Onderzoeksrapport

Verrijking zoekmachine

Expertise Management Wiki's

afstudeerscriptie van Joos Mesie,
studentnummer 48580
in het kader van de opleiding HBO-ICT aan de
HZ University of Applied Sciences, Vlissingen
juli 2016

Onderzoeksrapport

Verrijking zoekmachine

Expertise Management Wiki’s

Versiebeheer

Versie	Auteur	Omschrijving
0.1	J.	Onderzoeksvoorstel, eigen concepten
0.2	J.	Onderzoeksvoorstel, feedback Mischa verwerkt
0.3	J.	Onderzoeksvoorstel, feedback Wouter verwerkt
1.0	J.	Onderzoeksvoorstel, eerste formele concept
9.0	J.	Onderzoeksrapport, eigen concepten
10.0	J.	Onderzoeksrapport, eerste formele concept
10.1	J.	Actualisering methode en verwerking voorlopige resultaten
20.0	J.	Onderzoeksrapport, tweede formele concept
20.1	J.	Onderzoeksrapport, feedback Mischa verwerkt
30.0	J.	Onderzoeksrapport, definitief, eerste kans

Samenvatting

De Expertise Management Methodologie (EMM) is een methodologie ontwikkeld om kennis en ervaring te structureren en aan elkaar te verbinden. Deze kennis wordt vastgelegd op basis van Semantic MediaWiki (SMW) technologie in zogenaamde EMM wiki's.

De zoekmachine van deze wiki's werkt momenteel op basis van een index beheerd door een real-time, gedistribueerde zoek en analyse engine, namelijk Elasticsearch (ES). De zoekmachine zoekt echter alleen naar letterlijke overeenkomsten van de door de gebruiker ingevoerde zoektermen.

De vraag die in dit onderzoek wordt gesteld en beantwoord is hoe men relevantere zoekresultaten kan presenteren op basis van semantiek en statistiek.

Momenteel zijn er twee thesauri welke voor dit onderzoek van belang zijn, namelijk:

SKOS, beheerd door experts. Deze thesaurus is gespecificeerd in een Simple Knowledge Organization System en bevat allerlei domein specifieke termen (concepten) welke handmatig zijn aangelegd, met eventuele onderlinge relaties.

Non-SKOS (NSKOS), ontwikkeld door mede afstudeerder Thien Tin Lam Ngoc op basis van statistiek en big data. Deze thesaurus bevat alle andere (niet in SKOS) termen uit de content van de EMM wiki's. Deze termen worden genormaliseerd (als stam) en bevatten geen stopwoorden.

Deze twee thesauri worden samengevoegd tot een nieuwe thesaurus welke relaties tussen NSKOS termen en SKOS concepten beschrijft.

Verrijkte SKOS (SKOS*), conceptueel ontwikkeld door mede afstudeerder Thien Tin Lam Ngoc. Relaties bestaan uit term frequenties (welke het gewicht van de relatie representeren) en de herkomst (het brondocument).

Om te zorgen dat de zoekmachine relevantere zoekresultaten kan presenteren, wordt op basis van de relaties beschreven in SKOS*, term expansie toegepast. Hierbij worden de originele zoektermen uitgebreid met gerelateerde termen.

Op basis van de resultaten uit dit onderzoek, kan geconcludeerd worden dat term expansie een positief effect kan hebben op de precisie en relevantie van zoekresultaten. Soortgelijk onderzoek onderstreept deze conclusie.

Summary

The Expertise Management Methodology (EMM) is a methodology developed to structure and relate knowledge and experience. This is then stored in so called EMM wiki's based on Semantic MediaWiki (SMW) technology.

Currently the search engine of these wiki's uses an index which is maintained by a real-time distributed search and analysis engine, namely Elasticsearch (ES). However, the search engine only finds literal matches based on the user's search terms.

The question posed and answered by this study, is how more relevant search results can be presented, based on semantics and statistics.

There are currently two thesauri which are of interest to this study, namely:

SKOS, maintained by experts. This thesaurus is specified in a Simple Knowledge Organization System and contains various domain specific terms (concepts) and any relations between these concepts, which are manually added.

Non-SKOS (NSKOS), developed by fellow grad student Thien Tin Lam Ngoc, based on statistics and big data. This thesaurus contains all other (not in SKOS) terms in the content of the EMM wiki's. These terms are normalized (root form) and stop words are discarded.

These two thesauri are combined to form a new thesaurus, which describes relations between NSKOS terms and SKOS concepts.

Enriched SKOS (SKOS*), conceptually developed by fellow grad student Thien Tin Lam Ngoc. Relations consist of term frequencies (which represent the weight of a relation) and the origin (the document pertaining the relation).

To ensure that the search engine can present more relevant results, we apply term expansion based on the relation described in SKOS*. The original search terms are augmented with related terms.

Based on the results in this study, we can conclude that applying term expansion has a positive effect on precision and recall. Similar research underscores this conclusion.

Voorwoord

In de periode van februari 2016 tot en met juli 2016, heb ik in het kader van afstuderen, onderzoek gedaan voor het kenniscentrum Expertise en Valorisatie Management (EVM). Het onderzoek beantwoord de vraag hoe men relevantere zoekresultaten kan presenteren op basis van semantiek en statistiek. Dit rapport is één van de resultaten van dit onderzoek.

Hierbij ben ik ondersteund en begeleid door onder andere, Hans de Bruin, Wouter Everse, Mischa Beckers en Jethro Waanders, welke ondanks drukte en andere verplichtingen toch tijd vrij hebben gemaakt voor mij. Bij deze wil ik jullie hiervoor bedanken.

Ik wens u veel leesplezier.

Joos Mesie,

Vlissingen, 4 juli 2016

Inhoud

1 Inleiding

Eén probleem dat bedrijven ondervinden, is dat wanneer experts het bedrijf verlaten (hetzij door pensioen of andere redenen), zij vaak ook veel expertise en ervaring (zowel good- als bad practices) meenemen. Hoe kan men deze kennis en ervaring borgen en sterker nog, toegankelijk maken voor de rest van het bedrijf?

De Expertise Management Methodologie (EMM) is een methodologie ontwikkeld om kennis en ervaring te structureren en aan elkaar te verbinden. Deze kennis bestaat uit een grote en groeiende hoeveelheid gegevens welke zijn vastgelegd in zogenaamde EMM wiki's, op basis van Semantic MediaWiki (SMW) technologie (Rooij, 2014).

Er zijn momenteel verschillende EMM wiki's, waaronder www.deltaexpertise.nl. Het zoekalgoritme van deze wiki's maakt op het moment van schrijven, gebruik van een index welke deels is gebaseerd op een domein specifieke thesaurus[1], welke is gespecificeerd in een Simple Knowledge Organization System (SKOS, zie H1.3.1.3). In deze thesaurus (voortaan SKOS) zijn allerlei domein specifieke termen handmatig vastgelegd met eventuele onderlinge relaties.

Verder wordt er een zogenaamde Non-SKOS (voortaan NSKOS) ontwikkeld door mede afstudeerder Thien Tin Lam Ngoc (voortaan Tin), op basis van statistiek en big data. Hierin worden termen opgenomen uit alle beschikbare documenten van één of meer EMM wiki's, die zelf niet in SKOS voorkomen maar die wel met termen uit SKOS in een document zitten (waardoor een relatie wordt gesuggereerd). Deze termen worden genormaliseerd en bevatten geen stopwoorden.

Er zijn nu twee thesauri, één met SKOS termen en één met NSKOS termen. De relatie tussen SKOS en NSKOS termen bestaat uit een frequentie (het gewicht van de relatie) en de herkomst (het brondocument). Deze relaties worden samen met SKOS, beschreven in de "verrijkte" SKOS (voortaan SKOS*). Tin ontwikkeld het concept SKOS* als eindproduct, de vorm hiervan staat echter nog niet vast.

Stel voor dat in veel documenten op www.deltaexpertise.nl over *dijkbouw*, het woord *keien* voorkomt. Statistisch gezien zou dat kunnen wijzen op een relatie tussen het woord *keien* en *dijkbouw*. Wat de relatie precies is weten we niet, maar de relatie lijkt er te zijn. Stel verder voor dat het woord *dijkbouw* reeds bestaat in SKOS, maar het woord *keien* niet. In dat geval zal de term *keien* worden opgeslagen in NSKOS als relatie met *dijkbouw*, met de frequentie waarin het woord *keien* voorkomt en over welk brondocument het gaat.

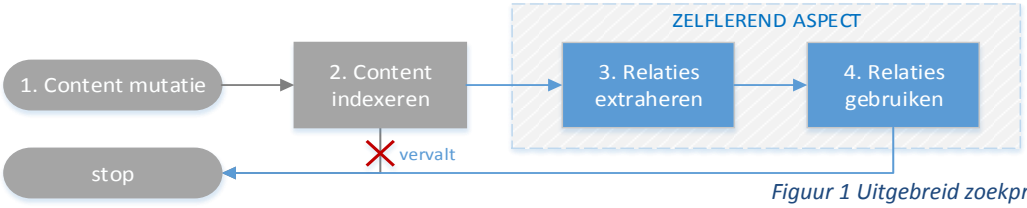
Momenteel zijn er twee projecten betrokken bij de zoekmachine van de EMM wiki's, namelijk HZ search en HZ indexing welke respectievelijk in de front-end en de back-end van de EMM wiki's actief zijn. Mijn opdracht zal raakvlakken hebben met beide. Tin zijn opdracht heeft voornamelijk met de back-end te maken. Op het moment van schrijven maakt de zoekmachine gebruik van een index die o.a. op SKOS is gebaseerd. De zoekresultaten zijn echter beperkt tot documenten waar de ingevoerde zoekterm letterlijk in staat.

1.1 Aanleiding

Om alle gegevens in de EMM wiki's gemakkelijk vindbaar en inzichtelijk te maken voor gebruikers, moet de zoekmachine op basis van een query, contextueel relevante en/of synonieme informatie kunnen weergeven. Verder, omdat de hoeveelheid gegevens zal blijven groeien, is het van belang dat de zoekmachine nieuwe content aan bestaande content kan relateren.

1 Een database met woordenlijsten waarvan de woorden overeenkomen, synoniemen zijn of gerelateerde concepten betreffen.

Dit wordt ook wel het *zelflerend* aspect van de zoekmachine genoemd. Het huidige zoekproces zal daarvoor uitgebreid moeten worden (aangegeven in blauw in onderstaand figuur).



Figuur 1 Uitgebreid zoekproces

1. De content wordt gemuteerd door toevoeging, wijziging, verwijdering of (ont)koppeling.
2. De gemuteerde content wordt verwerkt in de index.
3. Relaties worden uit de gemuteerde content geëxtraheerd en verwerkt in SKOS*.
4. De zoekmachine gebruik SKOS* om relevantere zoekresultaten te presenteren.

LET OP !

Bovenstaande stappen worden momenteel handmatig in werking gezet.

Het onderzoek van Tin (het conceptuele SKOS*) betreft grotendeels stap 3, relaties extraheren. Mijn onderzoek betreft grotendeels stap 4, relaties gebruiken.

1.2 Probleemstelling

Gebruikers moeten gemakkelijk relevante informatie kunnen opvragen aan de hand van een zoekterm. Omdat de EMM wiki's een grote (en groeiende) hoeveelheid gegevens bevatten wordt dit een steeds kritieker punt. Momenteel zijn de zoekresultaten beperkt tot documenten waar de ingevoerde zoekterm letterlijk in staat. Er is dus geen sprake van aanvullende relevante informatie.

Het conceptuele SKOS* is het vertrekpunt van mijn onderzoek waarbij de aanname is gemaakt dat deze reeds is ontwikkeld in een bepaalde vorm. De definitieve vorm moet nog vastgelegd worden. Vervolgens moet SKOS* verwerkt worden in het zoekproces door de zoekresultaten hiermee te optimaliseren.

1.2.1 Doelstelling

Het doel van dit onderzoek is een prototype zoekmachine ontwikkelen welke gebruikt maakt van SKOS*. Zo wordt het zoekproces uitgebreid met het *zelflerend* aspect.

Stel voor een gebruiker zoekt op de term *stuwmeer*, in plaats van documenten die letterlijk dit woord bevatten zullen de zoekresultaten aangevuld zijn met gerelateerde documenten. Bijvoorbeeld documenten over andere vormen van waterkeringen zoals *dijken* en *stormvloedkeringen*. Of documenten over *veiligheid in Nederland*.

Uiteindelijk wil men het *zelflerend* aspect automatiseren door toevoeging van de volgende stappen:

1. het zoekproces kan zelf constateren wanneer een content mutatie plaatsvindt;
2. de gemuteerde content wordt automatisch verwerkt in de index;
3. SKOS* wordt bijgewerkt met de gemuteerde content;

1.2.2 Relevantie

1.2.2.1 Bedrijfsrelevantie

Uit interesse heeft Hans de Bruin EMM ontwikkeld. Om EMM in een bruikbare vorm te kunnen realiseren zijn verschillende technieken en middelen gekozen waaronder SMW technologie. Zo kan nieuwe kennis gekoppeld worden aan bestaande kennis.

De HZ past EMM ook toe in de vorm van een SMW wiki, namelijk de HZ Portfolio wiki. Studenten kunnen daarin voor de studie relevante zaken vastleggen.

1.2.2.2 Maatschappelijke relevantie

Delen van de provincie Zeeland zijn door de Rijksoverheid aangewezen als krimpgebieden. Ook Vlissingen is één van de gemeenten waar het aantal inwoners is afgenomen de afgelopen jaren. Een mogelijk hulpmiddel om verdere krimping tegen te werken door groei te stimuleren, is door innovatie (in dit geval, door het ontwikkelen van EMM).

EMM biedt bedrijven de mogelijkheid om kennis en expertise te waarborgen en toegankelijk te maken binnen en buiten het bedrijf. (zie H1 Inleiding).

1.2.2.3 Kennisgebied en theoretische relevantie

Het onderzoek vindt plaatst in het kennisgebied Information Retrieval. Een (brede) definitie kan zijn:

> *Information retrieval is het vinden van materie (meestal documenten), binnen grote collecties, van een ongestructureerde natuur (meestal tekst), welk voldoet aan een informatie behoefte. (Manning, Raghavan, & Schütze, 2009)*

Het onderzoek komt specifiek in aanraking met zoekalgoritmiek, indexering en semantische data. Er moet namelijk gekeken worden hoe de zoekmachine kan werken met SKOS* en hoe de zoekresultaten geoptimaliseerd kunnen worden met de gerelateerde termen beschreven in NSKOS. Hiervoor zal het zoekalgoritme aangepast of aangevuld moeten worden in combinatie met het indexeringsproces.

Verder komen ook twee aspecten van het semantische web aan bod, namelijk thesauri en Simple Knowledge Organization Systems (SKOS) wat in principe een domein specifieke thesaurus is, alleen dan gespecificeerd in een Simple Knowledge Organization System (SKOS).

1.2.3 Vraagstelling

In het kort ter herhaling, wat men in dit onderzoek verstaat onder SKOS* (voor de uitgebreide beschrijving zie H1 Inleiding).

- SKOS* beschrijft relaties tussen SKOS en NSKOS termen middels een frequentie en herkomst;

1.2.3.1 Centrale vraag

De vraag die dit onderzoek probeert te beantwoorden luidt:

> Hoe kan de zoekmachine relevantere zoekresultaten presenteren door gebruik te maken van SKOS*?

1.2.3.2 Deelvragen

De hiervoor genoemde vraag valt uiteen in de volgende deelvragen:

1. Hoe werkt de huidige zoekmachine van input tot en met output?

2. Welke vorm van SKOS* is het meest geschikt als input voor de zoekmachine?

3. Hoe kan er in het bepalen van de relevantie van de zoekresultaten rekening gehouden worden met SKOS*?

1.2.4 Projectgrenzen en randvoorwaarden

Dit project heeft de volgende projectgrenzen:

- Realisatie en implementatie is beperkt tot conceptbewijzen;

- Er wordt alleen gekeken naar hoe SKOS* gebruikt kan worden om relevantere zoekresultaten te presenteren, het *zelflerend* aspect en eventuele optimalisaties aan de huidige zoekmachine wat betreft Elasticsearch, worden buiten beschouwing gelaten. Deze worden echter wel beschreven in een adviesrapport.

Aan het onderzoek worden de volgende randvoorwaarden gesteld:

- De zoekmachine is onderdeel van een bestaand systeem (namelijk de EMM wiki's, gebaseerd op SMW technologie) en zal dus met bepaalde componenten en technieken moeten samenwerken;

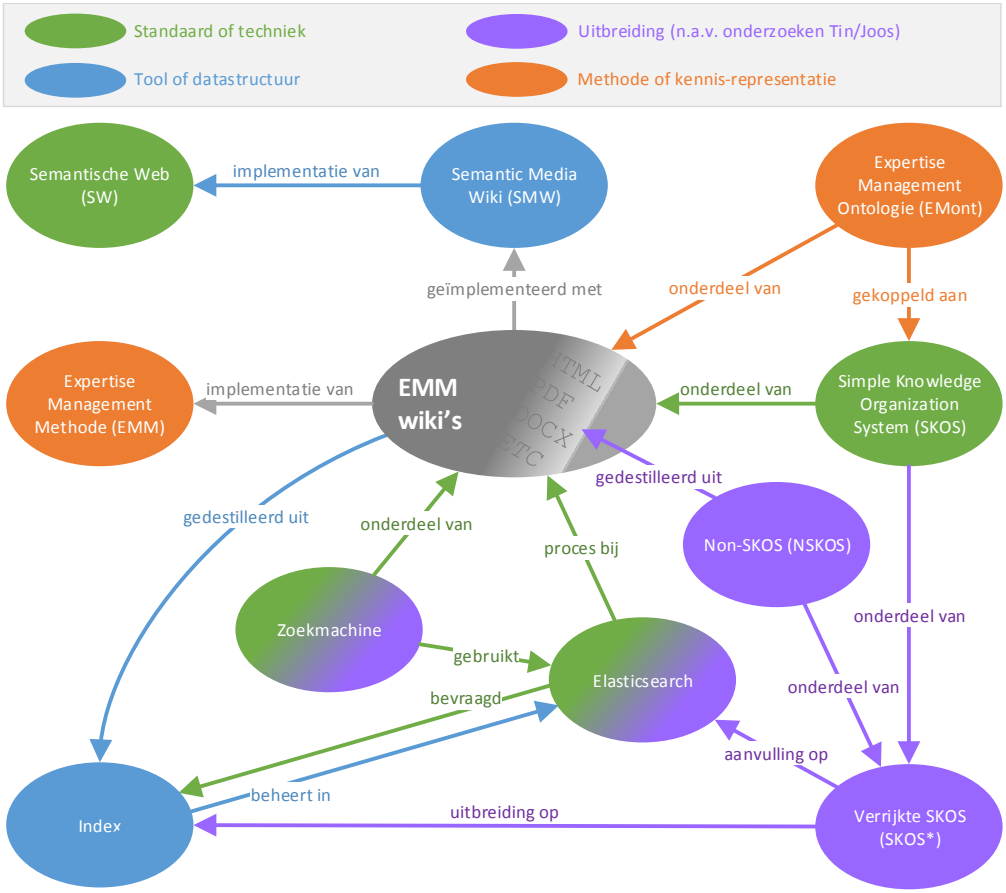
- Elasticsearch beheerd en bevraagd de index.

1.3 Theoretisch kader

Dit onderzoek gaat verder met eerder verricht werk en onderzoek door zowel het lectoraat, de HZ en andere afstudeerders, op het gebied van expertise management.

1.3.1 Belangrijkste concepten in samenhang

In Figuur 2 hieronder, zijn de belangrijkste concepten en de samenhang hiertussen weergegeven. Information retrieval is het overkoepelende concept waarin de EMM wiki's als use-case gezien kunnen worden.



Figuur 2 – Totaalbeeld belangrijkste concepten

In de volgende hoofdstukken wordt elk deel verder toegelicht.

H1.3.1.1 Semantische web
H1.3.1.2 Semantic MediaWiki (SMW)
H1.3.1.3 Simple Knowledge Organization System (SKOS)
H1.3.1.4 Non-SKOS (NSKOS)
H1.3.1.5 Verrijkte SKOS (SKOS*)
H1.3.1.6 Elastic Search
H1.3.1.7 Zoekmachine
H1.3.1.8 Index
H1.3.1.9 Expertise Management Ontologie (EMont)
H1.3.1.10 Expertise Management Methode (EMM)

1.3.1.1 Semantische web

Het semantische web is een heel groot begrip. In dit onderzoek wordt een kleine subset hieruit gebruikt, namelijk de zogenaamde thesauri en een standaard manier om kennis organisatie systemen te presenteren, namelijk de Simple Knowledge Organization System (SKOS) specificaties (zie H1.3.1.3). Toch is het wellicht handig om kort uit te leggen wat men verstaat onder het semantische web.

Het web zoals nu bekend (ook wel web 2.0) is in principe niks anders dan een grote verzamelbak met documenten (hetzij html-, pdf- docx documenten, wat dan ook). Het gaat hier voornamelijk om platte tekst, er zijn geen relaties tussen deze documenten.

> *De meeste informatie op het web is vertegenwoordigd in natuurlijke taal, wat begrijpelijk is voor mensen. Dezelfde informatie is echter wel leesbaar, maar niet begrijpelijk, voor machines. (Wahlster & Denge, 2006)*

Het semantische web (ook wel web 3.0) is een verzameling van entiteiten. Tussen deze entiteiten liggen mens- en belangrijker, machine leesbare uitdrukkingen van een relatie tussen een bron en een object. Met deze relaties wordt het mogelijk om EMM toe te passen via het web.

> *Het Semantische web is gebaseerd op de inhoudelijke beschrijving van documenten, met gestandaardiseerde woordenlijsten die machine begrijpelijke semantiek bieden. Het resultaat is de transformatie van een web van links, naar een web van betekenis. (Wahlster & Denge, 2006)*

Het semantische web biedt een gemeenschappelijk kader waarmee gegevens kruislings gedeeld en hergebruikt kunnen worden over applicaties, het bedrijfsleven en andere gemeenschappen.

1.3.1.2 Semantic MediaWiki (SMW)

Semantic MediaWiki is een open-source uitbreiding van Media Wiki (het platform voor bijvoorbeeld Wikipedia).

> *Alle gegevens in SMW kunnen gemakkelijk gepubliceerd worden via het Semantische Web, zodat andere applicaties en systemen naadloos deze gegevens kunnen gebruiken. (Semantic MediaWiki, 2016)*

> *Het doel van SMW is om semantische technieken naadloos te integreren in MediaWiki zodat dit voor een breed (niet deskundig) publiek bruikbaar wordt. (Krötzsch, Vrandečić, & Völkel, 2006)*

SMW is een implementatie van het Semantische Web. De EMM wiki's zijn geïmplementeerd met SMW.

1.3.1.3 Simple Knowledge Organization System (SKOS)

SKOS is een domein specifieke thesaurus, die is gespecificeerd in een Simple Knowledge Organization System (SKOS), welke weer formeel gedefinieerd is als een ontologie[2]. SKOS bestaat uit zogenaamde SKOS concepten, welke geïdentificeerd worden aan de hand van URI's.

2 Een informatiestructuur die alle relevante entiteiten en onderliggende relaties en regel binnen een gedefinieerd domein bevat.

SKOS is een model om de basis structuur en content van een set van concepten uit te drukken, met optioneel semantische relaties tussen deze concepten. (Miles, Matthews, Wilson, & Brickley, 2005)

SKOS concepten kunnen o.a. gekoppeld worden aan andere SKOS concepten via semantische relaties welke opgedeeld zijn in twee categorieën, namelijk hiërarchische en associatieve relaties. Hiermee kunnen dus relaties tussen entiteiten van het Semantische Web beschreven worden.

De betekenis van een concept is niet alleen gedefinieerd door de woorden beschreven in de labels, maar ook door de relaties met andere concepten in de vocabulaire. (Isaac & Summers, 2009)

Met betrekking tot de opdracht, beschrijft SKOS zogenaamde *Propositional* (ook wel *Descriptive*, *Declarative* of *Knowing that*) kennis.

Propositional kennis is het simpelweg weten, of hebben van kennis over een bepaald onderwerp. In andere woorden, kennis van iets, ten opzichte van kennis van hoe iets moet. Wiskundige uitdrukkingen zijn een goed voorbeeld hiervan. (Rooij, 2014)

1.3.1.4 Non-SKOS (NSKOS)

SKOS is niet volledig voor wat men wil realiseren. Om te zorgen dat gerelateerde termen toch op één of andere manier gekoppeld kunnen worden wordt er een NSKOS ontwikkeld.

NSKOS bevat termen uit alle beschikbare documenten die niet in SKOS voorkomen maar die wel met termen uit SKOS in één document zitten (waardoor een relatie wordt gesuggereerd). Deze termen worden genormaliseerd en bevatten geen stopwoorden. De relatie wordt beschreven in de vorm van een frequentie (de mate waarin NSKOS termen voorkomen m.b.t. een bepaalde SKOS term) en de herkomst (bronbestand).

Deze termen worden dus op basis van statistiek uit een grote hoeveelheid documenten en teksten gehaald, vandaar dat het gaat over *mogelijke* relaties en waar de frequentie een *aanduiding* is van de sterkte van de relatie en geen stellig weten.

1.3.1.5 Verrijkte SKOS (SKOS)*

Simpel gezegd, bestaat SKOS* uit SKOS en NSKOS. Zoals ook in H1 Inleiding beschreven, ontwikkeld Tin het concept SKOS* als eindproduct, de vorm hiervan staat echter nog niet vast. Het is op het moment van schrijven dus nog niet bekend of dit één groot bestand wordt of twee losse.

1.3.1.6 Elastic Search (ES)

Elasticsearch is een open-source, gedistribueerde en makkelijk schaalbare, full-tekst zoekmachine. ES is toegankelijk via een uitgebreide en gedetailleerde API waarmee men snelle zoekopdrachten op grote hoeveelheden (complexe) data kan uitvoeren. Tevens biedt het mogelijkheden voor het analyseren van deze data in nuttige vormen.

1.3.1.7 Zoekmachine

De zoekmachine van de EMM wiki's voorziet gebruikers momenteel in basis zoekfunctionaliteit. Ingevulde zoektermen worden letterlijk gezocht in de index. De zoekresultaten die worden weergeven bevatten de zoekterm dus letterlijk.

De zoekmachine gebruikt Elasticsearch om de index te bevragen en zodoende resultaten te verkrijgen.

Ten eerste wordt een normale full-tekst zoekopdracht uitgevoerd op de titel, content, context en eventueel andere relevante attributen. Het resultaat is een lijst met documenten met een toegewezen relevantiescore. Hierna wordt de overkoepelende context van de zoekactie bepaald waarin het merendeel van de gevonden documenten toebehoord. (Vogels, 2014)

De werking van de zoekmachine wordt beschreven als antwoord op de eerste deelvraag (zie H3.1). Mijn onderzoek betreft o.a. de uitbreiding van de huidige machine met SKOS*.

1.3.1.8 Index

De index is een datastructuur waarin verschillende soorten gegevens worden beschreven, waaronder de SKOS concepten. De NSKOS termen worden hier aan toegevoegd. Ze wordt SKOS* dus onderdeel van de index.

Elasticsearch beheerd en bevraagd de index. De zoekmachine gebruikt de index en de daarin beschreven relaties om de zoekresultaten te optimaliseren.

1.3.1.9 Expertise Management Ontologie (EMont)

De EMont is ook een ontologie, welke een andere vorm van kennis beschrijft, namelijk de zogenaamde *Procedural* (ook wel *Non-propositional* of *Knowing how*) kennis.

Procedural kennis is kennis die gebruikt kan worden, of anders gezegd, kan worden toegepast, bijvoorbeeld op een probleem. Andere (al dan niet simpele) manieren om Propositional- en Procedural kennis te omschrijven zijn respectievelijk educatie en (werk) ervaring.

Procedural kennis wordt dus verzameld door dingen 'te doen' en wordt op een soortgelijke manier vastgelegd als *Propositional* kennis. SKOS en de EMont zijn aan elkaar gekoppeld om een geheel aan kennis aan te kunnen bieden.

Door deze vormen van kennis te combineren wordt er inzicht verkregen in wat iemand kan en hoe iemand dit doet, waardoor de kennis van experts dus vastgelegd kan worden. (Rooij, 2014)

1.3.1.10 Expertise Management Methode (EMM)

De Expertise Management Methodologie (EMM) is een methodologie ontwikkeld om kennis te structureren en aan elkaar te verbinden. Hierdoor kan expertise die is opgedaan inzichtelijk gemaakt worden voor anderen.

2 Methode en materialen

In dit hoofdstuk wordt per deelvraag beschreven hoe gegevens worden verzameld en geanalyseerd om tot een antwoord te komen op de desbetreffende deelvraag. In H2.4 is een totaalbeeld opgenomen ter overzicht.

2.1 Deelvraag 1

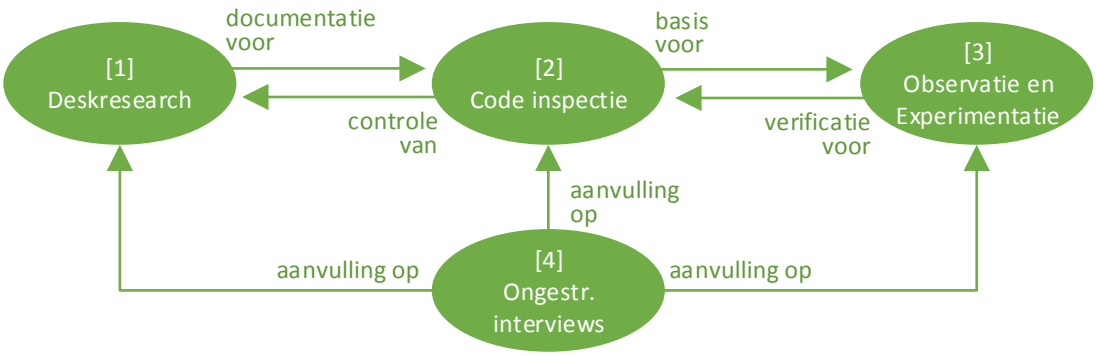
1. *Hoe werkt de huidige zoekmachine van input tot en met output?*

2.1.1 Meetinstrumenten en operationalisatie

Voordat de huidige zoekmachine verbeterd kan worden is het van belang te begrijpen hoe deze nu bestaat. Het indexeringsproces, de input (zoekterm), verwerking (resultaten ophalen en sortering) en output (presenteren van resultaten) van de zoekmachine, worden in deze deelvraag onderzocht en beschreven.

Er wordt gebruik gemaakt van (zie ook Figuur 3):

1. Deskresearch (voornamelijk bestaande documentatie);
2. Code inspectie van de zoekmachine;
3. Observatie van- en experimentatie met- de EMM wiki's zoekmachine;
4. Ongestructureerde interviews (het stellen van gerichte vragen wanneer nodig).



Figuur 3 – Samenhang meetinstrumenten, deelvraag 1

De verwachting is dat de bestaande documentatie een beter beeld geeft van de uiteindelijke implementatie. Tegelijk zal het inzicht geven waarom de implementatie zo is uitgevoerd. Het observeren en experimenteren, zal het geheel nog tastbaarder maken. Tussendoor zullen er geheid specifieke vragen naar voren komen welke middels de ongestructureerde interviews gesteld worden.

2.1.1.1 Onderbouwing keuze meetinstrument

Omdat de zoekmachine reeds bestaat wordt er code inspectie in combinatie met observatie en experimentatie uitgevoerd, om te bepalen hoe de zoekmachine technisch werkt.

De deskresearch en ongestructureerde interviews hebben een ondersteunende rol wanneer er vraag is naar meer informatie.

2.1.2 Analysemethodes

Er wordt ten eerste een visuele representatie van de globale werking van de zoekmachine gemaakt (zie Figuur 4). De hoofdfunctionaliteiten hiervan worden specifieker vastgelegd in aparte visuele representaties met aanvullende beschrijving.

Figuur 4 – Toegepaste analysemethode, deelvraag 1

De visualisaties kunnen zodoende gemakkelijk geverifieerd worden door de bedrijfsbegeleider.

2.2 Deelvraag 2

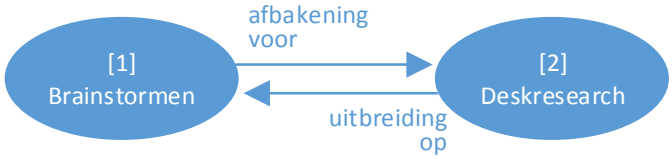
2. *Welke vorm van SKOS* is het meest geschikt als input voor de zoekmachine?*

2.2.1 Meetinstrumenten en operationalisatie

Nu bekend is hoe de zoekmachine werkt, kan gekeken worden in welke vorm het SKOS* het meest geschikt is als input voor de zoekmachine.

Er wordt gebruik gemaakt van (zie ook Figuur 5):

1. Brainstormen, bedenken welke mogelijke vormen er überhaupt zijn;
2. Deskresearch, aanvullend op het brainstormen en, omdat het SKOS* semantische informatie bevat (de relaties tussen allerlei SKOS en NSKOS termen), wordt met deskresearch gekeken naar manieren om semantische data te bewaren. Hiervoor wordt een zoekplan opgesteld.



Figuur 5 – Samenhang meetinstrumenten, deelvraag 2

De verwachting is dat door te brainstormen snel een afbakening gemaakt kan worden, wat vervolgens verder uitgewerkt kan worden middels deskresearch. Omdat het over semantische informatie gaat, is het waarschijnlijk dat uit de deskresearch ook concrete suggesties komen wat betreft opslag.

2.2.1.1 Onderbouwing keuze meetinstrument

Brainstormen is hier toegepast om een stap terug te nemen en na te denken over wat nou mogelijke vormen zijn. Met deskresearch wordt dieper ingegaan op deze mogelijke vormen en hoe semantische data opgeslagen kan worden.

2.2.2 Analysemethodes

De bevindingen worden verwerkt in voor- en nadelen per mogelijke vorm, zie ook Figuur 6. Zo wordt het bespreekbaar en vergelijkbaar gemaakt. In overleg met de bedrijfsbegeleider kan op basis hiervan discussie ontstaan en een passende vorm worden gekozen. Vervolgens wordt deze vorm in het klein ook technisch uitgewerkt als conceptbewijs.

Figuur 6 – Toegepaste analysemethodes, deelvraag 2

2.3 Deelvraag 3

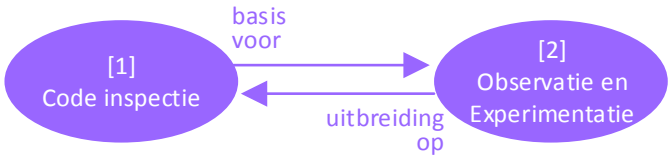
3. *Hoe kan er in het bepalen van de relevantie van de zoekresultaten rekening gehouden worden met SKOS*?*

2.3.1 Meetinstrumenten en operationalisatie

Nu de vorm van SKOS* bekend is resteert de vraag, hoe kunnen de relaties beschreven in SKOS*, gebruikt worden om relevantere zoekresultaten te presenteren.

In eerste instantie wordt onderzocht welke relevantiebepaling momenteel wordt gehanteerd. Hiervoor wordt gebruik gemaakt van (zie ook Figuur 7):

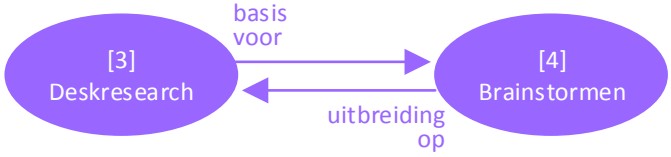
1. Code inspectie, om de huidige relevantiebepaling van zoekresultaten in kaart te brengen;
2. Observatie en experimentatie, ter ondersteuning van de code inspectie (1).



Figuur 7 – Samenhang meetinstrument 1 en 2, deelvraag 3

Omdat SKOS* o.a. relaties op basis van frequenties en herkomst beschrijft, is hiervoor een andere relevantiebepaling nodig en moet er een vertaalslag (of normalisatie) gemaakt worden tussen de huidige relevantiebepaling (1 en 2) en relevantie op basis van SKOS* (3 en 4). Hiervoor wordt gebruik gemaakt van (zie ook Figuur 8):

3. Deskresearch, om te onderzoeken wat relevantie is, wat iets over relevantie zegt en eventuele andere onderzoeken en/of bevindingen m.b.t. relevantiebepaling;
4. Brainstormen, hoe SKOS* te introduceren in de huidige relevantiebepaling.



Figuur 8 – Samenhang meetinstrument 3 en 4, deelvraag 3

Er wordt een requirementsanalyse uitgevoerd welke in eerste instantie naar eigen inzicht worden beschreven en geprioriteerd middels de MoSCoW methode. In overleg met de bedrijfsbegeleider worden deze eventueel aangevuld en /of anders geprioriteerd.

Ten behoeve van tijd is de keuze gemaakt om een vorm van Rapid Application Development (RAD) toe te passen, waar iteratieve ontwikkeling vooraan staat. In plaats van vooraf alles uit te denken en te ontwerpen, wordt het ontwerpen verweven met het daadwerkelijke ontwikkelen. De functionele- en technische uitwerking worden, samen met de requirements, opgenomen in een ontwerpspecificatie.

Met versiebeheer en het beschrijven en uitvoeren van test-cases wordt de oude situatie vergeleken worden met de nieuwe situatie, gebruik makend van éénzelfde data (sub)set.

Naar verwachting zal de code inspectie, samen met de observatie en experimentatie, duidelijk maken hoe relevantie momenteel wordt bepaald. Vervolgens kan er gedacht worden hoe de gegevens in SKOS* hierbij betrokken kunnen worden.

2.3.1.1 Onderbouwing keuze meetinstrument

Omdat de zoekmachine reeds bestaat wordt er code inspectie in combinatie met observatie en experimentatie uitgevoerd, om te bepalen wat de huidige manier van relevantiebepaling is.

Er wordt deskresearch uitgevoerd om beter inzicht te krijgen in wat relevantie is en welke factoren hier invloed op hebben. Met brainstormen wordt nagedacht hoe SKOS* gebruikt kan worden in de relevantiebepaling.

2.3.2 Analysemethodes

De huidige manier van zoekresultaten sorteren en presenteren wordt onderzocht en aan de hand van een voorbeeld gevisualiseerd. Dit wordt gebruikt om te bepalen hoe SKOS* geïntroduceerd kan worden in de huidige relevantiebepaling.

Figuur 9 – Toegepaste analysemethodes, deelvraag 3

De requirementsanalyse zorgt voor houvast tijdens het ontwikkelen. Tijdens en na ontwikkeling van het prototype, wordt de requirementsanalyse gebruikt als zijnde checklist, om te kijken in hoeverre het prototype hieraan voldoet.

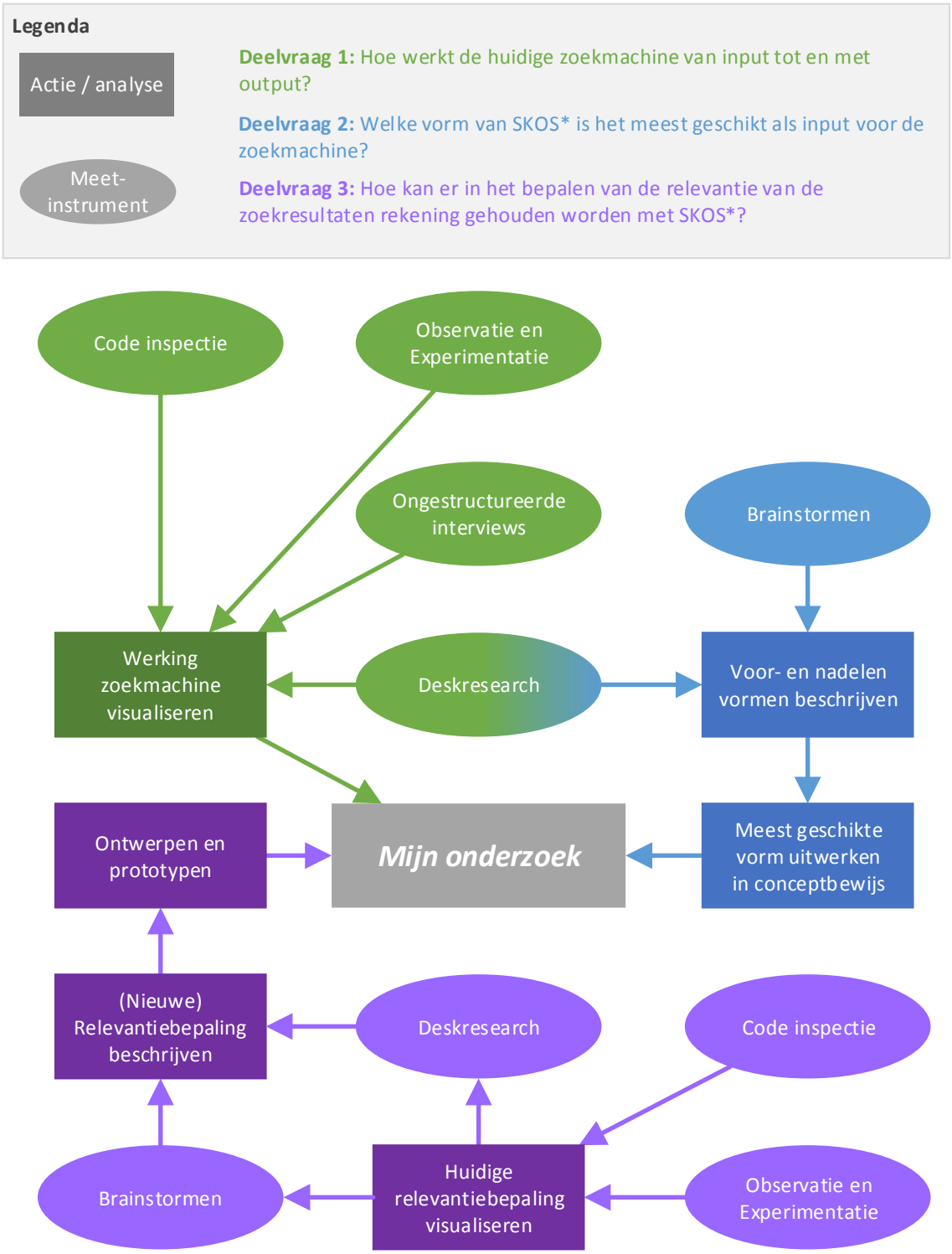
De gevonden manier(en) om SKOS* te gebruiken in de relevantiebepaling worden uitgewerkt in een ontwerpspecificatie met een prototype als conceptbewijs.

Figuur 10 – Toegepaste analysemethodes, deelvraag 3

Middels versiebeheer en het uitvoeren van test-cases, worden de zoekresultaten van de “oude” situatie net met de nieuwe vergeleken. De dataset (en daarmee de zoekresultaten) voor deze vergelijking worden t.z.t. bepaald. Hieruit kan geconcludeerd worden of de gemaakte wijzigingen de doelstelling hebben behaald.

2.4 Totaalbeeld

In Figuur 11 hieronder, is een totaalbeeld van de toegepaste meetinstrumenten, acties en analyses te zien (zie de legenda).



Figuur 11 – Totaalbeeld methode

3 Resultaten en analyse

3.1 Deelvraag 1

1. *Hoe werkt de huidige zoekmachine van input tot en met output?*

3.1.1 Resultaten

Voordat getracht wordt de zoekmachine te verrijken is het van belang te begrijpen hoe deze nu werkt. Specifiek, wat gaat er in, wat voor verwerkingen vinden plaats en tot slot wat voor output is er.

Hieronder wordt per meetinstrument (zie H2.1.1) het bijbehorende resultaat beschreven.

Code inspectie

Met code inspectie werd inzichtelijk hoe de zoekmachine technisch is uitgewerkt. Dit is vastgelegd in deelproduct 1 (zie Bijlage 1: Onderzoek zoekmachine).

Deskresearch

Met de deskresearch werd duidelijk wat het idee achter de zoekmachine is en hoe men getracht heeft deze te realiseren. Tevens was het een aanvulling op de code inspectie.

Observatie en experimentatie

Tijdens het observeren en experimenteren is geconstateerd dat er een mogelijk verschil is tussen de relaties beschreven in SKOS en de relaties die de zoekmachine destilleert uit de gevonden resultaten. Dit verschil ontstaat wellicht doordat de relaties beschreven in SKOS* een stuk complexer kunnen zijn dan alleen parent en child. Dit is genoteerd en in de toekomst aangekaart.

Er zijn ook andere eigenaardigheden, verbeteringen of fouten ontdekt in de werking van de zoekmachine en de EMM wiki's in het algemeen. Deze zijn vervolgens middels het JIRA ticketsysteem gemeld. Hieronder ter referentie de ticket nummers.

EMT-497	Bug met filteren in zoekresultaten;
EMT-500	Gebruik van speciale karakters in zoekveld gaat fout;
EMT-501	Uitgebreid zoeken toevoegen;
EMT-502	Gebruik stopwoorden bij zoeken;
EMT-503	Zoeken komt soms op niet VN-pagina of Speciaal zoeken-pagina uit;
EMT-504	404-error bij zoeken met redirect;
EMT-505	Geen auto-aanvullen zoekveld door JavaScript-error;
EMT-507	Lijst feiten sidebar SKOS-concept onvolledig;
EMT-508	Content pagina alternatieve benaming wijkt af van pagina originele benaming.

Ongestructureerde interviews

Een aantal keren is het voorgekomen dat ondanks de andere meetinstrumenten iets toch onduidelijk bleef. In dit geval zijn er één of meer vragen voorgelegd aan een collega en/of de bedrijfsbegeleider (zie ook Bijlage 5: Transcripties).

3.1.2 Analyse

Hier volgt een samenvatting van de werking van de zoekmachine. Zie Bijlage 1: Onderzoek zoekmachine voor de volledige uitwerking.

De zoekmachine werkt momenteel op basis van een index welke wordt beheerd door Elasticsearch (ES). Elasticsearch (ES) is een real-time, gedistribueerde zoek en analyse engine. ES biedt de mogelijkheid om data (in dit geval documenten op de EMM wiki's zoals html, pdf, etc.) te bevragen via één of een combinatie van:

- Full-tekst, het doorzoeken van tekst om te bepalen welke documenten het best passen bij de aanvraag (welke het meest relevant zijn);
- Gestructureerd zoeken, waarbij het niet om relevantie gaat maar of een document overeenkomt met een vraag of niet. Bijvoorbeeld alle zoekresultaten vanaf 1 januari 2016;
- Real-time analyse van data, zoals het normaliseren van zoektermen.

De zoekmachine vult input van de gebruiker automatisch aan. Elasticsearch wordt gevraagd een lijst met suggesties te produceren welke voldoen aan de ingevoerde karakters. De gebruiker krijgt een lijst met suggesties te zien (zie Figuur 12) waarmee gemakkelijk een zoekopdracht uitgevoerd kan worden.

Figuur 12 – Lijst met suggesties d.m.v. auto aanvullen

Bij het kiezen van een suggestie of het uitvoeren van een zoekopdracht, navigeert de zoekmachine naar een aparte zoekpagina waar de uiteindelijke resultaten worden gepresenteerd (zie Figuur 13).

Op basis van de gevonden resultaten wordt een overkoepelende context bepaald (dat is de context waarin minimaal 80% van de resultaten onder vallen). Deze context, samen met direct onderliggende contexten hiervan, dienen voor optionele filtering van de resultaten (zie Figuur 15).

Figuur 13 – Zoekresultaten

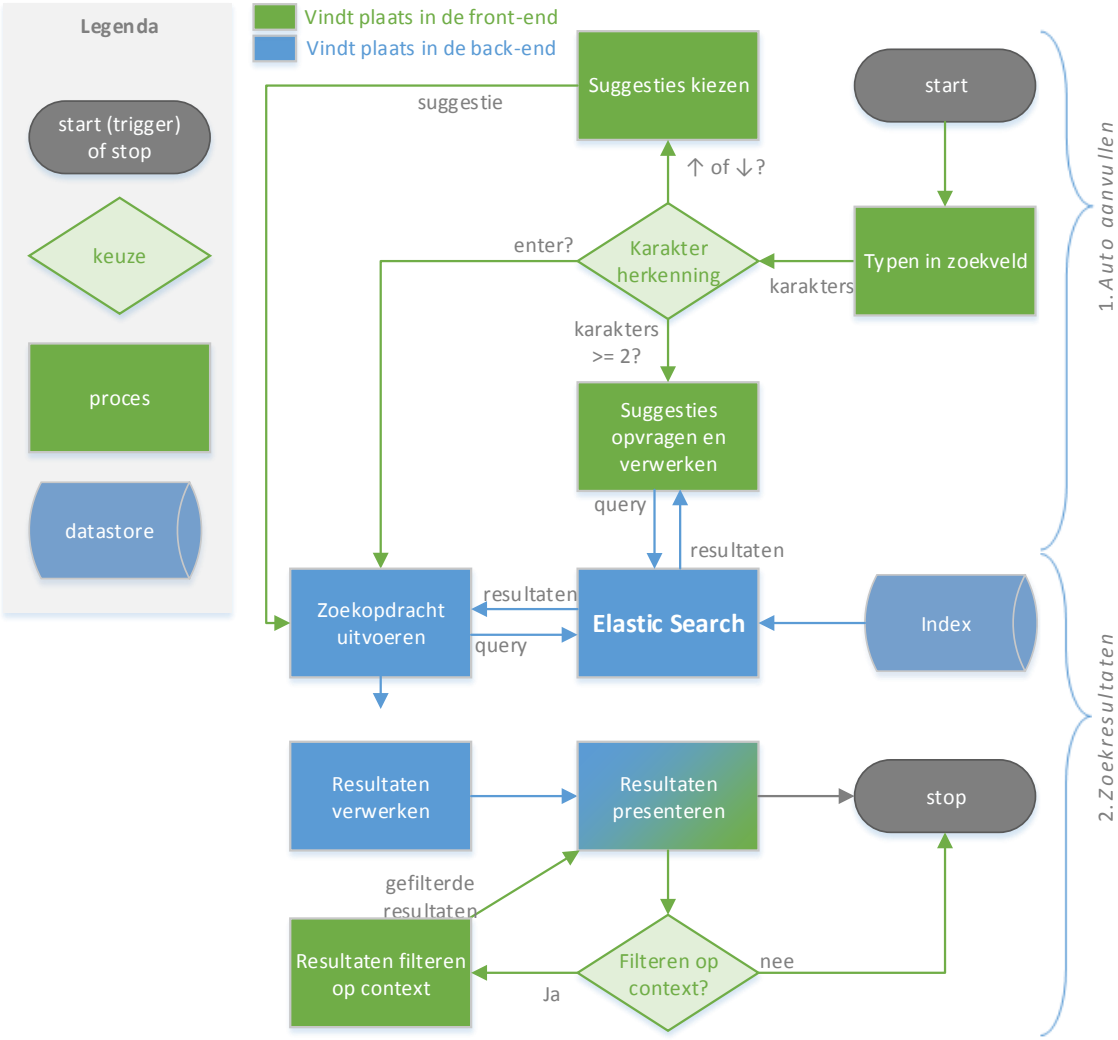
Figuur 14 – Contextfilters

Zoekresultaten worden gerangschikt op een relevantiescore (de mate waarin een document overeenkomt met de zoekterm). Deze score wordt bepaald door Elasticsearch (verder behandeld in deelvraag 3, zie H3.3.2) en staat verder los van de relaties beschreven in SKOS*.

Figuur 15 hieronder illustreert de globale werking van de zoekmachine, welke is op te delen in de volgende twee punten.

1. *Auto aanvullen* (typen in zoekveld, karakter herkenning, suggesties opvragen, verwerken en kiezen);

2. *Zoekresultaten* ophalen, verwerken en presenteren en eventueel filteren (zoekopdracht uitvoeren, resultaten verwerken, presenteren en eventueel filteren).



Figuur 15 – Overzicht werking zoekmachine

3.2 Deelvraag 2

2. Welke vorm van SKOS is het meest geschikt als input voor de zoekmachine?*

3.2.1 Resultaten

In eerste instantie is gekeken wat voor mogelijke vormen er überhaupt zijn, maar ook om wat voor soort data SKOS* nou precies beschrijft. Dit heeft immers invloed op het bepalen van de meest geschikte vorm van SKOS*.

Daarna wordt in overleg gekozen voor één vorm welke wordt uitgewerkt. Hiervoor worden een paar zelfverzonnen relaties gebruikt.

> *In deelvraag 3 (zie H2.3) wordt er een data(sub)set uit de EMM wiki's genomen, voor gebruik tijdens het ontwikkelen van het prototype en voor vergelijking van de zoekresultaten met- en zonder SKOS*.*

Hieronder wordt per meetinstrument (zie H2.2.1) het bijbehorende resultaat beschreven.

Brainstormen

Door het brainstormen kwamen drie categorieën van mogelijke vormen naar boven, wat een aantal handvaten gaf aan de uit te voeren deskresearch.

Deskresearch

De data die SKOS* beschrijft, betreft semantische data. Het is daarom niet meer dan logisch om deze data ook op een bijpassende manier op te slaan. Al vroeg in uitvoering van het zoekplan werd duidelijk dat SMW hier reeds een bijpassende vorm voor had. Hierdoor kon het vervolg onderzoek beter gestuurd worden door meer tijd toe te wijzen aan deze vorm.

Ongestructureerde interviews

Om de bevindingen beter te beschrijven zijn een aantal vragen gesteld m.b.t. kennis van de bevindingen tot nu toe (zie ook Bijlage 5: Transcripties).

3.2.2 Analyse

Hier volgt een samenvatting van de gekozen vorm van SKOS*. Zie Bijlage 2: Onderzoek vorm SKOS* voor de volledige uitwerking.

Verschillende formaten zijn mogelijk om SKOS* buiten de index te bewaren, echter wordt één specifiek formaat aangeraden vanuit SMW voor semantische data.

> *SMW slaat alle data op in een relationele database (RDB). Dit is echter geen ideale vorm voor opslag van semantische data. Een meer natuurlijke datamodel voor SMW data is RDF, een dataformaat die informatie organiseert in informatiegrafieken i.p.v. database tabellen. (Semantic MediaWiki, 2016)*

Deze manier van opslag, een zogenaamde RDF triple-store, is een reflectie van alle semantische data uit de RDB en is reeds in gebruik voor andere projecten. Een triple-store bevat triples, welke uitspraken zijn over web bronnen, als zogenaamde 'subject-predicate-object' expressies (ook wel *onderwerp-predicaat-object*). Het *onderwerp* is een aanduiding voor de bron, het *predicaat* een aanduiding voor de eigenschappen of aspecten van de bron, welke een relatie tussen *onderwerp* en *object* uitdrukt. Een voorbeeld kan zijn:

Onderwerp	*Predicaat*	*Object*
Dijk	gerelateerd aan	Kopsloot

Stel er bestaan twee relaties tussen het SKOS concept `Dijk` en de NSKOS term `Steen`, gevonden in de documenten `ontwerpnota.pdf` en `opbouw-dijken.pdf`. In Tabel 1, is uitgewerkt hoe dit door gebruik van triples verwerkt kan worden in de triple-store.

> *Het object van het predicaat `relatedToNonSkosTerm`, welke de relatie beschrijft tussen een SKOS concept en een NSKOS term, krijgt een unieke identifier (uuid) als waarde. Dit wordt gedaan omdat een specifieke relatie, bestaat uit meerdere triples (één voor het bronbestand, één voor de frequentie, etc.). Deze gegevens moeten allemaal op één of andere manier aan éénzelfde relatie gekoppeld kunnen worden.*

Onderwerp	Predicaat	Object
Dijk	relatedToNonSkosTerm	[uuid: 123]
[123]	hasFrequency	3x
[123]	hasOrigin	ontwerpnota-2016.pdf
[123]	concernsNonSkosTerm	Steen
Dijk	relatedToNonSkosTerm	[uuid: 720]
[720]	hasFrequency	5x
[720]	hasOrigin	opbouw-duiken.pdf
[720]	concernsNonSkosTerm	Steen

Tabel 1 – Twee relaties als triples

> *In overleg met Tin, is er een script gemaakt, waarmee een bestand met comma separated values (CSV) omgezet kan worden naar triples. Dit script is te vinden in de hz-search projectmap (hz-search/csv-to-triples), onder de naam 'csv-to-triples.php'.*

3.3 Deelvraag 3

3. Hoe kan er in het bepalen van de relevantie van de zoekresultaten rekening gehouden worden met SKOS?*

3.3.1 Resultaten

Naast een bestaande export van SKOS concepten als triples, is er een subset van 7 documenten uit de EMM wiki's genomen. Het proces waarmee het NSKOS wordt gemaakt is hiervoor gereproduceerd middels Microsoft Office (Word en Excel). Dit proces is uitgebreid beschreven in Bijlage 3: Onderzoek relevantiebepaling SKOS*.

Nu is er een (kleine, maar wel realistische) dataset beschikbaar voor gebruik tijdens het ontwikkelen van het prototype en voor vergelijking van de zoekresultaten met- en zonder SKOS*.

Voordat SKOS* betrokken kan worden bij de huidige relevantiebepaling is eerst gekeken hoe deze wordt bepaald. Daarna is in overleg met de bedrijfsbegeleider gekeken hoe de relaties beschreven in SKOS*, gebruikt kunnen worden als aanvulling hierop.

Dit is uitgewerkt in een requirementsanalyse en ontwerpspecificatie. Parallel aan het ontwerpen is door toepassing van Rapid Application Development (RAD) ook het prototype ontwikkeld. Dit voornamelijk uit tijdsoverweging maar ook voor flexibiliteit.

Hieronder wordt per meetinstrument (zie H2.3.1) het bijbehorende resultaat beschreven.

Code inspectie

Er wordt buitenom de relevantie die Elasticsearch bepaald, niets gedaan met relevantie. Daarom is beschreven hoe Elasticsearch relevantie bepaald, namelijk op basis van de volgende drie factoren:

- Term frequentie (TF), hoe vaak een term in een document voorkomt;
- Inverse document frequentie (IDF), hoe vaak de term voorkomt in alle documenten;
- Veld lengte, hoe langer een veld, hoe minder relevant de woorden in dat veld zijn. Een term in een titel weegt wellicht zwaarder dan diezelfde term in een omschrijving.

Het product van deze drie factoren is de relevantiescore van het desbetreffende document.

Observatie en experimentatie

Elasticsearch biedt de mogelijkheid om de waarden van bovengenoemde factoren mee te nemen in de zoekresultaten. Zo wordt inzichtelijk hoe deze wordt samengesteld. Hieruit is bijvoorbeeld gebleken dat de veld lengte factor veelal inaccuraat is.

> *De standaard scoring implementatie codeert de veld lengte waarden als "single byte" voordat het wordt opgeslagen. Tijdens het zoeken wordt deze waarde gelezen uit de index en gedecodeerd naar een "float". Dit coderen/decoderen reduceert de grootte van de index maar resulteert in precisieverlies. (Apache Software Foundation, 2014)*

Er zijn meer gevallen waar een waarde niet lijkt te kloppen als dit handmatig wordt uitgerekend. Er zijn wellicht meer situaties en uitzonderingen die invloed hebben op de totaalscore. Deze zijn echter niet gevonden, maar dit heeft verder geen invloed op de rest van het onderzoek.

> *Overigens zijn dit relatief kleine afwijkingen die over het algemeen verwaarloosbaar zijn.*

De zoekopdracht die Elasticsearch uitvoert is vrij simpel. Zoals in H1 Inleiding ook al staat beschreven, wordt er alleen gebruik gemaakt van letterlijke overeenkomsten. Dit betekend dus dat bij zoeken naar bijvoorbeeld Dijk, het woord Dijkbekleding niet overeenkomt.

> *Specifiek voor dit voorbeeld, wordt het SKOS concept Dijkbekleding wel gevonden. De reden is echter omdat één van de doorzochte velden toevallig letterlijk Dijk bevat. Zou dit niet zo zijn dan zou Dijkbekleding niet worden gevonden.*

Overigens is de zoekmachine wel ingesteld op gedeeltelijke overeenkomsten te kunnen vinden, dit middels de zogenaamde *fuzzy* query. Deze query past de zogenaamde Levenshtein edit distance toe tijdens het zoeken. Simpel gezegd is de Levenshtein distance tussen twee woorden, het minimum aantal karakter transformaties (zoals substitutie, verwijdering, etc.) die gedaan moeten worden om van het ene woord het andere woord te maken.

> *De instelling welke aangeeft hoeveel transformaties toegepast mogen worden staat echter momenteel op 0, waardoor het niet wordt toegepast.*

Deskresearch

Er zijn veel factoren die invloed kunnen hebben op relevantie, mede afhangend van wat voor een specifiek project als relevant wordt beschouwd door de stakeholders. De meer voor de hand liggende en erkende factoren zijn beschreven, waaronder ook TF en IDF, welke gebruikt worden door Elasticsearch.

Brainstormen

Er is een dataset van SKOS* beschikbaar, het is duidelijk wat relevantie is, wat er invloed op heeft, en hoe Elasticsearch relevantie bepaald. Hoe kan SKOS* nu geïntroduceerd worden in de huidige relevantiebepaling? Het antwoord is door het toepassen van term expansie, gebruik makend van de relaties beschreven in SKOS*.

> *Met term expansie doelt men het door uitbreiden van de door de gebruiker ingevoerde zoektermen, met gerelateerde termen.*

3.3.2 Analyse

Hier volgt een samenvatting van hoe term expansie, als algoritme, is geïntroduceerd in de zoekmachine. Zie Bijlage 3: Onderzoek relevantiebepaling SKOS* voor de volledige uitwerking.

Eerst een aantal definities:

- *SKOS*term* – Een SKOS concept of NSKOS term uit SKOS*;
- *Zoekterm* – Gebruiker input, bestaand uit één of meer termen;
- *Gewicht* – Representeert het belang van een *SKOS*term*, in de uiteindelijk op te stellen Elasticsearch (ES) query. Het *gewicht* is het product van vermenigvuldiger *startgewicht* en vermenigvuldigtal *gewichtfactor*;
- *Startgewicht* – Een vermenigvuldiger welke gebruikt wordt in de *gewicht* berekening. Dit is het *gewicht* van de voorgaande *SKOS*term* waarvoor recursie wordt uitgevoerd, of indien het de eerste recursie betreft, een instelbare waarde;
- *Gewichtfactor* – Een factor welke gebruikt wordt in de *gewicht* berekening als vermenigvuldigtal. Deze factor is voor SKOS relaties instelbaar en wordt voor NSKOS relaties berekend.

Het term expansie algoritme (zie hierna volgend), is een recursief algoritme welke met een gegeven *zoekterm*, de triple-store doorzoekt naar letterlijke overeenkomsten met *SKOS*termen*. Zijn er overeenkomsten? Dan worden alle directe gerelateerde *SKOS*termen* opgehaald.

Per gevonden *SKOS*term* wordt een *gewicht* berekend. De *SKOS*term* en het bijbehorende *gewicht* worden bewaard voor later gebruik.

Recursie vindt plaats wanneer een gevonden *SKOS*term* een `skos:broader` relatie bevat. In dat geval wordt het proces herhaald met het `skos:broader` concept als zoekterm en het berekende *gewicht* als *startgewicht*. Dit wordt gedaan tot er geen `skos:broader` meer is of het *gewicht* kleiner is dan een instelbaar *minimum*.

> *Hoe meer afstand van de initiële zoektermen, hoe lager de gewichten en hoe minder relevant de termen.*

Al deze gevonden *SKOS*termen* worden verwerkt in een Elasticsearch query, waarbij elke losse term wordt omwikkeld in een *should* clausule.

> *Als should clausules overeenkomen met een document, dan wordt de relevantiescore hiervan verhoogt. (Elasticsearch, 2016)*

Elke *should* clausule kan worden voorzien van een zogenaamde *boost* parameter, welke in dit geval het berekende *gewicht* van de desbetreffende *SKOS*term* betreft.

> *De boost parameter verhoogt of verlaagd het belang van een query clausule ten opzichte van een andere clausule. (Elasticsearch, 2016)*

Zo worden de initiële zoektermen uitgebreid met allerlei gerelateerde termen. Dit wordt als geheel aan Elasticsearch gegeven om de daadwerkelijke zoekopdracht mee uit te voeren.

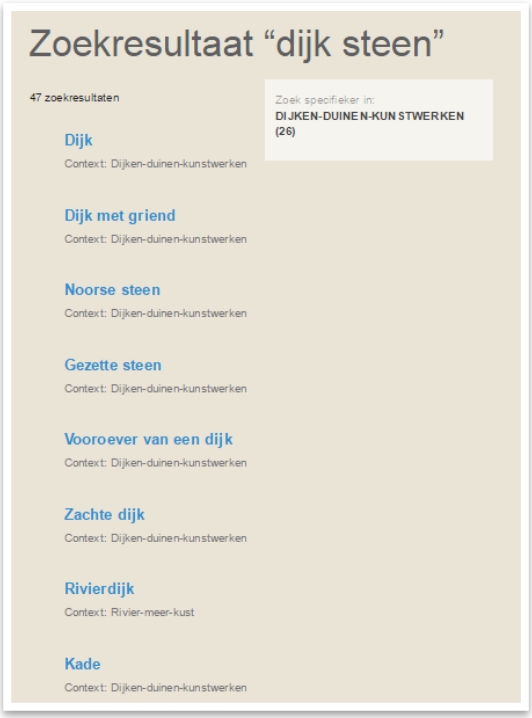
Eén van de uitgevoerde test-cases beschreven in Bijlage 3: Onderzoek relevantiebepaling SKOS*, betreft een SKOS concept en een NSKOS term, namelijk ‘Dijk’ en ‘Steen’. Met term expansie resulteerde dit in een lijst van 30 gerelateerde SKOS concepten en 28 (van de totaal 29 in de dataset) gerelateerde NSKOS termen.

In Figuur 16 en Figuur 17 hierna volgend, zijn de resultaten met en zonder term expansie, naast elkaar gezet.

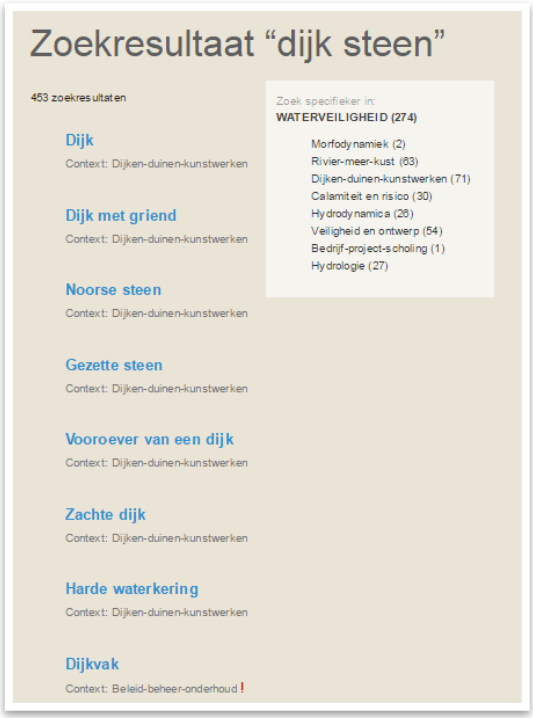
De volgende instellingen zijn gebruikt voor deze resultaten:

`Constant` Gebruik een zogenaamde `constant score query`, waar de score van een document gelijk is aan de boost, ongeacht hoeveel overeenkomsten er zijn gevonden. Immers, het genormaliseerde gewicht is de reflectie van de relevantie. Het is daarom onnodig om Elasticsearch ook een dergelijke berekening toe te laten passen.

> *Het verschil in gevonden zoekresultaten en het aantal bijhorend de context filters (rechterkant van de afbeeldingen) is een bekende fout in de zoekmachine. De aantallen bij de context filters zijn hier niet de juiste aantallen.*



Figuur 16 – Zonder term expansie



Figuur 17 – Met term expansie

Zoals hierboven te zien, komen de resultaten redelijk overeen, wat niet onlogisch is. Echter worden er met term expansie veel meer documenten gevonden doordat er met veel meer gerelateerde termen wordt gezocht. Dit is ook gereflecteerd in de context-filters rechts van de zoekresultaten.

3.4 Totaalbeeld

Alle deelvragen bij elkaar, geven antwoord op de centrale vraag. De samenhang wordt hier benadrukt, zie ook Figuur 18 rechtsonder.

Hoe kan de zoekmachine relevantere zoekresultaten presenteren door gebruik te maken van SKOS?*

Werking zoekmachine:

Het resultaat van deelvraag 1, de werking van de huidige zoekmachine is bekend en beschreven.

Uitwerking SKOS* vorm:

Het resultaat van deelvraag 2, SKOS* is uitgewerkt als triples, in de triple-store, waarin een data subset van SKOS en NSKOS is verwerkt.

Relevantiebepaling met SKOS*,
Ontwerpen en prototypen:

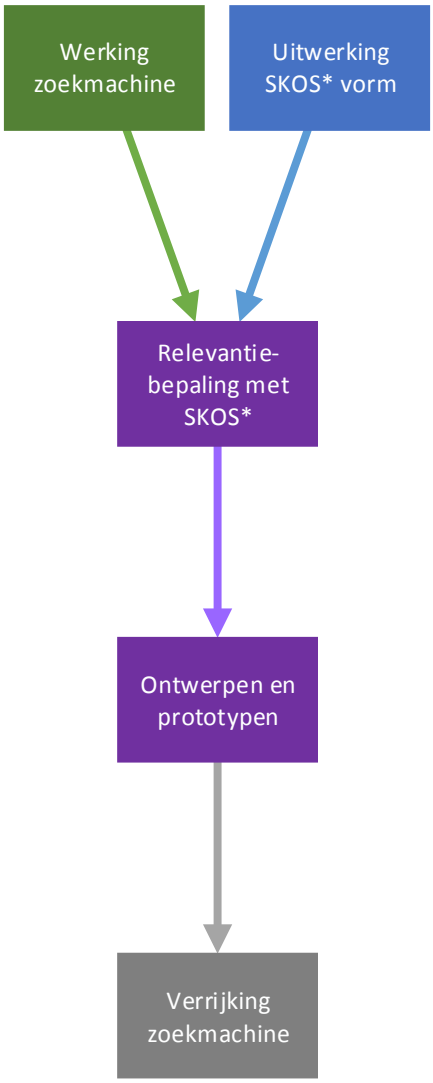
Het resultaat van deelvraag 3, er is onderzocht wat relevantie is, welke factoren er invloed en hoe SKOS* gebruikt kan worden om relevantie te bepalen of beïnvloeden. Dit is uitgewerkt in requirements, een ontwerpspecificatie en een prototype als conceptbewijs.

Verrijkte zoekmachine:

Zo is stapsgewijs gewerkt naar antwoord op de centrale vraag, namelijk:

Hoe kan de zoekmachine relevantere zoekresultaten presenteren door gebruik te maken van SKOS?*

Door het toepassen term expansie op de originele zoektermen, gebruik makend van de relaties beschreven in SKOS*.



Figuur 18 – Totaalbeeld resultaten

Door de requirements, welke ook in de ontwerp-specificatie zijn beschreven, als checklist naast het prototype te leggen, kan men verifiëren het prototype hieraan voldoet. Oftewel, is het systeem juist gemaakt.

Door middel van test-cases, wederom beschreven in de ontwerpspecificatie, kan men valideren dat het prototype ook doet wat het moet doen, namelijk gerelateerde termen verkrijgen, wegen, en gebruiken samen met de originele zoektermen. Oftewel, is het juiste systeem gemaakt.

4 Discussie

Dit hoofdstuk beschrijft hoe het onderzoek antwoord geeft op de in H1.2.3 geformuleerde vraagstelling.

Per deelvraag wordt discussie gevoerd over eventuele voordelen, beperkingen en implicaties van de toegepaste methode en bevindingen.

4.1 Deelvraag 1

1. *Hoe werkt de huidige zoekmachine van input tot en met output?*

4.1.1 Discussie

De resultaten van deze deelvraag zijn gebaseerd op bestaande documentatie, code inspectie en observatie en experimentatie.

Met code inspectie werd de bestaande documentatie concreter. Andersom, beschreef de documentatie het grotere plaatje achter de code. Observatie en experimentatie dienden hier als extra controle slag op de bevindingen. Tegelijk kon hiermee gekeken worden of de bestaande documentatie wel overeen komt met de huidige uitwerking.

Punten die niet overeen kwamen of meer vragen opriepen, zijn ofwel besproken met de bedrijfsbegeleider of andere collega's (zie ook Bijlage 5: Transcripties), of opgenomen als ticket in het gebruikte JIRA ticket systeem.

Observatie en experimentatie is niet per definitie uitputtend uitgevoerd. Het is geheel mogelijk dat er punten over het hoofd zijn gezien, echter de hoofdzaken, nodig voor de vordering van dit onderzoek, zijn bekeken en beschreven.

4.1.2 Deelconclusie

Voordat de zoekmachine verrijkt kan worden moet eerst duidelijk zijn hoe deze momenteel werkt. Dat is met deze deelvraag in kaart gebracht.

De zoekmachine vult input van de gebruiker automatisch aan in de vorm van suggesties. De gebruiker kan een zoekopdracht uitvoeren met de tot nu toe ingevoerde karakters of door één van de suggesties te kiezen.

Hier is wel gebleken dat het vinden van suggesties beperkt is door (te) strenge eisen. Er wordt namelijk gezocht naar letterlijke overeenkomsten, waarbij ook de volgorde van termen in acht wordt genomen.

Buitenom de suggesties wordt ook bij het uitvoeren van een zoekopdracht alleen gezocht naar documenten welke letterlijke overeenkomsten met de zoektermen bevatten. Echter wordt hier niet gekeken naar de volgorde van termen. De resultaten worden gesorteerd op basis van een door Elasticsearch berekende relevantie score. Hoe deze berekening precies werkt, wordt uitgelegd in Bijlage 1: Onderzoek zoekmachine.

Na het uitvoeren van een zoekopdracht, wordt op basis van de gevonden resultaten een overkoepelende context bepaald.

De overkoepelende 'zoek' context is de context waaronder minimaal 80% van de gevonden documenten vallen. (Vogels, 2014)

Deze overkoepelende context, samen met direct onderliggende contexten hiervan, dienen voor optionele filtering van de resultaten. Echter deze contexten gaan van een hiërarchische structuur uit, waardoor sommige contexten niet worden meegenomen. Dit veroorzaakt o.a. dat de aantallen bij de context filters niet overeenkomen met het totaal aantal gevonden resultaten.

Als alles bij elkaar wordt genomen, is de huidige werking van de zoekmachine zeker voor verbetering vatbaar. Er worden te strenge beperkingen gelegd op de invoer van de gebruiker. Ook zitten er hier en daar nog fouten in, zoals missende context filters of incorrect omgaan met speciale karakters.

Het lijkt vooralsnog dat het merendeel van deze problemen met minimale inspanning zijn op te lossen.

4.2 Deelvraag 2

2. Welke vorm van SKOS is het meest geschikt als input voor de zoekmachine?*

4.2.1 Discussie

Omdat SKOS* semantische gegevens bevat is in eerste instantie gezocht naar manieren om semantische data op te slaan. Hier kwam al snel een antwoord uit in de vorm van een suggestie van Semantic Media Wiki zelf, welke aanraadt om een RDF store te gebruiken.

Er was reeds al een triple-store in gebruik (voor een ander project) en de benodigde middelen voor beheer en communicatie waren ook al aanwezig, namelijk:

- De SPARQLStore, het component in Semantic Media Wiki, welke communicatie en beheer tussen de wiki en de triple-store verzorgt;

- Fuseki, als interface voor de triple-store, waarin deze bevraagd en onderhouden kan worden.

> *De naam triple-store is echter misleidend. Naast de drie bestaande attributen van een triple-store (subject, predicate, object) is het veelal mogelijk de triple te voorzien van een naam (als vierde attribuut). De triple-store heet in theorie dan een quad-store (ook wel named graph) en kan daarmee zogenaamde quads beschrijven.*

Door het beschrijven van een quad (een triple met een naam) kan de complete triple hergebruikt worden in een andere triple. Dit is handig voor het verwerken van SKOS* in de triple-store aangezien één relatie één triple vormt, maar aan die relatie moet ook nog frequenties en een herkomst gekoppeld wordenHet toevoegen van een quad is echter niet gelukt via Fuseki. De syntax die hiervoor nodig is lijkt niet te worden ondersteund door Fuseki of het is onbekend hoe dit wel gerealiseerd kan worden.

> *Dit heeft dus als gevolg dat er voor elke unieke relatie een extra triple opgenomen moet worden. Dit is momenteel nog geen groot probleem maar men moet hier wel voor waken naarmate de triple-store gaat groeien. Dit is nadelig wat betreft opslagruimte maar mogelijk ook voor de prestaties bij het bevragen van de triple-store.*

Ter illustratie, stel er is een relatie tussen het SKOS concept Dijk en de NSKOS term Steen, gevonden in het document ontwerpnota.pdf. In plaats van de vier triples in Tabel 2 hieronder, zijn er met quads maar drie triples nodig, zie Tabel 3.

Onderwerp	Predicaat	Object
Dijk	relatedToNonSkosTerm	[uuid: 123]
[123]	hasFrequency	3x
[123]	hasOrigin	ontwerpnota-2016.pdf
[123]	concernsNonSkosTerm	Steen

Tabel 2 – Vier triples voor één relatie

Onderwerp	Predicaat	Object	Naam (4de attribuut)
Dijk	relatedToNonSkosTerm	Steen	[uuid: 123]
[123]	hasFrequency	3x	
[123]	hasOrigin	ontwerpnota-2016.pdf	

Tabel 3 – Drie triples voor één relatie

Een ander discussiepunt is het opnemen van frequenties in de triple-store. Er wordt momenteel namelijk een relevantiescore, mede op basis hiervan, berekend. Echter dit is een berekening die beter voorhand gedaan kan worden. De berekende relevantiescore kan dan als enkele triple worden opgenomen ten opzichte van de huidige SKOS en NSKOS frequentie triples.

Ook is de herkomst triple uiteindelijk niet meer gebruikt in het prototype. Het doel van deze triple was in eerst instantie, afhankelijk van de sterkte van de relatie, de herkomst direct kunnen toevoegen in de zoekresultaten. Ten tweede is een relatie specifiek voor één document. Dit is echter in de loop van het onderzoek overbodig geworden doordat er term expansie wordt toegepast en de herkomst niet meer wordt gebruikt. Wellicht is deze overbodig of kan er een ander doel voor bedacht worden.

Wellicht kan worden volstaan met de in Tabel 4 hieronder beschreven triples voor één relatie.

Onderwerp	Predicaat	Object	Naam (4de attribuut)
Dijk	relatedToNonSkosTerm	Steen	[uuid: 123]
[123]	hasRelevance	0.73	

Tabel 4 – Twee triples voor één relatie

4.2.2 Deelconclusie

SKOS* bevat semantische gegevens en Semantic Media Wiki raadt aan dergelijke gegevens in een triple-store te bewaren. Uit overleg met Tin is gebleken dat SKOS* in een CSV formaat opgeleverd kan worden. Het CSV formaat kan vervolgens middels een simpel script, worden omgezet naar triples. Deze kunnen vervolgens 1 op 1 overgenomen worden in de triple-store middels Fuseki.

> *Oftewel, een hoop handmatige stappen, welke rijp zijn voor optimalisatie en automatisering.*

Momenteel worden, zoals ook benoemd in de discussie, een aantal gegevens bewaard als triples, welke men in de toekomst kan weggelaten of vervangen.

Wellicht kan dit verder worden doorgetrokken, waarbij in plaats van relaties per document, een aggregatie van relaties of alleen de berekende relevantiescores, ook volstaat. Bijvoorbeeld door over alle documenten, één genormaliseerd gewicht te berekenen voor de relatie tussen `Dijk` en `Steen`.

4.3 Deelvraag 3

3. *Hoe kan er in het bepalen van de relevantie van de zoekresultaten rekening gehouden worden met SKOS*?*

4.3.1 Discussie

Eén discussiepunt betreft de gebruikte dataset. Deze is namelijk relatief klein en niet actueel.

> *Twee document soorten (Intentional Elements en Resource Descriptions) kunnen, op het moment van schrijven, niet geïndexeerd worden in de testomgeving. Tevens is de database in de testomgeving een momentopname van de database in de productie omgeving.*

Ook SKOS* is uiteindelijk niet volledig ontwikkeld. Er is een kleine subset gemaakt welke een soortgelijke (al dan niet semi-handmatige) methode toepast. Deze methode is beschreven in H2 van Bijlage 3: Onderzoek relevantiebepaling SKOS*.

> *Voor de doeleinden van dit onderzoek is dit geen probleem aangezien er simpel gezegd minder- en tegelijk ook minder actuele gegevens beschikbaar zijn. De essentie blijft echter wel hetzelfde.*

De gebruikte instellingen voor de uitgevoerde test-cases zijn niet per definitie de 'juiste' instellingen, maar er zit over het algemeen wel een gedachte achter. De instellingen die voorlopig de meeste impact lijkt te hebben is het gebruiken van een `constant score` in plaats van de relevantieberekening die Elasticsearch toepast. Immers, het genormaliseerde gewicht is de reflectie van de relevantie. Het is daarom onnodig om Elasticsearch nogmaals een dergelijke berekening toe te laten passen.

Wat betreft het minimum gewicht, startgewicht, gewicht factoren, maar ook gewicht berekeningen en normalisatie, zijn nu 'vrij' ingevuld. Deze kunnen echter pas echt goed worden ingesteld wanneer de index en SKOS* meer gegevens bevatten en er beter getest kan worden.

Het laatste punt gaat over de huidige zoekopdracht die aan Elasticsearch wordt gegeven. Zoals in H1 Inleiding ook al staat beschreven, wordt er alleen gebruik gemaakt van letterlijke overeenkomsten. Dit geldt evenzo voor de termen die toegevoegd worden aan de originele zoekopdracht.

Stel een zoekopdracht voor *Dijk* levert met term expansie de term *Bekleding* op. Dat betekend niet dat een document *Dijkbekleding* gevonden zal worden.

> *In dit geval wel, maar de reden dat deze wordt gevonden is echter omdat één van de doorzochte velden toevallig letterlijk Dijk bevat. Zou dit niet zo zijn dan zou Dijkbekleding niet worden gevonden, immers Dijk is niet gelijk aan Dijkbekleding.*

4.3.2 Deelconclusie

De in dit onderzoek uitgewerkte manier om rekening te houden met SKOS* in de zoekresultaten is door middel van term expansie. Het algoritme wat term expansie toe past, levert wat gewenst is, namelijk logischerwijs gerelateerde en gewogen termen, hoewel nog in ruwe vorm.

Het zoeken, afgezien van term expansie is nog voor verbetering vatbaar. Afgezien daarvan is het resultaat (zie H3.3.1) een optimistisch resultaat, waarmee de geloofwaardigheid van de achterliggende gedachte wordt versterkt. Namelijk dat met genoeg ruwe data en statistiek, zinnige resultaten zullen blijken.

> *Wanneer de volledige SKOS* beschikbaar is zal er echter nog wel getest moeten worden met verschillende instellingen.*

5 Conclusie

Hoe kan de zoekmachine relevantere zoekresultaten presenteren door gebruik te maken van SKOS?*

5.1.1 Vergelijking met ander onderzoek/theorie

Term expansie (eigenlijk beter benoemd als query expansie) is een relatief nieuw, maar wel bekend onderwerp in andere onderzoeken op het gebied van information retrieval.

Eén onderzoek welke sterk overeenkomt met dit onderzoek, heeft als titel 'Keyword Query Expansion on Linked Data Using Linguistic and Semantic Features' (Shekarpour, Höffner, Lehmann, & Auer, 2013).

> *Effectief zoeken in gestructureerde informatie gebaseerd op tekstuele gebruiker invoer is van groot belang in duizenden applicaties. Query expansie methoden augmenteren de originele query met gerelateerde alternatieve elementen, om zo de kans op het vinden van passende bronnen te verhogen. (Shekarpour, Höffner, Lehmann, & Auer, 2013)*

Term expansie in dit onderzoek maakt o.a. gebruik van verschillende SKOS relaties, aangevuld met taalkundige kenmerken. Onder taalkundige kenmerken worden synoniemen, heteroniemen (generalisaties van een term) en homoniemen (specialisaties van een term) verstaan. Ook hier wordt er een relevantiescore toegekend aan de gevonden termen.

> *Met behulp van deze score wordt de lijst van gerelateerde woorden 'gesnoeid' om een evenwicht tussen precisie en herroeping (recall) te bereiken. (Shekarpour, Höffner, Lehmann, & Auer, 2013)*

Er wordt geconcludeerd dat semantische kenmerken minstens zo effectief zijn voor term expansie als taalkundige kenmerken.

> *De intelligente combinatie van beiden levert een betere precisie en relevantie. (Shekarpour, Höffner, Lehmann, & Auer, 2013)*

Een ander onderzoek welke ook raakvlakken heeft met dit onderzoek, heeft als titel 'Concept Based Query Expansion' (Qiu & Frei). Welke mede op basis van term frequentie en inverse document frequentie gelijkenis thesauri samenstellen. Vervolgens wordt er term expansie toegepast op basis van deze thesaurus.

> *Het lijkt erop dat de verbetering toeneemt met de omvang van de collectie. Deze verbetering is relatief aan de grootte van de collectie. Uiteraard bevatten grotere collecties meer domein specifieke kennis, met als bijgevolg dat de kwaliteit van de gelijkenis thesauri beter is. (Qiu & Frei)*

Allen komen tot de conclusie dat term expansie een positief effect kan hebben op de precisie en relevantie van zoekresultaten.

5.1.2 Suggesties voor vervolgonderzoek

In Bijlage 3: Onderzoek relevantiebepaling SKOS* zijn een aantal aspecten beschreven, welke invloed hebben op relevantie en daarmee wellicht gebruikt kunnen worden in toekomstige versies van de zoekmachine.

Waar nu in dit onderzoek alleen het term frequentie aspect is bekeken, zijn er veel meer aspecten welke gebruikt kunnen worden om gebruikers beter te faciliteren.

In Bijlage 4: Adviesrapport worden een aantal verbeterpunten, aandachtspunten en mogelijkheden voor doorontwikkeling, beschreven.

Eén van de grotere factoren is in mijn mening Elasticsearch en de functionaliteiten daarin. Afgezien van term expansie zijn daar waarschijnlijk de grootste stappen te behalen.

5.1.3 Tot besluit

Op basis van de resultaten uit dit onderzoek, kan geconcludeerd worden dat door toepassing van term expansie op basis van SKOS*, relevantere zoekresultaten gepresenteerd kunnen worden.

Dit is echter een begin en in huidige vorm zeker niet het volledige antwoord. Zoals eerder benoemd in H4.3.2, is het resultaat beschreven in H3.3.1 een optimistisch resultaat, waarmee de geloofwaardigheid van de achterliggende gedachte wordt versterkt. Namelijk dat met genoeg ruwe data en statistiek, zinnige resultaten zullen blijken.

Dit wordt versterkt door soortgelijk onderzoek zoals beschreven in H5.1.1.

Wanneer de openstaande problematiek en tekortkomingen van de zoekmachine worden opgelost zal de zoekmachine in mijn mening al aanzienlijk beter presteren. Met aanvulling van de aanbevelingen beschreven in Bijlage 4: Adviesrapport en de suggesties voor vervolgonderzoek in H5.1.2, kunnen deze prestaties nog verder toenemen.

6 Literatuur

Apache Software Foundation. (2014). *DefaultSimilarity (Lucene API)*. Opgehaald van Lucene core API: http://lucene.apache.org/core/4_10_2/core/org/apache/lucene/search/similarities/DefaultSimilarity.html

Elasticsearch. (2016). *Elasticsearch Reference.* Opgehaald van Elastic · Revealing Insights from Data: www.elastic.co/guide/en/elasticsearch/reference/current

Isaac, A., & Summers, E. (2009, Augustus 18). *SKOS Simple Knowledge Organization System Primer*. Opgehaald van World Wide Web Consortium (W3C): www.w3.org/TR/skos-primer

Krötzsch, M., Vrandečić, D., & Völkel, M. (2006). *The Semantic Web.* Athens, GA, USA: Springer Berlin Heidelberg.

Manning, C. D., Raghavan, P., & Schütze, H. (2009). *Introduction to Information Retrieval.* Cambridge: Cambridge University Press.

Miles, A., Matthews, B., Wilson, M., & Brickley, D. (2005). *SKOS Core: Simple Knowledge Organisation for the Web.* Madrid, Spanje.

Qiu, Y., & Frei, H.-P. (n.d.). *Concept based query expansion.* Zurich, Switzerland: Department of Computer Science, Swiss Federal Institute of Technology.

Rooij, F. d. (2014). *Een kijk op expertise.* Vlissingen: HZ University of Applied Sciences.

Semantic MediaWiki. (2016, april 2). *Semantic MediaWiki*. Retrieved from Semantic MediaWiki: www.semantic-mediawiki.org

Shekarpour, S., Höffner, K., Lehmann, J., & Auer, S. (2013). *Keyword Query Expansion on Linked Data Using Linguistic and Semantic Features.* Augustusplatz 10, 04109 Leipzig: University of Leipzig, Department of Computer Science.

Vogels, T. (2014). *Searching the Semantic Web.* Vlissingen: University College Roosevelt & HZ University of Applied Sciences.

Wahlster, W., & Denge, A. (2006). *Web 3.0: Convergence of Web 2.0 and the Semantic Web.* German Research Center for Artificial Intelligence (DFKI).

BIJLAGEN

Bijlage 1: Onderzoek zoekmachine

1

Onderzoek

Zoekmachine

Expertise Management Wiki’s

bevindingen van Joos Mesie
studentnummer 48580
in het kader van de opleiding HBO-ICT aan de
HZ University of Applied Sciences, Vlissingen

Inhoud

1 Inleiding

Dit document beschrijft de bevindingen bij de eerste deelvraag:

1. *Hoe werkt de huidige zoekmachine van input tot en met output?*

De volgende activiteiten zijn uitgevoerd:

1.1 De huidige zoekmachine (inclusief de index) onderzoeken wat betreft de input (zoekterm), verwerking (resultaten ophalen en sortering) en output (presenteren van resultaten);

1.2 Werking zoekmachine uitwerken in een visuele representatie en later verifiëren door de bedrijfsbegeleider.

1.1 Terminologie

Kort wat gebruikte termen toegelicht:

- Met **SKOS** doelt men in dit onderzoek op een domein specifieke thesaurus, die is gespecificeerd in een Simple Knowledge Organization System (SKOS);

- **NSKOS**, beschrijft genormaliseerde termen, afkomstig uit documenten van de wiki, waarbij de termen zelf niet in SKOS voorkomen maar wel met termen uit SKOS in één document zitten (waardoor een relatie wordt gesuggereerd);

- **SKOS***, er zijn nu twee thesauri, één met SKOS termen en één met NSKOS termen. De relatie tussen SKOS en NSKOS termen bestaat uit een frequentie (het gewicht van de relatie) en de herkomst (het brondocument). Deze relaties worden beschreven in de “verrijkte” SKOS (voortaan SKOS*).

- **SKOS semantische relaties** zijn verbanden tussen SKOS concepten, waarbij de koppeling onafscheidelijk is van de betekenis van de gekoppelde begrippen. Tevens zijn de relaties elkaars inverse waardoor de relatie niet in beide richtingen gedefinieerd hoeft te worden;

- Een **API** is een hulpmiddel of bibliotheek welke ontwikkelaars assisteert bij het schrijven van code die interacteert met andere software;

- **JSON** staat voor JavaScript Object Notation en is een manier om gegevens op te slaan in een gestructureerde, georganiseerde en gemakkelijk toegankelijke manier;

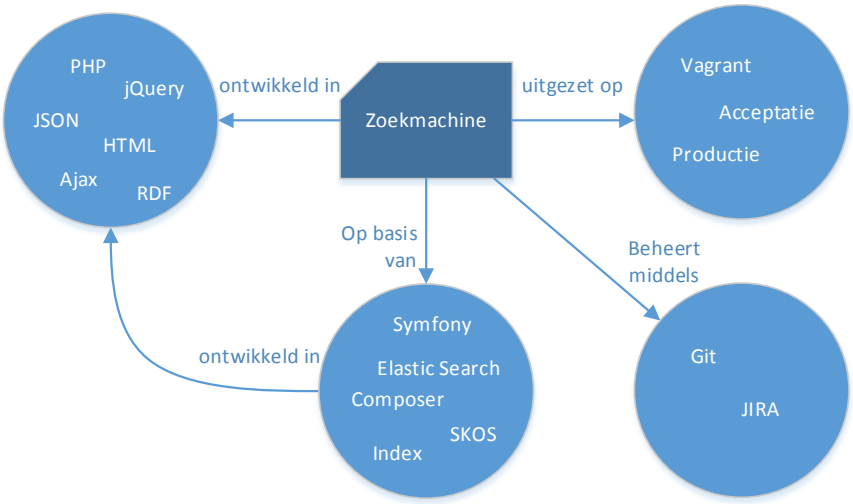
- **RDF** (Resource Description Framework) is een standaard voor het uitwisselen van gegevens op het web;

- **RESTful** applicaties gebruiken HTTP-requests om gegevens- te versturen (aanmaken en/of bijwerken), op te halen en te verwijderen.

1.2 Technieken en tooling

Hoewel dit niet perse nodig is voor deze deelvraag wil ik toch kort (in overzicht) toelichten welke technieken en tools er worden gebruikt.

In Figuur 1 (hieronder) is te grofweg te zien welke componenten voor de implementatie van de gehele zoekmachine gebruikt zijn of in aanraking komen daarmee.



Figuur 1 – Globaal overzicht technieken en tools

De zoekmachine is ontwikkeld in bekende web development talen, namelijk PHP, JavaScript en HTML. Wat betreft JavaScript wordt er voornamelijk gebruik gemaakt van JSON , jQuery en Ajax.

Aan de server kant wordt gebruik gemaakt van het Symfony web application framework samen met Composer (dependency management). De zoekmachine maakt verder gebruik van Elastic Search voor het indexeren en zoeken.

Wat betreft tooling wordt er gebruik gemaakt van Vagrant voor lokale (virtuele) ontwikkel en test omgevingen. Er is verder een acceptatie- en productie omgeving.

Versiebeheer wordt middels Git toegepast en change management wordt middels het JIRA-ticket systeem gedaan.

2 Indexering (HZ-indexing)

De huidige manier van indexering gebruikt content van de EMM wiki zelf. Deze content is voorzien van de nodige informatie en relaties, welke opgevraagd kunnen worden in verschillende standaard formaten. Deze content wordt vervolgens doorgegeven aan Elasticsearch welke de index aanmaakt en beheert.

Content op de EMM wiki's is over het algemeen voorzien van minimaal één (sub)categorie. Momenteel worden vier categorieën content geïndexeerd waaronder ook content in de categorie *SKOS concept*. Hieronder wordt besproken hoe het indexeren van deze specifieke categorie verloopt.

2.1 Indexering van SKOS concepten

HZ-indexing vraagt aan de MediaWiki API een lijst met documenten in JSON formaat, die aan de categorie SKOS Concept zijn gekoppeld. Vervolgens wordt van elk document een aantal extra gegevens opgevraagd waarvan hieronder een voorbeeld is te zien.

In Tabel 1 (rechts) staat een beknopt stuk JSON van het door Elasticsearch geïndexeerde SKOS concept *Dijkvak*. URL's zijn hierin ingekort en lege elementen zijn weggelaten. In dit geval waren er twee lege elementen, namelijk `Skos:narrower` en `Skos:broader`.

`Skos:` eigenschappen worden in H3.3.2 verder toegelicht.

Het SKOS concept `Dijkvak` bevat een element `printouts`. Dit element bevat alle extra opgevraagde gegevens zoals hierboven beschreven.

In dit geval zijn er twee gerelateerde SKOS concepten (`Skos:related`), namelijk `Waterkering` en `Legger`.

Verder is het concept `Dijkvak` onderdeel van twee concepten (`Skos:partOf`), namelijk `Dijkring` en `Dijktraject`.

Het concept `Dijkvak` bevat daarnaast ook nog een eigen `fulltext` en `fullurl`.

```
Dijkvak:{
 printouts:{
  Skos:altLabel:[ Dijksectie ],
  Skos:related:[{
    fulltext: Waterkering,
    fullurl: /Waterkering
   },{
    fulltext: Legger,
    fullurl: /Legger
   }
  ],
  Skos:partOf:[ {
    fulltext: Dijkring,
    fullurl: /Dijkring
   },{
    fulltext: Dijktraject,
    fullurl: /Dijktraject
   }
  ],
  Skos:definition:[
   Een deel van een waterkering.
  ],
  Context:[{
   fulltext: Beheer en onderhoud,
   fullurl: /Beheer_en_onderhoud
  }]
 },
 fulltext: Dijkvak,
 fullurl: /Dijkvak,
}
```

Tabel 1 – JSON resultaat van het SKOS concept Dijkvak

3 De Elasticsearch index

Elasticsearch (ES) is een real-time, gedistribueerde zoek en analyse engine. ES biedt de mogelijkheid om data (in dit geval documenten op de EMM wiki's zoals html, pdf, etc.) te bevragen via één of een combinatie van:

- Full-text, het doorzoeken van tekst om te bepalen welke documenten het best passen bij de aanvraag (welke het meest relevant zijn);
- Gestructureerd zoeken, waarbij het niet om relevantie gaat maar of een document overeenkomt met een vraag of niet. Bijvoorbeeld alle zoekresultaten vanaf 1 januari 2016;
- Real-time analyse van data, zoals het normaliseren van zoektermen.

Het is een losstaande database server, geschreven in Java. Hierin worden gegevens bewaard op een voor "taal gebaseerde zoekopdrachten" geoptimaliseerde manier. De database is aanspreekbaar via een RESTful API, middels HTML en JSON.

De index zoals opgesteld met Elasticsearch bevat een aantal eigenschappen. Zo zijn er eigenschappen die simpelweg een bepaalde waarde hebben (bijvoorbeeld `language: Dutch`), maar ook eigenschappen die beschrijven hoe de gegevens geïnterpreteerd of geanalyseerd moet worden. In dit hoofdstuk worden deze eigenschappen, namelijk de *analysis module*, *settings* en *mappings*, van de (huidige) index, toegelicht.

3.1 De analysis module

Deze module beschrijft zogenaamde *analyzers* welke kunnen worden toegepast tijdens het indexeren van documenten of het verwerken van een zoekopdracht. Denk bijvoorbeeld aan het analyseren van een tekst om stopwoorden eruit te halen, of om een eindig (bijvoorbeeld maximaal 500) aantal karakters te bekijken. *Analyzers* bestaan uit één *tokenizer* en nul of meer *token filters*.

Een *tokenizer* zet een tekenreeks om in zogenaamde tokens, bijvoorbeeld losse woorden of hele zinnen. Met *token filters* kunnen de tokens zelf gewijzigd, verwijderd of juist toegevoegd worden, bijvoorbeeld stopwoorden verwijderen of synoniemen toevoegen.

In de huidige index worden twee analyzers toegepast. Namelijk een *Snowball Analyzer* en *SKOS analyzer*.

3.1.1 Snowball Analyzer

Deze analyzer zit standaard bij Elasticsearch en is gebaseerd op het snowball project (snowball-stem.org).

> *Snowball is een tekstverwerkingstaal met als doel het reduceren van woorden naar de gemeenschappelijke, enkelvoudige, onvervoegde of onverbogen vorm van een woord. (Porter & Boulton, 2002)*

De analyzer bevat de volgende tokenizer en filters.

- *Standard Tokenizer*, voor standaard grammatica ondersteuning gebaseerd op een specifieke taal;
- *Standard Token Filter*, voor het normaliseren van tokens uit de *Standard Tokenizer*;
- *Lowercase Token Filter*, voor het normaliseren van tokens naar lowercase;

- *Stop Token Filter*, voor het verwijderen van stopwoorden;
- *Snowball Filter*, voor het "stemmen" van tokens (bijv. "sloot" en " gesloten" worden allebei "sluiten").

3.1.2 SKOS analyzer

Deze analyzer zit niet standaard bij Elasticsearch maar is ontwikkeld door Jörg Prante. Met deze analyzer kan op basis van bestaande SKOS concepten zogenaamde *term expansion* toegepast worden.

> *Het neemt bestaande SKOS concepten en voert term expansion toe voor gegeven documenten en/of queries. (Prante, 2016)*

Er zijn twee types voor *term expansion*, namelijk:

- Uitbreiding van URI termen naar SKOS labels: URI-verwijzingen naar SKOS concepten in bepaalde documenten, worden uitgebreid door de labels achter die concepten.

Bijvoorbeeld, het concept *Dijkvak* heeft een “onderdeel van” relatie met de concepten *Dijkring* en *Dijktraject*. Bij het zoeken naar de term *dijkvak* zullen de labels van *Dijkring* en *Dijktraject* ook worden bekeken en andersom.

- Uitbreiding van tekst naar termen SKOS labels: labels in bepaalde documenten, die worden gedefinieerd als de preferred labels, worden uitgebreid met de overige labels achter die concepten.

Bijvoorbeeld, het concept *Dijkvak* heeft een alternatieve naam *Dijksectie*. Bij het zoeken naar de term *dijksectie* zal ook de term *dijkvak* worden bekeken en andersom.

De SKOS analyzer gebruikt de SKOS (thesaurus) voor *term expansion*. De analyzer kan met verschillende bestandsformaten overweg. Momenteel wordt hier het Notation3 (N3) formaat voor gebruikt.

> *Notation3 (N3) is een uitbreiding op het RDF datamodel en heeft verschillende toevoegingen zoals het gebruik van variabelen en biedt verder een alternatieve tekstuele syntax van RDF/XML. (Berners-Lee & Connolly, 2011)*

Naast een verwijzing naar de SKOS en de keuze voor het type *term expansion* zijn er nog meer instellingen mogelijk. Twee mogelijk interessante instellingen zijn:

- `language`, een specifieke taal aan kunnen geven voor *term expansion* (interessant wanneer de thesaurus meertalig is);
- `skosType`, een spatie gescheiden lijst met SKOS eigenschappen voor *term expansion*. De volgende opties zijn momenteel mogelijk: `PREF`, `ALT`, `HIDDEN`, `BROADER`, `NARROWER`, `BROADERTRANSITIVE`, `NARROWERTRANSITIVE`, `RELATED`.

LET OP !

Momenteel wordt de SKOS analyzer alleen toegepast op het *subject* veld. Dit veld is echter alleen bij *resource descriptions* gevuld en is daarmee dus alleen voor *resource descriptions* van toepassing.

3.2 Instellingen (Settings)

De index bevat een aantal settings waaronder bijvoorbeeld *analyzers* (zie H3.1). Met de volgende HTTP-request kan men de settings van de deltaexpertise index ophalen.

```
GET deltaexpertise/_settings
```

In Tabel 2 (rechts) is de (gedeeltelijke) response van Elasticsearch te zien, met daarop een toelichting.

Er wordt een filter gedefinieerd (met de naam `skosfilter`) voor gebruik tijdens het uiteindelijke zoeken. Hiervoor wordt het eerder besproken (zie H3.1.2) N3 bestand gebruikt.

Er worden twee `analyzers` gedefinieerd, één van het `type custom`, voor de `skosfilter` (zie H3.1.2) en één van het `type snowball` voor het "stemmen" van woorden en het verwijderen van stopwoorden (zie H3.1.1).

```
analysis: {
  filter: {
    skosfilter: {
      path: /elasticindex,
      type: skos,
      skosFile: /deltaexp.n3,
      expansionType: URI
    }
  },
  analyzer: {
    skos: {
      type: custom,
      filter: skosfilter,
      tokenizer: keyword
    },
    my_analyzer: {
      type: snowball,
      stopwords: [
        aan,
        en,
        …
      ],
      language: Dutch
    }
  }
}
```

Tabel 2 – De index settings in JSON

3.3 Indelingen (Mappings)

Mapping is het proces van definiëren hoe een document, en de velden daarin, worden opgeslagen en geïndexeerd. Bijvoorbeeld in welk formaat datums geïnterpreteerd moeten worden, of in welke velden numerieke getallen zitten. (Elasticsearch, 2016)

De huidige index heeft een standaard mapping en categorie specifieke mappings. De standaard mapping en de SKOS Concept categorie mapping worden in dit hoofdstuk toegelicht. Met de volgende HTTP-request kan men de mappings van de deltaexpertise index ophalen.

```
GET deltaexpertise/_mapping
```

3.3.1 Standaard mapping

De standaard (`_default_`) mapping wordt gebruikt als basis voor andere mappings en beschrijft een aantal algemene `properties`.

Zo is er bijvoorbeeld altijd een veld `content` van het type `string` welke tijdens het bevragen en indexeren de in H3.1.1 beschreven Snowball analyzer (met de naam `"my_analyzer"`) toepast.

Het `subject` veld is wederom van het type `string` en tijdens indexering de in H3.1.2 besproken SKOS analyzer toepast. Tijdens bevraging wordt gebruik gemaakt van een `standard` analyzer.

Het `suggest` veld wordt met het type `completion` voorzien van basis autocomplete functionaliteit. Er wordt verder een `simple` analyzer op toegepast welke niks anders doet dan de tekenreeks lowercase bekijken.

De overige instellingen zijn niet relevant voor mijn onderzoek en worden niet toegelicht.

```
1  _default_: {
2    properties: {
3      content: {
4        type: string,
5        analyzer: my_analyzer
6      },
7      name: {
8        type: string,
9        fields: {
10         untouched: {
11           type: string,
12           index: not_analyzed
13         }
14       }
15     },
16     subject: {
17       type: string,
18       index_analyzer: skos,
19       search_analyzer: standard
20     },
21     suggest: {
22       type: completion,
23       analyzer: simple,
24       ...
25     },
26     title: {
27       type: string,
28       analyzer: my_analyzer
29     }
30   }
31 }
```

Tabel 3 – De standaard mapping in JSON

3.3.2 SKOS mapping

De `skos` mapping definieert onder andere de velden `context` en `context_readable`. Hierin wordt respectievelijk de URL en de naam van de context waartoe een SKOS concept behoort beschreven.

Verder worden een aantal SKOS eigenschappen beschreven welke hieronder kort worden toegelicht (met behulp van het voorbeeld besproken in H2.1).

`skos:prefLabel` (preferred label) en `altLabel` (alternative label) beschrijven de sterkste aanwijzingen over de betekenis van een concept. Bijvoorbeeld `prefLabel` `Dijkvak` en `altLabel` `Dijksectie`.

`skos:definition`, beschrijft de definitie van een concept. Bijvoorbeeld: `"Een deel van een waterkering..."`.

De volgende eigenschappen zijn SKOS semantische relaties.

`skos:broader` en `narrower` beschrijven de hiërarchische koppelingen, waarbij een concept breder of smaller is dan een ander concept.

`skos:related`, beschrijft een associatieve relatie tussen twee concepten. Zo is `Dijkvak` bijvoorbeeld gerelateerd aan `Waterkering`.

`skos:partOf`, is een uitbreiding op `skos:related` en beschrijft een gedeeltelijke relatie tussen twee concepten. Zo is `Dijkvak` bijvoorbeeld onderdeel van `Dijkring`.

Zoals in H3.1.2 besproken, past de SKOS analyzer term expansion toe. Hiervoor wordt het veld `skos:altLabel` gebruikt, waarin alternatieve benamingen beschreven kunnen worden.

```
skos: {
  properties: {
    context: {
      type: string
    },
    context_readable: {
      type: string
    },
    skos:altLabel: {
      type: string
    },
    skos:broader: {
      type: string
    },
    skos:definition: {
      type: string
    },
    skos:narrower: {
      type: string
    },
    skos:partOf: {
      type: string
    },
    skos:prefLabel: {
      type: string
    },
    skos:related: {
      type: string
    },
    url: {
      type: string
    }
  }
}
```

Tabel 4 – De SKOS mapping in JSON

Voor de EMM wiki's zijn de `broader`, `narrower` en `partOf` relaties uitgebreid met een gespecialiseerde variant, respectievelijk `skosem:broader`, `skosem:narrower`, `skosem:partOf`. De "`em`" in `skosem` staat hier voor expertise management.

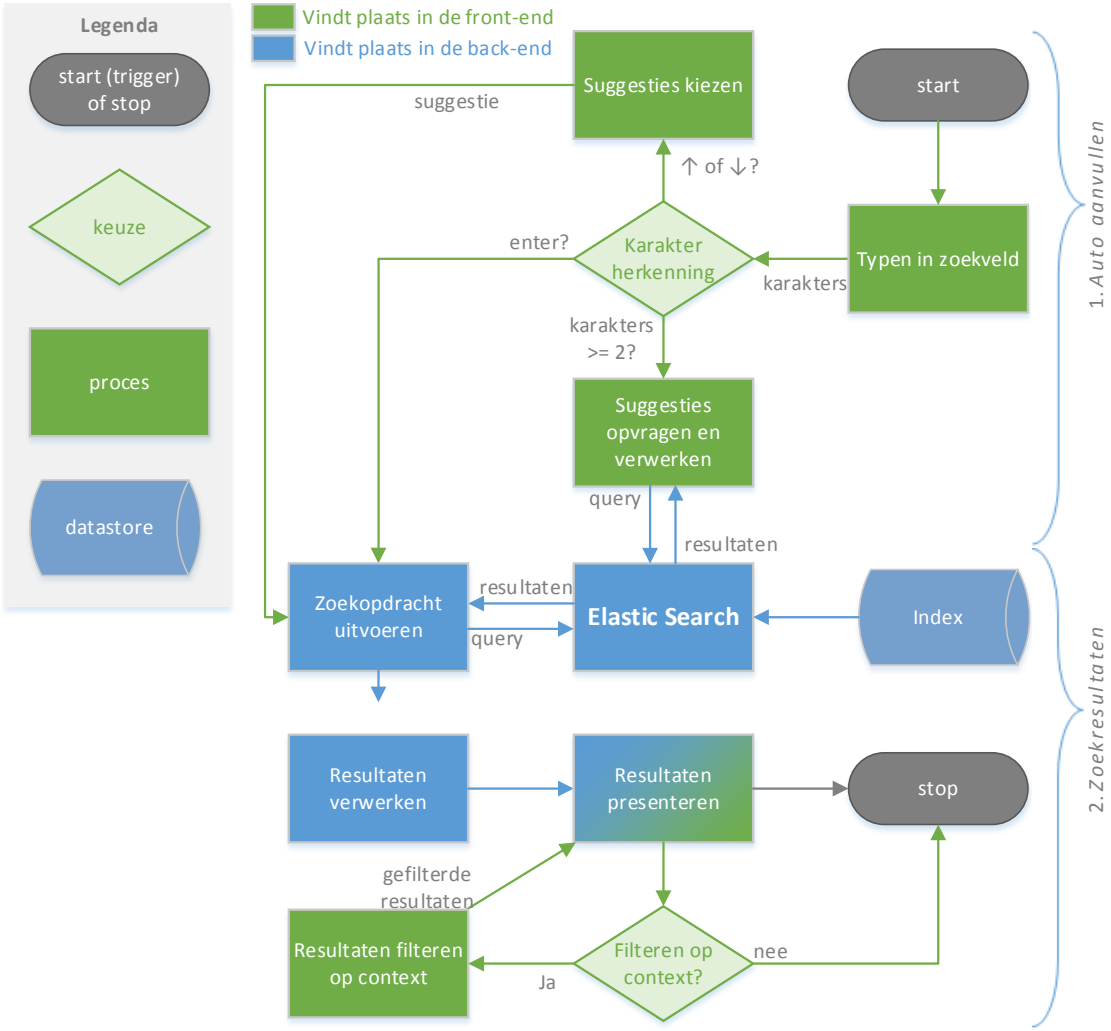
Waar deze specialisaties verder voor dienen valt verder buiten beschouwing.

4 Werking zoekmachine

Figuur 2 hieronder illustreert de globale werking van de zoekmachine. Op de volgende pagina's wordt de werking van de achterliggende code in meer detail toegelicht waaronder:

1. *Auto aanvullen* (typen in zoekveld, karakter herkenning, suggesties opvragen, verwerken en kiezen);

2. *Zoekresultaten* ophalen, verwerken en presenteren en eventueel filteren (zoekopdracht uitvoeren, resultaten verwerken, presenteren en eventueel filteren).



Figuur 2 – Overzicht werking zoekmachine

4.1 Auto aanvullen

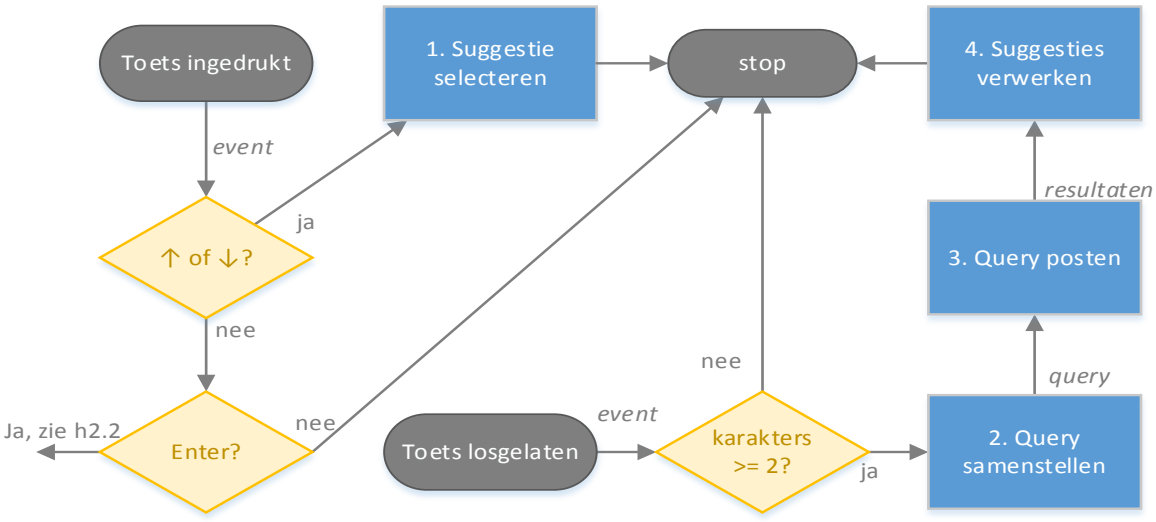
Tijdens het typen in het zoekveld worden suggesties weergegeven onder het zoekveld (zie Figuur 3).

Figuur 3 – Lijst met suggesties d.m.v. auto aanvullen

Figuur 4 illustreert de werking van het automatisch aanvullen en wordt hieronder verder toegelicht.

Wanneer de gebruiker een toets indrukt wordt gekeken of er een speciale actie moet worden uitgevoerd. De volgende toetsen worden momenteel herkent:

- pijl omhoog ↑ of omlaag ↓ (navigeren van de dropdown lijst met suggesties, *proces 1*);
- enter toets, zoekopdracht versturen (hier wordt genavigeerd naar de zoekpagina waar het vervolg in gang wordt gezet, maar dit valt buiten het auto aanvullen en wordt in H4.2 verder beschreven).



Figuur 4 – Suggesties onder het zoekveld

Als er geen speciale acties zijn uitgevoerd, gaat het script verder met het kijken naar het aantal ingevoerde karakters (in dit geval 2 of meer). Als het aantal voldoende is worden pas suggesties weergegeven. Dit om de suggesties relevanter te maken en ook om het aantal suggesties te beperken.

Er wordt een query samengesteld met de ingevoerde karakters (*proces 2*), bijvoorbeeld '*Dij*' . Deze query wordt gepost naar ES welke de query uitvoert en een resultaat terug geeft (*proces 3*).

De suggesties worden verwerkt in een dropdown lijst onder het zoekveld (*proces 4*). Een gebruik kan door het klikken op een suggestie direct zoeken naar de suggestie of handmatig zoeken naar wat is ingevuld in het zoekveld.

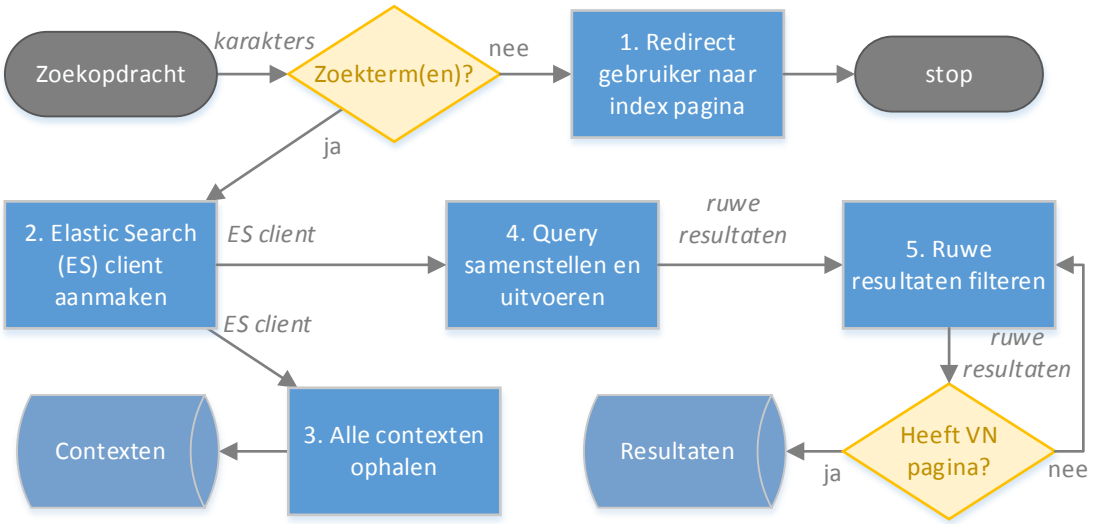
4.2 Zoekresultaten ophalen

Na het uitvoeren van een zoekopdracht worden op basis van de ingevulde zoekterm(en) achterhaald welke documenten in de zoekresultaten worden opgenomen.

Figuur 5 illustreert de werking van het automatisch aanvullen. Onderstaande uitleg is ter ondersteuning hiervan.

Als er geen zoekterm is ingevuld wordt de gebruiker doorverwezen naar de index pagina (*proces 1*).

Is er wel een zoekterm, dan wordt er een Elastic Search (ES) client aangemaakt (*proces 2*).



Figuur 5 – Contexten en zoekresultaten ophalen

Voordat er gezocht wordt met de ingevoerde zoekterm worden alle contexten opgehaald en tijdelijk opgeslagen (*proces 3*).

Daarna wordt een zogenaamde 'Multimatch' query opgesteld en uitgevoerd (*proces 4*). De Multi-match query beschrijft welke velden bekeken worden en welke velden belangrijker zijn dan andere. Dit laatste wordt gedaan door een zogenaamde 'boost' waarde te beschrijven. Bijvoorbeeld:

```
"title^3"
"content"
```

Waar zoekterm(en) gevonden in de titel 3x zwaarder mee tellen dan wanneer deze in de content worden gevonden.

Na het uitvoeren van de query worden de ruwe resultaten gefilterd zodat alleen resultaten met een zogenaamde *View-Navigation (VN)* pagina worden meegenomen (*proces 5*). VN pagina's zijn simpel gezegd de visuele schil om de gegevens van de pagina, bestemd voor de bezoeker.

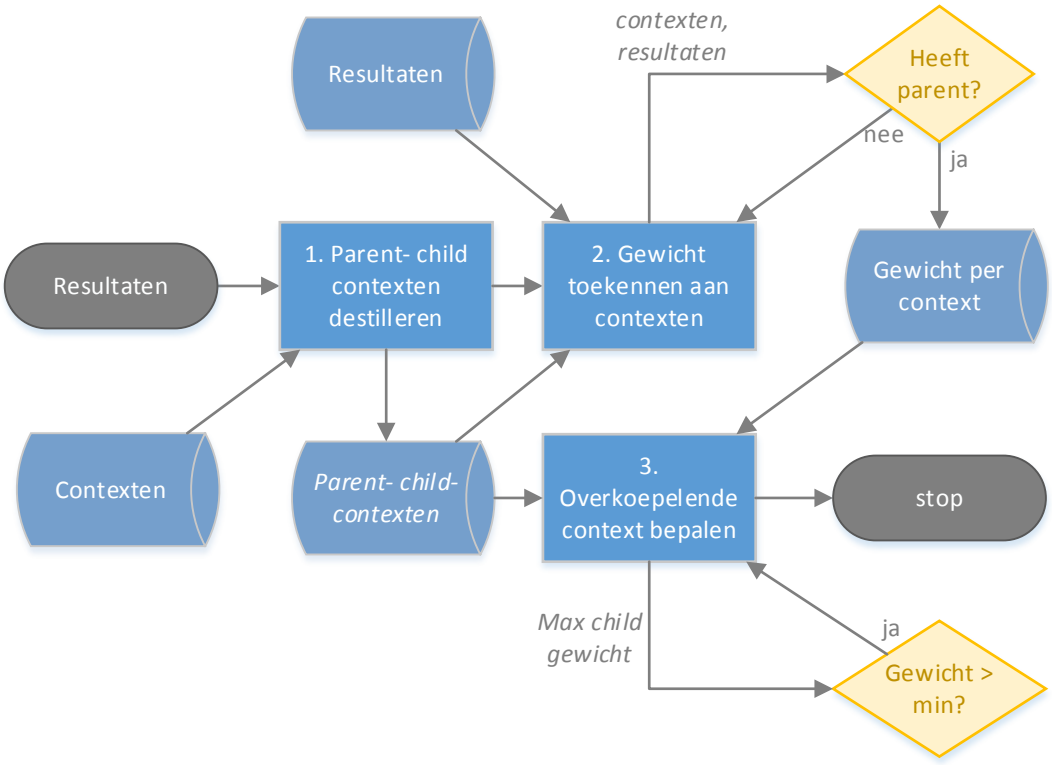
De resultaten worden opgeslagen voor latere verwerking.

4.3 Zoekresultaten verwerken

De resultaten zijn nu beschikbaar. Aanvullend wordt de overkoepelende context bepaald en worden de overkoepelende context met de child-contexten daarvan, weergegeven in een lijst voor eventuele filtering.

Figuur 6 illustreert hoe de overkoepelende context wordt achterhaald. Onderstaande uitleg is ter ondersteuning hiervan.

Elk gevonden resultaat is gekoppeld aan een bepaalde context. Om verder te kunnen filteren in de gevonden zoekresultaten wil men de overkoepelende context achterhalen. Om dit te doen worden de gevonden contexten verwerkt in twee data structuren met respectievelijk de parent contexten en de child contexten (*proces 1*). Nu kan men gemakkelijk de parent- of child- contexten ophalen van een willekeurige context.



Figuur 6 – Overkoepelende context bepalen

Gewichten worden iteratief toegekend aan de resultaten (*proces 2*). Hier wordt het gewicht van contexten met een parent context opgehoogd en opgeslagen. De 'root' context telt niet mee als parent.

Op basis van de gewichten wordt bepaald welke context overkoepelend is (*proces 3*). De overkoepelende context is de "diepste" context die meer dan 80% van de resultaten beschrijft. Met "diepste" bedoelt men een context die geen child-contexten te hebben die ook meer dan 80% beschrijven.

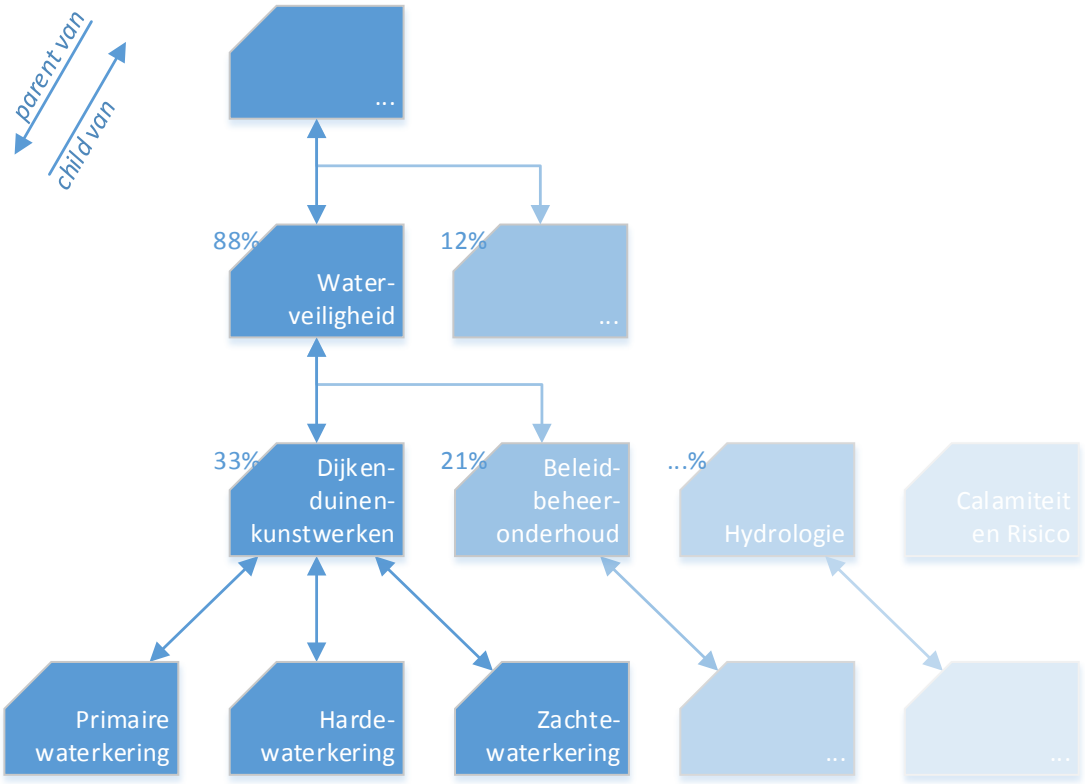
LET OP !

Tijdens het beschrijven is geconstateerd dat er een mogelijk verschil is tussen de relaties beschreven in SKOS en de relaties die de zoekmachine destilleert uit de gevonden resultaten. Dit verschil ontstaat wellicht doordat de relaties beschreven in SKOS* een stuk complexer kunnen zijn dan alleen *parent* en *child*. Dit is genoteerd en in de toekomst aangekaart.

4.3.1 Voorbeeld overkoepelende context

In Figuur 7 hieronder wordt een voor een voorbeeld zoekopdracht gevisualiseerd hoe de zoekmachine bepaald wat de overkoepelende context is en welke filters er worden weergegeven naast de zoekresultaten.

Zoals op de vorige pagina (H4.3) is beschreven wijken de relaties in dit voorbeeld af van de relaties zoals in SKOS beschreven. Afgezien van dit verschil is het onderstaande voorbeeld conform de werking van de huidige zoekmachine en is het een prima voorbeeld van de manier waarop de overkoepelende context wordt bepaald.



Figuur 7 – Voorbeeld overkoepelende context

Stel men zoekt naar *waterkering*. Er komen een aantal resultaten terug, verspreid over verschillende contexten. Ten minste 80% van de resultaten bevinden zich in de context *Waterveiligheid*. Er wordt geconstateerd dat deze context geen child-contexten heeft welke ook meer dan 80% van de resultaten beschrijven.

Dit is in dit voorbeeld niet het geval. Dat betekend dat de *Waterveiligheid* de overkoepelende context is.

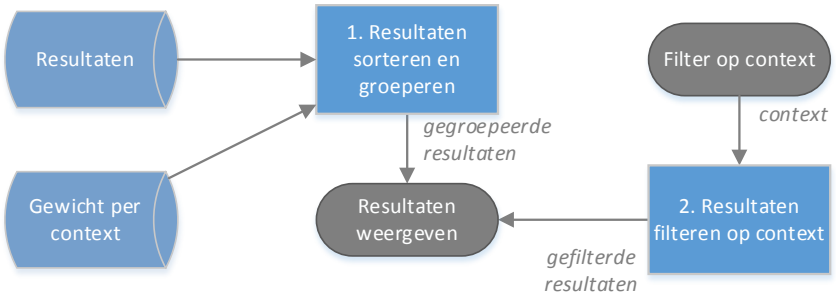
De overkoepelende context samen met de child-contexten hiervan, worden opgesteld als filters (zie ook Figuur 8,rechts).

Figuur 8 – Context filters

4.4 Zoekresultaten presenteren

De resultaten worden gesorteerd, gegroepeerd en gepresenteerd. Filters worden rechts van de zoekresultaten weergegeven.

In Figuur 9 hieronder, wordt gevisualiseerd hoe de zoekresultaten worden gepresenteerd (Figuur 10) en hoe men deze kan filteren middels contextfilters (Figuur 11).



Figuur 9 – Resultaten en filters presenteren

De zoekresultaten worden op basis van de eerder opgehaalde score (ook wel het gewicht) gesorteerd, waarbij het hoogst scorende document bovenaan staat (*proces 1*).

De resultaten worden tevens gekoppeld aan de *child*-contexten van de overkoepelende context. Hiermee kunnen de resultaten namelijk worden gefilterd (*proces 1*).

Wanneer men filtert worden in eerste instantie alle zoekresultaten 'verstopt'. Vervolgens worden alle zoekresultaten die bij de geselecteerde context horen weergegeven (*proces 2*).

Figuur 10 – Zoekresultaten

Figuur 11 – Contextfilters

5 Literatuur

Berners-Lee, T., & Connolly, D. (2011, Maart 28). *Notation3 (N3): A readable RDF syntax*. Retrieved from World Wide Web Consortium (W3C): www.w3.org/TeamSubmission/n3

Elasticsearch. (2016). *Mapping.* Retrieved from Elastic · Revealing Insights from Data: www.elastic.co/guide/en/elasticsearch/reference/2.3/mapping.html

Porter, M., & Boulton, R. (2002). *Snowball Manual.* Retrieved from Snowball: www.snowballstem.org/compiler/snowman.html

Vogels, T. (2014). *Searching the Semantic Web.* Vlissingen: University College Roosevelt & HZ University of Applied Sciences.

Bijlage 2: Onderzoek vorm SKOS*

Onderzoek

Vorm SKOS*

Expertise Management Wiki's

bevindingen van Joos Mesie
studentnummer 48580
in het kader van de opleiding HBO-ICT aan de
HZ University of Applied Sciences, Vlissingen

Inhoud

1 Inleiding

Dit document beschrijft de bevindingen bij de tweede deelvraag:

Welke vorm van SKOS is het meest geschikt als input voor de zoekmachine?*

De volgende activiteiten zijn uitgevoerd:

2.1 Deskresearch voor meer informatie over hoe semantische data kan worden opgeslagen;

2.2 Brainstormen naar mogelijke vormen;

2.3 Voor- en nadelen van bevindingen beschrijven;

2.4 In overleg met bedrijfsbegeleider een passende vorm kiezen voor uitwerking en toepassing in het resterende onderzoek. Tevens een aantal (verzonnen) SKOS* relaties uitwerking in deze vorm.

H2 beschrijft het idee achter het conceptueel SKOS* en de mogelijke vormen van SKOS*. In H3 worden de gevonden vormen uitgewerkt in voor- en nadelen. In H4 worden de uitwerking van een aantal SKOS* relaties beschreven.

1.1 Terminologie

Kort wat gebruikte termen toegelicht:

- Met **SKOS** doelt men in dit onderzoek op een domein specifieke thesaurus, die is gespecificeerd in een Simple Knowledge Organization System (SKOS);

- **NSKOS**, beschrijft genormaliseerde termen, afkomstig uit documenten van de wiki, waarbij de termen zelf niet in SKOS voorkomen maar wel met termen uit SKOS in één document zitten (waardoor een relatie wordt gesuggereerd);

- **SKOS***, er zijn nu twee thesauri, één met SKOS termen en één met NSKOS termen. De relatie tussen SKOS en NSKOS termen bestaat uit een frequentie (het gewicht van de relatie) en de herkomst (het brondocument). Deze relaties worden beschreven in de “verrijkte” SKOS (voortaan SKOS*).

- **JSON** staat voor JavaScript Object Notation en is een manier om data op te slaan in een gestructureerde, georganiseerde en gemakkelijk toegankelijke manier;

- **RDF** (Resource Description Framework) is een standaard voor het uitwisselen van data op het web;

2 Mogelijke vormen SKOS*

Het conceptueel SKOS* wordt ontwikkeld door Thien Tin Lam Ngoc (voortaan Tin). Voor mijn onderzoek is de aanname gemaakt dat SKOS* reeds is ontwikkeld in een bepaalde vorm. Deze vorm is echter niet per definitie de meest geschikte vorm als input voor de zoekmachine. In dit hoofdstuk worden verschillende mogelijkheden beschreven.

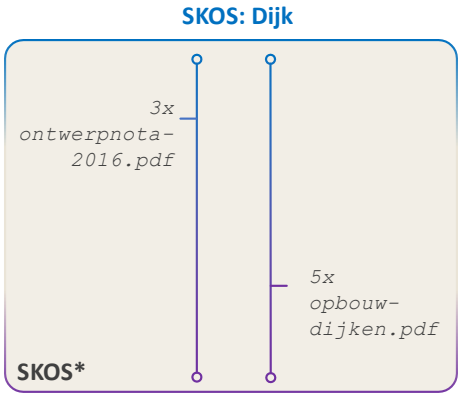
2.1 Conceptueel SKOS*

Het conceptueel SKOS* beschrijft relaties tussen SKOS concepten en NSKOS termen middels een frequentie (f) en het bijbehorende document.

Stel er zijn relaties gevonden tussen het SKOS concept Dijk en de NSKOS term Steen, namelijk in hetzelfde document waar Dijk in voorkomt, komt 3 keer ($f = 3$) Steen voor. In een ander document is dit 5 keer het geval ($f = 5$), zie ook Figuur 1. Oftewel:

SKOS | Dijk ↔ **NSKOS |** Steen
3x in ontwerpnota-2016.pdf
5x in opbouw-dijken.pdf

Het SKOS is een reflectie van de content op de EMM wiki's en kan geactualiseerd worden op basis van de content van de EMM wiki's.



*Figuur 1 – Voorbeeld van een relatie uit SKOS**

Het NSKOS bevat simpel gezegd alle overige (genormaliseerde) termen uit de content van de EMM wiki's, die niet al in het SKOS zijn opgenomen.

2.2 Mogelijke vormen

Mogelijke vormen van SKOS* kunnen in drie categorieën worden opgedeeld, namelijk:

H2.2.1 – Buiten de index

H2.3.1 – Onderdeel van de index

H2.3.2 – Aparte (dedicated) index

Deze drie categorieën worden in de volgende hoofdstukken toegelicht.

2.2.1 Buiten de index

Verschillende formaten zijn mogelijk om SKOS* buiten de index te bewaren, echter wordt één specifiek formaat aangeraden vanuit SMW. De data die SKOS* beschrijft, betreft namelijk semantische data. Het is daarom niet meer dan logisch om deze data ook op een bijpassende manier op te slaan.

> *SMW slaat alle data op in een relationele database (RDB). Dit is echter geen ideale vorm voor opslag van semantische data. Een meer natuurlijk datamodel voor SMW data is RDF, een dataformaat die informatie organiseert in informatiegrafieken in plaats van in vaste database tabellen. SMW ondersteund gebruik van RDF naast de standaard SQL-database. (Semantic MediaWiki, 2016)*

Deze manier van opslag, een zogenaamde RDF triple-store, is reeds in gebruik door de EMM wiki's. Deze triple-store is een reflectie van alle semantische data in de RDB. Een triple-store bevat triples, wat uitspraken zijn over bronnen (specifiek web bronnen) als zogenaamde 'subject-predicate-object' expressies, ook wel *onderwerp-predicaat-object*. Het *onderwerp* is een aanduiding voor de bron, het *predicaat* een aanduiding voor de eigenschappen of aspecten van de bron, welke een relatie tussen *onderwerp* en *object* uitdrukt. Een voorbeeld kan zijn:

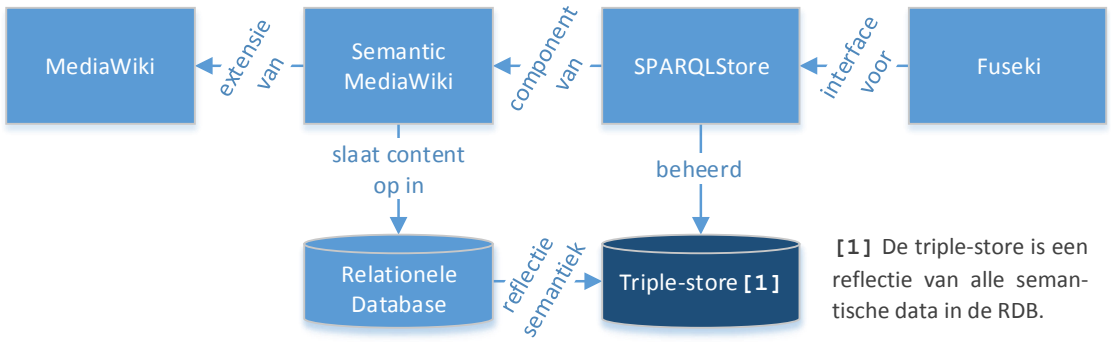
Onderwerp	*Predicaat*	*Object*
Dijk	gerelateerd aan	Kopsloot

Of in meer formele SKOS notatie:

<Dijk>	skos:related	<Kopsloot>

> *De SPARQLStore is de naam voor het component welke communicatie (inclusief data beheer) tussen Semantic MediaWiki en de RDF triple-store verzorgt. (Semantic MediaWiki, 2016)*

In Figuur 2 hieronder is te zien hoe de SPARQLStore, triple-store en andere onderdelen een geheel vormen.



Figuur 2 - Triple-store

De naam triple-store is echter misleidend. Naast de drie bestaande attributen van een triple-store (*onderwerp, predicaat, object*) is het veelal mogelijk de triple te voorzien van een naam (als vierde attribuut). De triple-store heet in theorie dan een *quad-store* (ook wel *named graph*) en beschrijft naast triples dan ook *quads*. De store zelf blijft hier benoemd als triple-store.

Door het beschrijven van een quad kan een complete triple hergebruikt worden in andere triples. Dit is handig voor het verwerken van SKOS* in de triple-store aangezien één relatie één triple vormt, maar aan die relatie moet ook nog een frequentie en herkomst gekoppeld worden.

In Tabel 1 hierna volgend, is een voorbeeld van de volgende relaties (zie ook Figuur 1, H2.1) uitgewerkt.

SKOS|Dijk ↔ **NSKOS**|Steen
3x in ontwerpnota-2016.pdf
5x in opbouw-dijken.pdf

Subject	Predicate	Object	Name
Dijk	relatedToNonSkosTerm	Steen	[uuid: 123]
Dijk	relatedToNonSkosTerm	Steen	[uuid: 720]
[123]	hasFrequency	3x	
[123]	hasOrigin	ontwerpnota-2016.pdf	
[720]	hasFrequency	5x	
[720]	hasOrigin	opbouw-dijken.pdf	

Tabel 1 - Voorbeeld van een SKOS relatie, verwerkt in de triple-store*

2.3 De alternatieven

2.3.1 Onderdeel van de index

Relaties in SKOS opnemen in de huidige index is een mogelijkheid. Dit kan door de relaties los op te nemen in de index (gescheiden van reeds bestaande delen van de index), maar het is wellicht logischer om de relaties als eigenschap onder een SKOS concept of andersom een NSKOS term toe te voegen.

Hier wordt dit laatste geval toegelicht, dus relaties als eigenschap van een SKOS concept.

Stel de regels in Tabel 2, rechts, worden toegevoegd aan de SKOS mapping. Neem de volgende relaties (zie ook Figuur 1, H2.1):

SKOS|Dijk ↔ **NSKOS**|Steen
3x in ontwerpnota-2016.pdf
5x in opbouw-dijken.pdf

```
relatedToNSKOS: {
  type: nested,
    properties: {
      nskos:term: {
        type: string
      },
      frequency: {
        type: int
      },
      origin: {
        type: string
      }
  }
}
```

Tabel 2 – Toevoeging SKOS aan SKOS mapping*

Omdat er meerdere relaties worden beschreven onder de `relatedToNSKOS` eigenschap, wordt er gebruik gemaakt van het type `nested`. Dit omdat Elasticsearch uit prestatie overwegingen een platte structuur (`key-value` paar) van de index wil maken. Dat wil zeggen dat alle waarden per veld, worden samengevoegd tot één grote verzameling. Hierbij gaat het unieke object (in dit geval de unieke relatie) verloren, bijv:

```
nskos:term   [ "Steen" ]
frequency    [ 3, 5 ]
origin       [ "ontwerpnota-2016.pdf", "opbouw-dijken.pdf" ]
```

Volgens de mapping in Tabel 2 zal bijvoorbeeld de hieronder beschreven JSON als resultaat op een zoekactie worden teruggegeven.

```
title:          "Dijk",
skos:prefLabel: "Dijk",
skos:altLabel:  "",
relatedToNSKOS: [
{
  nskos:term:   "Steen"
  frequency:    3
  origin:       "ontwerpnota-2016.pdf"
},{
  nskos:term:   "Steen"
  frequency:    5
  origin:       "opbouw-dijken.pdf"
}]
...
```

2.3.2 Aparte (dedicated) index

Men kan ook een nieuwe Elasticsearch index maken en deze compleet toewijden aan de relaties beschreven in SKOS*.

Dit lijkt grotendeels op het alternatief beschreven in H2.3.1. Het verschil is dat de lijst met relaties niet meer direct onder een SKOS concept of NSKOS term hangt maar apart wordt geïndexeerd.

Volgens de mapping in Tabel 3 zal bijvoorbeeld de hieronder beschreven JSON als resultaat op een zoekactie worden teruggegeven.

```
skosConcept:      "Dijk",
relatedToNSKOS: [
{
  nskos:term:     "Steen"
  frequency:      3
  origin:         "ontwerpnota-2016.pdf"
},{
  nskos:term:     "Steen"
  frequency:      5
  origin:         "opbouw-dijken.pdf"
}]
...
```

```
1  relatedToNSKOS: {
2    type: nested,
3    properties: {
4      nskos:term: {
5        type: string
6      },
7      frequency: {
8        type: int
9      },
10     origin: {
11       type: string
12     }
13   }
14 }
```

Tabel 3 – Toevoeging SKOS aan SKOS mapping*

3 Voor- en nadelen

3.1 Buiten de index

Voor- en nadelen van een RDF database volgens Semantic MediaWiki (Semantic MediaWiki, 2016).

Voordeel: Betere query prestaties

RDF stores zijn ontwikkeld voor query's in de SPARQL query language. SMW query's kunnen beter uitgedrukt worden in SPARQL dan in SQL of andere varianten. Tevens zijn veel optimalisatiemethoden voor relationele databases nutteloos of misleidend voor SMW query's. Het is daarom aan te nemen dat RDF stores superieure query prestaties leveren.

Voordeel: Additionele interfaces

RDF stores die de SPARQL standaard ondersteunen zijn ook gelijk bruikbaar door andere applicaties zonder naar de SMW front-end te moeten gaan. Zo kan de content van de media wiki's efficiënt gebruikt worden in andere applicaties.

Voordeel: Redenering functionaliteiten en ontologie gebaseerde data toegang

Semantische Web talen zoals RDF Schema en OWL bieden additionele expressieve mogelijkheden voor het modelleren van data. Bijvoorbeeld door de verklaring van afgeleide klassen of de verklaring van kenmerk eigenschappen (bijv. transitiviteit). Sommige SPARQL-geschikte databases kunnen deze eigenschappen evalueren tijdens zoekopdrachten. Bijvoorbeeld door het maken van "virtuele views".

Voordeel: Data integratie en ontologie hergebruik

Het is mogelijk om extra data op te slaan in de RDF database die SMW bijwerkt. Op deze wijze kan de RDF store fungeren als platform voor data-integratie en ontologie hergebruik.

Voordeel: Fysieke scheiding van IT-middelen

Omdat de SMW web front-end met een relationele database werkt en de RDF store gescheiden is, kunnen complexe query's verdeeld worden over meerdere (andere) servers waardoor de wiki zelf normaal blijft presteren.

Nadeel: Extra onderhoud

Omdat er een extra database bestaat kan dit wellicht voor meer onderhoud zorgen.

Nadeel: Hogere opslag eisen

De data van de wiki staan zowel in een relationele database en de RDF store.

Nadeel: Vraagstukken voor prestaties en stabiliteit

Er zijn verschillende soorten RDF databases, echter is er nog weinig ervaring van gebruik, zeker samen met SMW. Het is gelukkig makkelijk om te wisselen van RDF database wanneer nodig.

Deze nadelen zijn momenteel niet of nauwelijks een nadeel omdat de dataset nog relatief klein is. Het is echter wel belangrijk om dit mee te nemen voor de toekomst.

3.2 Onderdeel van de index of dedicated index

Ten eerste een kleine toelichting op verdeling van data en prestaties, binnen Elasticsearch.

Er vanuit gaand dat er geen verandering aan de instellingen plaatsvindt wat betreft de hoeveelheid *shards* per index, moet men rekening houden met de volgende implicaties:

Eén index met verschillende typen	**Meerdere indexen (bijvoorbeeld per type)**
Minder efficiënt zoeken omdat de hoeveelheid data per *shard* groter is, waardoor Elasticsearch meer tijd nodig heeft om door de documenten hierin te filteren.	Efficiënt en snel zoeken in de index, omdat de hoeveelheid data kleiner is per *shard*.
Er is minder overhead omdat de aanvraag minder *shards* hoeft aan te roepen. De *shards* zelf zijn echter wel groter.	Zoekopdrachten genereren meer overhead, de aanvraag moet naar meerdere *shards*, over meerdere indexen worden gestuurd, gecompileerd en terug gegeven aan de gebruiker.
Als de dataset relatief klein is, kan er meer opslagruimte nodig zijn voor elke additionele *shard* terwijl de prestatie winsten minimaal zijn. Is de dataset echter relatief groot, kan dit juist betere prestaties leveren.	Door datasets te scheiden kan Elasticsearch gemakkelijker specifieke typen verwerken en doorzoeken (specialisatie ten opzichte van generalisatie).

Tabel 4 - Implicaties één of meerdere indexen

Deze implicaties zijn echter grotendeels in te perken door wel overwogen Elasticsearch instellingen (specifiek het aantal *shards* per index) te gebruiken. De prestaties per shard zijn uiteraard wel weer afhankelijk van de beschikbare hardware.

Wanneer een zoekopdracht wordt uitgevoerd, wordt dit doorgestuurd naar een kopie van alle shards[1] in een index. Als we diezelfde zoekopdracht op meerdere indexen uitvoeren, gebeurt exact hetzelfde, alleen zijn er dan meer shards bij betrokken. Bijvoorbeeld, zoeken in 1 index met 50 shards is exact hetzelfde als zoeken in 50 indexen met elk 1 shard. Beide zoekopdrachten bevragen 50 shards. (Elasticsearch, 2016)

3.3 Conclusie

Zoals eerder in H2.2.1 beschreven, betreft de data in SKOS*, semantische data. Het is daarom niet meer dan logisch om deze data ook op een bijpassende manier op te slaan. Aangezien er reeds gebruik gemaakt wordt van een dergelijke manier (namelijk de triple-store), zal SKOS* verwerkt worden in de reeds bestaande triple-store (zie H2.2.1 en H3.1).

Dit wordt ondersteund op aanraden van SMW en instemming van de bedrijfsbegeleider.

[1] Een *shard* is een enkele instantie van Lucene[2] (simpel gezegd een enkele low-level "worker"), welke automatisch door Elasticsearch wordt beheerd. Een index is een verzameling van deze "workers".

[2] Lucene biedt krachtige full-text zoek functionaliteiten, geschreven in Java. Het kan gebruikt worden voor full-text indexing van database objecten en documenten in verschillende formaten.

4 Uitwerking

De in H2.2.1 beschreven vorm van SKOS* is momenteel niet volledig te realiseren. Er wordt gebruik gemaakt van Fuseki om de SPARQLStore aan te spreken, welke vervolgens de triple-store beheerd. Het toevoegen van een naam aan een triple wil echter niet lukken via Fuseki. De syntax die hiervoor nodig is lijkt niet te worden ondersteund door Fuseki of het is onbekend hoe dit wel gerealiseerd kan worden.

Neem bijvoorbeeld de in Tabel 5 beschreven triple inclusief naam:

Subject	Predicate	Object	Name
Dijk	*relatedToNonSkosTerm*	Steen	[uuid: 123]

Tabel 5 – Triple met naam

Door de toevoeging van de naam 123 aan de triple, ontstaat de volgende error in Fuseki:

```
Parse error: [line: #, col: #] Triples not terminated by DOT
```

In andere woorden, een vierde attribuut wordt gezien als het eerste attribuut van een nieuwe triple (waar normaliter dus een punt tussen moet staan). Dit veroorzaakt de error waardoor toevoeging niet mogelijk is.

Met een kleine wijziging is het wel mogelijk om de vorm toch te realiseren, al is het op een minder nette manier. Het object, Steen , wordt vervangen door de naam, 123 . Aan deze naam wordt vervolgens naast de frequentie en herkomst, ook het object Steen gekoppeld (zie Tabel 6 hieronder).

Subject	Predicate	Object	
Dijk	relatedToNonSkosTerm	[uuid: 123]	
[123]	hasFrequency	3x	
[123]	hasOrigin	ontwerpnota-2016.pdf	
[123]	concernsNonSkosTerm	Steen	← Extra triple

Tabel 6 – Toevoeging extra triple

> *Dit heeft dus als gevolg dat er voor elke unieke relatie een extra triple opgenomen moet worden. Dit is momenteel nog geen groot probleem maar men moet hier wel voor waken naarmate de triple-store, en met name het aantal SKOS* relaties daarin, gaat groeien. Dit is nadelig voor de nodige opslagruimte maar ook voor de prestaties bij het bevragen.*

Het is verder nog onbekend wat de impact hiervan is op het uiteindelijk te ontwikkelen prototype. De nodige gegevens (SKOS Concept, NSKOS term, frequentie en herkomst) zijn echter nu wel beschikbaar in de triple-store en kunnen opgevraagd worden voor gebruik in het prototype.

Er is een script gemaakt, waarmee SKOS* gegevens in CSV formaat, omgezet kunnen worden naar triples. Dit script, met de nodige documentatie, is te vinden in hz-search/csv-to-triples.php.

> *De SKOS concepten kunnen reeds geëxporteerd worden naar triple formaat (zoals bijvoorbeeld het hzbwnature.n3 bestand). Deze SKOS concepten zijn los toegevoegd aan de triple-store, samen met de nieuwe SKOS* triples.*

4.1 SPARQL query's

Hieronder is het voorbeeld uit H2.2.1, Tabel 1, uitgewerkt in SPARQL query's.

Er wordt hier verder niet ingegaan op de syntax van SPARQL. Neem de volgende relaties:

SKOS|Dijk ↔ **NSKOS|**Steen
3x in ontwerpnota-2016.pdf
5x in opbouw-dijken.pdf

In Tabel 7 hieronder staat de SPARQL query om deze relaties toe te voegen aan de triple-store:

```
@prefix eskos: <http://wiki.local/eskos#> .
@prefix wiki:  <http://wiki.local/index.php/> .

wiki:Dijk eskos:relatedToNonSkosTerm      <123> , <456> .

<123>
    eskos:hasFrequency          "3" ;
    eskos:hasOrigin             "ontwerpnota-2016.pdf" ;
    eskos:concernsNonSkosTerm   "Steen" .

<456>
    eskos:hasFrequency          "5" ;
    eskos:hasOrigin             "opbouw-dijken.pdf" ;
    eskos:concernsNonSkosTerm   "Steen" .
```

Tabel 7 – SPARQL query, SKOS relaties toevoegen*

In Tabel 8 hieronder staat de SPARQL query om deze gegevens op te halen:

```
PREFIX eskos: <http://wiki.local/eskos#>

SELECT ?Concept ?NSKOS Term ?Frequency ?Origin
WHERE
{
   ?Concept    eskos:relatedToNonSkosTerm   ?relation .
   ?relation   eskos:concernsNonSkosTerm    ?NSKOS Term .
   ?relation   eskos:hasFrequency           ?Frequency .
   ?relation   eskos:hasOrigin              ?Origin
}
```

Tabel 8 – SPARQL query, SKOS relaties opvragen*

In Tabel 9 staat het resultaat van deze query.

Concept	NSKOS_Term	Frequency	Origin
<http://wiki.local/index.php/Dijk>	Steen	3	ontwerpnota-2016.pdf
<http://wiki.local/index.php/Dijk>	Steen	5	opbouw-dijken.pdf

Tabel 9 - Voorbeeld van een SKOS relatie, verwerkt in de triple-store*

5 Literatuur

Elasticsearch. (2016). *Glossary of terms*. Opgehaald van Elastic · Revealing Insights from Data: www.elastic.co/guide/en/elasticsearch/reference/current/glossary.html#glossary-shard

Semantic MediaWiki. (2016, april 2). *Semantic MediaWiki*. Retrieved from Semantic MediaWiki: www.semantic-mediawiki.org

Bijlage 1: Zoekplan

Zoekplan

Semantische data opslaan

Expertise Management Wiki’s

zoekplan van Joos Mesie
studentnummer 48580
in het kader van de opleiding HBO-ICT aan de
HZ University of Applied Sciences, Vlissingen

Inhoud

1 Zoekplan

Door vooraf na te denken over zoektermen en invalshoeken, is het verzamelen van bronnen reproduceerdbaar. Tevens is er een duidelijk richting waardoor het makkelijker wordt om “on-topic” te blijven.

Omdat een gedeelte van de gevonden bronnen vermoedelijk internetbronnen zijn (blogs, forums, etc.) is ook gekozen om de CARS-checklist toe te passen op deze specifieke bronnen. Andere bronnen zoals wetenschappelijke artikelen of bronnen van erkende organisaties zoals Wikipedia etc., worden echter niet gecontroleerd met de CARS-checklist (zie H1.3).

Dit zoekplan wordt in rondes uitgevoerd. Tijdens het zoeken komen wellicht termen voorbij welke gebruikt kunnen worden als nieuwe zoektermen (ook wel sneeuwbal methode). Deze worden genoteerd en in een vervolg ronde gebruikt om door te zoeken.

Zoektermen zijn hier in het Engels beschreven, maar worden ook in het Nederlands ingevoerd.

1.1 Zoektermen

De zoektermen in ronde 1 (de initiële zoektermen) bedacht op basis van de vraag:

> *Hoe kan semantische data worden opgeslagen?*

De volgende afkortingen worden gebruikt:

SD	Semantic data
SMW	Semantic MediaWiki

1.1.1 Ronde 1

SD
How to store SD
How to store SD SMW
SD storage
SD storage SMW
Storing SD
Storing SD SMW

1.1.2 Ronde 2

Graph database
Named Graph
Relational databases SD
Triple store
SD management
SD model
SD model storage
Semantic modeling
Semantic Web Rule Language

1.1.3 Ronde 3

RDB2RDF
Large Triple store

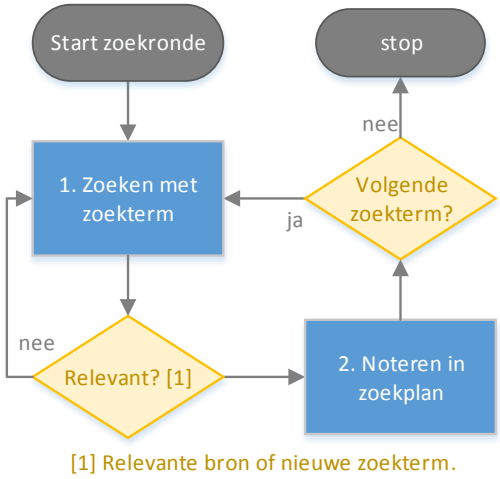
1.2 Zoekmethode

Figuur 1 (rechts) illustreert hoe een zoekronde wordt uitgevoerd.

1. Er wordt gezocht met een zoekterm uit deze zoekronde. Resultaten worden kort inhoudelijk bekeken en beoordeeld op relevantie.

2. Bronnen die relevant lijken of nieuw gevonden zoektermen (voor een vervolg zoekronde) worden genoteerd in het zoekplan;

Aan het einde van een zoekronde wordt gekeken of er een nieuw zoekronde gewenst is.



Figuur 1 – Uitvoering zoekronde

Na uitvoering van alle zoekrondes is er een lijst met gevonden bronnen. Deze bronnen worden vervolgens beter inhoudelijk bekeken en indien nog steeds relevant, getoetst aan de CARS-checklist (zie H1.3).

1.3 CARS checklist

Literatuur, wetenschappelijke artikelen, standaarden zoals beschreven op bijv. W3C of ISO of bronnen van erkende organisaties (bijvoorbeeld Wikipedia), hoeven niet te worden gecontroleerd op betrouwbaarheid. Deze worden immers voor publicatie al door (strengere en meer uitgebreide) soortgelijke processen gelopen of worden door meerdere personen en organisaties onderhouden. Andere bronnen, bijvoorbeeld website artikelen, blogs, forums etc., worden wel op betrouwbaarheid gecontroleerd.

De CARS-checklist (Credibility, Accuracy, Reasonableness, Support) toetst bronnen op vier hoofdpunten, als volgt:

Credibility: Een bron welke is gemaakt door een persoon of organisatie welke bekend en betrokken is met het onderwerp.
- Is er een uitgever of sponsorende organisatie? Is de organisatie een autoriteit m.b.t. het onderwerp?
- Zijn er spellingfouten, dode links, of andere problemen welke wijzen op een gemis aan kwaliteitscontrole?

Accuracy: Een bron met actuele, complete en correcte informatie.
- Komt de informatie uit de bron overeen met andere bronnen?
- Spreekt de bron zichzelf tegen?
- Wat is de datum van publicatie of copyright?
- Hoe recentelijk is de bron bijgewerkt?

Reasonableness: Een bron welke eerlijk en onbevooroordeeld is.
- Heeft de auteur, uitgever, of sponsor een vooroordeel?
- Wat is de motivatie of het doel van de bron? (Commercieel? Onderwijzen? Een bepaalde visie vooruit helpen?)

Support: Een bron met verifieerbare informatiebronnen.
- Zijn de bronnen vermeld? Kunnen ze gecontroleerd worden?
- Is er een manier om contact op te nemen met de auteur of organisatie?

2 Gevonden bronnen

Bronnen die niet met de CARS checklist gecontroleerd hoeven worden (literatuur en wetenschappelijke artikelen, standaarden).

Bron	Ronde - Zoekterm
wikipedia.org/wiki/Semantic_data_model	*1 - semantic data*
semantic-mediawiki.org/wiki/Help:Using_SPARQL_and_RDF_stores	*1 - how to store semantic data semantic media wiki*
wikipedia.org/wiki/Triplestore	*1 - semantic data storage*
noraonline.nl/wiki/Nationaal_Semantisch_Vlak	*1 - semantic data storage [NL]*
wikipedia.org/wiki/Graph_database	*2 - graph database*
wikipedia.org/wiki/Named_graph	*2- named graph*
w3.org/DesignIssues/RDB-RDF.html	*2 - relational databases semantic data*
ceur-ws.org/Vol-837/paper6.pdf	*2 - semantic data model storage*

2.1 CARS checklist bronnen

De overige bronnen zijn hieronder uitgewerkt met voor elke bron, per punt, een indicatie of de bron voldoet aan CARS. Voor de uitwerking hierop zie H2.2.

+	Goed
±	Voldoende
–	Onvoldoende
~	Onbekend

Bron	Ronde - Zoekterm	Toelichting	C	A	R	S
linkeddatatools.com/introducing-rdf	*1 - how to store semantic data*	H2.2.1	+	±	+	+
linkeddatatools.com/semantic-modeling	*2 - semantic data model*	H2.2.1	+	±	+	+
iks-project.eu/academy/storing-and-accessing-semantic-data	*1 - semantic data storage*	H2.2.2	+	+	+	+
dataversity.net/relational-database-and-the-semantic-web/	*2 - relational databases semantic data*		+	+	+	+
neo4j.com/developer/graph-db-vs-rdbms	*2 - graph database*		+	+	±	±

2.2 CARS toelichting

2.2.1 linkeddatatools.com/introducing-rdf

Credibility **Bron overzicht**

LinkedDataTools.com is opgezet door professionele ontwikkelaars welke de web community willen helpen over te stappen naar het semantische web (web 3.0).

Het is een simpele website die zoals ook daarop staat, voor training doeleinden is ontwikkeld. De content, met name het training gedeelte, is duidelijk, overzichtelijk en goed toegelicht.

Accuracy

Opgezet door verschillende ontwikkelaars, er lijkt geen tegenspreek voor te komen.

Publicatie datum: > 2009 Datum laatste update: 2015 Copyright: 2015

Reasonableness

Auteurs willen de web community helpen over te stappen naar het semantische web, het betreft grotendeels instructie zonder vooroordeel.

Support

Er wordt naar verschillende bronnen verwezen middels hyperlinks in de content zelf.

Bezoekers kunnen gebruik maken van een support forum en de organisatie is per e-mail te bereiken (support@linkeddatatools.com).

2.2.2 iks-project.eu/academy/storing-and-accessing-semantic-data

Credibility **Bron overzicht**

Publicerende organisatie: IKS Interactive Knowledge Stack

IKS wordt deels door de europese unie gefinancieerd met een subsidie van €6.58 miljoen. Het wordt bestuurd door een consortium van zeven onderzoekspartners en zes industriële partners. IKS is een open source technologie platform voor semantisch versterkte content management systemen.

Er werdt gewerkt aan een semantisch framework, nauwlettend op standaarden in het semantische web. Ook waren er verschillende samenwerkingsverbanden en werden er allerlei activiteiten georganiseerd, zoals ontmoetingen en workshops.

Sinds maart 2013 is het IKS project geëindigd. Alle bevindingen zijn beschikbaar op de bron website.

Accuracy

IKS beschrijft o.a. veel informatie en nieuws m.b.t. W3C standaarden en praktijken op het gebied van het semantische web.

Publicatie datum: 2012 Datum laatste update: 2013 Copyright: 2013

Reasonableness

Er is geen vooroordeel. Men wil een open source platform aanbieden zodat CMS ontwikkelaars semantisch versterkte software kunnen produceren. De motivatie is onderwijzen en instrueren.

Support

De website is één grote verzameling aan informatie en links naar relevante content. De organisatie kan benaderd worden via:

Project Coordinator:

Salzburg Research Forschungsgesellschaft

Jakob Haringer Straße 5/3

5020 Salzburg, Austria

iks-office@salzburgresearch.at

IKS Media Contact:

John Pereira

Salzburg Research Forschungsgesellschaft

Jakob Haringer Straße 5/3

5020 Salzburg, Austria

Tel: +43 662 2288 247

john.pereira@salzburgresearch.at

2.2.3 dataversity.net/relational-database-and-the-semantic-web

Credibility **Bron overzicht**

DATAVERSITY is een online educatie portal welke gericht zijn op het onderwerp van data beheer. Het bedrijf wordt ondersteund door allerlei experts, ontwikkelaars en gebruikers wereldwijd, welke in allerlei activiteiten zoals conferenties, live webinars, white papers en andere zaken deelnemen.

Accuracy

De informatie wordt door een community onderhouden en zal grotendeels overeen komen met andere bronnen.

Publicatie datum: jun 2010 Datum laatste update: 2010 Copyright: 2016

Reasonableness

Er is geen vooroordeel aanwezig. Men wil collectief de web community hoge kwaliteit bronnen aanbieden.

Support

De organisatie bevat een aantal contactpersonen zoals de executive editor en marketing manager. De auteur is niet persoonlijk aanspreekbaar via de website, maar auteurs die artikels leveren aan de website zijn over het algemeen bekend en contactgegevens zijn meestal te achterhalen via zoekmachines.

In dit geval is de auteur Juan Sequeda te bereiken op:

jsequeda@cs.utexas.edu

GDC (Gates Dell Complex) 7.416

2.2.4 neo4j.com/developer/graph-db-vs-rdbms

Credibility **Bron overzicht**

Alhoewel het een commercieel bedrijf betreft welke haar producten wil promoveren, heeft het bedrijf verschillende uitreikingen gehad en zijn er duizenden organisaties die haar software gebruiken. De software is open source sinds 2010 en er zijn aparte edities zoals de enterprise- en community edities.

De bron ziet er goed verzorgd uit en bevat naast software specifieke educatieve bronnen, ook bronnen wat betreft graph en graph databases in het algemeen.

Accuracy

De wat meer algemene informatie komt overeen met wat er tot nu toe is gelezen hierover.

Publicatie datum: mrt 2016 Datum laatste update: 2016 Copyright: 2016

Reasonableness

Op elke pagina worden de eigen producten wel op één of andere manier verwerkt in de content. Zoals “In onze software werkt dit zus en zo”. Maar afgezien van dat lijkt de algemene informatie onbevooroordeeld. De motivatie van de bron is gesplitst. Er is een duidelijk commercieel doel maar men wil ook de web community helpen om te groeien (het liefst natuurlijk met de software van het bedrijf zelf).

Support

Bronnen zijn vermeld maar betreffen voornamelijk pagina’s op de website zelf.

Er zijn verschillende manieren om in contact te komen met het bedrijf zoals bekende social media kanalen maar ook met onderstaande:

1-855-636-4532	US	info@neotechnology.com
+44 808 189 0493	UK	uk@neotechnology.com
Please email:	Germany	vertrieb@neotechnology.com
+33 (0) 8 05 08 03 44	France	ventes@neotechnology.com

Bijlage 3: Onderzoek relevantiebepaling SKOS*

Onderzoek

Relevantie bepaling SKOS*

Expertise Management Wiki's

bevindingen van Joos Mesie
studentnummer 48580
in het kader van de opleiding HBO-ICT aan de
HZ University of Applied Sciences, Vlissingen

Inhoud

1 Inleiding

Dit document beschrijft de bevindingen bij de derde en laatste deelvraag:

3. Hoe kan er in het bepalen van de relevantie van de zoekresultaten rekening gehouden worden met SKOS?*

De volgende activiteiten zijn uitgevoerd:

3.1 Dataset selecteren uit EMM wiki's en een kopie hiervan aanvullen met SKOS* relaties;

3.2 Onderzoeken hoe relevantie momenteel wordt bepaald en dit visualiseren;

3.3 Onderzoeken wat relevantie is, welke aspecten iets over relevantie kunnen zeggen en hoe de relaties beschreven in SKOS* gebruikt kunnen worden om relevantere zoekresultaten te presenteren;

3.4 Beschrijven hoe de relaties beschreven in SKOS* gebruikt kunnen worden om relevantere zoekresultaten te presenteren;

3.5 Requirements naar eigen inzicht opstellen, prioriteren met de MoSCoW methode en eventueel aanvullen in overleg met bedrijfsbegeleider;

3.6 Concept bewijzen ontwikkelen en tegelijkertijd functionele- en technische specificaties beschrijven. Tijdens en na ontwikkeling terugpakken naar de requirementsanalyse als zijnde checklist. Test-cases beschrijven en uitvoeren om de oude situatie met de conceptbewijzen te vergelijken.

De activiteiten 3.4, 3.5 en 3.6 zijn beschreven in de ontwerpspecificatie (zie Bijlage 1: Ontwerpspecificatie). H2 beschrijft de gebruikte dataset. Het onderzoek naar relevantie, de huidige relevantiebepaling, en hoe SKOS* hier een rol in gaat spelen, zijn in H3 beschreven.

1.1 Terminologie

Kort wat gebruikte termen toegelicht:

- Met **SKOS** doelt men in dit onderzoek op een domein specifieke thesaurus, die is gespecificeerd in een Simple Knowledge Organization System (SKOS);

- **NSKOS**, beschrijft genormaliseerde termen, afkomstig uit documenten van de wiki, waarbij de termen zelf niet in SKOS voorkomen maar wel met termen uit SKOS in één document zitten (waardoor een relatie wordt gesuggereerd);

- **SKOS***, er zijn nu twee thesauri, één met SKOS termen en één met NSKOS termen. De relatie tussen SKOS en NSKOS termen bestaat uit een frequentie (het gewicht van de relatie) en de herkomst (het brondocument). Deze relaties worden beschreven in de "verrijkte" SKOS (voortaan SKOS*).

- Een **triple-store** is een database specifiek ontworpen voor het opslaan en opvragen van **triples** door semantische query's. Veelal ondersteunen **triple-store**s ook zogenaamde **quads** en zijn het theoretisch gezien **quad-stores**;

- Een **triple** is een data entiteit samengesteld uit korte zinnen (subject, predicate, object), bijvoorbeeld: Duin onderdeelVan Dijkvak. Een **quad** is een **triple** met een specifieke naam, waarmee de hele triple hergebruikt kan worden in andere triples.

2 Dataset

Alle SKOS concepten zijn beschikbaar als triples (te vinden in het `hzbwnature.n3` bestand). Dit bestand is te vinden in . Deze worden aangevuld met een kleine set SKOS* relaties. Deze SKOS* relaties zijn met de volgende stappen gerealiseerd:

1. Alle resource descriptions (PDF documenten) van www.deltaexpertise.nl, exporteren in JSON:

```
deltaexpertise.nl/wiki/api.php?action=ask&format=json
      &query=[[Category:Resource Description]]|limit=2500
```

2. Zoeken op de woorden 'Dijk' en 'Steen' in de titels van deze documenten voor een redelijk specifieke set documenten. Daarna op basis van woorden uit de titels van de gevonden documenten uitgebreider zoeken, voor een wat bredere set documenten. In dit geval met de woorden 'Bekleding' en 'Klemming'. Er zijn totaal 7 documenten gekozen;

3. Middels een Microsoft Word macro het aantal unieke woorden tellen per specifiek document. Hieruit komt een lijst zoals bijvoorbeeld:

```
AfschuivingGepenetreerdeBekledingenOvergang.pdf
      62    de
      32    van
      28    een
      20    het
      16    toplaag
      15    op
      15    bekleding
      ...
```

4. De lijsten van de 7 gekozen documenten in Microsoft Excel naast elkaar gezet en daaruit een aantal **[woorden]** gekozen. Hiervoor is geselecteerd op basis van frequentie (hoog, middel en laag) maar ook de herkomst (sommigen komen niet in alle 7 documenten voor, andere wel). De volgende woorden zijn gekozen (eventuele meervoudsvormen zijn hierbij opgeteld):

```
afsluitdijk
bekleding
breuksteen
dijkbeheerder
dijkbekleding
dijkvak
element
grasbekleding
klei
klein
kleipakket
klemming
klemmingsonderzoek
klemsteen
materiaal
meting
mijnsteen
onderlaag
plaatbekleding
stabiliteit
steen
steenachtige
steenglooiing
steentoets
steenzetting
taludhelling
toplaagelement
toplaagstabiliteit
waddenzeedijk
```

5. Met *conditional formatting* in Excel, alle (gedeeltelijk) overeenkomende **[woorden]** per kolom gekleurd. Vervolgens deze kolommen gefiltert op "geen kleur" om zodoende alle stopwoorden en overige woorden uit de lijsten te verwijderen. Hiervoor is de volgende formule gebruikt:

> *'Format only cells that contain specific text' : [woorden]*

Aanvullend zijn hier vier SKOS concepten uit gekozen, op basis van verschillen in herkomst en frequentie, namelijk 'Dijk', 'Meerdijk', 'Steenbekleding' en 'Talud';

6. Per document de gekozen SKOS concepten in relatie brengen met alle overige NSKOS termen waarvan de frequentie > 10;

7. Deze relaties omgezet naar CSV middels de `concatenate` functionaliteit in Excel. Vervolgens zijn deze gegevens via een PHP script omgezet naar triples voor gebruik in de triple-store.

> *Dit script en gerelateerde bestanden, is te vinden in de `hz-search` projectmap:*
> *`hz-search/csv-to-triples/*`*
> *De output van dit script kan 1 op 1, via Fuseki, in de triple-store worden gezet.*

De dataset bestaat 365 SKOS concepten en 166 relaties van 4 SKOS concepten naar 29 NSKOS termen verspreid over 7 documenten. Er zijn totaal 830 triples nodig om deze 166 relaties te beschrijven. Dit komt omdat:

- De fuseki interface momenteel geen ondersteuning biedt voor quads, waardoor er per relatie één extra triple nodig is om eigenschappen aan een relatie te koppelen;

- Er is gekozen om de SKOS frequentie (hoe vaak het SKOS concept voorkomt in een document) ook te beschrijven in een relatie (hier is één extra triple voor nodig).

Elke relatie bestaat nu uit 5 triples, vandaar 830 triples (`166 * 5 = 830`). Naast deze triples bestaat SKOS zelf uit 2058 triples, voor een totaal van 2888 triples.

3 Relevantie en relevantiebepaling

Het bepalen van de relevantie van een document kan snel complex worden. Maar wat bedoelen wij nou eigenlijk met relevantie? Waardoor wordt relevantie bepaald? Wanneer is iets relevant en in welke mate is het dan relevant?

> *re·le·van·tie* *1. belang, betekenis* *(Van Dale, 2016)*

In dit hoofdstuk worden verschillende aandachtspunten beschreven voor het bepalen van relevantie. Let op, deze punten zijn niet uitputtend, er bestaan geheid meer mogelijkheden.

3.1.1 Datum van publicatie of wijziging;

3.1.2 Link analyse;

3.1.3 Naam- object herkenning;

3.1.4 Populariteit;

3.1.5 Term frequentie;

3.1.6 Inverse document frequentie (IDF)

3.1.7 Term locatie;

3.1.8 Term nabijheid (proximity of terms);

3.1.9 Term combinaties

Alle besproken punten zeggen op een bepaalde manier iets over relevantie en kunnen in zekere mate meewegen bij het bepalen van de relevantiescore. Daarnaast kunnen een aantal van deze punten gebruikt worden voor een ‘geavanceerde’ vorm van zoeken (denk aan filters en/of sortering op basis van datum of populariteit).

3.1.1 Datum van publicatie of wijziging

De leeftijd van een document (sinds publicatie of laatste wijziging) kan gebruikt worden om te bepalen hoelang een document “houdbaar” is.

Een document uit bijvoorbeeld 1990 is mogelijk minder relevant zijn dan een document uit 2010, maar dat hoeft uiteraard niet. Vermoedelijk zullen gebruikers, als zij een document moeten kiezen, de meer recente variant kiezen.

3.1.2 Link analyse

Een document met koppelingen naar andere documenten (bijvoorbeeld hyperlinks), kan worden beschouwd als zijnde relevanter dan een document zonder koppelingen. Men kan bijvoorbeeld stellen dat een document met veel koppelingen meer een deel van het geheel lijkt dan een document zonder koppelingen.

Tevens kunnen dergelijke koppelingen gebruikt worden om de gevonden resultaten aan te vullen met mogelijk gerelateerde documenten. Denk bijvoorbeeld aan een bibliografie, waarin potentieel gerelateerde documenten in kunnen staan. De relevantie kan wellicht per bron verder worden bijgesteld, op basis van bijvoorbeeld, de herkomst (wetenschappelijke artikelen of blog?) maar ook hoe recent een bron is bijgewerkt (zie ook H3.1.1).

3.1.3 Naam- object herkenning (ook wel Named Entity Recognition, NER)

Het komt voor dat zoekterm(en) niet letterlijk bedoeld zijn maar een bepaalde betekenis hebben voor de gebruiker en/of het systeem. NER betreft het identificeren en classificeren van elementen in tekst waar mogelijk een andere betekenis aan verleend kan worden.

> *Drie universeel aanvaarde classificaties zijn: persoon, locatie en organisatie. Andere mogelijkheden zijn erkenning van datum en tijd uitdrukkingen, maatstaven (procent, valuta, gewicht, etc.) en domein specifieke entiteiten zoals vakjargon. (Maynard, 2001)*

Een voorbeeld kan zijn het Projectbureau Zeeweringen. Stel men zoekt op PBZ, in plaats van een letterlijke zoekopdracht kan PBZ herkend worden als organisatie. Door dit te constateren kunnen de zoekresultaten relevanter worden gemaakt voor de gebruiker. Documenten met de letterlijke tekens PBZ zijn wellicht minder relevant zijn dan documenten die gaan over de organisatie (entiteit) PBZ. Bovendien ontstaat de mogelijkheid om de gebruiker te voorzien van gespecificeerde informatie, bijvoorbeeld de contactgegevens van PBZ.

Stel er worden twee zoekopdrachten uitgevoerd, namelijk “PBZ” en “projectbureau zeeweringen”. Het is te verwachten dat de zoekresultaten van beide zoekopdrachten grotendeels (al dan niet volledig) dezelfde resultaten zullen vinden. In Figuur 1 hieronder, is te zien dat wanneer NER niet is toegepast, deze zoekopdrachten totaal verschillende zoekresultaten vinden.

Figuur 1 – Zoekresultaten ‘PBZ’ en ‘Projectbureau Zeeweringen’

> *Het verschil in zoekresultaten is in dit geval gedeeltelijk op te lossen door de afkorting ‘PBZ’ als alternatieve naam of synoniem op te nemen voor ‘Projectbureau Zeeweringen’. Echter is daarmee het entiteit zelf niet herkend en wordt er nog steeds letterlijk gezocht naar de zoektermen.*

Een greep uit bestaande projecten welke intensief bezig zijn met NER:

- Apache Open NLP opennlp.apache.org;
- A Nearly New Information Extraction system (ANNIE) gate.ac.uk/ie/annie.html;
- Named Entity Recognition (NER) nlp.stanford.edu/ner;
- Open Calais opencalais.com.

3.1.4 Populariteit

Documenten of termen die vaker worden gebruikt of opgezocht, kunnen statistisch gezien als relevanter worden beschouwd.

> *In het geval van documenten kan dit overigens ook betekenen dat de vindbaarheid van soortgelijke (minder populaire) documenten slechter is, wat wijst op een mogelijke fout in de zoekmachine.*

Afgezien daarvan kan populariteit een goede manier zijn om de meest 'begeerde' content naar voren te halen (bijvoorbeeld in de suggesties onder het zoekveld).

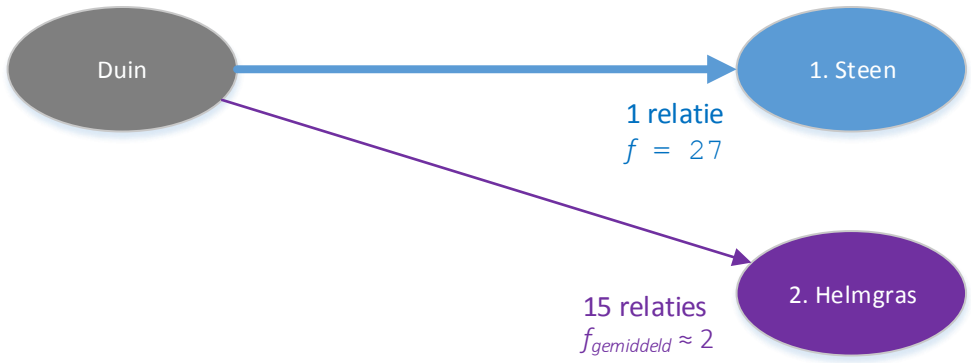
3.1.5 Term frequentie (TF)

TF is onderdeel van SKOS* (hoewel in simpele vorm). Elke relatie in SKOS* betreft een specifieke combinatie SKOS concept, NSKOS term en het desbetreffende document (herkomst). Een relatie tussen een SKOS concept en een NSKOS term bestaat uit het volgende:

- Een referentie naar de herkomst;
- De TF van het SKOS concept (f_{skos}), hier wordt echter nog niks mee gedaan in mijn onderzoek;
- De TF van de NSKOS term (f_{nskos} of gewoon f).

Eén aandachtspunt is het bekijken van individuele- maar ook globale relaties. De frequentie van één relatie, zegt statistisch gezien iets over de relevantie van de NSKOS term en het bijbehorende document. Er kunnen echter meerdere relaties bestaat tussen één SKOS concept en één NSKOS term, verspreid over meerdere documenten.

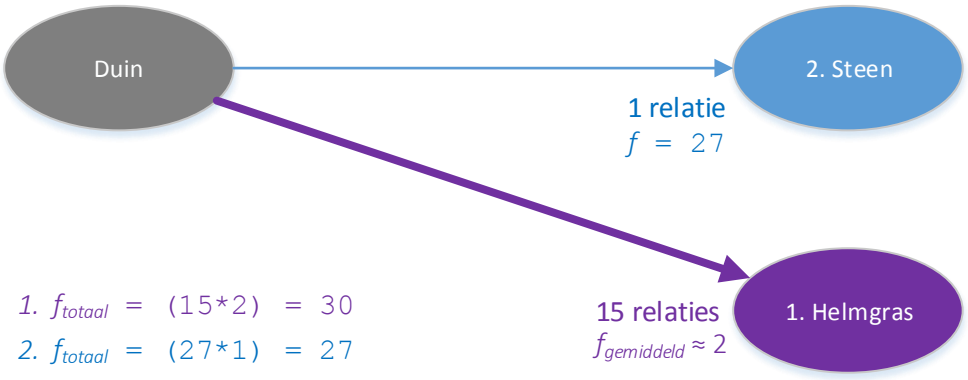
Als men kijkt naar individuele relaties, dan weegt volgens Figuur 2 (hieronder) de relatie tussen Duin en Steen zwaarder dan de relatie tussen Duin en Helmgras.



Figuur 2 – Welke is relevanter?

Kijkt men verder dan alleen individuele relaties, dan weegt de term Helmgras (en daarmee de bijbehorende documenten) mogelijk zwaarder dan de relatie tussen Duin en Steen. Immers, de enkele relatie met Steen kan een toevalligheid zijn en heeft alleen betrekking op één specifiek document. Helmgras daarentegen lijkt veel meer deel uit te maken van Duin omdat dit de relatie hiertussen veel vaker voorkomt.

Door gebruik van aggregatie is het mogelijk om betekenis te verlenen aan de frequenties van alle relaties tussen een specifiek SKOS concept en NSKOS term. Hiermee kan men vervolgens betere inschatten hoe sterk de gecombineerde relatie is. Op basis daarvan kunnen de specifiek onderliggende relaties eventueel worden versterkt. Neem bijvoorbeeld de relaties beschreven in Figuur 3 hieronder.

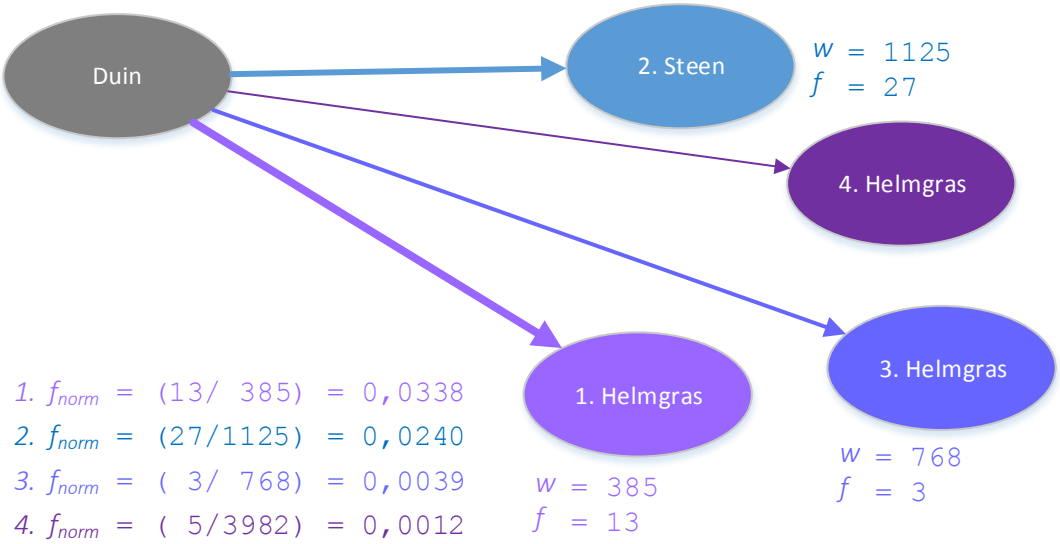


Figuur 3 – Aggregatie van frequenties

Een tweede aandachtspunt is het normaliseren van de TF (zie Figuur 4 hieronder). Denk aan een lage- of juist hoge frequentie in een groot- of juist klein document.

> *Sinds elk document verschilt in lengte, is het mogelijk dat een term veel vaker voorkomt in lange documenten dan korte documenten. Daarom wordt deze frequentie veelal gedeeld door het totale aantal termen in een document als een manier van normalisatie. (Manning, Raghavan, & Schütze, 2008)*

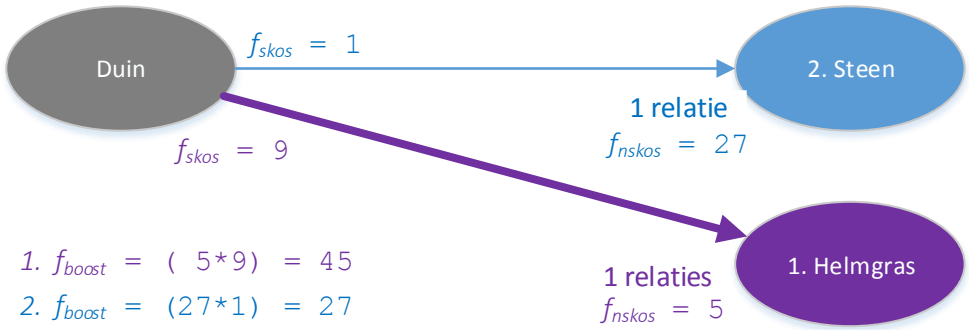
Het is wenselijk om een frequentie per bepaalde eenheid te hebben (de genormaliseerde frequentie, f_{norm}). Zo kan de TF geschaald worden naar de grootte van documenten. Bijvoorbeeld door de frequentie (f) te delen door het aantal woorden in een document (w): $f_{norm} = f / w$



Figuur 4 – Frequentie normalisatie

Een derde is de frequentie van het SKOS concept in een document (zie Figuur 5 hieronder). Deze frequentie kan worden gebruikt om de NSKOS frequentie bij te stellen, bijvoorbeeld door multiplicatie. Een relatie, gebaseerd op een SKOS frequentie van 1 is wellicht minder relevant dan een relatie gebaseerd op een SKOS frequentie van 9.

> *SKOS frequenties worden meegenomen in de triple-store. Er wordt echter, zoals eerder benoemd, in dit onderzoek nog geen gebruik van gemaakt.*



Figuur 5 – SKOS frequenties

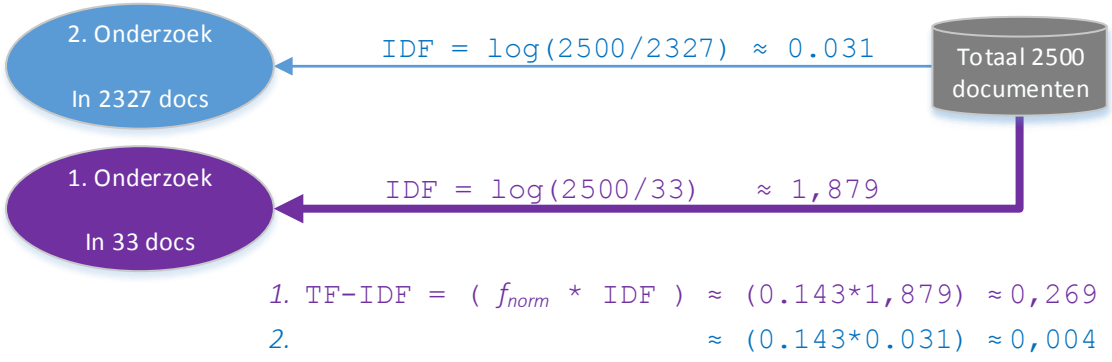
> *Het is aan te raden om buitenom dit onderzoek verder te kijken naar de mogelijkheden die term frequenties bieden. Een combinatie van de hierboven behandelde aandachtspunten (samen met de in H3.1.6 besproken frequentie) kan wellicht leiden tot verbetering van de zoekresultaten.*

3.1.6 Inverse document frequentie (IDF)

IDF wordt gebruikt om de TF (zie H3.1.5) van veel voorkomende termen (dus ook stopwoorden) te schalen met de TF van meer zeldzame termen. Sommige woorden komen nou eenmaal vaker voor in documenten dan anderen, terwijl deze niet perse meer relevant zijn.

> *Term frequentie-inverse document frequentie (TF-IDF) is een numerieke statistiek welke bedoeld is als reflectie van de relevantie van een woord in relatie tot een document in een collectie van documenten. (Wikipedia, 2016)*

Stel er is een relatie met de NSKOS term *Onderzoek* waarbij $f_{norm} = \mathbf{0.143}$. De relatie betreft één document van 1000 woorden ($w = 1000$). In Figuur 6Figuur 5 hieronder is te zien hoe de IDF wordt berekend wanneer het een veelgebruikte term betreft (komt voor in 2327 documenten) of juist een zeldzame (komt voor in 33 documenten).



Figuur 6 – IDF ter compensatie van TF

3.1.7 Term locatie

Met betrekking tot:

- **Document lengte**, een 'snippet' van enkele regels is wellicht minder relevant dan een rapport van tientallen pagina's. Het gaat hier niet perse om het aantal woorden en ook niet met de normalisatie van een frequentie op basis van het aantal woorden in een document. Hier wordt bedoeld dat een relatie gebaseerd op een klein document in z'n geheel wellicht minder zwaar moet meewegen dan een document met 20 pagina's, afgezien van de (genormaliseerde) frequentie;
- **Positie binnen een document**, komt een term voor in een titel dan zal dit wellicht zwaarder wegen dan in een paragraaf;
- **Soort document (herkomst)**, bepaalde documenten zullen zwaarder wegen omdat dit bijvoorbeeld wetenschappelijke publicaties zijn t.o.v. een jaaroverzicht of perspublicatie (afhankelijk van welke soort documenten belangrijker zijn voor een wiki);
- **Zinsontleding**, wellicht weegt een resultaat met de zoekterm als onderwerp zwaarder dan als zelfstandig naamwoord.

3.1.8 Term nabijheid (proximity of terms)

Bij zoekopdrachten met meerdere woorden, of ter versterking van relaties tussen twee termen, kan gekeken worden naar de nabijheid van deze termen. Neem bijvoorbeeld de zoekopdracht:

> "*opblaasbare waterkering*".

Neem verder de twee teksten:

> "…*moderne opblaasbare waterkering oplossingen*…";
>
> "…*opblaasbare zakken. Tot slot zijn waterkeringen*…"

De bovenste tekst zal wellicht relevanter zijn voor deze zoekopdracht dan de onderste tekst.

3.1.9 Term combinaties

Term combinatie is eigenlijk een uitbreiding van term locatie (zie H3.1.7) maar wordt hier los beschreven omdat dit in dit onderzoek een speciaal geval is. Het komt namelijk voor dat een SKOS concept (gedeeltelijk) gecombineerd is met andere SKOS concepten.

Neem bijvoorbeeld het SKOS concept *Bekleding*. Dit concept wordt meerdere malen gecombineerd met andere SKOS concepten om zo een nieuw SKOS concept te vormen, namelijk:

- Dijkbekleding
- Taludbekleding

Ook zijn er combinaties met NSKOS termen, namelijk:

- Gepenetreerde_bekleding
- Steenbekleding

Dit kan een rol spelen bij relevantiebepaling maar ook bij het vinden van mogelijk gerelateerde informatie, in dit geval bij het zoeken naar *Bekleding* kunnen de concepten Dijk en Talud ook naar voren komen samen met de NSKOS term Steen.

> *In ieder geval moet men opletten dat bij een zoekopdracht naar Dijk, het SKOS concept dijkbekleding alleen wordt gevonden doordat er toevallig in de definitie daarvan letterlijk 'Dijk' staat. Wordt het definitie veld echter niet doorzocht dan is Dijkbekleding niet onderdeel van de zoekresultaten wat natuurlijk niet klopt.*
> *Uiteindelijk zal dit niet vaak voorkomen, als je het document Dijkbekleding bekijkt dan moet daar haast wel ergens een keer 'dijk' los in staan, maar wel een punt om op te letten!*

3.2 Huidige relevantie bepaling

Het toekennen van een relevantiescore wordt in Elasticsearch gedaan. Buitenom deze score wordt eigenlijk niks gedaan wat betreft relevantiebepaling. De resultaten komen gesorteerd terug en worden ook zo weergegeven. Hoe die score wordt berekend hangt af van het type query dat wordt uitgevoerd. Standaard wordt er echter gebruik gemaakt van het zogenaamde 'similariteit algoritme' van ES, welke rekening houdt met de volgende factoren:

- Term frequentie (TF) , hoe vaak een term in een document voorkomt, zie H3.1.5;

De TF voor term t in document d is de wortel van het aantal keren dat een term voorkomt in een document. (Elasticsearch, 2016)

$$TF(t \text{ in } d) = \sqrt{frequency}$$

- Inverse document frequentie (IDF), hoe vaak de term voorkomt in alle documenten, zie H0;

De inverse document frequentie voor term t is het logoritme van het aantal documenten in de index (n), gedeeld door het aantal documenten welke de term bevatten (f). (Elasticsearch, 2016)

$$IDF(t) = 1 + \log(n / (f + 1))$$

- Veld lengte, hoe langer een veld, hoe minder relevant de woorden in dat veld zijn. Een term welke in bijvoorbeeld een titel veld zit zal zwaarder wegen dan diezelfde term in een content veld (vergelijkbaar met Figuur 4 – Frequentie normalisatie, zie H3.1.5).

De veld lengte normalisatie l_{norm} is de inverse wortel (ook wel de reciproke genoemd) van het aantal termen k in het veld. (Elasticsearch, 2016)

$$l_{norm} = 1 / \sqrt{k}$$

Deze drie factoren worden los berekend en het product van deze drie factoren is de relevantie score van het desbetreffende document.

3.2.1 Simpel voorbeeld

Neem het onderstaande document met ID 1, in een aparte index (waar niks anders in zit dan dit document).

```
PUT /losseindex/1
{
   text: "Oude paarden jaagt men aan de dijk, oftewel iemand aan
          de dijk zetten."
}
```

Vervolgens wordt er een zoekopdracht uitgevoerd met de term *Dijk* door middel van de onderstaande query naar Elasticsearch te sturen:

```
GET losseindex/_search?explain
{
   query: {
     match: {
        text : "dijk"
     }
   }
}
```

> *De `explain` parameter zorgt ervoor dat de berekende waarden van TF, IDF en l_{norm} ook worden meegegeven als resultaat.*

De volgende berekeningen vinden plaats in dit geval:

$$TF = \sqrt{2} = 1{,}414$$

$$IDF = 1 + \log\left(1 / (1 + 1)\right) = 0{,}307$$

> *Voor het berekenen van IDF wordt gebruik gemaakt van het natuurlijke logoritme, niet het base10 logoritme.*

$$l_{norm} = 1 / \sqrt{13} = 0{,}277$$

De standaard scoring implementatie codeert de veld lengte waarden als "single byte" voordat het wordt opgeslagen. Tijdens het zoeken wordt deze waarde gelezen uit de index en gedecodeerd naar een "float". Dit coderen/decoderen reduceert de grootte van de index maar resulteert in precisieverlies. (Apache Software Foundation, 2014)

> *De volgende expressie is niet gegarandeerd.*
> `decode( encode( x ) ) = x`
> *In het geval van de eerder berekende l_{norm} van 0.277 resulteert dit in:*
> `decode( encode( 0.277 ) ) = 0.25`

Dit wordt gerationaliseerd op basis van de moeilijkheid (en onnauwkeurigheid) waarmee een gebruiker zijn of haar informatiebehoefte uitdrukt. In andere woorden, alleen grote verschillen doen er toe. (Apache Software Foundation, 2014)

De relevantiescore wordt berekend door de bovenstaande drie producten met elkaar te vermenigvuldigen.

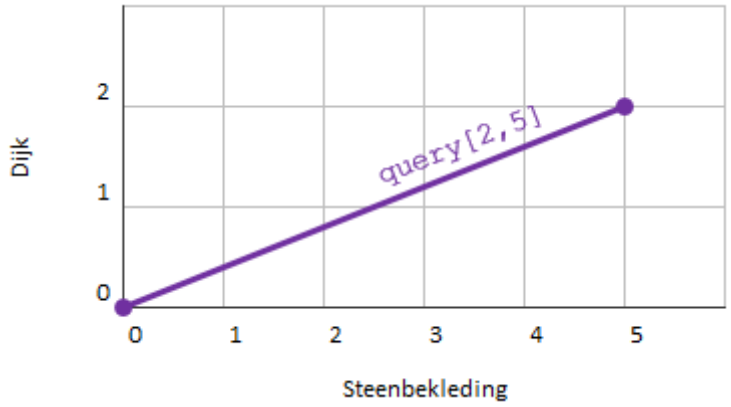
$$score = TF * IDF * l_{norm} = 1.414 * 0.307 * 0.25 = 0,1085$$

Uiteraard bestaan query's vaak uit één of meer termen. Daarom is er behoefte aan een manier om de gewichten van meerdere termen te combineren. Hiervoor maken wij gebruik van het vector space model. (Elasticsearch, 2016)

3.2.2 Vector Space Model (VSM)

Met VSM wordt in ES een query bestaand uit meerdere termen, met een document vergeleken. Hiervoor wordt van zowel de query als het document een vector gemaakt. Een vector is hier niks anders dan een lijst met nummers, in het geval van Elasticsearch, de TF-IDF zoals beschreven in H3.2.

Stel dat de term *dijk* vaak voorkomt en daarom een IDF heeft gekregen van 2. De term *steenbekleding* komt minder vaak voor en heeft daarom een IDF van 5 (zie Figuur 7). De vector voor de query is dan `[2,5]`.



Figuur 7 – Een query, gerepresenteerd als vector

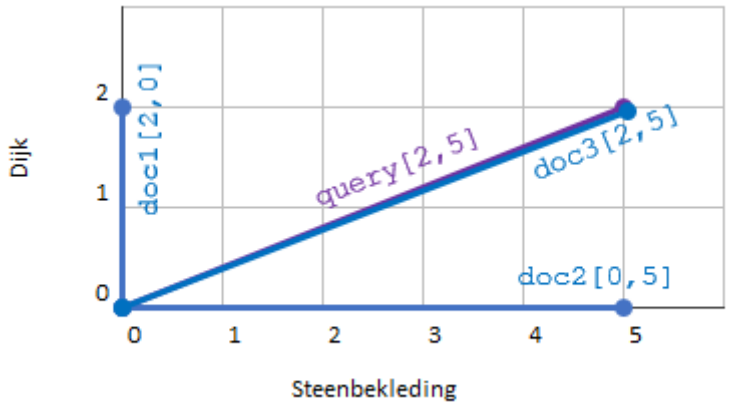
Stel verder dat er twee documenten zijn in de index, namelijk:

1. "Oude paarden jaagt men aan de *dijk*."
2. "De *dijk* bestaat uit gras- en *steenbekleding*."
3. "De *steenbekleding* wordt jaarlijks gecontroleerd."

Vervolgens wordt er een zoekopdracht uitgevoerd met de termen *dijk* en *steenbekleding*.

```
GET losseindex/_search?explain
{
   query: {
     match: {
        text : "dijk steenbekleding"
     }
   }
}
```

Per document wordt er een vector gemaakt, met het gewicht van elke query term in het document (zie Figuur 8).



Figuur 8 – Query en document vectoren

Deze vectoren kunnen vervolgens vergeleken worden door de hoek tussen de document en query vectoren. De hoek tussen document 1 en de query is het grootst en is daarmee het minst relevante document. Document 2 is iets relevanter. Document 3 komt overeen met de query vector en is het meest relevant.

BIJLAGEN

Bijlage 1: Ontwerpspecificatie

Ontwerpspecificatie

Verrijking zoekmachine

Expertise Management Wiki's

Ontwerpspecificatie van Joos Mesie
studentnummer 48580
in het kader van de opleiding HBO-ICT aan de
HZ University of Applied Sciences, Vlissingen

Ontwerpspecificatie

Verrijking zoekmachine

Expertise Management Wiki’s

Versiebeheer

Versie	Auteur	Omschrijving
1.0	J.	Ontwerpspecificatie

Inhoud

1 Inleiding

Op basis van SKOS* wordt een term expansie algoritme geïntroduceerd in de huidige zoekmachine. Hiervoor wordt een prototype ontwikkeld. Dit document beschrijft het algoritme en het te ontwikkelen prototype.

Gebruikte begrippen en definities:

*SKOS*term*	– Een SKOS concept of NSKOS term uit SKOS*;
Zoekterm	– Gebruiker input, bestaand uit één of meer termen;
Gewicht	– Representeert het belang van een *SKOS*term*, in de uiteindelijk op te stellen Elasticsearch (ES) query. Het *gewicht* is het product van vermenigvuldiger *startgewicht* en vermenigvuldigtal *gewichtfactor*;
Startgewicht	– Een vermenigvuldiger welke gebruikt wordt in de *gewicht* berekening. Dit is het *gewicht* van de voorgaande *SKOS*term* waarvoor recursie wordt uitgevoerd, of indien het de eerste recursie betreft, een instelbare waarde;
Gewichtfactor	– Een factor welke gebruikt wordt in de *gewicht* berekening als vermenigvuldigtal. Deze factor is voor SKOS relaties instelbaar en wordt voor NSKOS relaties berekend.

1.1 Het algoritme

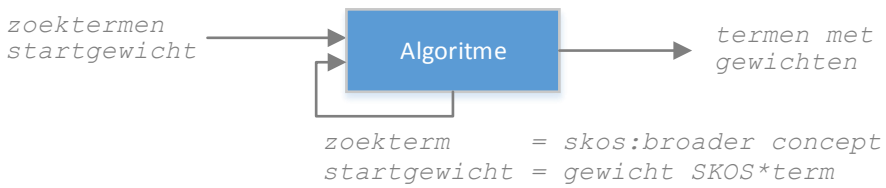
Het term expansie algoritme, is een recursief algoritme welke met de *zoekterm*, de triple-store doorzoekt naar letterlijke overeenkomsten met *SKOS*termen*. Zijn er overeenkomsten? Dan worden alle directe gerelateerde *SKOS*termen* opgehaald. Per gevonden *SKOS*term* wordt een *gewicht* berekend. De *SKOS*term* en het bijbehorende *gewicht* worden bewaard in een lijst voor later gebruik, tenzij de *SKOS*term* al aanwezig is in de lijst. In dat geval wordt het oude *gewicht* bijgewerkt indien het nieuwe *gewicht* groter is.

Het algoritme heeft de volgende input en output (zie Figuur 1) :

Input	De *zoekterm* en een *startgewicht*;
Output	Een lijst met *SKOS*termen* en bijbehorende g*ewichten*.

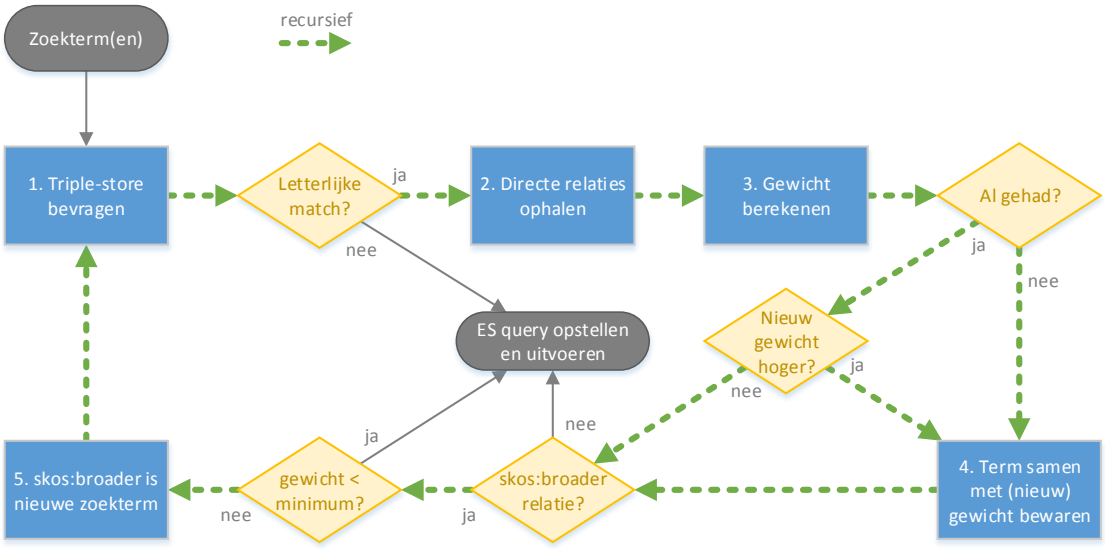
Recursie vindt plaats wanneer een gevonden SKOS concept een `skos:broader` relatie bevat. In dat geval wordt het proces herhaald met het `skos:broader` concept als *zoekterm* en het bijbehorende *gewicht* als *startgewicht*. Dit wordt gedaan tot er geen `skos:broader` meer is of het g*ewicht* kleiner is dan een instelbaar *minimum*.

> *Hoe meer afstand van de initiële zoektermen, hoe lager de gewichten en hoe minder relevant de termen.*



Figuur 1 – Input, recursie en output van het algoritme

In Figuur 2 hieronder, is de werking van het algoritme in detail beschreven.

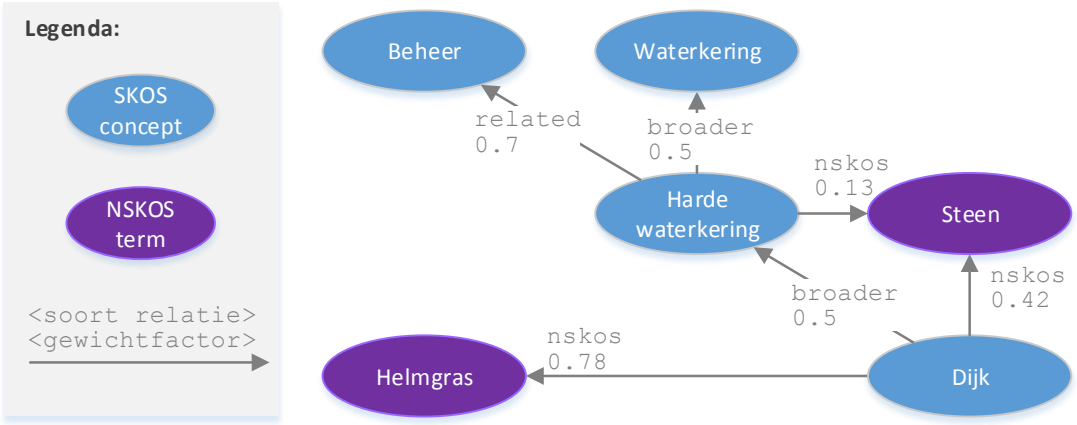


Figuur 2 – Werking term expansie algoritme

1.2 Voorbeeld

Neem de SKOS* dataset beschreven in Figuur 3 hieronder, met de volgende instellingen:

- *startgewicht* = 1
- *minimum* gewicht = 0.3



Figuur 3 – Voorbeeld SKOS dataset*

Stel dat een gebruiker zoekt naar `Dijk`, door de stappen in Figuur 2 te volgen, worden de volgende handelingen verricht- en resultaten verkregen:

1. De triple-store wordt bevraagd en geeft één resultaat terug, namelijk het SKOS concept `Dijk`;
2. `Dijk` heeft de volgende directe relaties:

`nskos:related`	`Helmgras`	*gewichtfactor*	= `0.78`
`nskos:related`	`Steen`	*gewichtfactor*	= `0.42`

```
skos:broader    Harde waterkering   gewichtfactor     = 0.5
```

3. De volgende *gewichten* (*startgewicht* = 1) worden berekend –
4. en per bijbehorende term, bewaard:

```
Helmgras                    gewicht = 1 * 0.78      = 0.78
Steen                       gewicht = 1 * 0.42      = 0.42
Harde waterkering           gewicht = 1 * 0.5       = 0.5
```

5. De term Harde waterkering is een broader concept met een groter gewicht dan het *minimum* (0.5 > 0.3). Er wordt opnieuw gezocht met deze term in plaats van de originele zoekterm Dijk, waarbij het *startgewicht* het *gewicht* van deze term is (0.5);

1.2.1 Tweede ronde

1. De triple-store wordt wederom bevraagd en geeft weer één resultaat terug, namelijk het SKOS concept Harde waterkering;

2. Harde waterkering heeft de volgende directe relaties:

```
nskos:related       Steen           gewichtfactor     = 0.13
skos:related        Beheer          gewichtfactor     = 0.7
skos:broader        Waterkering     gewichtfactor     = 0.5
```

3. De volgende *gewichten* (*startgewicht* = 0.5) worden berekend –
4. en per bijbehorende term, bewaard:

```
Steen                       gewicht = 0.5 * 0.13    = 0.065
```

Steen zit al in de lijst en het oude *gewicht* is groter dan het nieuwe (0.42 > 0.065). Het gewicht wordt daarom niet bijgewerkt.

```
Beheer                      gewicht = 0.5 * 0.7     = 0.35
Waterkering                 gewicht = 0.5 * 0.5     = 0.25
```

Waterkering is een broader concept, maar het *gewicht* hiervan is kleiner dan het *minimum*, (0.25 < 0.3). In dit geval zal het algoritme hier stoppen.

1.2.2 Resultaat

Het resultaat is een lijst met *SKOS*termen* en bijbehorende *gewichten*, namelijk:

```
Helmgras                    gewicht = 0.78
Steen                       gewicht = 0.42
Harde waterkering           gewicht = 0.5
Beheer                      gewicht = 0.35
Waterkering                 gewicht = 0.25
```

Hiermee kan vervolgens een ES query worden samengesteld. Dit wordt in H1.3 verder behandeld.

1.3 Elasticsearch (ES)

Stel voor dat, terugpakkend naar het voorbeeld besproken in H1.2, op basis van de *zoekterm* `Dijk`, de volgende lijst met *SKOS*termen* en bijbehorende *gewichten* beschikbaar is:

`Helmgras`	*gewicht* = `0.78`
`Steen`	*gewicht* = `0.42`
`Harde waterkering`	*gewicht* = `0.5`
`Beheer`	*gewicht* = `0.35`
`Waterkering`	*gewicht* = `0.25`

Nu moet er een ES query worden samengesteld, waarin elke losse *SKOS*term* kan worden voorzien van het bijbehorende *gewicht* als zogenaamde *boost* parameter.

> *De boost parameter verhoogt of verlaagd het belang van een query clausule ten opzichte van een andere clausule. (Elasticsearch, 2016)*

Momenteel wordt er gebruik gemaakt van een zogenaamde `multi_match` query (zie Tabel 1 hieronder), waarbij tijdens het zoeken wordt gekeken naar specifieke velden in de index. Deze specifieke velden (`fields`) worden in dit hoofdstuk weggelaten ten behoeve van het overzicht.

```
multi_match : {
    query: "dijk"
}
```

Tabel 1 – Huidige ES query

Om individuele *SKOS*termen* te kunnen *boosten*, is er een andere type query nodig, namelijk de zogenaamde *boolean* query. Er zijn verschillende soorten clausules mogelijk, waarvan de zogenaamde *should* clausule is gekozen om te gebruiken. Dit omdat er term expansie wordt uitgevoerd en het resultaat daarvan als uitbreiding op de originele zoekopdracht wordt gebruikt.

> *Als should clausules overeenkomen met een document, dan wordt de relevantiescore hiervan verhoogt. In combinatie met andere soorten clausules zijn deze echter niet verplicht. Zijn er alleen maar should clausules, dan moet één van de clausules overeenkomen. (Elasticsearch, 2016)*

Elke *should* clausule kan worden voorzien van een *boost*. De query in Tabel 1 hierboven, kan worden herschreven (met exact dezelfde resultaten) naar een *boolean* query (zie Tabel 2 hieronder).

```
bool: {
    should: [
        { multi_match: { query: "dijk",      boost: 1 }
    ]
}
```

Tabel 2 – Zoektermen omwikkeld in should clausules

Elke *SKOS*term* kan nu in een aparte *should* clausule worden opgenomen, met het bijbehorende *gewicht* als *boost*. De *zoektermen* krijgen een aparte boost (instelbaar, bijvoorbeeld 2).

De query voor het voorbeeld ziet er als volgt uit:

> *Er wordt nog steeds `multi_match` gebruikt omdat er wordt gezocht in meerdere velden.*

```
bool:
  should: [
    { multi_match: { query: "Dijk",              boost: 2    },
    { multi_match: { query: "Helmgras",          boost: 0.78 },
    { multi_match: { query: "Steen",             boost: 0.42 },
    { multi_match: { query: "Harde waterkering", boost: 0.5  },
    { multi_match: { query: "Beheer",            boost: 0.35 },
    { multi_match: { query: "Waterkering",       boost: 0.25 }
  ]
}
```

Tabel 3 – De nieuwe ES query

1.3.1 Minimum overeenkomsten

Omdat de query uit alleen *should* clausules bestaat moet een document met minimaal één clausule overeenkomen. Het is echter mogelijk om documenten te verplichten aan meerdere *should* clausules te voldoen middels de `minimum_should_match` parameter.

> *Het minimum aantal clausules waaraan het document overeen moet komen kan ingesteld worden met de `minimum_should_match` parameter. (Elasticsearch, 2016)*

Er zijn veel mogelijkheden wat betreft de waarde van deze parameter. Deze worden in Tabel 4 hieronder kort toegelicht. Er is echter geen “juiste” waarde. Het zal moeten blijken welke waarde en wenselijk is voor gebruik.

Soort	Voorbeeld	Toelichting
Positief integer	`3`	Moet overeenkomen met 3 clausules
Negatief integer	`-3`	Moet overeenkomen met het totaal aantal clausules - 3
Positief percentage	`75%`	Moet overeenkomen met 75% van de clausules
Negatief percentage	`-75%`	Hoeft niet overeen te komen met 75% van de clausules
Condities	`3 < 75%`	Als er <= 3 clausules zijn, dan moeten ze allemaal overeenkomen. Zijn er > 3, dan hoeft maar 75% overeen te komen.

Tabel 4 – Mogelijke waarden minimum_should_match parameter

2 Requirements

Het prototype past het in H1.1 beschreven term expansie algoritme toe, wanneer een zoekopdracht wordt uitgevoerd.

`[zoekterm]` = Gebruiker input;

`[skos*term]` = SKOS concept of NSKOS term uit SKOS*;

`[gewicht]` = representeert het belang van een `[skos*term]`, in de uiteindelijk op te stellen Elasticsearch query.

Het prototype moet -

1. op basis van de `[zoekterm]`, een lijst van overeenkomende `[skos*termen]` kunnen ophalen;
2. per `[skos*term]` direct gerelateerde `[skos*termen]` kunnen ophalen;
    a. *wanneer* `[skos*term]` *een SKOS concept betreft, alle direct gerelateerde SKOS concepten en NSKOS termen ophalen;*
    b. *wanneer* `[skos*term]` *een NSKOS term betreft, alle direct gerelateerde SKOS concepten ophalen.*
3. `[gewichten]` kunnen berekenen voor direct gerelateerde `[skos*termen]`, op basis van de soort relatie.
    a. *waarbij per SKOS relatie een instelbare factor beschikbaar is voor het berekenen van het gewicht;*
    b. *waarbij per NSKOS relatie, een gewicht wordt berekend aan de hand van de frequenties en relaties uit SKOS*.*
    c. *waarbij de* `[skos*termen]` *inclusief* `[gewichten]` *in een lijst worden bewaard, voor gebruik in Elasticsearch;*
4. recursief het `broader` pad kunnen volgen tot een bepaald minimum `[gewicht]` is bereikt of het einde van het pad is bereikt;
    a. *waarbij het start* `[gewicht]` *per recursie lager meeweegt, afhankelijk van de ingestelde gewicht factor voor* `broader` *relaties;*
5. de `[gewichten]` kunnen normaliseren naar de reeks [0 .. 1];
6. een Elasticsearch query kunnen samenstellen met de `[zoektermen]`, aangevuld met de `[skos*termen]` en de daarbij horende `[gewichten]`;
    a. *waarbij de* `[zoektermen]` *een gewicht van 1 toegekend krijgen.*
    b. *waarbij minimaal één van de* `[zoektermen]` *moet overeenkomen met documenten.*

> *Elasticsearch voert vervolgens de query uit en presenteert de resultaten zoals gewoonlijk.*

Verder zijn er een aantal niet functionele requirements en randvoorwaarden waar het algoritme aan moet voldoen, namelijk:

- Het algoritme treed pas in werking wanneer letterlijke matches van een `[skos*term]` wordt gevonden in de triple-store. Gedeeltelijke overeenkomsten worden, net als momenteel in de zoekmachine van toepassing is, niet bekeken;

- De gebruikte SKOS* is een data subset (zie Bijlage A: Termen in dataset), bestaand uit 365 SKOS concepten en 166 relaties, van 4 SKOS concepten naar 29 NSKOS termen verspreid over 7 documenten. Deze subset is dus niet per definitie representatief voor de complete dataset. Echter deze is momenteel niet beschikbaar;

- SKOS* relaties beschrijven nu nog een SKOS frequentie, NSKOS frequentie waar o.a. het `[gewicht]` mee wordt berekend. Dit gewicht zal in de toekomst onderdeel zijn van de SKOS* relaties in plaats van de losse frequenties, waardoor de run-time berekening weg kan;

- Een werkend prototype moet uiterlijk per 1 juli worden opgeleverd, ter ondersteuning van het onderzoeksrapport.

2.1 Prioritering

Het idee was om de requirements te prioriteren. De beschreven requirements zijn echter allemaal vereist om het prototype als een succes te kunnen zien. Daarom wordt prioritering middels de MoSCoW methode overgeslagen.

Er zijn echter nog wel wat wensen aan de dataset, namelijk:

- Er wordt nu een gewicht berekend op basis van de SKOS- en NSKOS frequenties en het aantal gevonden relaties. Dit is echter een onnodige query-time berekening die bij elke zoekopdracht wordt berekend per gevonden relatie. De frequenties die nu als triples zijn opgenomen in de triple-store, zullen in de toekomst worden vervangen door één triple met een voorhand berekend gewicht;

- De gebruikte data subset is relatief klein maar wel gebaseerd op statistiek en bestaande documenten. Wanneer de complete SKOS* beschikbaar is in de triple-store kunnen de gewichten en gewicht factoren in het prototype beter worden afgesteld;

- Niet alle soorten relaties zijn aanwezig in de dataset. Zo is er behoefte om relaties zoals de `instanceOf` en `dct:subject` ook toe te voegen. Overigens is het niet erg dat deze nu niet in het prototype zitten. Per soort relatie is een gewicht factor instelbaar. Deze twee relaties kunnen dus later worden toegevoegd en worden vervolgens meegenomen tijdens de term expansie.

3 Het prototype

In dit hoofdstuk wordt de uitwerking van het prototype toegelicht. Het prototype is ontwikkeld in PHP en bevat tevens uitgebreide documentatie in de broncode en in de `/doc` directory.

3.1 Instellingen

Er zijn een aantal constanten en variabelen toegevoegd voor gebruik in het prototype. Deze worden in Tabel 5 hieronder op alfabetische volgorde, toegelicht.

Naam	Waarde	Type
`$FIELDS_TO_SEARCH`	`[ 'veldnaam^veldboost ... ]`	array
Elasticsearch index, velden om te doorzoeken met optionele `^boost`		
`$SKOS_PROPERTY_WEIGHTS`	`[ 'skosBroader' => [ 'p' => 'skos:broader', 'w' => 0.7 ], ... ]`	array
Een gewichtfactor tabel voor elke soort `skos:property` (`p`) met gewichtfactor (`w`). De `key` moet overeenkomen met de alias uit de SPARQL query's.		
`DATASET_URL`	`http://localhost:3030/skosstar`	string
De URL van de dataset in de triple-store.		
`MINIMUM_WEIGHT`	`0.2`	float
Als dit gewicht wordt bereikt stopt het algoritme met recursie.		
`NSKOS_RELATION_WEIGHT`	`0.7`	float
Gewichtfactor voor gerelateerde NSKOS termen.		
`RECURSE_PROPERTY`	`skosBroader`	string
De alias van de SKOS eigenschap waarmee recursie wordt uitgevoerd.		

RECURSE_STARTWEIGHT	1.0	float
Startgewicht wanneer recursive plaatsvindt.		
SEARCH_START_WEIGHT	2.0	float
Gewicht van initiële zoektermen.		
TYPE_NSKOS	NSKOS	string
Type definitie voor NSKOS termen, voor onderscheid met SKOS concepten		
TYPE_SKOS	SKOS	string
Type definitie voor SKOS concepten, voor onderscheid met NSKOS termen		
WIKI_BASE_URL	http://wiki.local/index.php	string
De basis URL van alle SKOS concepten, dit wordt gebruikt om het uiteindelijk SKOS concept te achterhalen (door deze URL eraf te halen). *Dit in de veronderstelling dat de `skos:preflabel` eigenschap niet altijd aanwezig hoeft te zijn en ook niet altijd dezelfde naam als het SKOS concept hoeft te hebben. In ieder geval is het niet mogelijk te zoeken met complete URL's (i.v.m. speciale karakters).*		

Tabel 5 – Instellingen

3.2 URL Parameters

3.2.1 Test-case specifieke parameters

Met de `case` parameter kunnen individueel vier 'voorgeprogrammeerde' test-cases worden uitgevoerd, tevens kunnen de resultaten van test-cases naast elkaar worden weergegeven.

Param	Waarde	Toelichting
`case`	`noexp`	Geen term expansie toepassen, zoek met de termen: 'Dijk' en 'Steen'
	`expskos`	Term expansie met als zoekterm het SKOS concept: 'Dijk'
	`expnskos`	Term expansie met als zoekterm de NSKOS term: 'Steen'
	`expcombo`	Combinatie van `expskos` en `expnskos`, met de termen: 'Dijk' en 'Steen'
	`compare`	De resultaten van `expskos`, `expnskos` en `expcombo` naast elkaar weergeven, voor gemakkelijke vergelijking.

De resultaten van deze test-cases worden beschreven in H4.

3.2.2 Overige URL parameters

Er zijn een aantal URL parameters beschikbaar om de werking van het prototype te beïnvloeden. Deze worden in Tabel 6, hieronder op alfabetische volgorde, toegelicht.

Parameter	Waarde	Toelichting
`constant`	`isset`	Gebruik een zogenaamde `constant score query`, waar de score van een document gelijk is aan de boost, ongeacht hoeveel overeenkomsten er zijn gevonden. **Dit kan wenselijk zijn omdat op basis van SKOS* al een relevantieberekening wordt gedaan welke als boost wordt gerepresenteerd**.
	`!isset`	Laat elasticsearch zelf de relevantiescore berekenen, met de standaard TF-IDF calculatie
`dbg`	`isset`	Print debug informatie en sta de functie `prePrint` (zie H0) toe om gegeven functie parameters af te drukken
	`!isset`	Print geen debug informative of functie parameters
`matchcount`		Vertaald naar de `minimum should match` parameter (zie H1.3.1), welke definieerd hoeveel should clausules van het `TOTAL` aantal clausules, welke overeen moeten komen om een document mee te nemen
	`!isset`	Default is 0
	`3`	Must match `3` optional clauses
	`-3`	Moet overeenkomen met `TOTAL - 3` clausules
	`75%`	Moet overeenkomen met 75% van `TOTAL` clausules
	`-75%`	`75%` van `TOTAL` mogen missen
	`3<75%`	Zijn er `<= 3` clausules, dan moeten ze allemaal overeenkomen, bij `> 3` clausules hoeft alleen `75%` overeen te komen. See www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-minimum-should-match.html
`original`	`isset`	Overschrijf alle andere parameters zodat het pre- SKOS* gedrag kan worden hersteld, impliceert [exp=NONE | constant=false | matchcount=0]

Tabel 6 – URL parameters

3.3 Gebruik

De `QueryExpander` klasse, is een verzameling van functies voor het toepassen van term expansie op basis van een gegeven zoekterm, door gebruik van de SKOS en NSKOS relaties beschreven in SKOS*.

De klasse wordt als volgt geinitialiseerd:

```
$qe    = new QueryExpander($searchTerms, $SKOS_PROPERTY_WEIGHTS);
```

Parameters:

`$searchTerms`	`De zoektermen om te expanden`
`$propertyWeightsFactors`	`Dit is de $SKOS_PROPERTY_WEIGHTS, zie H3.1`

> *De `$propertyWeightsFactors`, net als andere instellingen, zijn nu geen onderdeel van de klasse zelf, zodat deze bijvoorbeeld verwerkt kunnen worden in een apart instellingen bestand.*

Vervolgens kan met de `applyTermExpansion` functie, term expansie worden toegepast. Met `getExpandedTerms` kunnen de gevonden termen en gewichten worden opgehaald.

3.4 Overzicht van functies

Een overzicht van alle functies in de `QueryExpander` klasse, op alfabetische volgorde.

addToExpandedTerms

Toevoegen van een SKOS concept of NSKOS term met gewicht. Als de term al is toegevoegd werk het gewicht bij als deze groter is dan het eerder toegevoegde gewicht.

```
private function addToExpandedTerms()
```

Parameters:

`$term`	`De term om toe te voegen`
`$w`	`Het gewicht bijhorend de term`
`$type`	`Het type term TYPE_SKOS | TYPE_NSKOS`

Deze functie wordt in het prototype niet direct aangeroepen. In plaats daarvan worden twee helper functies aangeroepen, namelijk:

```
private function addToExpandedTermsAsNskosTerm ($term, $w)
private function addToExpandedTermsAsSkosTerm ($term, $w)
```

applySkosTermExpansion

Toepassen van SKOS term expansie, door alle direct gerelateerde SKOS concepten en NSKOS termen op te halen en gewichten te berekenen.

> *Deze functie is recursief en zal wanneer er een `skos:broader` relatie wordt gevonden, zichzelf aanroepen met het `skos:broader` concept als `$skosConcept` en het daarvan berekende gewicht als `$startWeight`.*

```
private function applySkosTermExpansion()
```

Parameters:

`$skosConcept`	Het SKOS concept om term expansie op toe te passen
`$startWeight`	Het startgewicht voor verdere gewicht berekeningen, default = RECURSE_STARTWEIGHT

> *Deze functie roept ook applyNskosTermExpansion aan om alle direct gerelateerde NSKOS termen (van het huidige $skosConcept) mee te nemen.*

applyNskosTermExpansion

Afhankelijk van de `$applySkosExp` parameter, wordt gezocht naar letterlijke NSKOS termen en alle daaraan gerelateerde SKOS concepten, of de direct gerelateerde NSKOS termen aan een gegeven SKOS concept. Indien `$applySkosExp = true`, wordt per gevonden SKOS concept de functie `applySkosTermExpansion` aangeroepen.

```
private function applyNskosTermExpansion()
```

Parameters:

`$searchTerms`	De string om mee te zoeken
`$applySkosExp`	**true**, verdere term expansie toepassen op gevonden SKOS concepten **false**, geen verdere term expansie toepassen, default = **true**
`$startWeight`	Het startgewicht voor verdere gewicht berekeningen, default = RECURSE_STARTWEIGHT

applyTermExpansion

De functie die alles in werking zet. Op basis van de gegeven zoektermen wordt gezocht naar letterlijke SKOS concepten en NSKOS termen en wordt het term expansie algoritme in werking gezet.

```
public function applyTermExpansion()
```

calculateNskosWeight

Helper functie om het NSKOS gewicht te bereken. Een soortgelijke berekening die Elasticsearch toepast bij het bepalen van de inverse document frequentie wordt hier gebruikt. Namelijk

```
( 1 + log( n / ( f + 1 ) ) ) * startWeight
```

Waar:

```
n    Het totaal aantal direct gerelateerde NSKOS termen
f    De SKOS frequentie / de NSKOS frequentie
```

```
private function calculateNskosWeight()
```

Parameters:

$n	n
$f	f
$startWeight	Het startgewicht

countSparqlResults

Voert een gegeven SPARQL query uit en geeft het aantal gevonden resultaten terug.

```
private function countSparqlResults()
```

Parameters:

$sparqlQuery	De query om uit te voeren

getExpandedTerms

Haal het term expansie resultaat op. Dit is een lijst met alle gevonden termen en gewichten, gegroepeerd op SKOS en NSKOS.

```
private function getExpandedTerms()
```

Parameters (deze zijn in eerste instantie aanwezig voor test doeleinden):

$sort	Sorteren van de lijst op basis van gewicht 'weight', of term 'term'
$ascDesc	Oplopend 'asc' of aflopend 'desc' sorteren
$normalize	Wel of geen normalisatie toepassen op de gevonden NSKOS termen. default = **true**

> *De `$normalize` parameter zal in de toekomst overbodig worden omdat normalisatie voorhand berekend zal worden en in de triple-store worden bewaard. Het normaliseren van de NSKOS termen wordt nu in deze functie gedaan omdat de nodige waarden voor de berekening pas beschikbaar zijn wanneer term expansie is voltooid. De berekening is:*
>
> *`norm = ( $weight - $minWeight ) / ( $maxWeight - $minWeight )`*

getNskosTerms

Haal alle letterlijk overeenkomende NSKOS termen op, of alle NSKOS termen gerelateerd aan een specifiek SKOS concept.

```
private function getNskosTerms()
```

Parameters:

$searchTerms	De term(en) om mee te zoeken
$bySkosConcept	**true**, alleen direct gerelateerde NSKOS termen van een SKOS concept zoeken **false**, alle letterlijke overeenkomende NSKOS termen zoeken, default = **false**

getSkosConcepts

Haal alle letterlijk overeenkomende SKOS concepten op bijvoorbeeld alle `skos:related`.

```
private function getSkosConcepts()
```

Parameters:

$searchTerms	De term(en) om mee te zoeken

getSparqlResultsAsArray

Haal alle gevonden SPARQL resultaten op van een gegeven SPARQL result object of query, als array.

```
private function getSparqlResultsAsArray()
```

Parameters:

$sparqlResultOrQuery	Het SPARQL result object of de query om uit te voeren en resultaten van op te halen

inArray

Helper functie om te controleren of een gegeven string in een array zit. De functie is niet hoofdletter gevoelig en biedt de optie om een `boolean` te retourneren of het gevonden element zelf.

```
private function inArray()
```

Parameters:

$needle	De string om te zoeken
$haystack	De array om te doorzoeken
$returnBool	**true**, boolean teruggeven, **false**, gevonden element teruggeven, default = **true**

insertSearchTerms

Helper functie om gegeven zoektermen in een gegeven query te plaatsen.

```
private function insertSearchTerms()
```

Parameters:

$sparqlQuery	De query om zoektermen in te voegen
$searchTerms	Array of string met zoektermen om in de query toe te voegen. Locatie wordt aangeduid met #SEARCHTERMS#

prePrint

Helper functie om alle gegeven parameters in `<pre></pre>` HTML code te printen.

```
public function prePrint()
```

stripWikiUrl

Helper functie om de `WIKI_BASE_URL` te strippen van een SKOS concept, en tegelijk underscores te vervangen met spaties, zodat we de juiste tekstuele representatie hebben.

```
private function stripWikiUrl()
```

Parameters:

$url	De url om te strippen

3.5 Libraries

Om een connectie met de SPARQL store te realiseren in PHP, is gekozen om de PHP SPARQL Library van Christopher Gutteridge & University of Southampton te gebruiken.

```
PHP SPARQL Lib, Copyright 2012 Christopher Gutteridge &
University of Southampton, Licensed as LGPL

https://github.com/cgutteridge/PHP-SPARQL-Lib
```

Met deze library wordt een verbinding met de SPARQL store gemaakt, waar vervolgens query's naar gestuurd worden. Er zitten verder geen speciale of geavanceerde functies in en wordt daarom niet verder toegelicht. De functies die (direct) worden aangeroepen:

alive	Controle of de connection nog open is
query	Uitvoeren van een query op de triple-store
sparql_connect	Connectie met de triple-store

De library is te vinden in `/HZ-search/vendor/sparqllib`

4 Test-cases en resultaten

Dit hoofdstuk beschrijft vier test-cases, voor het testen van de functionaliteit van- en het vergelijken van de resultaten uit- de term expansie:

H4.2 Geen term expansie

H4.3 Term expansie met SKOS concept

H4.4 Term expansie met NSKOS term

H4.5 Term expansie met SKOS concept en NSKOS term

> *Deze test-cases zijn uitgevoerd op een kleine SKOS* dataset. De gevonden termen, berekende gewichten en uiteindelijk gepresenteerde zoekresultaten, zijn daarom niet representatief voor het volledige SKOS*.*

4.1 Gebruikte instellingen

Voor alle test-cases zijn de volgende instellingen (zie H3.1) toegepast:

```
MINIMUM_WEIGHT          =    0.2

NSKOS_RELATION_WEIGHT   =    0.7

RECURSE_STARTWEIGHT     =    1.0

SEARCH_START_WEIGHT     =    2.0

$SKOS_PROPERTY_WEIGHTS:

      skosRelated       =    0.7
      skosPartOf        =    0.5
      skosAltLabel      =    1.0
      skosBroader       =    0.5
```

De volgende URL parameters (zie H3.2) worden gebruikt:

`constant`	Staat standaard uit, oftewel er wordt gebruik gemaakt van de TF-IDF berekening van Elasticsearch.
`dbg`	Wordt expliciet aangezet, debug informatie wordt afgedrukt, in dit geval de lijst met termen en gewichten.
`matchcount`	Staat standaard uit, en resulteert dus in de waarde 0, echter omdat er alleen should-clausules zijn moet er minimaal één clausule overeenkomen.

4.2 Geen term expansie

Parameters:

```
case        = noexp
zoektermen  = 'Dijk Steen'
```

Deze test-case is een reflectie van hoe de zoekmachine nu werkt op de productie omgeving. Het enige wat is veranderd, is de soort query die wordt uitgevoerd, zie Tabel 1 en Tabel 2 in H1.3 .

> *De query's beschreven in Tabel 1 en Tabel 2, leveren exact dezelfde resultaten op.*

4.3 Term expansie met SKOS concept

Parameters:

```
case        = expskos
zoektermen  = 'Dijk'
```

Deze test-case begint met een SKOS concept en zoekt daarmee direct gerelateerde SKOS concepten en NSKOS termen. Vervolgens wordt het `skos:broader` pad gevolgd.

4.4 Term expansie met NSKOS term

Parameters:

```
case        = expnskos
zoektermen  = 'Steen'
```

Deze test-case begint met een NSKOS term en zoekt daarmee directe gerelateerde SKOS concepten. Vervolgens wordt met deze SKOS concepten SKOS term expansie toegepast (zie H4.3).

4.5 Term expansie met SKOS concept en NSKOS term

Parameters:

```
case        = expcombo
zoektermen  = 'Dijk Steen'
```

Deze test-case is een combinatie van de test-cases `expskos` (H4.3) en `expnskos` (H4.4).

4.6 Vergelijking

Middels de `case=compare` parameter (zie 3.2.1), worden hier de resultaten van de test-cases `expskos` (H4.3) en `expnskos` (H4.4) en `expocombo` (H4.5), naast elkaar weergegeven.

4.6.1 SKOS concepten

In Tabel 7 hieronder, zijn per test-case, de door term expansie gevonden SKOS concepten te zien, waarbij de `rood gemarkeerde cellen` van een test-case missen ten opzichte van de `groen gemarkeerde cellen` in andere test-cases.

	Dijk	Steen	Dijk + Steen
	case=expskos	case=expnskos	case=combo
1	(het) bezwijken	(het) bezwijken	(het) bezwijken
2	actuele sterkte	actuele sterkte	actuele sterkte
3	beheer	beheer	beheer
4	beheersregister	beheersregister	beheersregister
5	binnendijks	binnendijks	binnendijks
6	bres	bres	bres
7	buitendijks	buitendijks	buitendijks
8		buitenwater	buitenwater
9	deltahoogte	deltahoogte	deltahoogte
10		dijk	
11	dijkring	dijkring	dijkring
12	dijkvak	dijkvak	dijkvak
13	duinvak	duinvak	duinvak
14		golfoploop	golfoploop
15	harde waterkering	harde waterkering	harde waterkering
16	hydraulische belasting	hydraulische belasting	hydraulische belasting
17	intreepunt	intreepunt	intreepunt
18	keur	keur	keur
19	kopsloot	kopsloot	kopsloot
20	kustgebied	kustgebied	kustgebied
21	legger	legger	legger
22		macrostabiliteit	macrostabiliteit
23		meer	meer
24		meerdijk	meerdijk
25		microstabiliteit	microstabiliteit
26	normaal onderhoud	normaal onderhoud	normaal onderhoud
27		ondergrond	ondergrond
28		primaire waterkering	primaire waterkering
29		steenbekleding	steenbekleding
30		talud	talud
31	waterkering	waterkering	waterkering

Tabel 7 – De resulterende SKOS concepten

De NSKOS term expansie levert aanzienlijk meer resultaten op dan de SKOS concept term expansie. Dit komt omdat met de NSKOS term steen, alle gerelateerde SKOS concepten, worden opgehaald. In deze dataset (zie Bijlage A: Termen in dataset) zijn dat de volgende:

Dijk

Meerdijk

Talud

Steenbekleding

Vervolgens wordt met deze vier SKOS concepten SKOS expansie toegepast en het `skos:broader` pad gevolgd. De verschillen tussen de drie test-cases uit Tabel 7 behoeven extra toelichting. Hieronder wordt per gevonden SKOS concept de SKOS expansie, stap voor stap gevolgd.

10. Dijk

Dijk (10) is bij de andere twee test-cases een zoekterm, welke los worden meegenomen met een hogere boost. Bij `expnskos` niet, vandaar dat *Dijk* hier wel mee wordt genomen.

24. Meerdijk

23. *Meerdijk* (24) heeft als `skos:related` *Meer* (23)

28. Volgt `skos:broader` *Primaire waterkering* (23)

8. *Primaire waterkering* (28) heeft als `skos:related` *Buitenwater* (8)

30. Talud

14. *Talud* (30) heeft als `skos:related` *Golfoploop* (14)

22. *Talud* (30) heeft als `skos:related` *Macrostabiliteit* (22)

25. *Talud* (30) heeft als `skos:related` *Microstabiliteit* (25)

27. *Talud* (30) heeft als `skos:related` *Ondergrond* (27)

29. Steenbekleding

Heeft verder geen `skos:properties`.

> *Dit verschil ontstaat doordat bij* `expskos` *,* Dijk *(10) geen relatie heeft met de andere SKOS concepten (24, 29 en 30) en daardoor een aantal gerelateerde SKOS concepten mist ten opzichte van* `expnkos` *.*

4.6.2 NSKOS termen

In Tabel 8 hieronder, zijn per test-case, de door term expansie gevonden NSKOS termen te zien, waarbij de `rood gemarkeerde cellen` van een test-case missen ten opzichte van de `groen gemarkeerde cellen` in andere test-cases.

	Dijk	Steen	Dijk + Steen
	case=expskos	case=expnskos	case=combo
1	afsluitdijk	afsluitdijk	afsluitdijk
2	bekleding	bekleding	bekleding
3	breuksteen	breuksteen	breuksteen
4	dijkbeheerder	dijkbeheerder	dijkbeheerder
5	dijkbekleding	dijkbekleding	dijkbekleding
6	dijkvak	dijkvak	dijkvak
7	element	element	element
8	grasbekleding	grasbekleding	grasbekleding
9	klei	klei	klei
10	klein	klein	klein
11	kleipakket	kleipakket	kleipakket
12	klemming	klemming	klemming
13	klemmingsonderzoek	klemmingsonderzoek	klemmingsonderzoek
14	klemsteen	klemsteen	klemsteen
15	materiaal	materiaal	materiaal
16	meting	meting	meting
17	mijnsteen	mijnsteen	mijnsteen
18	onderlaag	onderlaag	onderlaag
19	plaatbekleding	plaatbekleding	plaatbekleding
20	stabiliteit	stabiliteit	stabiliteit
21	steen		
22	steenachtige	steenachtige	steenachtige
23	steenglooiing	steenglooiing	steenglooiing
24	steentoets	steentoets	steentoets
25	steenzetting	steenzetting	steenzetting
26	taludhelling	taludhelling	taludhelling
27	toplaagelement	toplaagelement	toplaagelement
28	toplaagstabiliteit	toplaagstabiliteit	toplaagstabiliteit
29	waddenzeedijk	waddenzeedijk	waddenzeedijk

Tabel 8 – De resulterende NSKOS termen

Hier is niet veel toelichting nodig. *Steen* (21) is alleen bij `expskos` geen zoekterm en is daarom alleen daar aanwezig. De SKOS concepten hebben allemaal wel één relatie met de 29 NSKOS termen.

4.6.3 Gewichten

Het is misschien opgevallen dat de berekende gewichten voor de gevonden SKOS* termen niet zijn meegenomen in H4 Test-cases en resultaten. Hier zijn een aantal redenen voor, namelijk:

- De Elasticsearch index is incompleet (bepaalde documenten kunnen door een bekend probleem niet worden geïndexeerd) en is daarmee niet representatief;
- De SKOS* dataset is relatief klein en daarmee niet representatief;
- De uitgevoerde berekeningen voor NSKOS termen is gebaseerd op de TF-IDF berekening die Elasticsearch gebruikt. Wellicht is er een betere berekening te verzinnen.
- De berekening voor de normalisatie van de gevonden NSKOS termen zorgt voor twee uiterste waarden, namelijk 0 en 1, waardoor de NSKOS term met het hoogste gewicht een 1 krijgt, en de NSKOS term met het laagste gewicht een 0 krijgt. Het is wenselijk te normaliseren naar een waarde tussen 0 en 1.

Echter, om dit toch te kunnen weergeven zijn hieronder de resultaten, inclusief berekende gewichten, weergeven.

4.6.3.1 SKOS concepten

meerdijk	1	keur	0.175
steenbekleding	1	duinvak	0.175
talud	1	beheersregister	0.175
kopsloot	0.7	binnendijks	0.175
macrostabiliteit	0.7	beheer	0.175
golfoploop	0.7	actuele sterkte	0.175
microstabiliteit	0.7	(het) bezwijken	0.175
ondergrond	0.7	intreepunt	0.175
primaire waterkering	0.5	bres	0.175
meer	0.5	hydraulische belasting	0.175
harde waterkering	0.5	dijkvak	0.175
buitenwater	0.35	dijkring	0.175
waterkering	0.25	buitendijks	0.175
legger	0.175	deltahoogte	0.175
normaal onderhoud	0.175	kustgebied	0.125

4.6.3.2 NSKOS termen

bekleding	0.700	taludhelling	0.670
materiaal	0.698	steenachtige	0.667
dijkbekleding	0.696	grasbekleding	0.665
element	0.694	steentoets	0.658
breuksteen	0.693	klemsteen	0.656
steenzetting	0.687	kleipakket	0.619
stabiliteit	0.687	klemmingsonderzoek	0.600
onderlaag	0.686	klei	0.587
klein	0.686	mijnsteen	0.479
dijkbeheerder	0.684	toplaagstabiliteit	0.435
klemming	0.681	afsluitdijk	0.290
toplaagelement	0.673	dijkvak	0.211
meting	0.673	steenglooiing	0.064
plaatbekleding	0.672	waddenzeedijk	0.000

BIJLAGEN

Bijlage A: Termen in dataset

> *De dataset bestaat uit 365 SKOS concepten en 166 relaties van 4 SKOS concepten naar 29 NSKOS termen verspreid over 7 documenten.*

365 SKOS concepten

Zie het `hzbwnature.n3` bestand.

4 SKOS concepten met NSKOS relaties

dijk

meerdijk

talud

steenbekleding

29 NSKOS termen

afsluitdijk

bekleding

breuksteen

dijkbeheerder

dijkbekleding

dijkvak

element

grasbekleding

klei

klein

kleipakket

klemming

klemmingsonderzoek

klemsteen

materiaal

meting

mijnsteen

onderlaag

plaatbekleding

stabiliteit

steen

steenachtige

steenglooiing

steentoets

steenzetting

taludhelling

toplaagelement

toplaagstabiliteit

waddenzeedijk

Bijlage 4: Adviesrapport

Adviesrapport

Doorontwikkeling zoekmachine

Adviesrapport van Joos Mesie
studentnummer 48580
in het kader van de opleiding HBO-ICT aan de
HZ University of Applied Sciences, Vlissingen

Inhoud

Inleiding

Dit rapport beschrijft een aantal verbeterpunten, aandachtspunten en mogelijkheden voor doorontwikkeling. Het is op geen enkele manier een volledige lijst en spoort over het algemeen aan tot meer onderzoek.

Onder voorbehoud dat sommige punten wellicht al lang bekeken zijn.

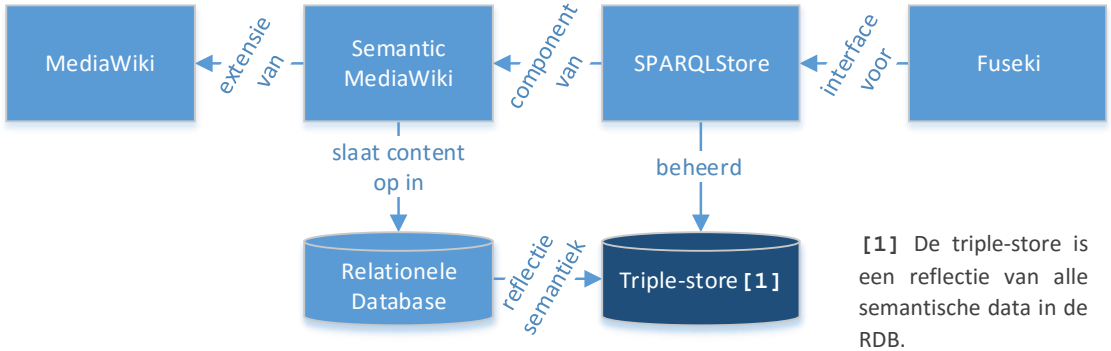
Hieronder wordt de gebruikte software ontwikkel methode en architectuur kort toegelicht.

Software ontwikkel methode

Ten behoeve van tijd is de keuze gemaakt om een vorm van Rapid Application Development (RAD) toe te passen, waar iteratieve ontwikkeling vooraan staat. In plaats van vooraf alles uit te denken en te ontwerpen, wordt het ontwerpen verweven met het daadwerkelijke ontwikkelen. De functionele- en technische uitwerking worden, samen met de requirements, opgenomen in een ontwerpspecificatie.

Architectuur

In Figuur 1 hieronder is te zien hoe de SPARQLStore, triple-store en andere onderdelen een geheel vormen.



Figuur 1 - Triple-store

Het gebruik van de SPARQLStore en daarmee de triple-store, is optioneel in Semantic MediaWiki. Fuseki is een daadwerkelijke keuze geweest vanuit de organisatie, welke op dat moment de beste keuze was.

Wellicht zijn er tegenwoordig betere varianten. In mijn persoonlijke ervaring laat de interface nog wat te wensen over, zeker bij gebruik van grotere datasets. Er waren ook wat problemen waarbij de service er zonder pardon mee stopte, echter dit bleek, zoals ook in H2 beschreven, te liggen aan een tekort aan werkgeheugen.

Afgezien van deze (kleine) problemen is dit een robuust en grootschalig toegepaste architectuur.

1 Elasticsearch

Een aantal verbeterpunten, aandachtspunten en ontwikkel mogelijkheden wat betreft Elasticsearch.

> *Mijn advies hier is simpelweg, neem de tijd om eens goed te kijken wat voor mogelijkheden Elasticsearch allemaal biedt. Het is geheel mogelijk dat met een paar kleine instellingen en wijzigingen het auto-aanvullen en het zoeken in het algemeen, flink verbeterd kunnen worden.*

1.1 Shards

Ten eerste een kleine toelichting op verdeling van data en prestaties, binnen Elasticsearch.

Een shard is simpel gezegd een enkele low-level "worker", welke automatisch door Elasticsearch wordt beheerd. Een index is een verzameling van deze "workers".

> *Wanneer een zoekopdracht wordt uitgevoerd, wordt dit doorgestuurd naar een kopie van alle shards in een index. Als we diezelfde zoekopdracht op meerdere indexen uitvoeren, gebeurt exact hetzelfde, alleen zijn er dan meer shards bij betrokken. Bijvoorbeeld, zoeken in 1 index met 50 shards is exact hetzelfde als zoeken in 50 indexen met elk 1 shard. Beide zoekopdrachten bevragen 50 shards. (Elasticsearch, 2016)*

Er vanuit gaand dat er geen verandering aan de instellingen plaatsvindt wat betreft de hoeveelheid *shards* per index, moet men rekening houden met de volgende implicaties:

Eén index met verschillende typen	**Meerdere indexen (bijvoorbeeld per type)**
Minder efficiënt zoeken omdat de hoeveelheid data per *shard* groter is, waardoor Elasticsearch meer tijd nodig heeft om door de documenten hierin te filteren.	Efficiënt en snel zoeken in de index, omdat de hoeveelheid data kleiner is per *shard*.
Er is minder overhead omdat de aanvraag minder *shards* hoeft aan te roepen. De *shards* zelf zijn echter wel groter.	Zoekopdrachten genereren meer overhead, de aanvraag moet naar meerdere *shards*, over meerdere indexen worden gestuurd, gecompileerd en terug gegeven aan de gebruiker.
Als de dataset relatief klein is, kan er meer opslagruimte nodig zijn voor elke additionele *shard* terwijl de prestatie winsten minimaal zijn. Is de dataset echter relatief groot, kan dit juist betere prestaties leveren.	Door datasets te scheiden kan Elasticsearch gemakkelijker specifieke typen verwerken en doorzoeken (specialisatie ten opzichte van generalisatie).

Tabel 1 - Implicaties één of meerdere indexen

Zie ook: www.elastic.co/guide/en/elasticsearch/guide/current/mapping.html

1.2 Auto-aanvullen

Momenteel wordt voor auto-aanvul functionaliteit gebruik gemaakt van een zogenaamde `Completion Suggester`.

> *De `Completion Suggester` is een `prefix suggester`. Het past geen spellingscorrectie toe zoals de `term-` of `phrase suggesters`, maar biedt basis auto-complete functionaliteit.* ***Ongeldige bron opgegeven.***

Dit is op zich prima, het werkt en dergelijke suggesters zijn volgens Elasticsearch ook zeer snel.

De `Completion Suggester` past een zogenaamde *`fuzzy`* query toe. Dit betekend simpelweg dat wanneer een typefout wordt gemaakt, er toch nog resultaten terug kunnen komen. Deze query bevat een `fuzziness` instelling.

> *Wanneer string velden worden bevraagd, wordt `fuzziness` geinterpreteerd als Levenshtein Edit Distance. Dit is het aantal keren dat een enkel karakter moet wijzigen om één string te veranderen in een andere string.* ***Ongeldige bron opgegeven.***

Stel een gebruiker typt “Dijle”, terwijl eigenlijk “Dijken” wordt bedoeld. De Levenshtein Edit Distance tussen deze woorden is 2, namelijk:

1. Substitutie van “L” naar “K”;
2. Toevoeging van “N” aan het eind.

Deze instelling staat in de huidige index echter op nul. Oftewel, er mogen geen wijzigingen worden toegepast en wat de gebruiker typt moet letterlijk overeenkomen. Dit is te zien als er expres een typefout wordt gemaakt op www.deltaexpertise.nl. De lijst met suggesties is bij een typefout abrupt verdwenen.

Dit is een kleine wijziging uiteraard, alleen is de vraag of een dergelijke *`suggester`* überhaupt gewenst is. Er zijn namelijk alternatieven. Neem bijvoorbeeld de *`match_phrase_prefix`* query. Deze query werkt op een soortgelijke manier, namelijk door een term als prefix te gebruiken, oftewel vindt alles wat begint met bijvoorbeeld “Dij”. Echter werkt deze query anders door alleen het laatste woord als prefix toe te passen.

Typt een gebruiker bijvoorbeeld: “Building with na” dan wordt dit gezien als:

1. Het woord “building”;
2. Gevolgd door het woord “with”;
3. Gevolgd door woorden die beginnen met “na”.

Door toepassing van een zogenaamde `slop` parameter, wordt er meer flexibiliteit geïntroduceerd wat betreft de volgorde van termen. Zo kan de gebruiker bijvoorbeeld zoeken naar “Nature building” en nog steeds uitkomen op “Building with nature”.

> *Dit zomaar één voorbeeld gevonden in de Elasticsearch reference. Maar er zijn meer mogelijkheden ook in combinatie met elkaar.*

1.3 More like this

Een ander voorbeeld, niet specifiek voor Auto-aanvullen, is de meer gespecialiseerde *More Like This (MLT)* query. Deze query vindt documenten die "overeenkomen" met een set documenten door een representatieve set termen samen te stellen op basis van de input documenten. Deze representatieve termen worden bepaald aan de hand van de Term Frequentie – Inverse Document Frequentie.

De MLT query extraheert simpelweg de tekst uit een input document, analyseert dit en selecteert de top K termen met de hoogste TF-IDF. Met deze termen wordt een zogenaamde disjunctive query (OR query) samengesteld, om soortgelijke documenten te zoeken. ***Ongeldige bron opgegeven.***

1.3.1 Analyse prestaties

Het analyse proces gebruikt veelal term vectoren. Deze worden echter (met standaard instellingen) voor elke uit te voeren query opnieuw gegenereerd.

Om analyse te versnellen kan het helpen om term vectoren op te slaan tijdens indexeren. ***Ongeldige bron opgegeven.***

De TF-IDF wordt reeds tijdens indexeren berekend en opgeslagen in de index.

1.4 Synoniemen

Elasticsearch kent ook vormen van term expansie en term samentrekking, alleen dan speciaal voor synoniemen, namelijk:

Simple expansion, uitbreiden van één synoniem naar alle aanwezige synoniemen. Als op één synoniem wordt gezocht, alle andere synoniemen ook meenemen in de zoekopdracht.

Simple contraction, een groep synoniemen toewijzen aan één synoniem. Als op één of meer synoniemen wordt gezocht, wordt dit onder water vertaald naar één specifieke term. Een soort stemming, alleen dan voor synoniemen.

Genre expansion, in plaats van alle synoniemen als gelijkwaardig te beschouwen, wordt hier de betekenis van een synoniem verbreed om meer generiek te zijn.

Deze laatste heeft wat toelichting nodig. Bij gebrek aan een beter voorbeeld wordt hier het voorbeeld uit de Elasticsearch reference beschreven:

Stel de volgende regels zijn van toepassing:

```
cat          => cat,pet,
kitten       => kitten,cat,pet,
dog          => dog,pet
puppy        => puppy,dog,pet
```

Bij het toepassen van *genre expansion* tijdens indexeren zal-

- een query voor `kitten`, alleen documenten over `kittens` vinden;
- een query voor `cat`, alle documenten over `kittens` en `cats` vinden;

- een query voor `pet`, alle documenten over `kittens`, `cats`, `puppies`, `dogs` of `pets` vinden.

Bij het toepassen van *genre expansion* tijdens een zoekopdracht, zal een query voor `kitten` uitgebreid worden zodat alle documenten waar `kittens`, `cats` of `pets` in wordt benoemd, geretourneerd worden.

1.5 Named entity recognition en resolution

Het herkennen van entiteiten in tekst, en het uitzoeken naar wat een entiteit refereert.

Bijvoorbeeld de tekst “PBZ bestaat 50 jaar”. Met Named Entity Recognition kan PBZ herkend worden als entiteit. Met Named Entity Resolution (ook wel Named Entity Normalisation) kan PBZ gekoppeld worden aan Projectbureau Zeeweringen in plaats van bijvoorbeeld de op 1 na grootste bank in Kroatië, Privredna banka Zagreb.

Dit is een lastig onderwerp en wellicht ook niet realistisch om te bekijken. Echter kan dit wel in zekere mate “genept” worden. Hiermee doel ik op het speciaal afhandelen van typische entiteiten op een specifieke wiki. Neem bijvoorbeeld weer PBZ. Als er nu gezocht wordt op PBZ komt er van alles naar voren, behalve het Projectbureau Zeeweringen.

Een oplossing kan zijn deze entiteiten voorzien van synoniemen. Wellicht is het ook mogelijk om een lijst met typische entiteiten apart te indexeren en daar iets mee te doen bij zoekopdrachten.

Een oplossing voor meerwoordige entiteiten is bijvoorbeeld de Shingling token filter, www.elastic.co/guide/en/elasticsearch/reference/current/analysis-shingle-tokenfilter.html

1.5.1 Elasticsearch plugin

Toevallig kwam ik deze plugin tegen voor Named Entity Resolution. Wellicht dat hier ook alternatieven voor zijn te vinden.

www.github.com/YannBrrd/elasticsearch-entity-resolution

2 Triple-store

> *Niet zozeer een advies, maar meer een opmerking wat betreft de gegevens die nu als triples worden opgeslagen, een aandachtspunt wat betreft stabiliteit en een notitie voor als men wil overstappen naar gedeeltelijke overeenkomsten.*

SKOS* is verwerkt in de triple-store, echter zijn er nu veel extra triples nodig voor het beschrijven van de SKOS frequentie, NSKOS frequentie en herkomst.

Met de triples die de frequenties bevatten wordt nu een gewicht berekend voor een specifieke relatie. Het is echter beter om deze berekening voorhand uit te voeren en het berekende en genormaliseerde gewicht kant en klaar op te nemen in de triple-store.

De herkomst triple is uiteindelijk niet meer gebruikt in het prototype. Het doel van deze triple was in eerst instantie, afhankelijk van de sterkte van de relatie, de herkomst direct kunnen toevoegen in de zoekresultaten. Ten tweede is een relatie specifiek voor één document. Dit is echter in de loop van het onderzoek overbodig geworden doordat er term expansie wordt toegepast en niet perse meer naar de herkomst wordt gekeken. Wellicht is deze overbodig of kan er een ander doel voor bedacht worden.

Eén aandachtspunt is de stabiliteit van Fuseki. In eerste instantie kwam het regelmatig voor dat de Fuseki service zonder pardon stopte. Dit bleek later gerelateerd aan een tekort aan werkgeheugen. Dit lijkt na toekenning van meer geheugen aan de virtuele omgeving en de Fuseki service zelf, te zijn opgelost.

> *Het is echter niet extensief bekeken en is iets om in de gaten te houden wanneer dit in productie wordt genomen!*

2.1 Gedeeltelijke overeenkomsten

Momenteel wordt net als in Elasticsearch, ook in de triple-store gezocht naar letterlijk overeenkomende triples. Met een kleine aanpassing in de query is dit echter te veranderen naar gedeeltelijke overeenkomsten.

Er wordt nu gebruik gemaakt van een filter welke de functie `SAMETERM` toepast om een letterlijke overeenkomst te vinden. Door deze functie te veranderen in de `CONTAINS` functie, kunnen gedeeltelijke overeenkomsten worden gerealiseerd.

In plaats van `SAMETERM`:

```
FILTER (
  SAMETERM( LCASE(?skosPrefLabel), LCASE(?search) )
)
```

De `CONTAINS` functie gebruiken:

```
FILTER (
  CONTAINS( LCASE(?skosPrefLabel), LCASE(?search) )
)
```

Zie: https://www.w3.org/TR/sparql11-query/#func-contains.

3 SKOS

> *Mijn advies hier is eigenlijk hetzelfde als bij Elasticsearch, namelijk neem de tijd om eens goed te kijken wat voor mogelijkheden SKOS allemaal biedt.*

Een aantal SKOS properties (predicates) zijn momenteel aanwezig in de EMM wiki's. Er zijn zoals waarschijnlijk wel bekend, meer properties gedefinieerd in de W3 SKOS reference.

Zo zijn er bijvoorbeeld ook de zogenaamde *mapping properties*. Het betreft de volgende properties:

`broadMatch` en `narrowMatch` voor hiërarchische toewijzingen;

`relatedMatch` een associatieve toewijzing;

`closeMatch` een link tussen concepten die voldoende vergelijkbaar zijn dat deze door elkaar gebruikt kunnen worden;

`exactMatch` een link tussen concepten die praktisch hetzelfde zijn en dus door elkaar gebruikt kunnen worden.

3.1.1 Transitiviteit

Hiërarchische relaties kunnen ook transitief (overdraagbaar) zijn. Neem bijvoorbeeld:

```
<Roofdier> skos:broader <Zoogdier>;
<Beer> skos:broader <Roofdier>.
```

Zoogdier is generieker dan *Roofdier* en *Roofdier* is weer generieker dan *Beer*. Echter, door de relaties zo te definiëren wordt niet afgeleid dat *Zoogdier* ook generieker is dan *Beer*. Wil men dit wel dan kan men de transitieve varianten gebruiken:

```
<Roofdier> skos:broaderTransitive <Zoogdier>;
<Beer> skos:broaderTransitive <Roofdier>.
```

Bijlage 5: Transcripties

Onderzoek

Zoekmachine

Transcriptie ongestructureerde interviews

transcriptie van Joos Mesie
studentnummer 48580
in het kader van de opleiding HBO-ICT aan de
HZ University of Applied Sciences, Vlissingen

1 Inleiding

Dit is de transcriptie van de ongestructureerde interviews bij de eerste en tweede deelvragen:

1. *Hoe werkt de huidige zoekmachine van input tot en met output?*
2. *Welke vorm van SKOS* is het meest geschikt als input voor de zoekmachine?*

De volgende personen zijn bevraagd, de afkortingen worden in de vraag en antwoord tabel gebruikt.

AB – Anton Bil (collega)
HB – Hans de Bruin (bedrijfsbegeleider)
JW – Jethro Waanders (collega)
TL – Thien Tin Lam Ngoc (mede-afstudeerder)
WE – Wouter Everse (bedrijfsbegeleider)

1.1 Vraag en antwoord

Huidige situatie	
In welke servers/omgevingen draait deltaexpertise.nl?	
Vagrant (lokaal), acceptatie en productie	**JW**
Waarom zijn er "normale" pagina's en VN pagina's? En waarom staat in de pagina die juist de gebruiker krijgt te zien _ *VN* achter de url?	
VN pagina's zijn de View-Navigation pagina's. Je kan het zien als de view in MVC. Dat achter de URL wordt nog naar gekeken.	**JW, WM**
Hoe zit het datamodel van de EMM wiki's (globaal) in elkaar?	
Model besproken (dit is echter niet toegevoegd i.v.m. mogelijke digitale verspreiding).	**JW, WM**

Index, NSKOS en SKOS vragen	
De herkomst en frequentie, ga je op basis daarvan ook nog een soort relevantie score maken? Of gebeurt dat al automatisch of?	
Gebeurt al automatisch.	**TL**

Welke “soorten” informatie zitten allemaal in de index?

Intentional Elements, Contexten, Resource Descriptions en SKOS concepten. **WM**

Kunnen we door het indexeringsproces lopen? Wat gebeurt er allemaal?

Ja. <<dit hebben we toen doorlopen en dat is uitgewerkt in het document bij deelvraag 1>> **WM**

Waarom zijn sommige skos eigenschappen zoals *skos:broader* toegevoegd als de eigenschap *skosem:broader*? En waar staat dat *em* dan voor?

Dit zijn eigenlijk verbijzonderingen van de corresponderende SKOS eigenschap. De toevoeging EM staat voor Expertise Management. **WM**

Zoekmachine en Elasticsearch

Is de *spread activation technique* zoals besproken in een rapport van Thijs Vogels toegepast? Zo nee, wordt dat nog toegepast?

Dat was origineel gepland maar is niet meer van gekomen. Wellicht wordt er in de toekomst nog naar gekeken. **WM**

Het lijkt erop dat de overkoepelende context en de daarbij horende filters hiërarchisch bekeken worden wat lijdt tot fouten op de zoekresultaten pagina (de totaal aantal gevonden en de onderverdeling in de overkoepelende context en de daarbij horende filters komt niet overeen). Wat is daar de redenering achter?

Dat is zo opgeleverd en verwerkt door Thijs Vogels, wellicht is daar een goede reden voor maar die heb ik niet paraat. Moet nog naar gekeken worden. **WM**

Waarom wordt voor de auto aanvul functionaliteit, gebruik gemaakt van *fuzzy* search met vervolgens de *fuzziness* factor 0?

Onbekend. **WM**

De SKOS analyzer wordt zo lijkt het niet gebruikt. De analyzer is ingesteld op een subject veld maar dit lijkt niet gevuld te zijn. Hoe zit dat?

Wordt wel gebruikt, alleen dat subject veld is alleen bij resource descriptions gevuld. Waarom het niet op meerdere velden is toegepast is onbekend. **WM**

*Triple-stores en SKOS**	
Is er reeds gekeken, in het kader van SKOS* opnemen in de triple-store, naar quad-stores?	
Niet dat ik weet.	**WM**
Is het concept quad-stores wel bekend?	
Ja, enigszins.	**WM**

Bespreking Hans de Bruin wat betreft het algoritme en wat er allemaal verwacht wordt.	
Op basis van de zoekopdracht meteen term expansion doen? Dus triple vragen om alle SKOS relaties en NSKOS relaties en dan die termen toevoegen al dan niet met een boost/gewicht?	
Ja	**HB**
^ en als er niks matched?	
Dan geen query expansion	**HB**
Doen we nog iets met de herkomst verder?	
Nee eigenlijk niet, wellicht later	
Wat doen we met woorden zoals bijvoorbeeld kruinbekleding, dijkbekleding, steenbekleding? Oftewel, combinaties van SKOS met NSKOS als één woord of soms zelfs SKOS met SKOS als één woord?	
Niet zeker of dat een vulfout is of anders, wordt later nog naar gekeken.	
Meertaligheid meenemen?	
Nee voorlopig niet	
Partial matches meenemen?	
Nee voorlopig niet	