RDFLib Workshop

Introduction et utilisation dans une application de Web Sémantique

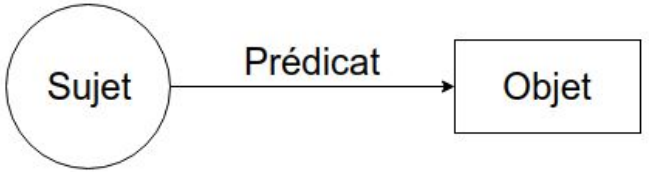
Lien vers les labs : https://github.com/MoohShadox/Web_Semantique_Workshop

Histoire et Définition des Graphes RDF

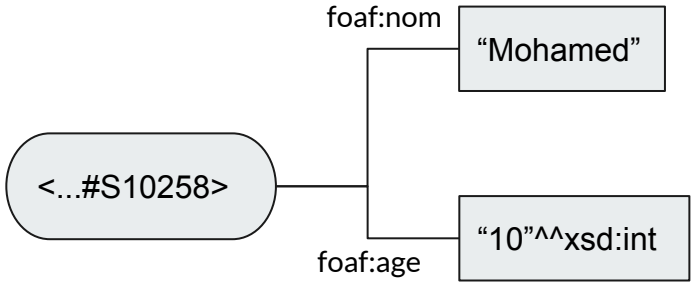
Historique:
1990 : Invention du Web
1999 : Schémas RDF
2004 : Annotations Sémantiques (RDFS) et Ontologies (OWL)

Un graphe RDF (Resource Description Framework) est un ensemble de triplets où chaque triplet à la forme <Sujet, Predicat, Objet>.

Forme générale :



Exemple :



Etape 01 : Base d’individus RDF

Sujet	Prédicat	Objet	Exemple
Ressource	ObjectProperty	Ressource	<...#Bart> monOnt:Frere_De <...#Lisa>

Etape 01 : Base d'individus RDF

Sujet	Prédicat	Objet	Exemple
Ressource	ObjectProperty	Ressource	<...#Bart> monOnt:Frere_De <...#Lisa>
Ressource	DataProperty	Litéral	<...#Bart> monOnt:Surnom “Bartman”
Ressource	DataProperty	Litéral(Typé)	<...#Bart> monOnt:Surnom “Bartman”^^xsd:string

Etape 01 : Base d’individus RDF

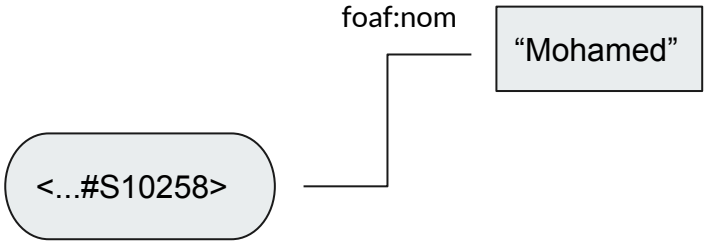
<...#S10258>

```
rdf_url = "https://Data#"
def ajouter_patient(nom,prenom,nss):
    graphe = rdflib.Graph()
    uri_patient = rdf_url+nss
    patient_r = rdflib.URIRef(uri_patient)
```

Etape 01 : Base d’individus RDF

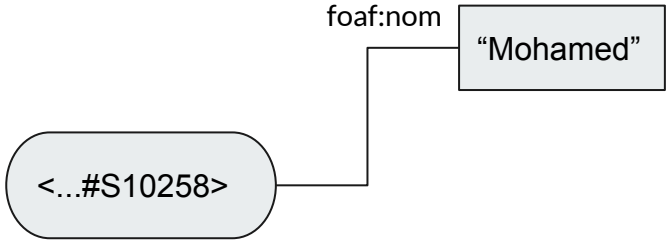

```
myont_ns = "http://www.co-ode.org/ontologies/ont.owl#"
foaf_ns = "http://xmlns.com/foaf/0.1/"
rdf_url = "https://Data#"
def ajouter_patient(nom,prenom,nss):
    graphe = rdflib.Graph()
    uri_patient = rdf_url+nss
    patient_r = rdflib.URIRef(uri_patient)
    name_relation = rdflib.URIRef(foaf_ns+"givenName")
    familyname_relation = rdflib.URIRef(foaf_ns+"nom")
```

Etape 01 : Base d’individus RDF



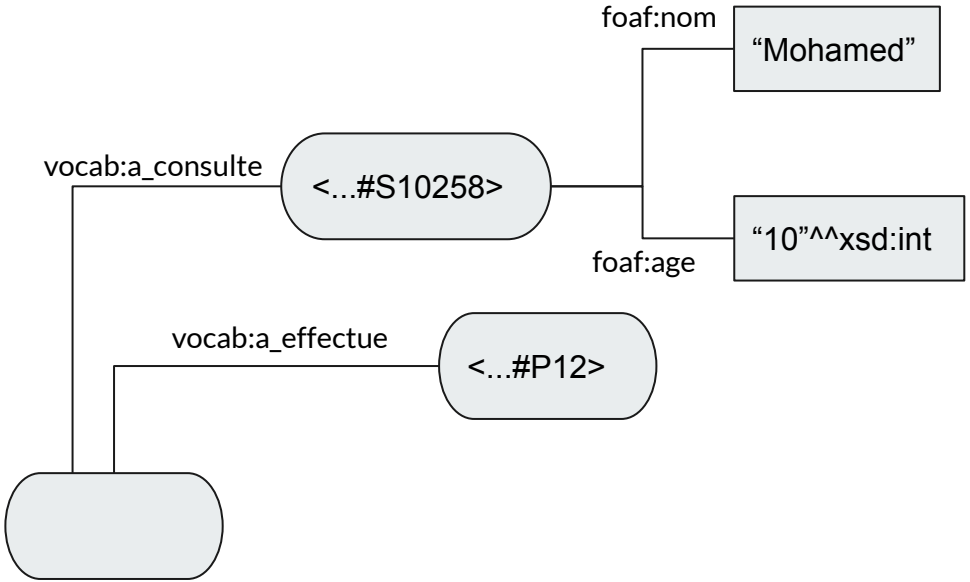

```
myont_ns = "http://www.co-ode.org/ontologies/ont.owl#"
foaf_ns = "http://xmlns.com/foaf/0.1/"
rdf_url = "https://Data#"
def ajouter_patient(nom,prenom,nss):
    graphe = rdflib.Graph()
    uri_patient = rdf_url+nss
    patient_r = rdflib.URIRef(uri_patient)
    name_relation = rdflib.URIRef(foaf_ns+"givenName")
    familyname_relation = rdflib.URIRef(foaf_ns+"nom")
    name_literal = rdflib.Literal(nom)
```

Etape 01 : Base d'individus RDF

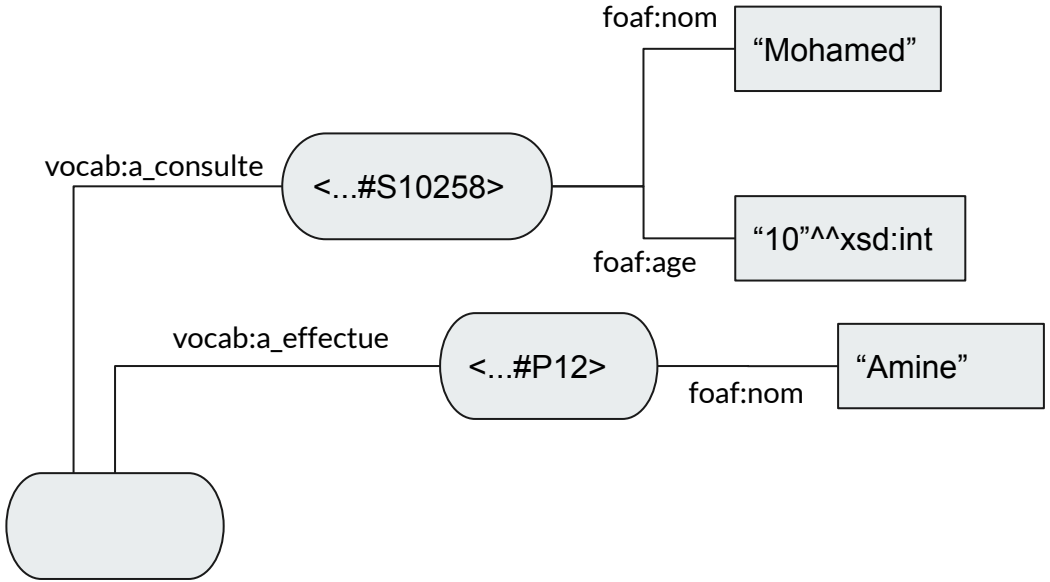



```python
foaf_ns = "http://xmlns.com/foaf/0.1/"
rdf_url = "https://Data#"
def ajouter_patient(nom,prenom,nss):
    graphe = rdflib.Graph()
    uri_patient = rdf_url+nss
    patient_r = rdflib.URIRef(uri_patient)
    name_relation = rdflib.URIRef(foaf_ns+"givenName")
    familyname_relation = rdflib.URIRef(foaf_ns+"familyName")
    name_literal = rdflib.Literal(nom)
    familyname_literal = rdflib.Literal(prenom)
    graphe.add((patient_r,name_relation,name_literal))
    return graphe
```

Etape 01 : Base d’individus RDF



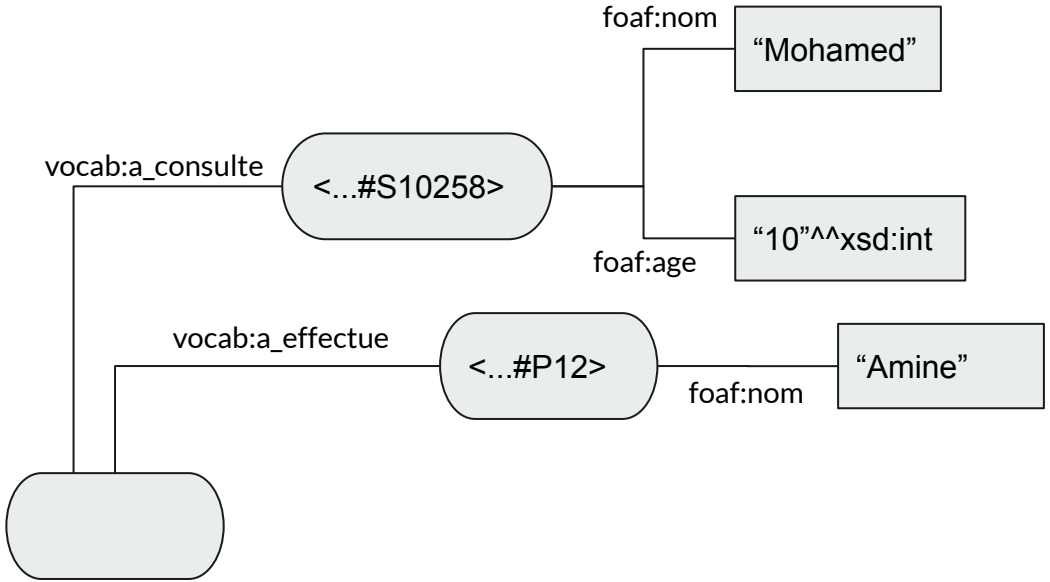
Etape 01 : Base d'individus RDF



Problème :

Comment différencier un patient d'un praticien ?

Etape 01 : Base d'individus RDF



Problème :

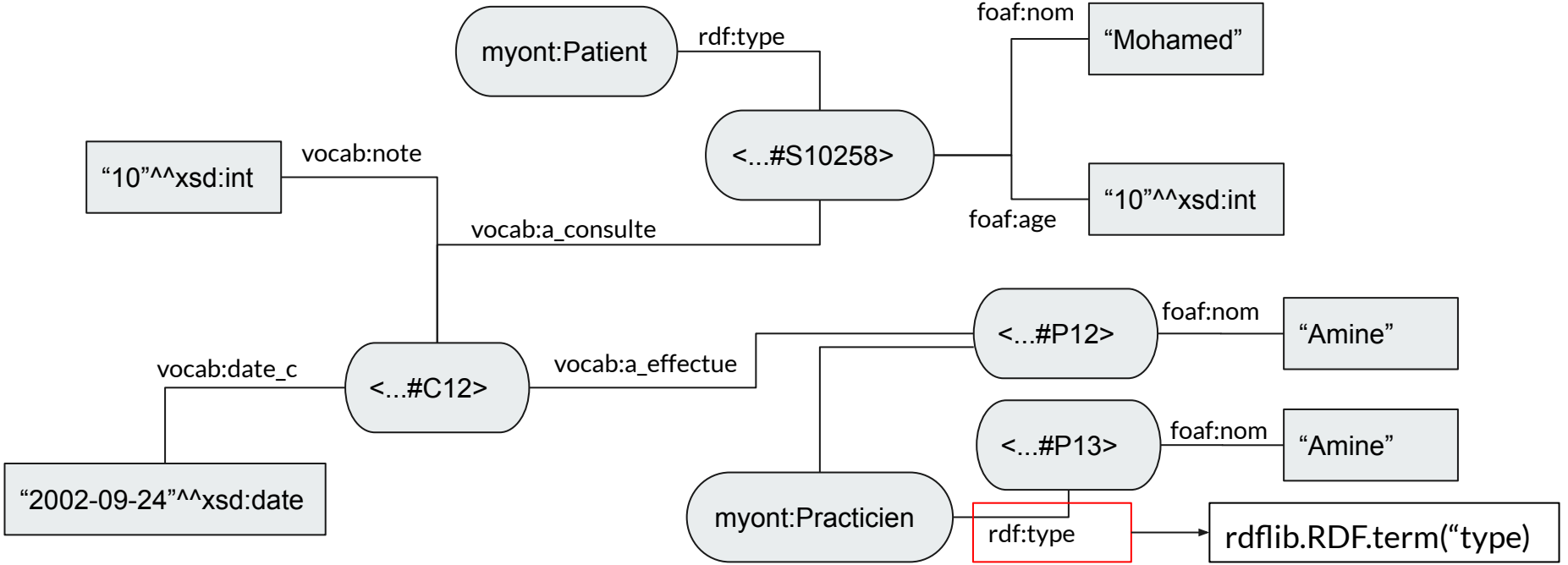
Comment différencier un patient d'un praticien ?

⇒ En utilisant des annotations sémantiques

Etape 02 : Annotations sémantiques RDF

Sujet	Prédicat	Objet	Exemple
Ressource	ObjectProperty	Ressource	<...#Bart> monOnt:Frere_De <...#Lisa>
Ressource	DataProperty	Litéral	<...#Bart> monOnt:Surnom “Bartman”
Ressource	DataProperty	Litéral(Typé)	<...#Bart> monOnt:Surnom “Bartman”^^xsd:string
Sujet	rdf:type	Concept	<...#Simpson> rdf:type monOnt:Famile

Etape 02 : Annotations sémantiques



Etape 03 : Annotations sémantiques RDFS

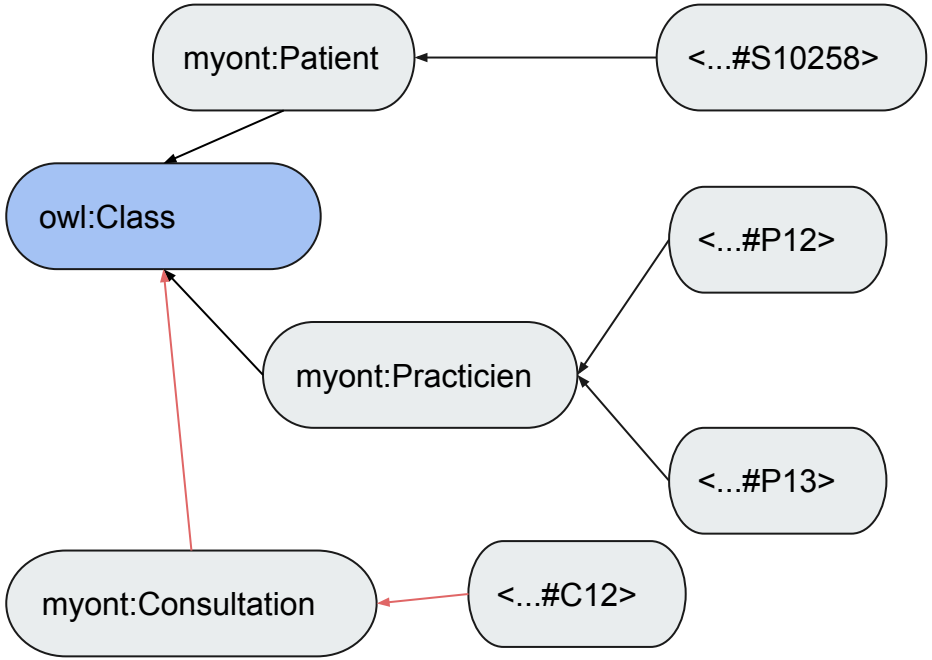
Sujet	Prédicat	Objet	Exemple
Ressource	ObjectProperty	Ressource	<...#Bart> monOnt:Frere_De <...#Lisa>
Ressource	DataProperty	Littéral	<...#Bart> monOnt:Surnom “Bartman”
Ressource	DataProperty	Littéral(Typé)	<...#Bart> monOnt:Surnom “Bartman”^^xsd:string
Sujet	rdf:type	Concept	<...#Simpson> rdf:type monOnt:Famile
Ressource	rdfs:label	Literal	<...#Bart> rdfs:label “Personnage des Simpsons”
Concept	rdfs:subClassOf	Concept	monOnt:Chat rdfs:subClassOf monOnt:Animal
ObjectProperty	rdfs:domain	Concept	monOnt:Frere_De rdfs:domain monOnt:Humains

Etape 04 : Annotations sémantiques OWL

Sujet	Prédicat	Objet	Exemple
Concept	rdf:type	owl:Class	monOnt:Etudiant rdf:type owl:Class
Relations	rdf:type	owl:ObjectProperty	monOnt:Soeur_de rdf:type owl:ObjectProperty
Relations	rdf:type	owl:DataProperty	monOnt:age rdf:type owl:DataProperty

Sujet	Prédicat	Objet	Exemple
Ressource	ObjectProperty	Ressource	<...#Bart> monOnt:Frere_De <...#Lisa>
Ressource	DataProperty	Littéral	<...#Bart> monOnt:Surnom “Bartman”
Ressource	DataProperty	Littéral(Typé)	<...#Bart> monOnt:Surnom “Bartman”^^xsd:string
Sujet	rdf:type	Concept	<...#Simpson> rdf:type monOnt:Famile
Ressource	rdfs:label	Literal	<...#Bart> rdfs:label “Personnage des Simpsons”
Concept	rdfs:subClassOf	Concept	monOnt:Chat rdfs:subClassOf monOnt:Animal
ObjectProperty	rdfs:domain	Concept	monOnt:Frere_De rdfs:domain monOnt:Humains
Concept	rdf:type	owl:Class	monOnt:Etudiant rdf:type owl:Class
Relations	rdf:type	owl:ObjectProperty	monOnt:Soeur_de rdf:type owl:ObjectProperty
Relations	rdf:type	owl:DataProperty	monOnt:age rdf:type owl:DataProperty

Etape 03 : Annotations sémantiques OWL




```
consultation_t = URIRef(myont_ns+"Consultation")
consultation_r = rdflib.URIRef(rdf_url+"C"+id_consultation)
graphe.add((consultation_t,rdflib.RDF.term("type"),rdflib.OWL.term("Class"))
)
graphe.add((consultation_r,rdflib.RDF.term("type"),concept_practicien))
```

Représenter les ressources

- Créer des ressources
- Décrire les propriété des ressources
- Décrire les relations entre les ressources

Annoter les ressources

- Spécifier les types des ressources
- Associer aux ressources des label, des commentaires, une langue etc..
- Hiérarchiser les types et les relations

Représenter les connaissances

- Utiliser le concept de Class pour définir des classes
- Définir des relations
- Définir des contraintes sur les classes et sur les relations
- Définir des relations d’ équivalences et de Subsomption

Récupérer les données d'une base RDF

- On peut utiliser un filtre pour conserver une partie seulement des triplets.
- On peut utiliser le langage SPARQL pour construire des requêtes plus complexes.

```
graphe.triples((None,rdflib.RDF.term("type"),rdflib.URIRef('...#Acteur')))
```

Récupérer les données d'une base RDF

- On peut utiliser un filtre pour conserver une partie seulement des triplets.
- On peut utiliser le langage SPARQL pour construire des requêtes plus complexes.

```
graphe.triples((None,rdflib.RDF.term("type"),rdflib.URIRef('...#Acteur')))
```

```
nom = "Daniel"
prenom = "Antoine"
requete = """
prefix ns1: <http://xmlns.com/foaf/0.1/>
prefix ns2: <http://www.co-ode.org/ontologies/ont.owl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?s
WHERE{
?s rdf:type ns2:Acteur .
?s ns1:givenName "%nom" .
?s ns1:familyName "%prenom"
}
""".replace("%nom",nom).replace("%prenom",prenom)
print(requete)
```

SPARQL ?

Présentation :

SPARQL est le langage développé par le W3C pour interroger des descriptions RDF. La documentation complète du langage se trouve à l'adresse suivante : http://www.w3.org/TR/rdfsparqlquery/.

Syntaxe courante :

La structure d'une requête SPARQL est très similaire à celle employée dans le langage SQL:

```
SELECT ?v1 ?v2 ... ?vn
FROM <description.rdf>
WHERE {
(sujet1 | vi) (predicat1 | vj) (objet1 | vk)
.
...
(sujetx | va) (predicaty | vb) (objetz | vc) .
}
```

SPARQL ?

Remarques

Il existe également d'autres éléments dans le langage SPARQL qui permettent de spécifier des préfixes (PREFIX), des conditions (FILTER), des disjonctions (UNION), des filtres sur la production des résultats (LIMIT et OFFSET).

Il est possible de limiter le nombre de réponses à afficher à l'utilisateur grâce aux termes LIMIT p et OFFSET q (avec p>0 et q>0) situés après la clause WHERE. Le terme LIMIT permet de spécifier le nombre maximum p de résultats à afficher, tandis que le terme OFFSET débute l'affichage des réponses au qième résultat.

Le terme FILTER dans une clause WHERE permet de contraindre certaines variables d'une requête SPARQL.

Inconvénients liés à la manipulation d'ontologies via RDFLib

RDFLib sert principalement à créer et à maintenir une base de faits, pour faire de l'ingénierie de connaissances en dépit du fait qu'on puisse utiliser le vocabulaire OWL pour faire des annotations sémantiques ce genre de manipulation présente des inconvénients :

- Impossibilité d'exploiter le python orienté objet qui propose déja une façon efficace de construire des classes et des entités.
- Difficulté de représentation de certaines annotations complexes tel qu'une définition par restriction.

La solution ? utiliser owlready une librairie plus abstraite permettant de manipuler les ontologies en tant que ontologies et non en tant que fichiers rdf

Que permet de faire owlready ?

- Décrire une ontologie en même temps que les classes d'objets instanciable.
- Instancier les individus aussi facilement que les objets
- Créer facilement les relations
- Enregistrer et charger les ontologie en utilisant où pas un stockage en mémoire permanente
- Extrêmement bien documenté
- Fournit un support pour manipuler le raisonneur

Que permet de faire owlready ?

- Décrire une ontologie en même temps que les classes d'objets instanciable.
- Instancier les individus aussi facilement que les objets
- Créer facilement les relations
- Enregistrer et charger les ontologie en utilisant où pas un stockage en mémoire permanente
- Extrêmement bien documenté
- Fournit un support pour manipuler le raisonneur

```
with onto:
    class Acteur(Thing):
        def presente_toi(self):
            print("Je suis acteur ")
    class Contenu(Thing):
        pass
    class type_contenu(Contenu >> str):
        pass
    class givenName(Thing >> str):
        pass
    class familyName(Thing >> str):
        pass
    class description(Contenu >> str):
        pass
    class title(Contenu >> str):
        pass
    class releaseYear(Contenu >> int):
        pass
    class playedIn(ObjectProperty):
        domain    = [Acteur]
        range     = [Contenu]
```

Utiliser le raisonneur

Pourquoi faire ?
En ingénierie de connaissance un raisonneur sert principalement à deux choses :

- Vérifier la consistance d'une base de connaissances ce qui est très utile pour les applications qui font de l'apprentissage car ça permet de leur fournir :
  - Une résilience aux défaillances provenant de la source des données si le modèle reçoit en continue de nouvelles données.
  - Une structure plus adaptables aux changements de modèle d'apprentissage
- Faire de l'inférence automatique

Utiliser le raisonneur

Comment le faire ?
Pour faire de l'inférence automatique nous pouvons utiliser le raisonneur qui fournit un support relativement efficace pour les relations de subsomption et d'équivalences entre les conceptions.

Attention cependant à la complexité !

```
with onto:
    sync_reasoner()
```

Utiliser le raisonneur

Une alternative à simplement utiliser le raisonneur
Une autre façon de faire de l'inférence automatique ciblée et personnalisée est d'utiliser une règle d'inférence.
Les règles d'inférence se formalisent en utilisant le langage SWRL (Semantic Web Rule Language).

```
rule = Imp()
rule.set_as_rule("""film(?f), actor(?a1), actor(?a2), played_in(?a1,?f), played_in(?a2,?f) -> knows(?a1,?a2),knows(?a2,a1)""")
rule.set_as_rule("""Drug(?d), price(?d, ?p), number_of_tablets(?d, ?n), divide(?r, ?p, ?n) -> price_per_tablet(?d, ?r)""")
```

Utiliser le web sémantique

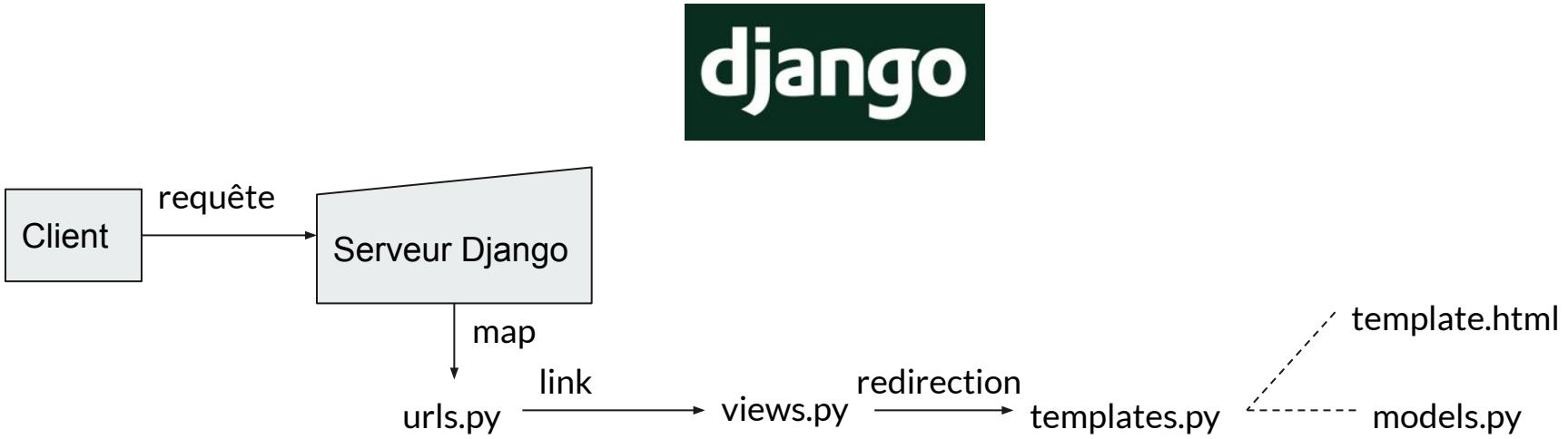
Utilité 01 : Utiliser du contenu issue du Web Sémantique

Les données du web sémantique sont traditionnellement open source ce qui en fait une source de données utiles gratuite et accessibles par requête posts en utilisant un “endpoint” .
(cf Notebook : Lab 03 - Using Existing Ontology où on récupère des données par requête SPARQL depuis DBPédia) .

Utiliser le Web Sémantique

<u>Utilité 02 :</u> Développer un site web 3.0

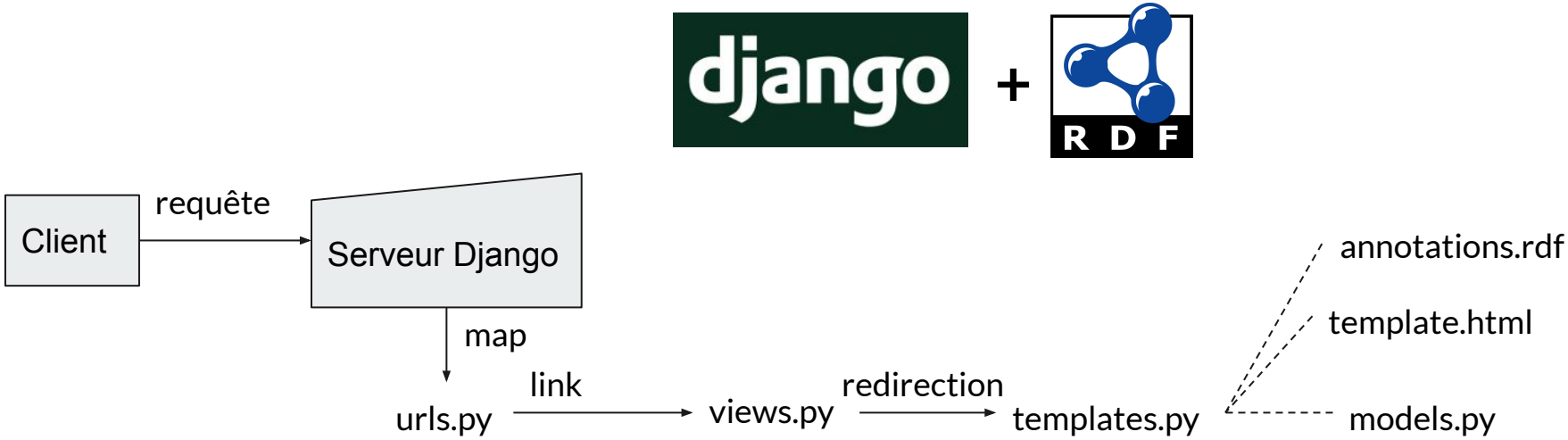
L'utilisation du web sémantique la plus courante reste dans le cadre du développement comme nous l'expliquerons rapidement dans ce qui suit :



Utiliser le Web Sémantique

Utilité 02 : Développer un site web 3.0

L'utilisation du web sémantique la plus courante reste dans le cadre du développement comme nous l'expliquerons rapidement dans ce qui suit :



Merci pour votre attention !