Semantic Web with Python -RDFLib

Ana-Maria Ghiran

Agenda

- Introducere în Web-ul Semantic
  - Principiile ce stau la baza Web-ului Semantic, Linked Data
  - Modelul RDF
  - SPARQL
- RDFLIB
  - Creare date RDF prin RDFLib
  - Manipulare date
  - Extragere date

Introducere în Web-ul Semantic

- Evoluția web-ului spre Web Semantic, Web Pragmatic ?
- Care sunt așteptările și posibilitățile?
  - Motoare de căutare semantice
  - Site-uri web cu meta-date (cunoștințe)
- Pentru ca motorul de căutare să permită căutări semantice trebuie să „înțeleagă” un concept adică să dețină cunoștințe despre acesta. Cunoștințele iau forma afirmațiilor (faptelor) și a regulilor.
- Site-urile web pentru a oferi cunoștințe relevante trebuie să descrie detaliat (fapte+reguli) ce concepte folosesc/furnizează.

Principiile după care se ghidează Web-ul Semantic:

- Principiul identității – Everything can be identified by URI’s
- Principiul numirii multiple – non unique naming
- Principiul validității – no need for absolute truth
- Principiul AAA - Anyone can Say Anything about Anything
- Principiul tolerării informației parțiale – Open World

Linked Data - primul pas spre Web-ul Semantic

- Linked Data – permite ca datele să fie publicate, conectate și descoperite. La fel cum hyperlegăturile sunt folosite pentru a conecta documentele în web-ul clasic, Linked Data va permite specificarea de legături între concepte aparținând diverselor surse: va interconecta aceste surse într-un spațiu global de date.
- Interconectarea datelor distribuite în Web solicită adoptarea unui mecanism standard atât pentru specificarea existenței și sensului legăturii dintre concepte. Acest mecanism este oferit de RDF – Resource Description Framework.
- RDF permite o modalitate de a descrie lucrurile (oricare ar fi acestea: oameni, locații, concepte abstrtacte) dar și modul în care acestea sunt în legătură cu altele.

Modelul RDF - Resource Description Framework

- RDF e un model, un set de reguli pentru structurarea cunoştinţelor
- Afirmaţiile în RDF iau forma unor propoziţii cu 3 părţi: subiect, predicat, obiect (tripleţi);
- Fiecare din cele 3 părţi trebuie exprimată prin identificator universal (URI); acesta va exprima *orice concept*
  - fie un obiect X din univers, http://www.paginamea.ro#anamaria , http://www.paginamea.ro#labus
  - fie un concept clasă (o mulțime a obiectelor X), http://dbpedia.org/resource/Dog
  - fie o relație dintre concepte sau o proprietate pe care o au conceptele. http://www.paginamea.ro#este , http://xmlns.com/foaf/0.1/knows
- Acești identificatori pot fi:
  - Standardizați – creați de diferite organizații și toți cei care vor, pot să îl folosească când fac referire la un anumit concept în afirmațiile lor
  - Personalizați – pot fi creați de oricine, pot fi creați pornind de la adresele URL alocate acestora
  - Locali – pentru a referi concepte cu identitate locală care nu prezintă interes decât pentru conectarea între afirmații (anonime)
- Ca și recomandare se vor folosi identificatori diferiți pentru a face referire la conceptul desemnat din lumea reală și pagima web care îl va descrie (ex: http://dbpedia.org/resource/Dog reprezintă conceptul câine - la introducerea acestui URI într-un browser web vom fi redirecționați către http://dbpedia.org/page/Dog)
- Modalităţi de reprezentare:
  - grafică (sub forma unui graf cu noduri pentru conceptele subiect , obiect și arce pentru conceptele proprietăți - predicatele)
  - și serializată (cu diverse sintaxe posibile pentru stocarea cunoștințelor în fișiere text și care pot fi convertite de parsere în modele obiectuale)

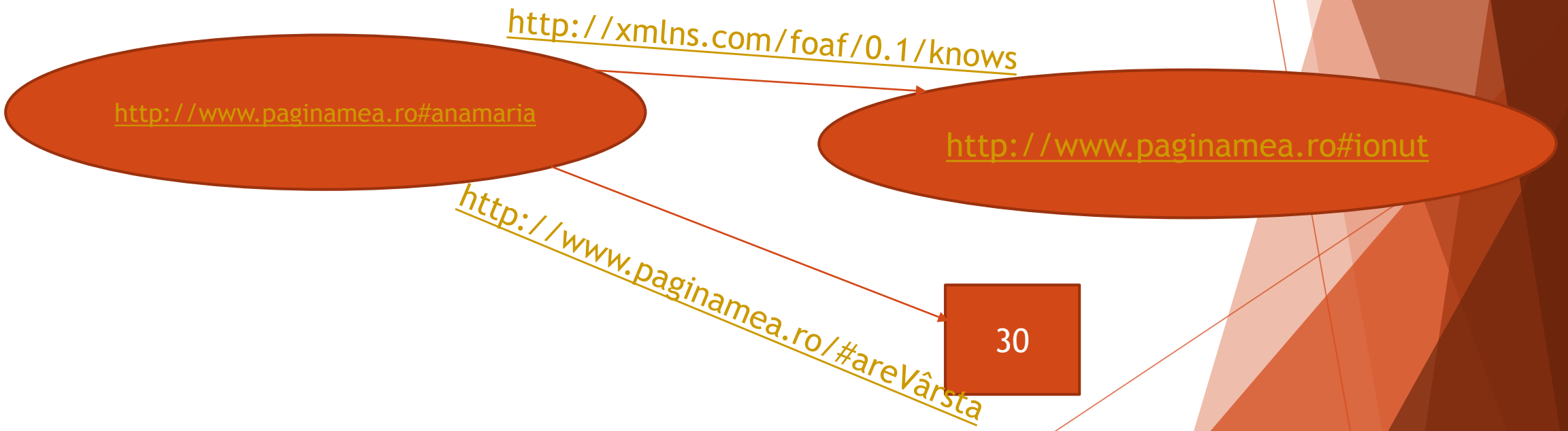
Exemple afirmații in RDF

- Avem afirmațiile:

Anamaria cunoaștePe Ionut.

Anamaria areVârsta 30.

Reprezentarea grafică



Un neajuns al reprezentărilor grafice: dificultatea exprimării situațiilor în care o proprietate devine și subiect al altor afirmații.

Exemple afirmații in RDF

- Aceleași afirmații într-o **reprezentare serializată:**

În N-Triples:

<http://www.paginamea.ro#anamaria> <http://xmlns.com/foaf/0.1/knows>
<http://www.paginamea.ro#ionut> .

<http://www.paginamea.ro#anamaria> <http://www.paginamea.ro/#areVârsta>
”30”^^<http://www.w3.org/XMLSchema#int> .

<http://paginamea.ro#anamaria> <http://paginamea.ro#locuiesteIn> <http://paginamea.ro#Cluj> .

<http://paginamea.ro#Cluj> <http://paginamea.ro#areNumele> „Cluj-Napoca"@ro .

Exemple afirmații in RDF

În Turtle:

@prefix : <http://paginamea.ro#>.

@prefix foaf:<http://xmlns.com/foaf/0.1>

:anamaria foaf:knows :ionut ;
:areVarsta 30 .
:anamaria :areAdresa _:x.
_:x :areLocalitate :Cluj; areStrada :Teleorman.
:anamaria :locuiesteIn :Cluj.
:Cluj :areNumele „Cluj-Napoca”@ro .

O alternativa la varianta cu prefixe este folosirea unor URI relative la un URI de baza

@base <http://paginamea.ro/>.

<anamaria> <cunoastePe> <ionut>.

Exemple afirmații in RDF

RDF-XML Reprezentarea serializată ce oferă interoperabilitate sintactică și semantică

Anamaria cunoaștePe Ionut.
Anamaria areVârsta 30.

```
<Description about="http://paginamea.ro#anamaria">                    (subiect)
    <http://paginamea.ro#cunoastePe>                (proprietate-relație)
        <Description about ="http://paginamea.ro#ionut" />          (obiect)
    </http://paginamea.ro#cunoastePe>
</Description>
<Description about="http://paginamea.ro#anamaria">                (subiect)
    <http://expl.ro#areVarsta>                    (proprietate-atribut)
        30                              (obiect-valoare)
    </http://paginamea.ro#areVarsta>
</Description>
```

Interogarea bazelor de cunoştinţe cu limbajul SPARQL

SPARQL – Simple Protocol and RDF Query Language

- Un limbaj de interogare a cunoștințelor bazat pe șabloane.

SELECT <variabile>

WHERE <sablon sintaxa RDF>

Ex:

SELECT * WHERE {?s ?p ?o}

-> selecteaza toate afirmaţiile din tripleţii existenţi în baza de cunoştinţe

SELECT ?s

WHERE {?s ?p ?o}

-> selecteaza doar subiecţii din tripleţii existenţi în baza de cunoştinţe

- obs.: nu este casesensitive, nu trebuie neaparat puse pe randuri separate insa e recomandat

Exemple interogari SPARQL

```
SELECT   ?prieten
    WHERE {:anamaria :cunoastePe ?prieten .}
```

sau

```
SELECT   ?locatie
    WHERE {?x :cunoastePe :ionut.
                ?x :locuiesteIn ?locatie.}
```

SPARQL returnează un rezultat pentru FIECARE triplet ce corespunde şablonului! Dacă dorim să nu se repete rezultatele, adăugăm clauza DISTINCT:

SELECT **DISTINCT** ?s WHERE {?s ?p ?o}

Rezultatele pot fi ordonate:

SELECT * WHERE {?s ?p ?o} **ORDER BY ?s**

Alte exemple SPARQL

```
SELECT   ?prieten
    WHERE {:anamaria :cunoastePe ?prieten .
                FILTER  regex(?prieten, '^i')}
```

Obs: SPARQL nu permite ca după SELECT să apară altceva decât necunoscute: în rezultate vor aparea subiecții și predicatele dar fară obiect ceea ce duce la dificultate în interpretare

Solutie:

```
select  (:anamaria as ?x) ?y (:ionut as ?prieten)
where {?x ?y :ionut}
```

Alte exemple SPARQL

- Interogari ce returneaza expresii:

select (concat(?x," are varsta ",str(?varsta)) as ?afirmatie)

where { ?x :areVarsta ?varsta}

-> selecteaza afirmații în limbaj natural din componentele RDF, prin funcția de concatenare

- Același rezultat se poate obține și cu clauza BIND (care funcționează ca o atribuire a valorii unei expresii spre o variabilă):

select ?afirmatie

where { ?x :areVarsta ?varsta

bind(concat(?x," are varsta",str(?varsta)) as ?afirmatie) }

Python for Semantic Web

- Cum poate fi folosit Python pentru a crea cunoștințe sau pentru a folosi cunoștințe din Semantic Web?
- Librăria RDFLib https://github.com/RDFLib/rdflib
- Ofera:
  - posibilitatea de a crea RDF direct în Python;
  - posibilitatea de a converti RDF în diverse sintaxe și de a le stoca persistent, pe disc;
  - posibilitatea de a prelua RDF din surse externe (servicii, servere precum Sesame) și a-l procesa, modifica, inclusiv interoga direct în Python, fără a mai apela la protocoale bazate pe HTTP.

Creare date – termeni RDF

- **BNodes** - blank nodes, o resursa pentru care nu s-a acordat nici un identificator, resursa anonima

  ex: from rdflib import BNode()

  unnod = BNode() sau

  unnod=rdflib.Bnode()

- **URIRef** – identificatorul unei resurse RDF

  from rdflib import URIRef

  resursa=URIRef('http://paginamea.ro#ana')

- **Literal** – atribute valoare in RDF, pot fi insotiti de declararea unu itip de data sau o eticheta pentru limba

  a=rdflib.Literal

  sau from rdflib import Literal, XSD

  lit2015 = Literal('2015-01-01',datatype=XSD.date)

Creare date – spatii de nume RDF

- Rdflib permite lucrul cu mai multi identificatori URI in cadrul unui spatiu de nume – namespace

  ex: from rdflib import Namespace

  n=Namespace('http://paginamea.ro')

- Astfel putem scrie ca si *atribut*

  >>> n.Persoana

pentru a face referire la identificatorul URI http://paginamea.ro/Persoana

Creare date

- Folosind URIRef, Bnode, Literal se pot construi noduri care sa fie atasate grafului

```
from rdflib import URIRef, BNode, Literal

bob = URIRef("http://paginamea.ro/persoane/Bob")
linda = BNode()
areNume = URIRef("http://paginamea.ro/persoane/areNume")

nume = Literal('Bob')
varsta= Literal(24)
inaltime= Literal(76.5)
```

Adaugarea de triplete in graf:

Graph.add((s,p,o))

ex: g.add((bob,areNume, nume))

In mod similar se pot sterge triplete in graf:

Graph.remove((s,p,o))

Manipulare date RDF

- Putem interoga daca exista o anumita asertiune in graf (un triplete):

```
from rdflib import URIRef
from rdflib.namespace import RDF, FOAF
ana= URIRef("http://paginamea.ro#anamaria")
if ( ana, RDF.type, FOAF.Person ) in g:
   print „Acest graf stie ca ana este o persoana!„
```

- Legarea nu trebuie sa fie completa!!!

```
Ex: if (ana, None, None) in g:
   print „Graful contine asertiuni despre Ana!"
```

Preluare directă de date de pe Web

Pentru exemplificare vom extrage date de pe DBPedia

Pașii urmați:

1. Se creează un graf gol, care va constitui un fel de container pentru datele aduse de pe web.

```
import rdflib
g=rdflib.Graph()
```

2. Se încarcă date de pe web

```
g.parse("http://dbpedia.org/resource/Guido_van_Rossum")
```

Se pot adăuga și date care sunt stocate în fișiere locale

```
g.parse("fisier", format="turtle/nt/xml/")
```

3. Extragem date
   a) Brut

```
print g.serialize(format="turtle/nt/xml")
```

   b) Selectiv

```
for s, p, o in g:
        if s==rdflib.URIRef("http://www.paginamea.ro#anamaria"):
                print s, p, o
```

Preluare directă de date de pe Web(2)

- Putem folosi o metodă proprie grafului numită *suject_object* care returnează tuple sub forma de (subiect, obiect) pentru proprietatea transmisă ca și argument :

```
for x in g.subject_object(rdflib.URIRef(”http://dbpedia.org/ontology/birthDate)):
    print x
```

Tuplul returnat contine valori pentru subiect si obiect sub forma unor obiecte URIRef sau obiecte Literal, dar pot fi convertite usor in reprezentarile string: str(x[0]) +str([1])

- Metoda *subject_predicates* care returnează subiectele şi predicarele pentru un anumit obiect dat

```
for x in g.subject_predicates(rdflib.URIRef(”http://dbpedia.org/resource/
Python_(programming_language”)):
    print x
```

- Metoda *triples* a grafului se comportă precum o interogare pentru tripleții ce iau forma șablonului specificat

```
list(g.triples((None, rdflib.URIRef("http://dbpedia.org/ontology/birthPlace"),None)))
```

Preluare date prin interogare SPARQL

- Unele site-uri oferă un SPARQL endpoint ce va permite adresarea de interogări și returnarea de date în diverse formate.
- Interogarea SELECT ne va permite să interogăm baza de cunoștințe și extrage coloane (potrivit variabilelor interogate) și rânduri (corespunzătoare numărului de rezultate)
- Formularea interogarii: se importă prepareQuery

```
from rdflib.plugins.sparql import prepareQuery

q=prepareQuery('SELECT ?y ?x WHERE {?y dbpedia-owl:birthDate
?x.}',initNs={'dbpedia-owl':'http://dbpedia.org/ontology/'})

g.query(q))
```

Pastrare date RDFLIB

- Grafurile in Rdflib au mai multe metode de a fi pastrate:

1. in memorie – Memory (nu este persistent) : graful creat implicit cu rdflib.Graph()
2. pe disk – Sleepycat (se va salva): graful trebuie creat prin rdflib.Graph(store='Sleepycat'); acesta va trebui deschis prin metoda open() si inchis la terminare cu close()

```
g=rdflib.Graph(store="Sleepycat")
g.open('C:\Python27\exemple',create=True)
g.add()
g.commit()
g.close()
```

Alte modalitati de pastrare (rdfextras):

- Berkeley DB
- MySQL
- PostgreSQL
- SQLite
- Zope Object Database (ZODB3)

Va multumesc

Intrebari?