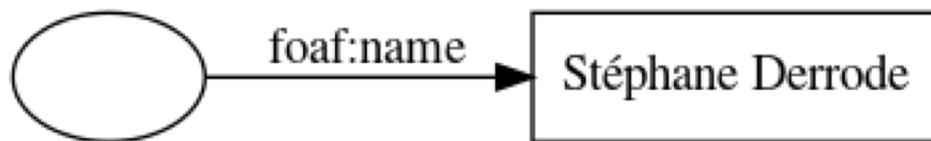


2. RDF : exemple de graphe

Enfin, RDF permet de parler d'une ressource sans connaître son IRI. Cela revient en logique à utiliser une variable quantifiée existentiellement.

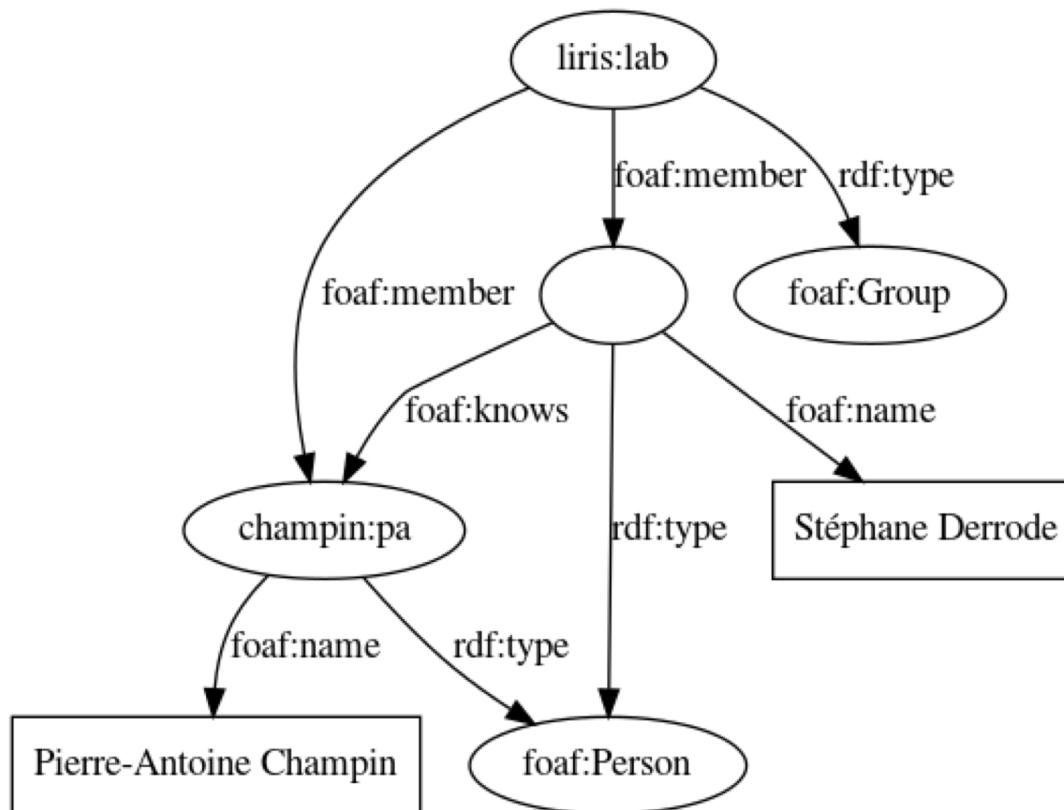
(quelque chose) foaf:name "Stéphane Derrode"

On parle alors de nœud *muet* (par analogie aux variables muettes). Graphiquement, on représente cette ressource par un nœud vierge (*blank node*).



2. RDF : les nœuds muets

Un ensemble de triplets forme un graphe orienté étiqueté.



2. RDF : syntaxes concrètes

RDF / XML

- syntaxe originale recommandée par le W3C (1999)
- basée sur XML
- relativement complexe et verbeuse

Syntaxe : <http://www.w3.org/TR/rdf-syntax-grammar/>

Validator : <http://www.w3.org/RDF/Validator/>

2. RDF : syntaxes concrètes

RDF / XML : exemple

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:foaf="http://xmlns.com/foaf/0.1/"
    <foaf:Group rdf:about="http://liris.cnrs.fr/#lab">
        <foaf:member>
            <foaf:Person>
                <foaf:name>Stéphane Derrode</foaf:name>
                <foaf:knows
                    rdf:resource="http://champin.net/#pa"/>
                </foaf:Person>
            </foaf:member>
            <foaf:member>
                <foaf:Person rdf:about="http://champin.net/#pa">
                    <foaf:name>Pierre-Antoine Champin</foaf:name>
                </foaf:Person>
            </foaf:member>
        </foaf:Group>
    </rdf:RDF>
```

2. RDF : syntaxes concrètes

RDF / Turtle

- dérivée du langage [N3](#)
- adoptée dans RDF 1.1 en 2014
- vise la simplicité et la compacité

Syntaxe : <http://www.w3.org/TR/turtle/>

Validator : <http://www.rdfabout.com/demo/validator/>

2. RDF : syntaxes concrètes

RDF / turtle : exemple

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix champin: <http://champin.net/#> .

liris:lab
  a foaf:Group ;
  foaf:member champin:pa, _:sd .
champin:pa
  a foaf:Person ;
  foaf:name "Pierre-Antoine Champin" .
_:sd
  a foaf:Person ;
  foaf:name "Stéphane Derrode" ;
  foaf:knows champin:pa .
```

2. RDF : syntaxes concrètes

RDF / JSON

- Rappel : JSON est un langage d'échange de données, basé sur Javascript, et très utilisé en développement web.
- JSON-LD (JSON Linked Data) permet d'interpréter une structure JSON comme du RDF, grâce à un *contexte* (implicite ou explicite).
- Objectif : faciliter l'adoption de RDF (syntaxe abstraite) auprès des développeurs d'applications web.

Syntaxe : <http://www.w3.org/TR/json-ld-syntax/>

Validator : <http://json-ld.org/playground/>

2. RDF : syntaxes concrètes

RDF / JSON : exemple

```
{ "@context": { /* ... */ },  
  "@id": "http://liris.cnrs.fr/#lab",  
  "@type": "Group",  
  "member": [  
    {  
      "@id": "http://champin.net/#pa",  
      "@type": "Person",  
      "name": "Pierre-Antoine Champin"  
    },  
    {  
      "@type": "Person",  
      "name": "Stéphane Derrode",  
      "knows": "http://champin.net/#pa"  
    }  
  ]  
}
```

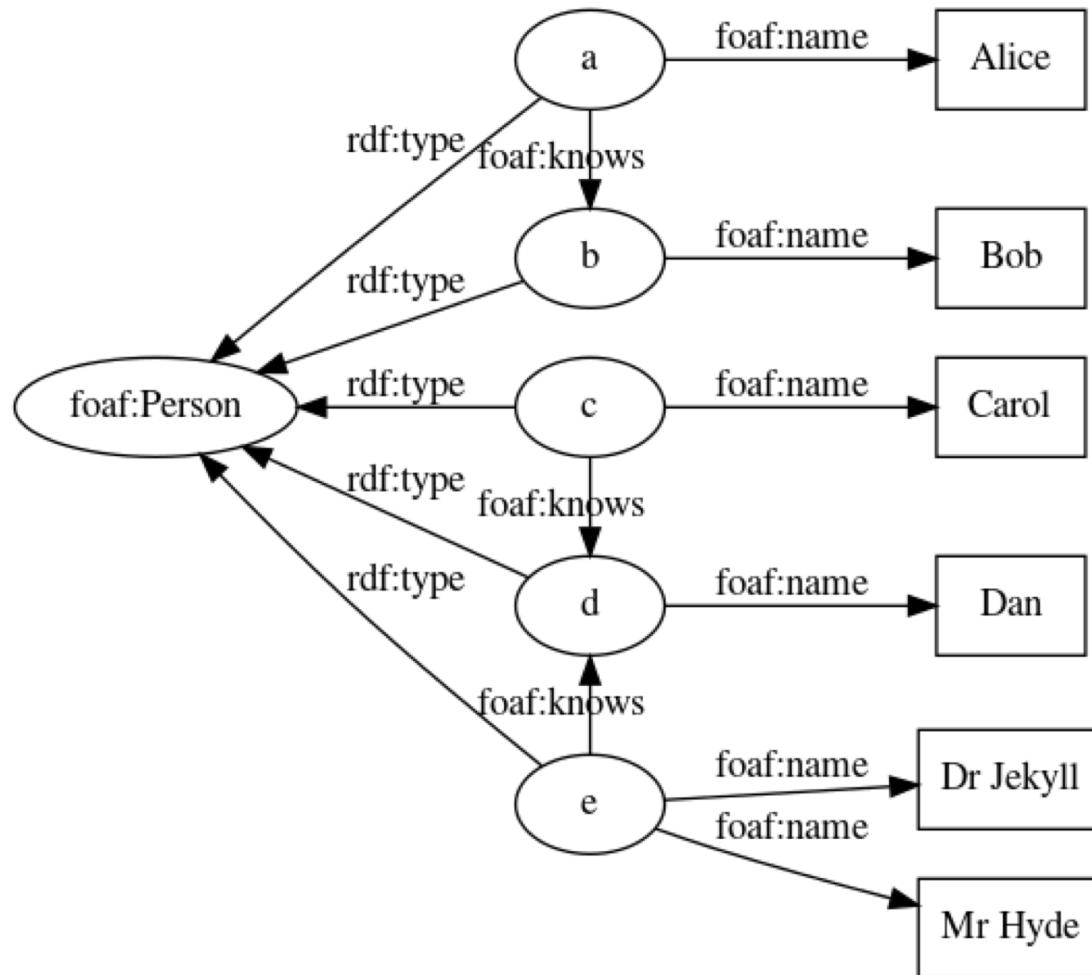
3. SparQL : objectifs

- Vous donner des bases pour écrire des requêtes SPARQL.
- Bonus: lire/écrire du Turtle (très proche de SPARQL).
- Ce n'est qu'une introduction ; pour en savoir plus :

<http://www.w3.org/TR/sparql11-overview/>

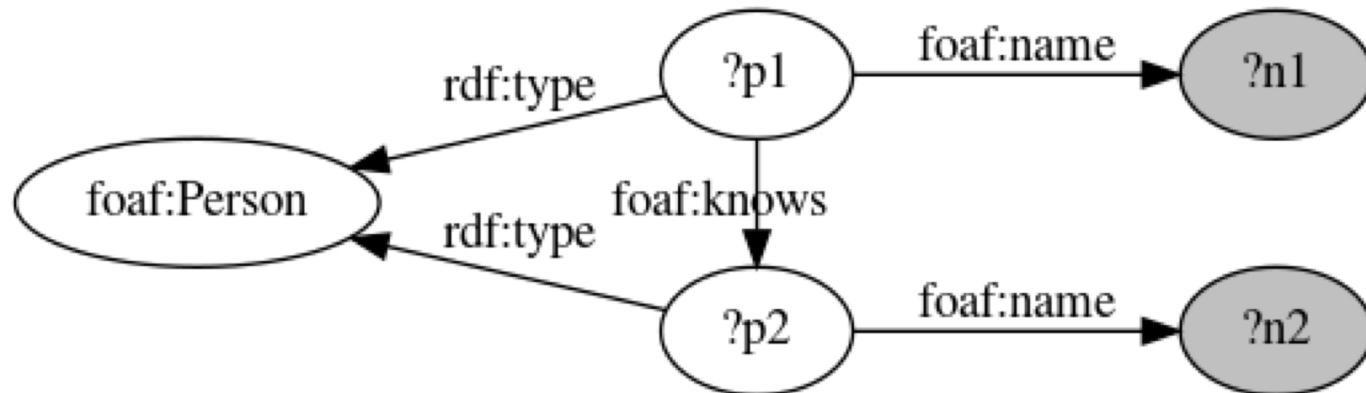
3. SparQL : requête simple

- Considérons les données suivantes



3. SparQL : requête simple

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?n1 ?n2
WHERE {
    ?p1 a foaf:Person;
        foaf:name ?n1;
        foaf:knows ?p2.
    ?p2 a foaf:Person;
        foaf:name ?n2.
}
```



3. SparQL : requête simple

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?n1 ?n2
WHERE {
    ?p1 a foaf:Person;
        foaf:name ?n1;
        foaf:knows ?p2.
    ?p2 a foaf:Person;
        foaf:name ?n2.
}
```

n1	n2
Alice	Bob
Carol	Dan
Dr Jekyll	Dan
Mr Hide	Dan

3. SparQL : description

Préfixes : les préfixes servent à abréger les IRIs.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
PREFIX : <http://example.com/>
```

Termes

IRI en extension (relatif ou absolu) :

```
<http://xmlns.org/foaf/0.1/Person>
<../other-file.rdf>
<#something>
<>
```

IRI abrégé :

```
foaf:Person
:something
```

3. SparQL : description

Littéraux :

```
"Hello"@en          # avec tag de langue
"123^^xsd:integer # typé

"Bonjour"           # equiv. "Bonjour"^^xsd:string
42                  # equiv. "42"^^xsd:integer
1.5                 # equiv. "1.5"^^xsd:decimal
314e-2              # equiv. "314e-2"^^xsd:double
true                # equiv. "true"^^xsd:boolean
```

Nœud muet :

```
_:toto
[]      # voir ci-après
```

Variables :

```
?x
$y
```

3. SparQL : triplets

- 3 termes (sujet, prédicat, objet) séparés par des espaces et suivis d'un point ":" :

```
?pl foaf:name "Pierre-Antoine Champin".
```

- Cas particulier : le mot clé "a" en position de prédicat est un raccourci pour <<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>> :

```
?pl a foaf:Person.
```

3. SparQL : factorisation

- On peut « factoriser » plusieurs triplets ayant le même sujet en séparant les couples <prédicat, objet> par un point-virgule ";" :

```
?p1 a foaf:Person;
    foaf:givenName "Pierre-Antoine";
    foaf:familyName "Champin".
```

- On peut « factoriser » plusieurs triplets ayant le même sujet et le même prédicat en séparant les objets par une virgule "," :

```
?p1 a foaf:phone <tel:+33-472-44-82-40>,
    <tel:+33-472-49-21-73>.
```

- On peut bien sûr combiner les deux types de factorisation.
- On n'est jamais obligé de factoriser, on peut aussi répéter les termes.

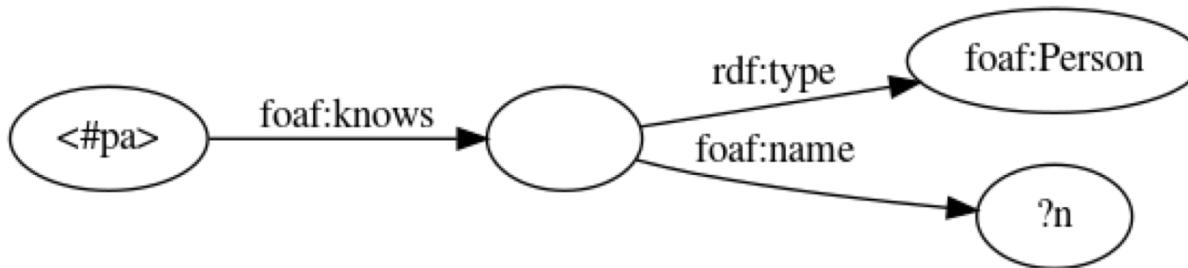
3. SparQL : nœuds muets

- Lorsqu'un nœud muet n'a qu'un seul arc entrant, au lieu de lui inventer un identifiant local :

```
<#pa> foaf:knows _:quelqun.  
_:quelqun a foaf:Person;  
faof:name ?n.
```

- on peut utiliser la notation [] :

```
<#pa> foaf:knows [  
    a foaf:Person;  
    faof:name ?n.  
].
```



3. SparQL : union

- Pour exprimer un "ou" logique entre plusieurs contraintes, on place chaque alternative entre accolades, séparées par le mot-clé UNION :

```
<#pa> foaf:knows ?pl.  
{ ?pl a foaf:Person; foaf:name ?n}  
UNION  
{ ?pl a schema:Person; schema:name ?n}
```

3. SparQL : sous-graphe optionnel

On peut accepter qu'une partie du graphe ne soit pas satisfaite :

```
?p1 a foaf:Person;
    foaf:name ?n.
OPTIONAL {?p1 foaf:img ?img}
OPTIONAL {?p1 foaf:phone ?tel}
```

Ou

```
?p1 a foaf:Person;
    foaf:name ?n.
OPTIONAL {?p1 foaf:img ?img. ?p1 foaf:phone ?tel}
```

Dans le résultat, les variables des clauses optionnelles peuvent donc ne recevoir aucune valeur (null).

3. SparQL : filtres

On peut ajouter des contraintes sur les valeurs des résultats, avec la clause FILTER.

```
?p foaf:age ?a.  
FILTER (?a >=18)
```

On peut combiner des conditions avec les opérateurs logiques « et » (`&&`), « ou » (`||`) et « non » (`!`).

```
FILTER (?a >=18 && a<30)
```

3. SparQL : filtres / opérateurs et fonctions

- comparaisons : =, !=, <, >, <=, >=
- opérateurs arithmétiques : +, -, *, /
- nature d'un nœud : isIRI, isBLANK, isLITERAL, isNUMERIC
- vérifier qu'une variable (utilisée avec OPTIONAL) a bien une valeur : Bound
- recherche de texte : REGEX(<variable>, <texte>)

Pour plus d'information, consultez la [documentation de SPARQL](#).

3. SparQL : requête SELECT

Similaire au SELECT de SQL :

projection sur un sous-ensemble des variables du graphe

Résultat : tableau

une colonne par variable sélectionnée
une ligne par résultat

Structure :

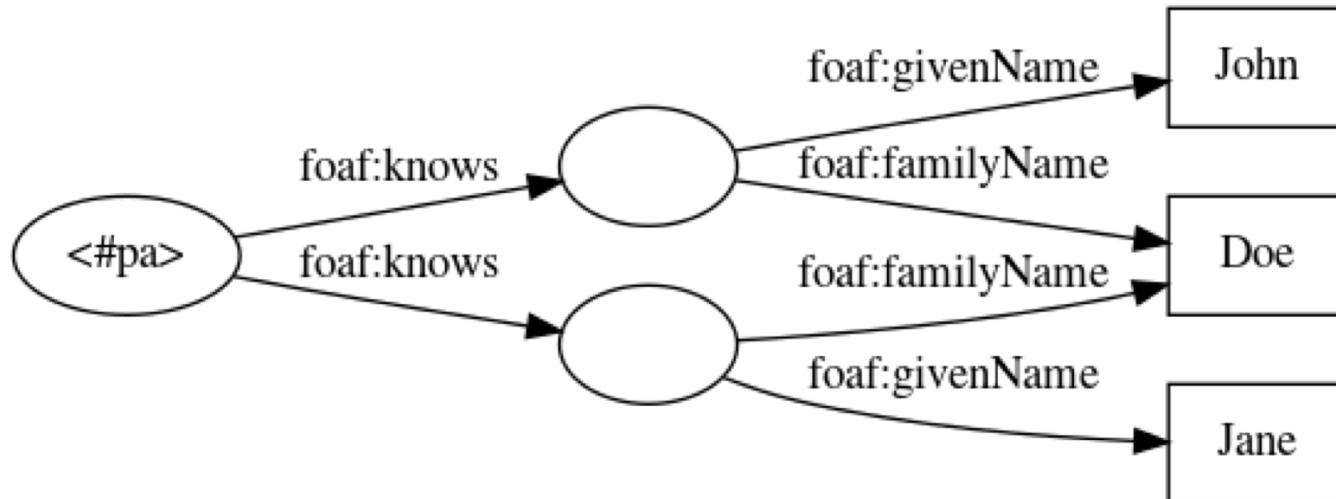
SELECT <variables/expression> WHERE { <graphe> }

Les résultats du SELECT peuvent être des expressions complexes, calculées à partir des variables de la clause WHERE.

```
SELECT ?p (concat(?gn, " ", ?fn) as ?name)
WHERE {
    ?p foaf:givenName ?gn;
        foaf:familyName ?fn.
}
```

3. SparQL : Distinct

```
SELECT DISTINCT ?sn
WHERE {
  <#pa> foaf:knows ?p.
  ?p foaf:familyName ?sn.
}
```



Sans le DISTINCT, la requête renverra deux fois le résultat sn="Doe".

3. SparQL : LIMIT et OFFSET

Pour obtenir les 10 premiers résultats :

```
SELECT ?p
WHERE {
  <#pa> foaf:knows ?p.
} LIMIT 10
```

Pour obtenir les 5 résultats suivants :

```
SELECT ?p
WHERE {
  <#pa> foaf:knows ?p.
}
LIMIT 10
OFFSET 5
```

3. SparQL : ORDER BY

```
SELECT ?p ?n
WHERE {
  <#pa> foaf:knows [
    foaf:givenName ?p ;
    foaf:familyName ?n ]
}
ORDER BY ?n ?p
```

On peut aussi trier par ordre descendant :

```
SELECT ?name ?age
WHERE {
  <#pa> foaf:knows [
    foaf:name ?name ;
    foaf:age ?age ]
}
ORDER BY DESC(?age)
LIMIT 1
```

3. SparQL : GROUP BY

Sert à *aggréger* certaines valeurs avec l'une des fonctions d'aggrégations : Count, Sum, Avg, Min, Max, GroupConcat et Sample.

```
SELECT ?p1 (count(?p2) as ?cp2)
WHERE {
    ?p1 foaf:knows ?p2
}
GROUP BY ?p1
```

On peut combiner GROUP BY avec ORDER BY et LIMIT (attention à l'ordre) :

```
SELECT ?p1 (count(?p2) as ?cp2)
WHERE {
    ?p1 foaf:knows ?p2
}
GROUP BY ?p1
ORDER BY DESC(?cp2) LIMIT 3
```

3. SparQL : sous-requête

Il est possible d'inclure, dans une clause WHERE, une requête SELECT entre accolades.

Par exemple, la requête suivante donne, pour chaque personne, son écart par rapport à l'âge moyen.

```
SELECT ?p ((?age - ?ageMoyen) as ?ecart)
WHERE {
    ?p a foaf:Person ;
        foaf:age ?age.
{
    SELECT (avg(?age2) as ?ageMoyen) {
        ?p2 a foaf:Person ;
            foaf:age ?age2.
    }
}
}
```

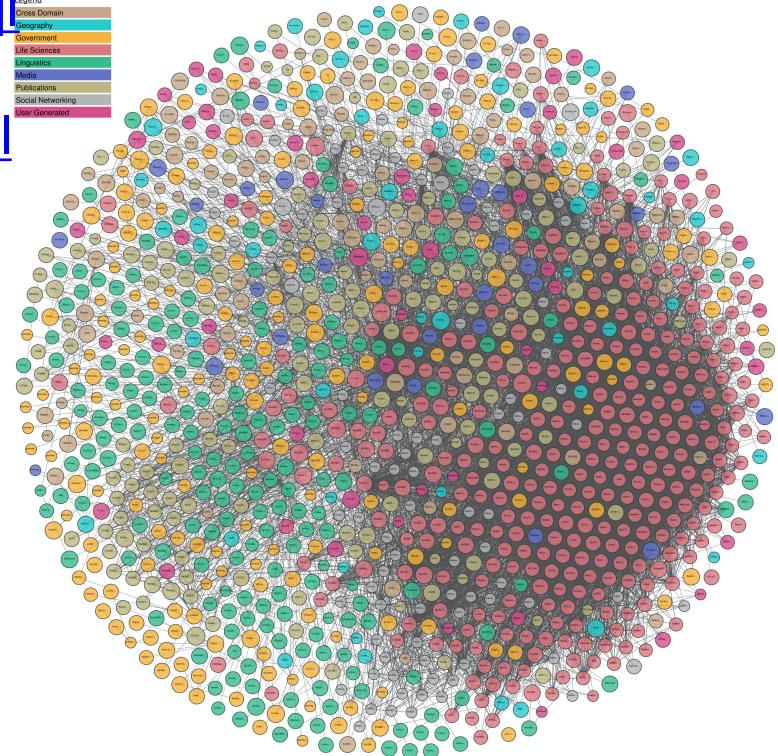
3. SparQL : points d'accès

Point d'accès :

- DBpedia : <http://dbpedia.org/sparql>
- Nobel prizes : <http://data.nobelprize.org/sparql>
- Muziekweb :
<https://api.data.muziekweb.nl/datasets/MuziekwebOrganization/Muziekweb/services/Muziekweb/sparql>
- LOV : *list of vocabularies*
<https://lov.linkeddata.es/dataset/lov/sparql>

Liste des points d'accès :

- The Linked Open Data Cloud :
<https://lod-cloud.net>



3. SparQL : Exploration des ressources

Sur *Musiekweb*, la requête suivante

```
SELECT ?type
WHERE { ?o a ?type }
GROUP BY ?type
ORDER BY DESC(count(?o))
LIMIT 30
```

génère

	type
1	< https://data.muziekweb.nl/vocab/OrderInformation >
2	< http://schema.org/MusicAlbum >
3	< https://data.muziekweb.nl/vocab/Album >
4	< http://schema.org/MusicGroup >
5	< https://data.muziekweb.nl/vocab/PopularAlbum >
6	< https://data.muziekweb.nl/vocab/PopularPerformer >

3. SparQL : exploration des propriétés

Sur *Musiekweb*, la requête suivante

```
prefix vocab: <https://data.muziekweb.nl/vocab/>
SELECT DISTINCT ?prop
WHERE {
    ?o a vocab:Performer;
       ?prop ?val.
}
LIMIT 30 OFFSET 4
```

génère

	prop
1	vocab:album
2	vocab:alias
3	vocab:beginYear
4	vocab:contemporary
5	vocab:endYear

3. SparQL : Trouver un vocabulaire

- Faire une démo de *Linked open vocabularies*
<http://lov.okfn.org/>
- Requête à partir de LOV (*list of vocabularies*)

```
PREFIX vann:<http://purl.org/vocab/vann/>
PREFIX voaf:<http://purl.org/vocommons/voaf#>

### Vocabularies contained in LOV and their prefix
SELECT DISTINCT ?vocabPrefix ?vocabURI {
  GRAPH <https://lov.linkeddata.es/dataset/lov>{
    ?vocabURI a voaf:Vocabulary.
    ?vocabURI vann:preferredNamespacePrefix ?vocabPrefix.
  }
}
ORDER BY ?vocabPrefix
```

3. SparQL : Trouver un vocabulaire

	vocabPrefix	vocabURI
1	SAN	< http://www.irit.fr/recherches/MELODI/ontologies/SAN >
2	a-loc	< https://w3id.org/arco/ontology/location >
3	acco	< http://purl.org/acco/ns >
4	acl	< http://www.w3.org/ns/auth/acl >
5	acm	< http://www.rkbexplorer.com/ontologies/acm >

Il y a 723 résultats!