Het semantisch web

Practicum Internettechnologie

Academiejaar 2014–2015

1 Semantisch-webtechnologieën

Om het semantisch web te realiseren zijn een aantal specifieke technologieën nodig. In dit practicum zullen we met een viertal van deze technologieën werken die de basis vormen voor het semantisch web: RDF, RDF Schema, OWL en SPARQL. Voor een compleet overzicht van alle semantisch-webtechnologieën en specificaties kan je terecht op de pagina van de W3C Semantic Web Activity (*http://www.w3.org/2001/sw/*).

1.1 Resource Description Framework (RDF)

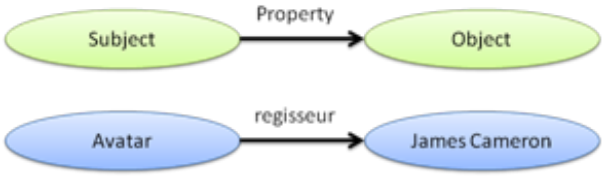
RDF laat toe om informatie in een graafstructuur te modelleren. Meer bepaald worden resources beschreven door middel van *triples: (subject, property, object)*. Subject en object overeen met knopen van de graaf, terwijl de property overeenkomt met een boog in de graaf. In Figuur 1 is een voorbeeld te zien van hoe informatie in RDF-triples voorgesteld wordt. De informatie "Avatar werd geregisseerd door James Cameron" komt overeen met het RDF-triple *(Avatar, regisseur, James Cameron)*. Elk van de drie triple-componenten identificeren we met een URL. Enkel objecten kunnen in plaats van een URL ook tekst ("literals") bevatten, al dan niet met een datatype. Dit zijn twee triples in de Turtle-notatie:

```
<http://dbpedia.org/resource/Avatar> <http://xmlns.com/foaf/0.1/name> "Avatar".
<http://dbpedia.org/resource/Avatar> <http://dbpedia.org/ontology/Film/director>
                                        <http://dbpedia.org/resource/James_Cameron>.
```

Via `PREFIX` kan je gebruik maken van afkortingen voor een compactere notatie:

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

dbpedia:Avatar foaf:name "Avatar".
dbpedia:Avatar <http://dbpedia.org/ontology/Film/director> dbpedia:James_Cameron.
```



Figuur 1: RDF-triples voor de voorstelling van informatie

1.2 RDF Schema (RDFS)

Met RDFS is het mogelijk om resources te groeperen in verschillende klassen en de semantische betekenis van properties te beschrijven. RDFS is eigenlijk niets anders dan een verzameling resources met een specifieke betekenis. Om bijvoorbeeld uit te drukken dat personen gegroepeerd kunnen worden in de klasse `foaf:Person`, kan de volgende constructie gebruikt worden.

```
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.
```

Daarnaast kan je ook het domein en bereik van properties vastleggen:

```
:studeertAan rdf:type rdf:Property;
             rdfs:domain :Student;
             rdfs:range :University.
```

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"@nl;
      rdfs:comment "Deze klasse laat toe om boeken te modelleren."@nl.
```

1.3 Web Ontology Language (OWL)

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. Merk op dat OWL heel complex is en een brede waaier voorziet aan concepten. 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 dergelijke definities van klassen:

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

:Persoon rdf:type owl:Class;
         owl:unionOf (:Linkshandig :Rechtshandig).
:Linkshandig rdf:type owl:Class.
:Rechtshandig rdf:type owl:Class.
```

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 `:Linkshandig` en `:Rechtshandig`. 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 (het bereik bestaat uit literals) en een object-property (het bereik bestaat uit OWL-klassen). Het volgende voorbeeldje toont een datatype-property:

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

Een invers-functionele property wordt gekenmerkt door het feit dat twee verschillende subjects geen identieke objects kunnen hebben. In dit voorbeeld is de property `foaf:email` 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 bij RDFS, kan je met OWL-concepten nieuwe kennis 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@example.org".
:Jan_2 :email "mailto:jan@example.org".
```

Merk op dat een verschillende URL op zich *niet* voldoende is om te besluiten of twee concepten al dan niet verschillend zijn. Omdat de property `:email` echter invers-functioneel is, kunnen twee verschillende subjecten geen identiek object kunnen hebben. Anders gezegd, doordat we twee RDF-triples tegenkomen die hetzelfde object hebben in combinatie met dezelfde invers-functionele property, kunnen we afleiden dat `:Jan_1` en `:Jan_2` gelijk zijn:

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

1.4 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 10,000 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 client en een SPARQL-server. De syntax van de SPARQL-querytaal is gelijkaardig en gebaseerd op SQL.

Merk op dat we ook hier geen volledig overzicht geven van de SPARQL-specificatie, maar enkel wat 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 zoals in Turtle;
- `SELECT`: namen van variabelen 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 zoals `?x`. Het `WHERE`-gedeelte is gelijkaardig aan een lijst van RDF-triples in Turtle-stijl, waarbij sommige componenten vervangen zijn door variabelen. Binnen het semantisch web worden SPARQL-queries typisch verstuurd over het HTTP-protocol. Daarom definieert SPARQL naast ook het protocol om SPARQL-vragen en antwoorden te versturen. Een SPARQL-query versturen naar een SPARQL-endpoint kan eenvoudig aan de hand van een HTTP `GET`-request. In de URL van dit request plaats je een URL-geëncodeerde SPARQL query, bijvoorbeeld: *http://example.com/sparql?query=SELECT+%3Fname...*

2 Opgave

2.1 Een ontologie maken

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 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.

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. Zorg evoor

dat de afbeelding die je annoteert beschikbaar is op het Web. Alle RDF-data dient in Turtle opgesteld te worden. Gebruik als naamruimte van je eigen concepten *http:/itech.ugent.be/ontology/<groepsnummer>/*. Gebruik als bestandsnamen `foto_ontologie.ttl` en `foto_instanties.ttl`.

2.2 Ontologische concepten linken

Nadat een eigen foto-ontologie werd opgesteld, is het de bedoeling om concepten die daarin gedefinieerd werden te linken met bestaande properties. Meer bepaald leg je in deze opgave 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 allerhande resources. 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 `foaf:Image`, idie een digitale afbeelding identificeert. 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 Turtle opgesteld te worden. Gebruik de bestandsnaam `mapping.ttl`.

2.3 Een semantische toepassing bouwen

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. Zoeken enkel mag gebeuren op basis van Dublin Core-properties en de klasse `foaf:Image`, niet van je eigen concepten uit 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 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 op Minerva. De Turtle-bestanden met RDF-triples van Opgaven 1 en 2 worden dan in de zelfde folder als deze bestanden geplaatst. Bij het opstarten zullen alle Turtle-bestanden in dezelfde folder automatisch ingeladen worden. Om het SPARQL-endpoint op te starten gebruik je:

```
java -jar itech-jar-with-dependencies.jar <poort>
```

Kies als poortnummer je groepsnummer + 11000. Het SPARQL-endpoint kan dan getest worden via de volgende URL: *http://localhost:110XX/sparql/query?query=...*.

Je applicatie 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 weergeven.
- iedere foto slechts eenmaal weergeven, 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.

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*.

Als je werkt op Athena, voer dan de server en de webapplicatie uit op dezelfde node. Tussen verschillende Athena-servers is er immers geen communicatie mogelijk. Vanuit de opdrachtprompt in Aptana Studio kan je het SPARQL-endpoint starten; dan ben je zeker dat dit op dezelfde machine als de server draait.

2.4 Indienen

Dien de drie opgaven via de Minerva-dropbox samen in als `itech-p3-g<groep>.zip`. Voor Opgave 3 sluit je de volledige Ruby on Rails-projectfolder in, **zonder** de SPARQL-endpoint-software.

Met originaliteit en gebruiksvriendelijkheid (ontologie en webtoepassing) kan je bonuspunten verdienen, die eventueel gemaakte fouten kunnen compenseren. Dit betekent echter ook dat geen punten afgetrokken worden als deze aspecten ontbreken.