Het semantisch web

Practicum Internettechnologie

Academiejaar 2013–2014

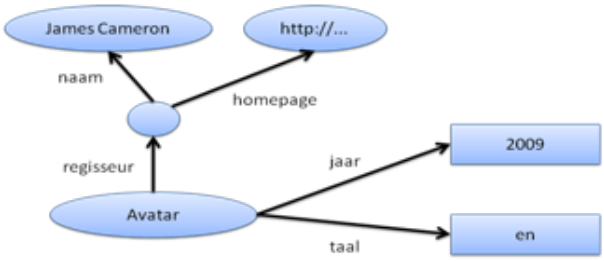
1 Het semantisch web: wat en waarom?

Huidige internetzoekmachines doorzoeken miljarden pagina's van het internet op zoek naar de opgegeven zoekwoorden. Vervolgens toont de zoekmachine een lijst met pagina's waarin die zoekwoorden voorkomen. De zoekmachine weet wel dat de zoekwoorden voorkomen in de pagina's in de lijst, maar de inhoud van die pagina's is hem echter onbekend. Dit komt omdat de huidige zoeksystemen op het internet HTML-tekst georiënteerd zijn. De meeste hebben geen notie van structuur of relaties binnen een kennisdomein. Als er complexe zoekvragen worden gesteld schieten huidige zoekmachines daarom te kort. Bijvoorbeeld, een overzicht van alle staatshoofden van Europa kan alleen beantwoord worden als er een webpagina over bestaat.

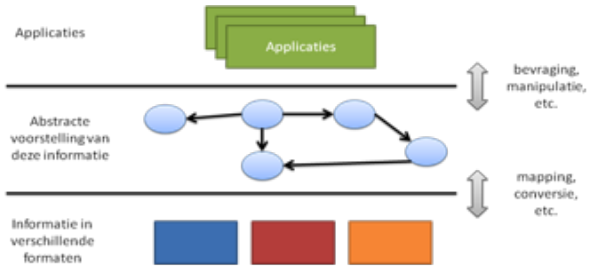
Het huidige web kunnen we daarom beschouwen als een web van documenten. Voor mensen zijn deze documenten perfect verstaanbaar. Echter, ondanks dat de documenten op het Web op een handige manier gelinkt zijn, is het voor webtoepassingen onmogelijk om te begrijpen wat er precies in die documenten staat. Meer bepaald kunnen webtoepassingen de structuur en syntax van deze documenten wel lezen, maar ze zijn zich niet bewust van de onderliggende semantische betekenis van de documenten. Voorbeelden van dergelijke semantische betekenissen zijn mensen, gebeurtenissen, bedrijven, landen, sport, eten, etc. Vandaag de dag vereisen heel wat taken dat verschillende data op het web gecombineerd moet worden. Dit komt omdat de vereiste kennis vaak in verspreide vorm op het web aanwezig is. Voorbeelden zijn het zoeken naar boeken op verschillende online bibliotheken of het zoeken naar een geschikte vlucht over verschillende luchtvaartmaatschappijen heen. Voor mensen is het vanzelfsprekend om deze verspreide en heterogene data te combineren, maar machines zijn momenteel niet in staat om dit op een automatische manier te doen. Meer bepaald, voor machines is het heel wat moeilijker om bestaande informatie te combineren omwille van het feit dat er veel verschillende dataformaten in gebruik zijn op het web, dat voor machines onvolledige informatie quasi onbruikbaar is, dat inhoud van afbeeldingen niet zomaar toegankelijk is, dat het maken van analogieën niet vanzelfsprekend is, etc.

Samengevat moet er een webarchitectuur komen waarin de beschikbare informatie wel begrijpbaar is voor machines en dit op een ondubbelzinnige manier. De informatie moet tevens gemakkelijk te combineren en uitwisselbaar zijn. Verder moeten machines ook in staat zijn om over de informatie te kunnen redeneren en dus nieuwe kennis afleiden op basis van de bestaande informatie. In plaats van een web van documenten hebben we dus nood aan een web van data: het semantisch web.

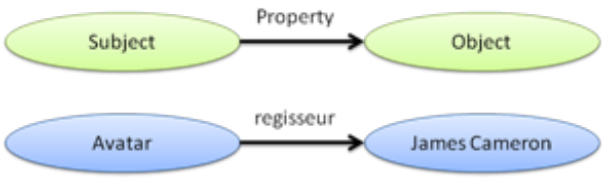
Informatie in het semantisch web wordt voorgesteld aan de hand van gerichte grafen. Een eenvoudig voorbeeld van een gerichte graaf die de beschrijving van een film voorstelt is afgebeeld in Figuur 1. Entiteiten komen typisch overeen met de knopen van de graaf, terwijl eigenschappen van deze entiteiten en relaties tussen entiteiten worden voorgesteld door gerichte takken. Het werken met gerichte grafen kan gezien worden als een abstracte voorstelling van de informatie. Meer bepaald kunnen de vele verschillende dataformaten afgebeeld of geconverteerd worden naar deze abstracte voorstelling. De webtoepassingen die gebruikmaken van de informatie beschikbaar op het web werken dus niet meer rechtstreeks op de verschillende dataformaten, maar baseren zich op de abstracte voorstelling van deze informatie. Op die manier kunnen toepassingen de beschikbare data doorzoeken, en/of manipuleren onafhankelijk van de onderliggende dataformaten. Een visuele voorstelling van deze structuur van het semantisch web is weergegeven in Figuur 2.



Figuur 1: Voorstelling van een film door middel van een gerichte graaf



Figuur 2: Structuur van het semantisch web



Figuur 3: RDF-triples voor de voorstelling van informatie

2 Semantisch-webtechnologieën

Om het semantisch web te realiseren zijn een aantal specifieke technologieën nodig (dwz. semantisch-webtechnologieën). In dit practicum zullen we met een viertal van deze technologieën werken: RDF, RDF Schema, OWL en SPARQL. Deze vier technologieën vormen tevens de basis voor het semantisch web. Het is niet de bedoeling dat we deze technologieën volledig in detail bekijken, enkel de basiselementen en de functionaliteit die nodig zijn om dit practicum op te lossen zullen toegelicht worden. Voor een compleet overzicht van alle semantisch-webtechnologieën en hun corresponderende specificaties kan je terecht op de pagina van de W3C Semantic Web Activity: *http://www.w3.org/2001/sw/*.

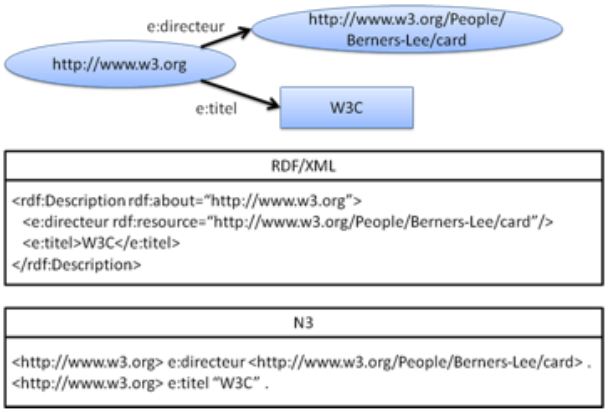
2.1 Resource Description Framework (RDF)

Het Resource Description Framework (RDF) is de bouwsteen van het semantisch web. RDF is de technologie die ons zal toelaten om informatie in een graafstructuur te modelleren. Meer bepaald worden resources beschreven door middel van 'triples': (subject, property, object). De graafstructuur in acht genomen komen het subject en object overeen met knopen van de graaf, terwijl de property overeenkomt met een tak in de graaf. In Figuur 3 is een voorbeeld te zien van hoe informatie in een RDF-triple gerepresenteerd wordt. De informatie "Avatar werd geregisseerd door James Cameron" komt overeen met het RDF-triple (Avatar, regisseur, James Cameron).

Een resource wordt in RDF uniek geïdentificeerd aan de hand van een Uniform Resource Identifier (URI). Meer bepaald worden subject en property typisch voorgesteld door een URI; het object kan een URI zijn of een literal (bv. integer, datum, string, etc.). Hieronder staan twee voorbeelden van geldige RDF-triples:

```
(http://dbpedia.org/page/Avatar_%282009_film%29, http://xmlns.com/foaf/0.1/name, "Avatar")
(http://dbpedia.org/page/Avatar_%282009_film%29, http://dbpedia.org/ontology/Film/director,
http://dbpedia.org/resource/James_Cameron)
```

Figuur 4: Anonieme knopen in een RDF-graaf



Figuur 5: Verschillende manieren om RDF voor te stellen

Merk op dat resources kunnen overeenkomen met om het even welke URI. Het is dus niet verplicht om een bestaande (zogenaamde 'dereferencable') URI te gebruiken voor resources. In RDF is het ook mogelijk om met interne knopen te werken. In dit geval komt een knoop niet overeen met een URI of een literal, maar met een anonieme knoop (Eng.: *blank node*). Met behulp van anonieme knopen is het mogelijk om het volgende statement uit te drukken: "een regisseur is 'iets' dat een naam en een homepage heeft", zoals voorgesteld in Figuur 4. In de logica komen anonieme knopen overeen met een existentie: "er bestaat een resource zodat…"

Er zijn een aantal verschillende mogelijkheden om RDF weer te geven en/of op te slaan. Eén manier is grafisch zoals reeds gedemonstreerd in Figuur 3 en Figuur 4. Deze manier is echter niet praktisch om door machines gelezen te worden. Daarom werden er een aantal formaten ontwikkeld waarin RDF kan voorgesteld en opgeslagen worden. Een eerste manier om RDF op te slaan is door gebruik te maken van de Extensible Markup Language (XML). Meer bepaald werden een aantal XML-elementen en XML-attributen gedefinieerd die het mogelijk maken om RDF-triples in XML voor te stellen. Het betreffende formaat wordt dan ook RDF/XML genoemd. In Figuur 5 staat een eenvoudig voorbeeld van hoe twee RDF-triples in XML-formaat kunnen neergeschreven worden. Een RDF-triple wordt beschreven door een rdf:Description, de properties komen voor als kinderen van het rdf:Description-element. Omdat het gebruik van XML om ingewikkelde RDF-grafen voor te stellen nogal onoverzichtelijk wordt en moeilijk te onderhouden is, is er een tweede manier ontwikkeld om RDF-triples op te slaan: Terse RDF Triple Language of N3. N3, dat geïntroduceerd werd door Sir Tim Berners-Lee, is een leesbaar en compact alternatief voor RDF/XML. RDF-triples worden neergeschreven als zinnen van drie woorden gevolgd door een punt, zoals geïllustreerd in Figuur 5. Merk op dat we in dit practicum dan ook gebruik zullen maken van N3 om RDF-triples voor te stellen. De oplossingen voor dit practicum zullen tevens in N3 moeten gemaakt worden.

Zoals daarnet reeds aangegeven worden RDF-triples in N3 voorgesteld als een zin van drie woorden gevolgd door een punt: subject property object . Voorts worden URI 's tussen vishaken geschreven: `<http://www.w3.org>` en staan literals tussen dubbele quotes, eventueel gevolgd door hun `^^` en het datatype: `"2009"^^xsd:int`. Net zoals in XML kan er in RDF ook gewerkt worden met naamruimten. Een naamruimte komt in RDF typisch overeen met een URI-prefix en kan de notatie van RDF-triples aanzienlijk verkorten. In N3 worden naamruimten vooraan in het document gedeclareerd aan de hand van de @prefix-constructie, zoals geïllustreerd in Figuur 6. Verder voorziet N3 een aantal shortcuts voor de notatie van RDF-triples met hetzelfde subject en/of property. Meer bepaald worden RDF-triples die hetzelfde subject hebben gegroepeerd met behulp van ';' en worden RDF-triples met dezelfde subject en property gegroepeerd met behulp van ','. Een voorbeeld van dergelijke verkorte N3-notatie is te zien in Figuur 6. Ten slotte worden anonieme knopen in N3 voorgesteld door `_:xxxx` met `xxxx` de naam van de anonieme knoop (bv. `_:bnode0`). Er is tevens een verkorte notatie in N3 mogelijk voor anonieme knopen. Meer bepaald kunnen de properties van anonieme knopen voorgesteld worden tussen

Figuur 6: Verkorte notaties in N3

vierkante haken (dwz. '[' en ']'), zoals te zien is in Figuur 6.

2.2 RDF Schema (RDFS)

Nu we gezien hebben hoe informatie op een ondubbelzinnige en formele manier kan beschreven worden in RDF, introduceren we een laag bovenop RDF: RDF Schema (RDFS). Met RDFS is het mogelijk om kennis op een geavanceerdere manier te beschrijven. Meer bepaald is het mogelijk om resources te gaan groeperen in verschillende klassen en de semantische betekenis van properties te beschrijven. RDFS is eigenlijk niets anders dan een verzameling resources die een speciale betekenis gekregen hebben. Om bijvoorbeeld uit te drukken dat personen gegroepeerd kunnen worden in de klasse `foaf:Person` kan de volgende constructie gebruikt worden (in N3):

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

foaf:Person rdf:type rdfs:Class .
<http://www.w3.org/People/Berners-lee/card> rdf:type foaf:Person .
```

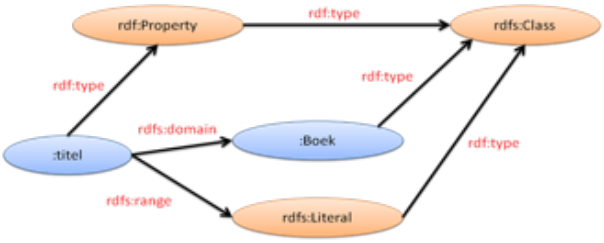
Het eerste triple drukt uit dat `foaf:Person` een klasse is; het tweede triple definieert een instantie van de klasse foaf:Person. Het is ook mogelijk om een klassenhiërarchie te maken met behulp van RDFS. We kunnen bijvoorbeeld uitdrukken dat de klasse `:Student` een subklasse is van de klasse `foaf:Person`:

```
:Student rdf:type rdfs:Class ;
         rdfs:subClassOf foaf:Person .
```

Naast het definiëren van een klassenstructuur is het met RDFS ook mogelijk om de semantiek van properties te beschrijven. Hierbij is het zo dat elke property lid is van de klasse `rdf:Property`. Gelijkaardig aan de definitie van een klassenhiërarchie is het mogelijk om een hiërarchie van properties te definiëren. Meer bepaald kunnen properties met elkaar gelinkt worden via de `rdfs:subPropertyOf`-property. Bijvoorbeeld, de property `:isBroerVan` is een subproperty van de property `:isFamilieVan`:

```
:isFamilieVan rdf:type rdf:Property .
:isBroerVan rdf:type rdf:Property ;
            rdfs:subPropertyOf :isFamilieVan .
:Jan :isBroerVan :Piet .
```

Verder is het ook mogelijk om de klassen en/of literals vast te leggen die kunnen voorkomen in het subject of object van de RDF-triples waarin een bepaalde property voorkomt. Meer bepaald, wanneer een bepaalde property gebruikt wordt in een RDF-triple, dan geeft de range (`rdfs:range`) aan welk type resource er zal voorkomen in het object, terwijl het domain (`rdfs:domain`) aangeeft welk type resource er zal voorkomen in het subject. In Figuur 9 staat een voorbeeld van de modellering van properties door gebruik te maken van RDFS. In dit voorbeeld wordt een property :titel gedefinieerd, die als range een literal zal hebben en als domain een resource van de klasse `:Boek`. De `:titel`-property kan dan bijvoorbeeld als volgt gebruikt worden:



Figuur 7: Modellering van properties in RDFS

```
:VoorbeeldBoek rdf:type :Boek ;
               :titel “Voorbeeld titel”^^xsd:string .
```

Ten slotte biedt RDFS nog een tweetal properties aan om commentaar en uitleg toe te voegen aan klassen, properties en instanties:

```
:Boek rdf:type rdfs:Class ;
      rdfs:label “Boek” ;
      rdfs:comment “Representatie van een boek” .
```

De property `rdfs:comment` laat toe om een beschrijving te geven van een resource, terwijl `rdfs:label` typisch een korte benaming van deze resource is. Merk op dat `rdfs:label` vaak gebruikt wordt voor de visualisatie van de resource. Het gebruik van RDFS laat niet enkel toe om bepaalde kennis op een geavanceerdere manier te beschrijven dan RDF, maar laat ook toe om, op basis van de bestaande kennis, nieuwe kennis af te leiden. Met andere woorden, RDFS-concepten laten toe om over de RDF-triples te redeneren (dwz. RDFS inferencing). Merk op dat hiervoor wel een specifieke RDFS-reasoner nodig is: een stuk software dat de speciale betekenis kent van de RDFS-concepten. We illustreren het redeneren over RDF-triples met behulp van RDFS met de volgende RDF-triples:

```
:Student rdfs:subClassOf foaf:Person .
:studeertAan rdf:type rdf:Property ;
             rdfs:domain :Student .
:Jan :studeertAan <http://www.ugent.be> .
```

Uit bovenstaande RDF-triples kunnen we nu, rekeninghoudende met de semantische betekenis van de RDFS-properties, nieuwe RDF-triples gaan afleiden. In het laatste RDF-triple wordt gebruikgemaakt van de property `:studeertAan` en komt `:Jan` voor als subject. Aangezien het domain van de property `:studeertAan` de klasse `:Student` is, kunnen we hieruit afleiden dat `:Jan` een instantie is van de klasse `:Student`. Verder kunnen we ook afleiden dat `:Jan` tevens een instantie is van de klasse `foaf:Person`, omdat de klasse `:Student` een subklasse is van de klasse `foaf:Person`. Volgende RDF-triples kunnen dus afgeleid worden:

```
:Jan rdf:type :Student, foaf:Person .
```

2.3 Web Ontology Language (OWL)

RDFS voorziet een aantal basisconcepten om een kennisdomein te beschrijven. Dergelijke beschrijving van een kennisdomein wordt ook wel een ontologie genoemd. Een ontologie definieert concepten en relaties die gebruikt worden om kennis van een bepaald domain te beschrijven op een rijke en ondubbelzinnige manier. Echter, wanneer we een gedetailleerde domeinbeschrijving willen maken op een formele en een voor machines verstaanbare manier, dan heeft RDFS toch een aantal beperkingen:

- de karakterisatie van properties is heel beperkt;
- hoe ga je om met verschillende URI 's die dezelfde resource voorstellen;
- het is niet mogelijk om equivalenties tussen klassen en tussen properties uit te drukken;
- er is nood aan een rijkere constructie van klassen.

Om een antwoord te bieden op de tekortkomingen van RDFS heeft men binnen het W3C de Web Ontology Language (OWL) ontwikkeld. Daar waar RDFS kan beschouwd worden als een heel eenvoudige ontologietaal,

is OWL een extra laag bovenop RDFs waarmee het mogelijk is om veel geavanceerdere domeinbeschrijvingen te maken. Net zoals RDFs definieert OWL een aantal concepten die een speciale betekenis hebben, en die door reasoners zullen gebruikt worden om nieuwe kennis af te leiden. De eerste versie van OWL werd gefinaliseerd in 2004. Ondertussen is er reeds een tweede versie van OWL ontwikkeld en gestandardiseerd in 2009. In dit practicum houden wij het echter bij de eerste versie van OWL. Merk op dat OWL heel complex is en een brede waaier voorziet aan concepten. Het is in dit practicum niet de bedoeling om een volledig overzicht te geven van OWL; er wordt enkel een beperkt overzicht gegeven van concepten die nuttig zijn voor het oplossen van dit practicum. Voor een volledige uitleg over OWL verwijzen we naar de specificatie: http://www.w3.org/TR/owl-features/. OWL definieert een nieuw concept voor de definitie van een klasse, gebaseerd op `rdfs:Class`: `owl:Class`. Naast het feit dat `owl:Class` een subklasse is van `rdfs:Class` laat het ook toe om klassen te construeren op basis van bestaande klassen of instanties. Dit kan door een klasse te creëren uit het complement, de doorsnede of de unie van twee of meerdere klassen. Verder is het ook mogelijk om een klasse te definiëren aan de hand van een opsomming van mogelijke instanties. Het volgende voorbeeld illustreert geavanceerde creaties van klassen:

```
@prefix : <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

:Resultaat rdf:type owl:Class ;
           owl:oneOf (:Goed :Fout) .
:Goed rdf:type :Resultaat .
:Fout rdf:type :Resultaat .

:Persoon rdf:type owl:Class ;
         owl:unionOf (:Man :Vrouw) .
:Man rdf:type owl:Class .
:Vrouw rdf:type owl:Class .
```

Merk op dat RDF-lijsten in N3 zeer compact genoteerd kunnen worden met behulp van de ronde haakjes; de leden van de lijst worden gescheiden door een spatie. In bovenstaand voorbeeld zien we dat de klasse `:Resultaat` enkel maar bestaat uit de instanties `:Goed` en `:Fout`. Verder zien we dat de klasse `:Persoon` een unie is van de klassen `:Man` en `:Vrouw`. Naast de vele verschillende mogelijkheden om klassen te definiëren, biedt OWL de mogelijkheid om het gedrag van properties te beschrijven. Meer bepaald kan aangegeven worden indien een property symmetrisch, transitief, functioneel of invers functioneel is. Verder wordt er een onderscheid gemaakt tussen een datatype-property (dwz. de range correspondeert met een literal) en een object-property (dwz. de range correspondeert met een OWL-klasse). Het volgende voorbeeldje toont een datatype-property:

```
foaf:email rdf:type owl:DatatypeProperty, owl:InverseFunctionalProperty ;
           rdfs:range xsd:string ;
           rdfs:domain foaf:Person .
```

Merk op dat een inverse functionele property gekenmerkt wordt door het feit dat twee verschillende subjects geen identieke objects kunnen hebben. In dit voorbeeld is de property `foaf:email` dus invers functioneel omdat geen twee dezelfde personen hetzelfde e-mailadres kunnen hebben. Ten slotte kunnen we OWL ook gebruiken voor het uitdrukken van equivalenties en verschillen tussen instanties, klassen en properties. Deze speciale OWL-properties worden veelvuldig gebruikt (samen met de `rdfs:subClassOf` en `rdfs:subPropertyOf` properties) om verschillende ontologieën met elkaar te linken. Dus, voor zowel klassen, properties als instanties zijn er properties die equivalentie en verschil uitdrukken:

- klassen: `owl:equivalentClass` en `owl:disjointWith`
- properties: `owl:equivalentProperty` en `owl:propertyDisjointWith`
- instanties: `owl:sameAs` en `owl:differentFrom`

Hieronder staan enkele voorbeelden van het gebruik van bovenstaande properties:

```
:Mens rdf:type owl:Class .
:Persoon rdf:type owl:Class ;
```

```
        owl:equivalentClass :Mens .

:Jan rdf:type :Persoon .
:Piet rdf:type :Persoon ;
      owl:differentFrom :Jan .
```

Net zoals we gezien hebben bij RDFs, kun je met OWL-concepten nieuwe kennis gaan afleiden op basis van de bestaande kennis. Gelijkaardig aan RDFs kunnen we de semantiek van de OWL-concepten gaan interpreteren. Stel dat we beschikken over de volgende RDF-triples:

```
:email rdf:type owl:InverseFunctionalProperty .
:Jan_1 :email “mailto:jan@b.c” .
:Jan_2 :email “mailto:jan@b.c” .
```

Omdat de property :email invers functioneel is, betekent dit dat er twee verschillende subjecten geen identiek object kunnen hebben. Anders gezegd, doordat we twee RDF-triples tegenkomen die hetzelfde object hebben (dwz. “mailto:jan@b.c”) in combinatie met dezelfde invers functionele property, kunnen we afleiden dat `:Jan_1` en `:Jan_2` niet verschillend, en dus gelijk zijn:

```
:Jan_1 owl:sameAs :Jan_2 .
```

Het afleiden van nieuwe kennis in OWL is dus heel gelijkaardig aan RDFs, alleen kan je met OWL veel geavanceerdere zaken gaan afleiden. Merk op dat je, net zoals je een specifieke RDFs-reasoner moet hebben voor RDFs-afleiding, een specifieke OWL-reasoner moet hebben om automatisch aan OWL-afleiding te doen.

2.4 SPARQL Protocol And RDF Query Language

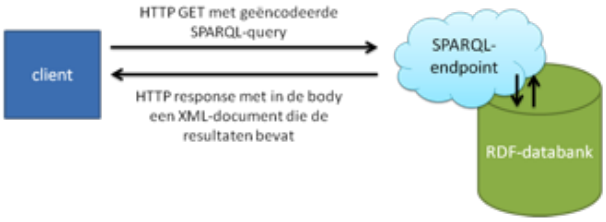
SPARQL Protocol And RDF Query Language (SPARQL) is een technologie om toegang te krijgen tot RDF-triples. SPARQL biedt ons de mogelijkheid om bijvoorbeeld alle instanties van het type foaf:Person op te vragen. Ook geavanceerde vragen zoals alle personen die in een stad wonen waarvan het inwonersaantal groter is dan 10000 zijn mogelijk. De SPARQL-specificatie is, zoals de naam reeds laat vermoeden, opgesplitst in twee delen: een specificatie voor de syntax van de queries en een specificatie voor de boodschappen die worden uitgewisseld tussen een gebruiker en een SPARQL-server. De syntax van de SPARQL-querytaal is gelijkaardig en gebaseerd op SQL, de querytaal die gebruikt wordt om relationele databanken te bevragen. Merk op dat we ook hier geen volledig overzicht geven van de SPARQL-specificatie, maar enkel datgene dat nodig is om het practicum tot een goed einde te kunnen brengen. Een volledig overzicht kan teruggevonden in de specificatie: http://www.w3.org/TR/rdf-sparql-query/. De structuur van een eenvoudige SPARQL-query bestaat typisch uit de volgende componenten:

- `PREFIX`: declaratie van naamruimten, gelijkaardig aan de `@prefix`-constructie in N3;
- `SELECT`: identificatie van de waarden die teruggegeven moeten worden;
- `WHERE { }`: het triplepatroon waarvoor een match gevonden moet worden.

Stel dat we alle instanties van het type foaf:Person willen bekomen, samen met de naam van die personen, dan kunnen we dat doen aan de hand van de volgende SPARQL-query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns\#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?person
WHERE {
    ?person rdf:type foaf:Person .
    ?person foaf:name ?name .
}
```

De waarden die we willen bekomen worden voorgesteld aan de hand van variabelen ?x waarbij x de naam is van de variabele. Het `WHERE`-gedeelte is heel gelijkaardig aan een lijst van RDF-triples in N3-stijl, waarbij sommige resources vervangen zijn door variabelen. Merk op dat je niet alle variabelen die voorkomen in het `WHERE`-gedeelte hoeft op te vragen via het `SELECT`-gedeelte. Binnen het semantisch web worden SPARQL-queries



Figuur 8: Overzicht van het SPARQL-protocol over HTTP

typisch verstuurd over het HTTP-protocol. Daarom definieert SPARQL naast een specificatie van de syntax van een query ook het protocol dat moet gebruikt worden om SPARQL-vragen en antwoorden te versturen. Met andere woorden, de syntax van de vraag (dit kan via een HTTP `GET` of `POST`) en het XML-formaat waarin het antwoord verstuurd worden vastgelegd. Een overzicht van deze twee communicatiestappen is afgebeeld in Figuur 8. Merk op dat de webservice die SPARQL-queries kan interpreteren en voldoet aan het SPARQL-protocol een SPARQL-endpoint genoemd wordt. Voorbeelden van publieke SPARQL-endpoints zijn *http://dbpedia.org/sparql/* en *http://data.linkedmdb.org/sparql/*.

Een SPARQL-query versturen naar een SPARQL-endpoint kan heel eenvoudig gebeuren aan de hand van een HTTP `GET` request. Stel dat we bijvoorbeeld de volgende SPARQL-query over HTTP willen versturen:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?person
WHERE {
    ?person rdf:type foaf:Person .
    ?person foaf:name ?name .
}
```

De corresponderende HTTP-vraag ziet er dan als volgt uit:

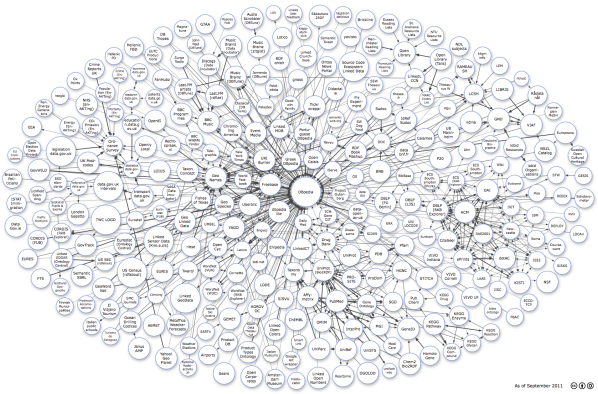
```
GET /sparql?query=<SPARQLQuery>  HTTP/1.1
User-Agent: my-sparql-client/0.0
Host: server.com
Accept: application/sparql-results+xml
```

Waarbij `<SPARQLQuery>` gelijk is aan een URL-geëncodeerde versie van de SPARQL-query:

```
PREFIX+rdf\%3A+\%3Chttp\%3A\%2F\%2Fwww.w3.org\%2F1999\%2F02\%2F22-rdf-syntax-ns\%23\%3E+REFIX+foaf\%3A+\%
```

Belangrijk bij de HTTP-vraag is de Accept-header, die aangeeft wat het outputformaat moet zijn van het SPARQL-endpoint (in dit geval dus SPARQL-resultaten in XML-formaat). Indien alles goed gaat, antwoordt de server met code 200 (dwz. OK) en worden de resultaten in XML doorgestuurd. Een voorbeeld van dergelijk resultaat is als volgt:

```
<sparql xmlns="http://www.w3.org/2005/sparql-results\#>
  <head>
    <variable name="name"/>
    <variable name="person"/>
  </head>
  <results>
    <result ordered="false" distinct="false">
      <binding name="name"><literal>Tim Berners-Lee</literal></binding>
      <binding name="person"><uri> http://.../People/Berners-Lee/card</uri></binding>
    </result>
    <result>
      ...
```



Figuur 9: De Linked Open Data cloud in september 2011

```
    </result>
  </results>
</sparql>
```

De XML-structuur die teruggegeven wordt door een SPARQL-endpoint begint steeds met een head-element waarin de variabelen voorkomen die gevraagd werden (komt dus overeen met de variabelen na de SELECT). Het results-element bevat een lijst van result-elementen die elke één resultaat van de SPARQL-query beschrijven. Dergelijk resultaat bevat voor elke variabele die voorkomt in het head-element een binding-element waarin de waarde van die variabele afgebeeld wordt. Afhankelijk van het type van de variabele is het resultaat opgeslagen als een literal of een uri.

3 Linked Open Data: het semantisch web in de praktijk

Het Linked Open Data (LOD)-project is een initiatief van de semantisch-webcommunity om zoveel mogelijk beschikbare data als RDF te publiceren en te linken met elkaar. Op die manier worden grote datasets in RDF gepubliceerd en via SPARQL toegankelijk gemaakt. Momenteel bevat de verzameling reeds miljarden triples en miljoenen links tussen de verschillende datasets. Een grafische voorstelling van deze datasets wordt ook wel de LOD cloud genoemd en is te zien in Figuur 9. Er zijn heel uiteenlopende datasets beschikbaar in de LOD cloud. Algemene datasets zoals DBPedia en Freebase bevatten informatie over verschillende domeinen. Merk op dat DBPedia letterlijk de RDF-versie is van wikipedia. Andere datasets, zoals bijvoorbeeld Geonames, zijn dan weer gespecialiseerd in één specifiek domein (Geonames is een verzameling van geografische locaties). In dit practicum is het voldoende om ons te beperken tot DBPedia voor het (her)gebruiken van RDF-data. Het SPARQL-endpoint van DBPedia is beschikbaar op http://dbpedia.org/sparql. Merk op dat DBPedia ook beschikt over een Google-achtige zoekinterface: http://lookup.dbpedia.org. Naast deze zoekmogelijkheden is het ook mogelijk om gewoon door de RDF-data te browsen. Surf bijvoorbeeld eens naar de resource die Barack Obama voorstelt (*http://dbpedia.org/resource/Barack_Obama*). Daar zul je merken dat je kunt navigeren naar andere RDF-data die gelinkt is met deze resource.

4 Opgave practicum 3

4.1 Ontwikkelen van een ontologie

In de eerste opgave is het de bedoeling om een OWL-ontologie te ontwikkelen voor de beschrijving van digitale foto's. Met deze ontologie moet het dus mogelijk zijn om digitale foto's te gaan annoteren. De volgende zaken moeten zeker opgenomen zijn in de ontologie:

- de maker van de foto;
- wat er te zien is op de foto;
- het thema van de foto;

- de plaats waar de foto gemaakt is;
- de datum waarop de foto gemaakt is;
- een korte beschrijving van de foto;
- de titel van de foto.

Merk op dat het thema enkel kan overeenkomen met de volgende resources (gebruik daartoe de gepaste OWL-constructie):

- http://dbpedia.org/resource/Category:People
- http://dbpedia.org/resource/Category:Animals
- http://dbpedia.org/resource/Category:Architecture
- http://dbpedia.org/resource/Category:Nature
- http://dbpedia.org/resource/Category:Politics
- http://dbpedia.org/resource/Category:Humor
- http://dbpedia.org/resource/Category:Culture
- http://dbpedia.org/resource/Category:News

Naast het opmaken van een ontologie is het ook de bedoeling om een aantal voorbeeldinstanties aan te maken die compatibel zijn met de ontworpen ontologie (minimum twee instanties). Probeer bij het opstellen van dergelijke instanties zoveel mogelijk gebruik te maken van bestaande RDF-data (die bv. aanwezig is op DBPedia). Niet alles wat in de ontologie beschreven staat, moet ook daadwerkelijk in de instanties voorkomen. Belangrijk: zorg er ook voor dat de afbeelding die je annoteert beschikbaar is op het Web (dwz. waarbij X staat voor het groepsnummer. Alle RDF-data dient in N3 opgesteld te worden.

Bestandsnamen:

- foto_ontologie.n3
- foto_instanties.n3

4.2 Linken van ontologie concepten

Nadat een eigen foto-ontologie werd opgesteld, is het de bedoeling om concepten die daarin gedefinieerd werden te linken met andere (veel gebruikte) properties. Meer bepaald is het in deze opgave de bedoeling om de link te leggen met de Dublin Core properties. Dublin Core is een verzameling van een 15-tal algemene metadata-elementen die gebruikt kunnen worden voor de annotatie van Web resources (bv. zie UGent-website). Uiteraard kunnen deze elementen ook gebruikt worden voor de annotatie van digitale afbeeldingen. Uitgebreide documentatie over deze Dublin Core properties kan je terugvinden op *http://dublincore.org/documents/dces/*. Merk op dat de properties ook formeel in RDF beschreven staan: *http://dublincore.org/2008/01/14/dcelements.rdf*. Naast de link met Dublin Core moet er ook voor gezorgd worden dat de foto-ontologie gelinkt is met de klasse http://xmlns.com/foaf/0.1/Image (foaf:Image), welke een digitale afbeelding representeert. Belangrijk om te weten is ook dat er enkel gebruik kan gemaakt worden van RDFS-logica om de mapping te maken. De reden hiervoor is dat de reasoning engine die zal gebruikt worden in de derde opgave geen ondersteuning biedt voor OWL-afleidingen en OWL reasoning. De RDF-triples die de mapping beschrijven tussen de foto-ontologie en Dublin Core dienen opnieuw in N3 opgesteld te worden.

Bestandsnaam:

- mapping.n3

4.3 Semantische-toepassing ontwikkelen

In de laatste opgave van dit practicum is het de bedoeling om een Ruby on Rails-toepassing te ontwikkelen die toelaat om in geannoteerde foto's te zoeken. Merk op dat het zoeken enkel mag gebeuren op basis van Dublin Core properties en de klasse foaf:Image. Er mag dus geen gebruikgemaakt worden van de eigen concepten die voorkomen in de foto-ontologie die gedefinieerd werd in opgave 1! Op deze manier kan er getest worden of de mapping in opgave 2 wel degelijk correct werd opgesteld. Deze functionaliteit illustreert op een eenvoudige manier de mogelijkheden die het Semantisch Web biedt: door het linken van verschillende metadataschema's (propriëtair en/of gestandaardiseerd) kan nieuwe kennis gegenereerd worden (bv. meer zoekresultaten). Zo zullen de instanties die reeds aanwezig in de databank samen getoond worden met de nieuwe toegevoegde instanties uit opgave 3.1 zonder dat de gebruiker zich bewust is van de verschillende dataschemas in de databank.

Het is uiteraard nodig dat deze schema's en links 'gepubliceerd' worden via een RDF-databank. Er is een RDF-databank en bijhorend SPARQL-endpoint beschikbaar voor deze opgave. De uitvoerbare bestanden voor deze databank kunnen gedownload worden in de Minerva files sectie. De N3-bestanden met RDF-triples van opgave 1 en 2 worden dan in de zelfde folder als deze bestanden geplaatst. Bij het opstarten zullen alle n3 files in de zelfde folder automatisch ingeladen worden. Om het sparql endpoint op de starten gebruik je volgende opdrachtregel:

```
java -jar itech-jar-with-dependencies.jar poortnr
```

Als poortnummer wordt het groepsnummer + 11000 gebruikt. Het SPARQL-endpoint kan dan getest worden via de volgende url: *http://localhost:110XX/sparql/query?query=<SPARQLQuery>* waarbij `<SPARQLQuery>` een -geëncodeerde versie is van de SPARQL-query. Dus de volgende URL haalt bijvoorbeeld alle tripels in de databank op in XML:

```
http://localhost:110XX/sparql/query?format=xml&query=select%20*%20 where%20{?s%20?p%20?o}.
```

Merk op dat dit niet strookt met de REST principes, want het bestandsformaat wordt in de *m*eegegeven. Dit creert twee verschillende resources, terwijl het er maar één is. Dit zou natuurlijk enkel met *Content Negotiation* mogen gebeuren (zie Practicum 1). Zoek zeker uit of dit kan met deze endpoint, want dit wordt beloond in de quotatie.

In Ruby kan de verbinding met het sparql endpoint gerealiseerd worden met behulp van de `Net::HTTP`-klasse en `URI`-klasse uit de packages *net/http* en *uri*. De Rails-toepassing zelf moet

- de mogelijkheid bieden om naar foto's te zoeken op basis van de Dublin Core properties. Dit kan bijvoorbeeld gebaseerd zijn op dropdown boxen of invulvelden, afhankelijk van het soort property. Er moet ten minste apart gezocht kunnen worden op thema en een tweede property naar keuze, alsook op twee properties naar keuze tegelijkertijd ('AND').
- initieel worden alle foto's in de databank weergegeven.
- geef elke foto slechts eenmaal weer, samen met alle bijhorende metadata.
- SPARQL gebruiken om zoekopdrachten uit te voeren op de server (SPARQL-endpoint), maar moet SPARQL verborgen houden voor de eindgebruiker.

Validatie N3-beschrijvingen kunnen (bijvoorbeeld) gevalideerd worden via *http://www.rdfabout.com/demo/validator/*.

Gebruik op Athena Athena is een cloud computing systeem. Wanneer de gebruiker een programma opstart zal het niet noodzakelijk op de zelfde node uitgevoerd worden. Aangezien het niet toegelaten is poorten te openen tussen nodes, is het bij het testen van de server dus belangrijk dat alle programma's (de Rails server én het Sparql endpoint) op de zelfde machine uitgevoerd worden.

4.4 Indienen

Dien de drie opgaven in via de Minerva dropbox als: *itech-p3-g<groep>.zip*.

Voor opgave 3 stuur je de volledige Ruby on Rails projectfolder, ZONDER de sparql endpoint software! Zorg er voor dat de geüploade N3-bestanden de correcte naamruimte gebruiken (*http://berio.elis.ugent.be/ontology/itechX/* met X = groepsnummer).

Opmerkingen Originaliteit en layout worden in dit practicum beloond (ontologie en webtoepassing). Deze punten dienen als extra beschouwd te worden ter compensatie van eventueel gemaakte fouten (dit betekent ook dat geen punten afgetrokken zullen worden op dit aspect zolang aan alle andere vereisten van de opgave voldaan is).