	**Apache Solr 1.4**	
	Zoekoplossing	
	Systeemvereisten: Java 1.5 of hoger Java-container	
	Ontwikkeld door:	Apache http://lucene.apache.org/solr
Licentie: Apache License	**Contactpersoon:**	Bob.Lannoy@smals.be

1. Beschrijving product

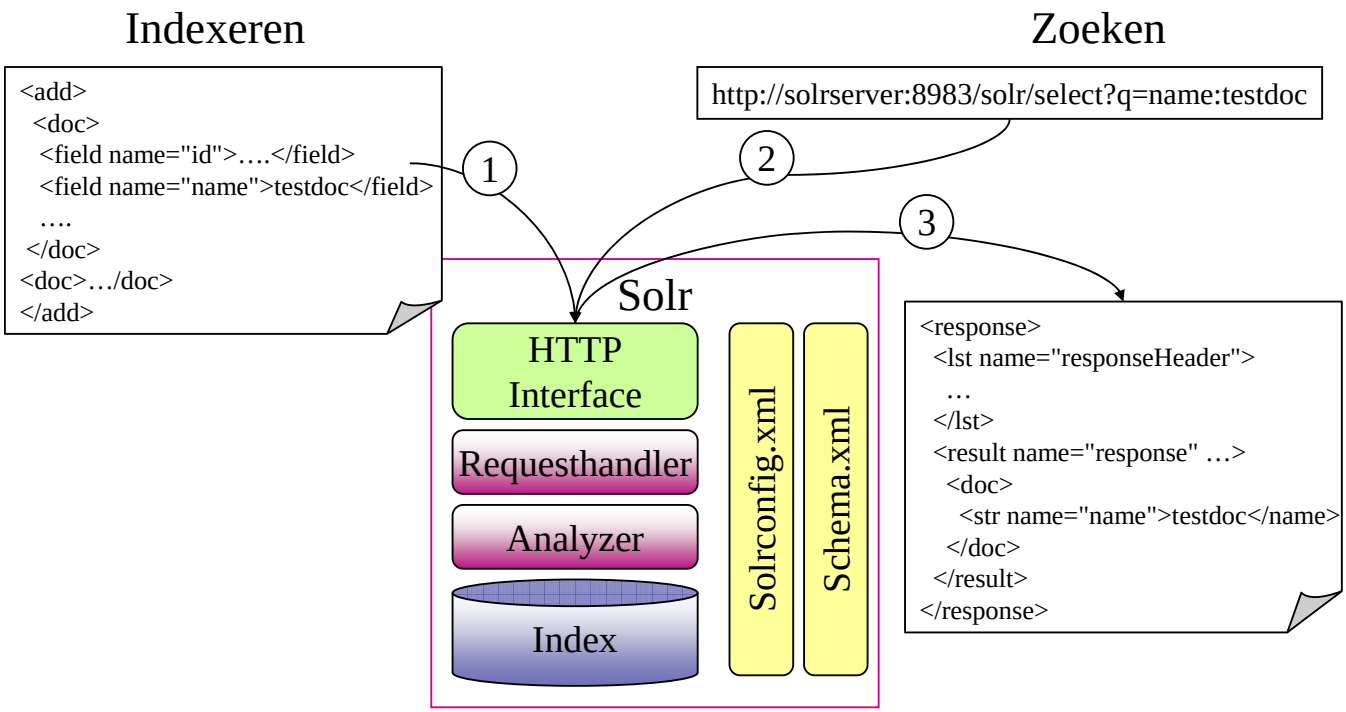
Apache Solr is een zoekplatform dat deel uitmaakt van het Apache Lucene project[1]. Lucene is een bekende softwarebibliotheek met uitgebreide functionaliteit voor de bouw van een eigen zoekmotor. Lucene vervult twee nodige functionaliteiten van een zoekoplossing, namelijk het opbouwen van de index en het ondervragen ervan. Rond deze kernfunctionaliteiten werd een server gebouwd, Solr genaamd, die uitgebreider is qua functionaliteit dan enkel Lucene.

Lucene als softwarebibliotheek is van nature zeer technisch van aard. Met de ontwikkeling van Solr heeft men deze techniciteit lichter gemaakt maar het platform blijft nog steeds gericht op ontwikkelaars. Deze nota zal dus georiënteerd zijn op een technisch publiek. Solr is Java-gebaseerd en maakt gebruik van een applicatieserver (Tomcat, Jboss, Jetty, WebLogic…).

Er zijn intussen al heel wat websites en producten (bijv. content management systemen) die gebruik maken van deze zoekoplossing, zowel op kleine als grote schaal. Een voorbeeld van een complexe implementatie is deze van Hathitrust[2] waar doorheen miljoenen documenten kan worden gezocht. Dat Solr heel wat interesse opwekt, wordt ook aangetoond door het gebruik ervan door PHP-gebaseerde oplossingen zoals

[1] http://lucene.apache.org/
[2] http://www.hathitrust.org/blogs

Drupal en EzPublish. Door zijn technologie-agnostische interface kan men makkelijk integreren met deze externe zoekoplossing.



Figuur 1: Algemene werking Solr

In Figuur 1 wordt de algemene werking van Solr samengevat. Solr draait binnen een Java-container en steunt hoofdzakelijk op 2 configuratiebestanden (*solrconfig.xml* en *schema.xml*) voor zijn werking. Bovenop de eigenlijke index is er functionaliteit die de verwerking van documenten en zoekopdrachten in goede banen leidt.

Via HTTP wordt de server aangesproken om zowel de documenten in de index te beheren als zoekopdrachten te lanceren. Door een XML-bestand te POST-en geef je instructies aan Solr om dit bestand te indexeren (stap 1 uit de figuur). Dit XML-bestand bevat de opdracht (add, update, delete) en de velden die geïndexeerd dienen te worden. Elke interactie met Solr gebeurt met een *requesthandler* die bepaalt wat er moet gebeuren met de aangevraagde operatie.

Op basis van een voorgedefinieerd schema kan je bepalen welke types van velden er bestaan, op welke manier ze zullen behandeld worden en welke velden welk type bezitten. Deze velden hoeven echter niet statisch gedefinieerd te zijn. Dynamische velden laten toe om op basis van een onderdeel van de veldnaam een veldtype toe te wijzen.

In de bouw van een index wordt gebruik gemaakt van een "*analyzer*" die de tekst of informatie verwerkt in verschillende stappen. We denken hierbij aan het verwijderen van stopwoorden (bijv. "een", "het", …), het terugbrengen van woorden tot hun stam (bijv. "woorden" -> "woord"), rekening houden met geaccentueerde karakters, … Uiteindelijk bevat de zoekindex dan woorden of onderdelen ervan die dan als basis dienen voor de zoekopdracht.

De te indexeren documenten dienen een vaste structuur te volgen. Eigen documenten dienen dus omgezet te worden naar dit formaat om geïndexeerd te kunnen worden. Voor dit formaat zijn er een aantal opties. Standaard is dit een XML-document, maar er zijn ook andere mogelijkheden zoals een CSV-bestand, rechtstreeks inhoud uit een databank halen m.b.v. de *DataImportHandler*, documentconversie (bijv. Microsoft Word → XML) via *Solr Cell* (*Apache Tika*), … Daarnaast kan je met een Java-API werken, *SolrJ*

genaamd. Op die manier kan je bijvoorbeeld documenten in een index stoppen via een eigen Java-programma.

Eens je weet welke velden je wenst op te slaan in de index en op welke manier je die wenst te behandelen is het niet zo ingewikkeld om op te zetten. Een aspect dat een belangrijke rol speelt is de meertaligheid. In de behandeling van een tekstfragment is de taalanalyse immers afhankelijk van de gebruikte taal. Gelukkig biedt de administratie-interface de middelen om tot in detail de indexering en zoekopdrachten te debuggen. Zo kan je nagaan op welke manier de ingevoerde tekst wordt behandeld (zie ook paragraaf 2.2.4).

Bij het uitvoeren van een zoekopdracht (stap 2 in Figuur 1) kan je dan aangeven in welke velden je wenst te zoeken naar welke term, eventueel gebruik makend van wildcards, range-operatoren (bijv. tussen bepaalde waarden) en booleaanse operatoren. Deze zoekopdracht is dus een gewone HTTP-request naar de server toe. Daarnaast kan je via HTTP-parameters heel wat extra features gebruiken zoals:

- Formaat van het resultaat: XML (standaard zoals weergegeven in stap 3 van Figuur 1) of JSON[3].
  Een interessante functionaliteit is het gebruik van een XSLT-stylesheet. Je kan bij een zoekopdracht aan de server meegeven dat je de resulterende XML wil laten weergeven volgens een specifieke XSLT. Op deze wijze bouw je het resultaatscherm in Solr en niet in een aparte toepassing.
  Daarnaast zijn er ook mogelijkheden om een volledige zoekinterface te bouwen op de server met behulp van Velocity of een andere technologie (Java, Ruby, …).
- Sortering van resultaten volgens verschillende criteria.
- *Facetten*: weergeven van termen met het aantal maal dat ze voorkomen. Dit kan dan gebruikt worden om je zoekopdracht te verfijnen.
- *Highlighting*: aanduiden van gevonden termen in een tekst.
- *Spellcheck*: wordt gebruikt om suggesties te doen voor een juiste zoekterm indien men de zoekterm vermoedelijk verkeerd gespeld heeft.
- *Clustering*: clusteren van zoekresultaten rond termen.

Standaard beschikt Solr over een administratie-interface. Deze heeft zeer interessante functionaliteit zoals de weergave van de gebruikte velden, debugging van de tekstanalyse van een veld en index-statistieken.

Solr is zeer schaalbaar en performant. Deze goede performantie wordt bereikt met behulp van caching maar een index kan ook gerepliceerd worden over verschillende servers om grote aantallen zoekopdrachten te verwerken. In implementaties waar de zoekindex te groot wordt, kan deze opgesplitst worden in *shards*. Solr zal dan de resultaten uit de verschillende *shards* samenvoegen.

2. Uitgevoerde testen

Om een aantal elementen van Solr te onderzoeken, werd een kleine reeks testen uitgevoerd. Solr werd gebruikt in zijn voorbeeldimplementatie, namelijk in een Jetty-container. Deze kan heel eenvoudig opgestart worden waarna de server beschikbaar is.

2.1. Standaardfunctionaliteit

Vooreerst werd de korte tutorial vanop de Solr-website[4] doorlopen. Deze geeft al een goed inzicht in de manier waarop je documenten in de index moet stoppen en wat de verschillende types van zoekopdrachten zijn.

Met behulp van een voorgemaakt Java-programma (*post.jar*) kan je makkelijk XML-documenten die voldoen aan de Solr-syntaxis naar het systeem sturen, bijvoorbeeld:

[3] *JavaScript Object Notation*: een populair bestandsformaat voor AJAX-applicaties, zie ook [3]
[4] http://lucene.apache.org/solr/tutorial.html

```
java –jar post.jar document.xml
```

Zoeken gebeurt door bijvoorbeeld in een internetbrowser een URL in te geven. Als we wensen te zoeken naar de documenten die het woord *video* bevatten en waarvan we in het resultaat alle velden wensen met een score (meest waarschijnlijke documenten) is dit:

```
http://solrserver:8983/solr/select/?q=video&fl=*,score
```

Als resultaat krijg je een XML-document terug in de vorm zoals weergegeven in Figuur 2. Je krijgt wat informatie over de uitgevoerde zoekopdracht en dan een lijst met documenten. Voor elk document worden dan de gevraagde velden weergegeven, in dit geval allemaal (*) en de score.

```
<response>
  <lst name="responseHeader">
…
  </lst>
  <result name="response" numFound="3" start="0" maxScore="0.4153909">
    <doc>
      <float name="score">0.4153909</float>
      <arr name="cat">
        <str>electronics</str>
        <str>music</str>
      </arr>
      <str name="id">MA147LL/A</str>
      <str name="name">Apple 60 GB iPod with Video Playback Black</str>
      <bool name="inStock">true</bool>
…
    </doc>
…
  </result>
</response>
```

Figuur 2: Voorbeeld van antwoord op zoekopdracht (details van 1 document)

2.2. Integratie met een bestaand systeem

Met behulp van *SolrJ* (Java-API van Solr) hebben we zelf een brug gebouwd tussen het web content management systeem Daisy[5] en Solr.

Daisy is een Java-gebaseerd web content management systeem dat gebruikt wordt door de sectie Onderzoek van Smals voor het beheer van de “Inventaris open source software en (open) standaarden”[6]. Daisy beschikt uiteraard zelf over een zoekmachine, die gebaseerd is op Lucene, maar hier wordt voor het prototype uiteraard geen gebruik van gemaakt.

Als prototype werd een Java-programma geschreven dat alle documenten uit Daisy haalt en deze in Solr stopt. In een typische implementatie zou deze integratie dynamisch moeten zijn zodat bij nieuwe, gewijzigde of verwijderde documenten de nodige operaties op de Solr-index gebeuren.

[5] http://www.daisycms.org/daisy/index.html
[6] http://inventarisoss.smals.be

Om Solr en Lucene makkelijker aan te sluiten op document-repositories is er een incubatieproject bij Apache met de naam *Lucene Connectors Framework* (*LCF*)[7]. LCF moet toelaten om via plugins de connectie met document-repositories te vergemakkelijken. Momenteel vermeldt de site onder andere connectoren met EMC Documentum en Livelink, maar het is niet duidelijk wat de kwaliteit hiervan is. Dit connectorframework kan ook gebruikt worden om eigen connectoren te bouwen. In het kader van dit prototype werd deze aanpak niet gevolgd.

De te indexeren documenten uit dit prototype kunnen onderverdeeld worden in verschillende types van documenten. Naargelang het type, bezit het document een aantal velden. Deze velden dienen dus per document in Solr overgebracht te worden.

2.2.1 Schema

Zoals eerder vermeld maakt Solr gebruik van een schema om aan te geven welke verschillende velden er bestaan. Dit schema is een XML-bestand dat je zelf dient te wijzigen. Er is hier geen specifieke grafische interface voor.

Om de velden uit Daisy te indexeren, hebben we twee mogelijkheden:

- Ofwel elk veld definiëren in het schema.
- Ofwel gebruik maken van de dynamische velden. Dit zijn velden die een wildcard in de veldnaam bevatten. Een dynamisch veld met de naam *_s* zal worden toegepast op een onbestaand veld van die vorm, bijvoorbeeld *type_s.*

Voor de test werd er gebruik gemaakt van de tweede aanpak, namelijk het gebruik van dynamische velden. Zo worden dus niet-gedefinieerde velden automatisch aangemaakt in de index. Voor prototyping is dit een heel vlotte manier van werken.

Aangezien de inventaris in twee talen (NL en FR) werd opgesteld, moet er een verschil gemaakt worden tussen Nederlandstalige en Franstalige tekst. Hiervoor werden in het schema een aantal taalafhankelijke veldtypes toegevoegd die dan werden gekoppeld aan een dynamisch veld (*attr_nl_** en *attr_fr_**).

```
<fieldType name="textnl" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
       <charFilter class="solr.HTMLStripCharFilterFactory"/>
       <charFilter class="solr.MappingCharFilterFactory" mapping="mapping-ISOLatin1Accent.txt"/>
       <tokenizer class="solr.WhitespaceTokenizerFactory"/>
       <filter class="solr.StopFilterFactory" ignoreCase="true" words="NL_stopwords.txt" … />
       <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1" … />
       <filter class="solr.LowerCaseFilterFactory"/>
       <filter class="solr.SnowballPorterFilterFactory" language="Dutch"/>
   </analyzer>
  <analyzer type="query">
…
       <filter class="solr.SynonymFilterFactory" synonyms="NL_synonyms.txt" … />
…
   </analyzer>
</fieldType>
…
<dynamicField name="attr_nl_*" type="textnl" indexed="true" stored="true" multiValued="true"/>
```

Figuur 3: Definitie van eigen velden in Solr

[7] http://incubator.apache.org/connectors/

In Figuur 3 wordt als voorbeeld de definitie van het Nederlandstalige veldtype (*textnl*) weergegeven met de toewijzing aan het dynamische veld (*attr_nl_**). Dit betekent dat velden met een naam van deze vorm, bijvoorbeeld "*attr_nl_desc*" dit type zullen toegewezen krijgen.

Zonder in detail te treden zien we dat er twee analyzers zijn, één voor het indexeren (*index*) en één voor het verwerken van de zoekopdracht (*query*). Dan volgen er een aantal filters die onder andere eventuele HTML-tags verwijderen, speciale en geaccentueerde karakters omzetten, stopwoorden verwijderen, alles omzetten in *lowercase* en de woorden terugbrengen tot hun stam.

Zoals eerder besproken, staan analyzers in voor het bewerken van de tekst vooraleer deze ofwel in de index belandt ofwel vooraleer de zoekopdracht wordt uitgevoerd.

De analyzer voor de zoekopdracht (*query*) is verschillend van de *index*-analyzer door bijvoorbeeld de mogelijkheid om synoniemen van de zoektermen toe te voegen zodat er ook resultaten worden teruggegeven die het synoniem van het gezochte woord bevatten. Merk op dat je zelf deze synoniemenlijst moet voorzien. Je kan deze lijst baseren op bestaande gratis of commerciële synoniemenlijsten maar deze moeten uiteraard eerst omgezet worden in het juiste formaat.

Het gedrag van deze analysers kan je perfect nagaan via de administratie-interface van Solr. Zie hiervoor paragraaf 2.2.4.

Men kan zich wel de vraag stellen of dynamische velden wel aangewezen zijn indien Solr als bedrijfswijde zoekoplossing voor diverse content management systemen en websites zou opgezet worden. Het wordt dan vooral oppassen met de wildgroei van velden. Het lijkt op dat moment beter om het gebruik van dynamische velden sterk te beperken. Maar om snel en flexibel aan de slag te kunnen, biedt deze manier van werken wel haar voordelen.

2.2.2 Indexeren

In SolrJ kan je gebruik maken van Java-beans die een voorstelling vormen van het te indexeren document. Elke property van de bean met zijn getter-methode resulteert in een veld in de Solr-index. Met behulp van annotaties kan je in de bean aangeven met welk veld in de index een bepaalde property overeenkomt. In het onderstaande voorbeeld komt de *text_NL* property overeen met het *attr_nl_desc* (een dynamisch veld zoals hierboven vermeld).

```
import org.apache.solr.client.solrj.beans.Field;

public class DaisyItem {
    @Field
    String id;

    @Field
    String name;

    @Field("type_s")
    String type;

    @Field("attr_nl_desc")
    String text_NL;
…
```

Na de creatie van de connectie met de Solr-server kan je dan deze bean met een eenvoudig *server.add(bean)* en een *server.commit()* toevoegen aan de index.

Indien dit document reeds bestaat (zelfde ID), dan wordt het in de index geüpdatet.

Van zodra je dus de structuur van de bean en het corresponderende schema hebt bepaald, is het indexeren een koud kunstje.

2.2.3 Zoekoperaties

SolrJ kan je ook gebruiken om de zoekinterface te bouwen. In het kader van het prototype hebben we ons echter beperkt tot het gebruik van pure URL's op de Solr-server.

De geïndexeerde documenten zijn perfect doorzoekbaar. Als je bijvoorbeeld zoekt op de term lucene (http://solrserver:8983/solr/select?q=lucene), krijg je mooi de velden van het resultaatdocument (Figuur 4) terug. Wens je het originele document weer te geven, dan dien je wel zelf nog de URL te reconstrueren. Deze zou ook bij aanvang mee opgeslagen kunnen worden in de index, zodat er geen bijkomende logica nodig is achteraf.

```
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">0</int>
    <lst name="params">
      <str name="q">lucene</str>
    </lst>
  </lst>
  <result name="response" numFound="1" start="0">
    <doc>
      <str name="id">249-DSY</str>
      <str name="name">Apache Lucene</str>
      <arr name="attr_nl_desc">
        <str>"Apache Lucene" is een Java library voor geavanceerde
        en hoogperformante zoekmachine -functionaliteiten. De
        …
         </str>
      </arr>
       …
    </doc>
  </result>
</response>
```

Figuur 4: Resultaat van een zoekopdracht met Solr

Standaardfunctionaliteit van veel zoekmachines is de duidelijke aanduiding (highlight) van de gezochte termen in de zoekresultaten door deze bijvoorbeeld te markeren met een tag (standaard is dit **). Het gedrag van de highlighter is configureerbaar in de configuratiefile (*solrconfig.xml*). Via de aanduiding *hl=true* en een aantal andere parameters zal Solr in het zoekresultaat naast de gevonden documenten een aparte XML-structuur met de highlights weergeven.

```
http://solrserver:8983/solr/select?q=attr_nl_desc:lucene&hl=true&hl.fl=*&hl.fragsize=300
```

```
…
<lst name="highlighting">
  <lst name="249-DSY">
    <arr name="attr_nl_desc">
      <str>"Apache <em>Lucene</em>" is een Java library voor geavanceerde en
hoogperformante zoekmachine -functionaliteiten. De technologie is geschikt voor vrijwel
alle toepassingen waarbij de volledige tekst van documentcollecties doorzocht moet
kunnen worden, in het bijzonder in cross</str>
    </arr>
  </lst>
…
```

Net zoals bij Lucene laat Solr ook complexere zoekopdrachten toe, waar je verschillende zoektermen en velden kan combineren met booleaanse operatoren. Daar bovenop kan je ook gewichten meegeven aan de te doorzoeken velden zodat bijvoorbeeld zoekresultaten die het gezochte woord in een titel bevatten

hoger gerankschikt zullen worden dan deze waar het woord enkel voorkomt in een algemeen tekstveld. De mogelijkheden van Solr zijn op dat vlak dus dezelfde als Lucene.

Tijd om naar de wat meer geavanceerde functionaliteit te kijken. In het geval je gebruikers wenst te helpen met eventuele spelfouten in hun zoekopdracht, kan je in Solr een vorm van spellingscontrole definiëren. Deze zal via een eigen index bepalen of een bepaalde zoekterm overeenkomt met een veel voorkomend woord in deze index. Dit zie je typisch ook bij zoekmachines die bovenaan de zoekresultaten de vraag stellen “Bedoel je <*dit woord*>?”.

Hiervoor dienen een paar additionele velden in het schema (Figuur 5) gedefinieerd te worden en dient het configuratiebestand aangepast te worden. In dit geval komt het erop neer dat er een veldtype wordt aangemaakt dat een beperkte taalanalyse zal uitvoeren om zoveel mogelijk het oorspronkelijke woord te bewaren. Een zogenaamd *copy-field* (*spell_nl*) wordt volgens dat type gedefinieerd. Dit veld bevat een kopie van alle inhoud van zelf gedefinieerde velden, in dit geval alle tekstvelden (attr_nl_*) en het naamveld.

```
<fieldType name="textSpellNl" class="solr.TextField" positionIncrementGap="100">
  <analyzer>
    <charFilter class="solr.HTMLStripCharFilterFactory" />
    <filter class="solr.StopFilterFactory" ignoreCase="true"
    words="NL_stopwords.txt" enablePositionIncrements="true" />
    <tokenizer class="solr.StandardTokenizerFactory" />
    <filter class="solr.LowerCaseFilterFactory" />
    <filter class="solr.RemoveDuplicatesTokenFilterFactory" />
  </analyzer>
</fieldType>
…
<field name="spell_nl" type="textSpellNl" indexed="true" stored="false"
multiValued="true"/>
…
<copyField source="attr_nl_*" dest="spell_nl"/>
<copyField source="name" dest="spell_nl"/>
```

Figuur 5: Schema voor spellingscontrole

In het configuratiebestand (*solrconfig.xml*) geef je aan in de spellCheckComponent op welke manier je de spellchecker wenst aan te spreken en met welke velden. In dit geval willen we een onderscheid maken tussen Nederlandstalige en Franstalige tekst, met hun eigen naam, het gedefinieerde copy-field en de corresponderende index.

```
<searchComponent name="spellcheck" class="solr.SpellCheckComponent">
…
    <lst name="spellchecker">
      <str name="name">defaultnl</str>
      <str name="field">spell_nl</str>
      <str name="spellcheckIndexDir">./spellcheckernl</str>
    </lst>
…
```

Figuur 6: Configuratie voor spellingscontrole

Als je dan de spellchecker-functie (*spell*) aanroept op een beschikbare index, krijg je een mooi resultaat. Bij de zoekterm “*lucien*” suggereert de spellchecker “*lucene”* als zoekterm. Daarna dien je nog een tweede zoekopdracht te lanceren met de juiste term, net zoals dit bij andere zoekmachines het geval is.

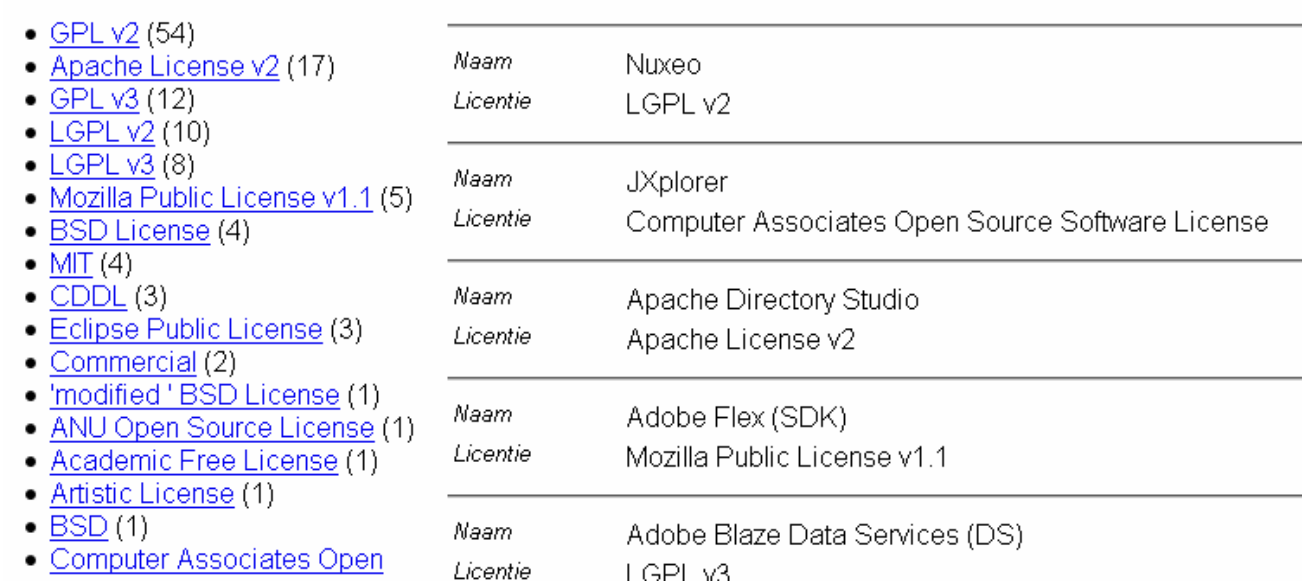
```
http://solrserver:8983/solr/spell?q=lucien&spellcheck=true&spellcheck.dictionary=defaultnl
```

```
…
  <lst name="spellcheck">
    <lst name="suggestions">
      <lst name="lucien">
…
        <arr name="suggestion">
          <str>lucene</str>
…
```

Figuur 7: Zoekopdracht en -resultaat van spellingscontrole

Een functionaliteit die ook heel bruikbaar is, is de zogenaamde *facet-query*. Deze laat toe om zoekresultaten te classificeren volgens de waarden in een specifiek veld. Bevat dit veld enkel woorden, dan krijg je per woord de aantallen te zien. Je kan echter bij numerieke velden een range aangeven. Dit is een functionaliteit die je vaak ziet bij webwinkels die je toelaten om hun catalogus te filteren volgens diverse criteria zoals functionaliteit, prijsrange, …

In ons voorbeeld met Daisy (inventaris open source software) bevatten de gebruikte documenten een veld met de naam *softwarelicentie*. Als we eenvoudigweg dit als facet gebruiken, dan kunnen we een lijst bekomen van zoekresultaten met het voorkomend aantal documenten met deze bepaalde veldwaarde. Als we de weergave door Solr laten doen aan de hand van een eenvoudige XSLT-stylesheet, krijgen we het resultaat uit Figuur 8. Aan de linkse zijde worden de facetten weergegeven, bijvoorbeeld 54 documenten die “GPL v2” hebben als softwarelicentie. Dit kan dan gebruikt worden om de resultaatlijst verder te filteren.



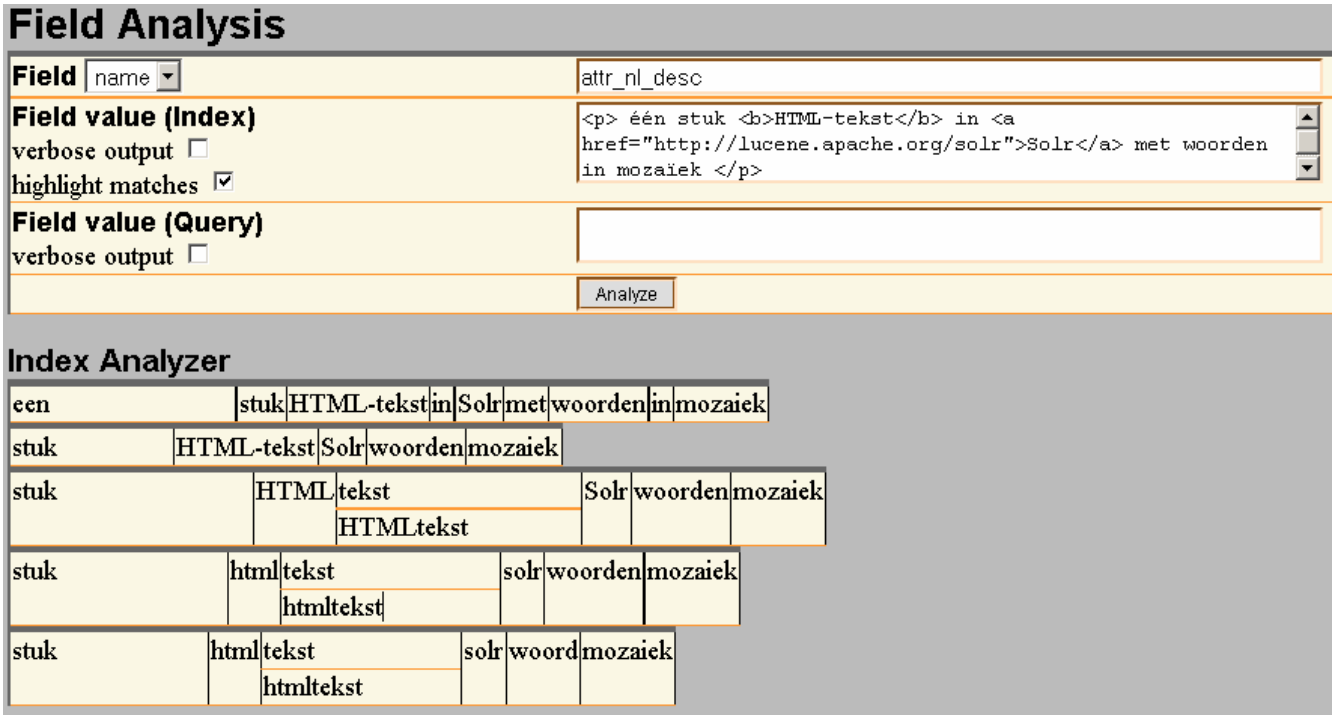
Figuur 8: Resultaat van een facet-query

2.2.4 Administratie-interface

Solr beschikt over een administratie-interface die op het eerste gezicht beperkt lijkt maar wel heel wat functionaliteit herbergt. Je kan er zoekopdrachten in uitvoeren en de configuratiebestanden van het systeem bekijken.

Een interessante functie is het gedeelte “analysis”. Daar kan je testen hoe een bepaald veld zich zal gedragen bij het indexeren en uitvoeren van een zoekopdracht. Figuur 9 toont bijvoorbeeld hoe een stuk HTML-tekst met geaccentueerde karakters (“één stuk HTML-tekst in Solr met woorden in mozaïek”) zich

zal gedragen in het eerder gedefinieerde dynamische *attr_nl_** veld. De onderste reeks “woorden” worden opgeslagen in de index. Bemerk hierbij de verwijdering van HTML-tags, omzetting van accenten, het weglaten van “een”, de verschillende wijzen van opslag van “HTML-tekst” (“html”, “tekst”, “htmltekst”), … Bij de *verbose output* krijg je nog meer info te zien.



Figuur 9: Debuggen van tekstanalyse via administratie-interface

3. Evaluatie maturiteit

Aan de hand van het maturiteitsmodel voor open source software[8] van de sectie Onderzoek van Smals werden enkele niet-functionele criteria nagegaan met betrekking tot de software.

Figuur 10 bevat een overzicht van de evaluatie. De gedetailleerde criteria, hun waarden en scores zijn terug te vinden in de bijlagen (paragraaf 6.1). Solr bezit een heel degelijke algemene score (3,9/5). Als gewichtsverdeling werd de standaard “routine use” gekozen. Ter vergelijking werden een aantal andere gewichtsverdelingen uitgeprobeerd (Mission critical en Internal development). Deze geven echter ongeveer hetzelfde resultaat.

[8] http://inventarisoss.smals.be/nl/160-RCH.html

Evaluated Technology			
Product Name:	**Apache Solr**	**Total Score (/5)**	**3,89**
Product Website:	**http://lucene.apache.org/solr/**		
Version :	**1,4**		
Project size:	**130000**		
Evaluation date:	**7/07/2010**		
Evaluated by:	**Bob Lannoy**		
Product type :	**Enterprise Search**		
Usage Setting:	**Routine use**	Put your custom weights in the table on the 'Weights' sheet	

Rank	Category	Weight	Unweighted score	Weighted score
1	Installation	23,00%	5,00	1,15
2	Quality	23,00%	2,40	0,55
3	Security	0,00%	4,60	0,00
4	Performance	15,00%	4,00	0,60
5	Scalability	0,00%	4,00	0,00
6	Architecture	0,00%	4,20	0,00
7	Support	15,00%	5,00	0,75
8	Documentation	5,00%	4,40	0,22
9	Adoption	14,00%	2,80	0,39
10	Community	5,00%	4,50	0,23
11	Professionalism	0,00%	3,60	0,00
12	License	0,00%	0,00	0,00

	Total Weight	Average unweighted score	Total weighted score
	100,00%	**4,01**	**3,89**

Figuur 10: Evaluatie van de maturiteit van Solr 1.4

Als we naar de signaalcriteria kijken in Figuur 11, dan valt op dat Solr een zeer actieve community heeft en over professionele support beschikt door derden. De stichters van Lucene hebben het bedrijf LucidImagination[9] opgestart met een commercieel aanbod maar met ook heel wat technische documentatie die beschikbaar is voor iedereen. Er zijn ook heel wat consultancybedrijven die ontwikkelingen voor Lucene & Solr uitvoeren.

Het model geeft aan dat er mogelijke kwaliteitsproblemen zijn met de code. Dit heeft te maken met de gegevens die uit het Jira-bugtrackingsysteem van Solr komen en die een ietwat vertekend beeld geven. Aangezien Solr heel modulair is, zijn er heel wat bugs in diverse onderdelen van Solr, zij het dan niet-kritieke onderdelen van Solr. Over het algemeen gezien kan de kwaliteit van de code als goed beschouwd worden.

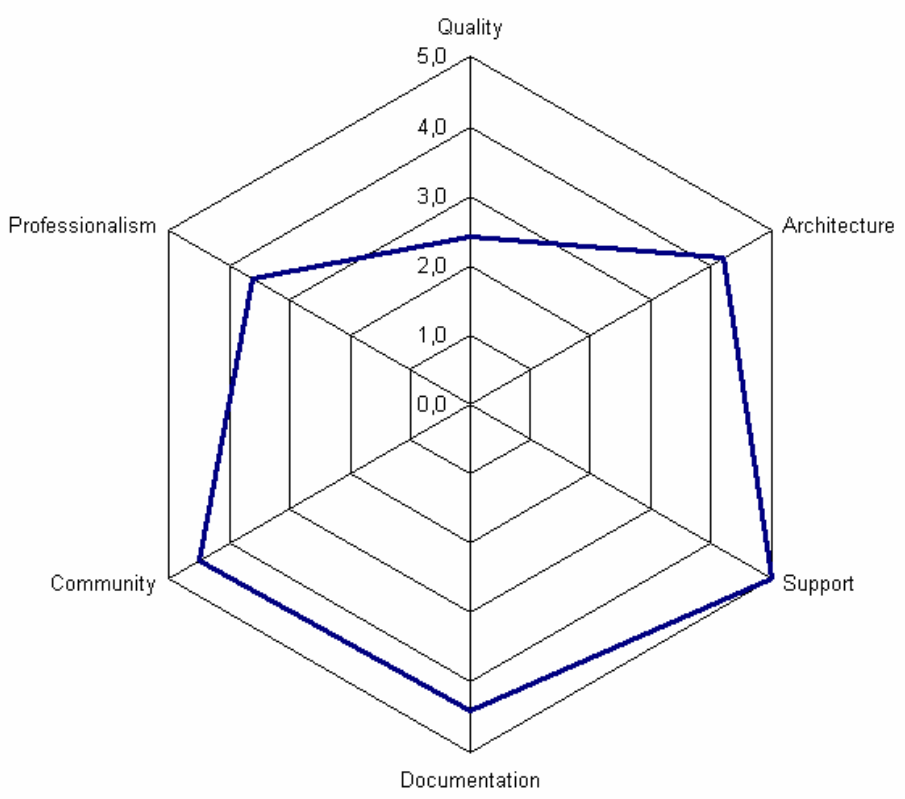
Solr is beschikbaar onder een Apache License die een zeer vrij gebruik van het pakket toelaat. Dit heeft uiteraard als nadeel dat er heel wat bedrijven zijn die Solr kunnen aanpassen of verbeteren zonder dat ze verplicht zijn om deze aanpassingen terug te geven aan de community (vandaar de lage score wat betreft “Protection against proprietary forks”).

Tot slot geeft Figuur 12 nog eens de belangrijkste scores weer in grafische vorm. Zoals vermeld geeft het kwaliteitscriterium een vertekend beeld en is het eigenlijk beter dan weergegeven. Het pakket is redelijk goed gedocumenteerd. Al moet gezegd dat de documentatie vooral te vinden is op de Solr-wiki en het soms moeilijk is om deze in één geheel te zien. Daarnaast zijn er ook diverse websites die informatie rond Solr/Lucene-implementaties geven.

[9] http://www.lucidimagination.com

This spreadsheet contains a number of key elements that should be considered as indicators for possible problems. Low scores could indicate serious issues.		
Category	**Unweighted rating (/5)**	**Score**
Number of bugs fixed in last 6 months (compared to # of bugs opened)	3	45% - 60%
Standards	5	Latest industry standards
Average volume of general mailing list in the last 6 months	5	> 720 msg per month
Quality of professional support	5	Installation + troubleshooting + integration / customization support
Assessed paid support	5	>5
Reference deployment	5	Yes, with publication of user's size
Age	5	> 3 years
Status	5	Mature, stable
Possible license issues		
Protection against proprietary forks (GPL preferred)	1	Very permissive like BSD or Apache licenses.
Permissiveness preferred	5	Very permissive like BSD or Apache licenses.
Multiple licenses?	5	Only open source license
Limitations of community or free edition	5	No different flavors or flavors offer same functionality. Commercial license offers e.g. services, GPL protection, …

Figuur 11: Signaalcriteria Solr 1.4



Figuur 12: Diagram met de diverse scores voor Solr 1.4

4. Besluit & aanbevelingen

Apache Solr is een zeer degelijke zoekserver met een uitgebreide set aan functionaliteit zoals taalanalyse, facetclassificatie, spellingscontrole en resultaat-highlighting. De server kan als zelfstandige zoekmachine gebruikt worden in alle projecten die nood hebben aan een zoekfunctie. Het platform beschikt over mechanismen om het systeem te schalen tot zeer grote volumes.

Zijn programmeertaal-agnostische interface via HTTP/XML/JSON maakt het mogelijk om onafhankelijk van de gebruikte programmeertaal van deze infrastructuur gebruik te maken. Daarnaast zijn er een aantal specifieke mogelijkheden om bijvoorbeeld in Java of Ruby met de server te werken.

Solr is geen plug-and-play oplossing. Het grootste probleem met Solr is de aansluiting van *content*-bronnen. Te indexeren documenten moeten immers omgezet worden naar een standaardformaat. In tegenstelling tot commerciële zoekoplossingen beschikt Solr over vrijwel geen connectoren naar content management systemen. Het Lucene Connectors Framework probeert hieraan te verhelpen maar staat nog in zijn kinderschoenen.

Ook voor het zoekresultaat moet je zelf de nodige schermen bouwen. Dit is op zich niet zo een ingewikkelde materie. Solr laat ook toe om een volledige zoekinterface te bouwen die op zichzelf kan staan. Zo dien je niet in alle toepassingen een eigen zoekscherm te bouwen.

Het implementeren van een zoekoplossing is een project op zich. Je moet bepalen wat je wenst te doorzoeken uit welke systemen. Daar dienen dan de nodige connectoren voor gebouwd te worden. Er moet uitgebreid stilgestaan worden bij de definitie van het schema met de gewenste functionaliteit voor de zoekresultaten. Daarna dient de zoekinterface gebouwd te worden voor de presentatie van de zoekresultaten.

Solr biedt aan programmeurs een snelle en degelijke manier om een zoekoplossing in een project te integreren. Gebruikers van Lucene zullen heel wat features van Solr (her)kennen. Solr steunt immers op heel wat bestaande Lucene-functionaliteit. Gezien de sterke overlap tussen Solr en Lucene werken deze nu met een gemeenschappelijke codebasis. Beide projecten zullen echter wel blijven bestaan.

Je kan je de vraag stellen waarom je zou kiezen voor Solr en niet puur voor Lucene. Solr verbergt voor een groot stuk het werk dat elke Lucene-gebruiker moet verwerken in code. Aangezien dit een server is, kan deze onafhankelijk van elk project beheerd worden door een team dat de zoekproblematiek door en door kent. Vaak wordt de zoekfunctie in veel systemen achteraf toegevoegd en op een weinig doordachte manier ingevoerd. Een centrale zoekinfrastructuur kan ervoor zorgen dat de aanpak rond zoekfunctionaliteit meer transversaal gebeurt.

Kortom, Solr is zeker het bekijken waard, maar dient met de nodige voorzichtigheid aangepakt te worden.

5. Referenties

[1] Algemene Solr-website: http://Lucene.apache.org/solr
[2] Apache Solr tutorials bij IBM: http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=apache+solr
[3] "*Rich Internet Applications: bruikbare en optimaal toegankelijke applicaties*" – Delivrable - A. Groebbens – Juni 2008

6. Bijlagen

6.1. Open Source Selectiemodel

Zoals besproken in paragraaf 3 werd het maturiteitsmodel voor Open Source Software[10] ingevuld.

De onderstaande tabel bevat alle ingevulde criteria.

Evaluated Technology							Score
Apache Solr v.1,4							**3,89**
7/07/2010							
	Category Title	**Measured value**	**Score**	**Weight**	**Unweighted Rating**	**Weighted Rating**	
	For more information open the outline (+ / -) left from row number	*Please put your description, comments and measured value here for future reference*	*Select appropriate range or value*				
1	**Installation**			23%	**5,00**	**1,15**	
1.1	Time for setup pre-requisites for installing open source software	Java and J2EE container but not necessary from start (Jetty used)	< 10 minutes	50%	5	2,50	
1.2	Time for vanilla installation/configuration	Sample runs directly	< 10 minutes	50%	5	2,50	
2	**Quality**			23%	**2,40**	**0,55**	
2.1	Number of minor releases in past 12 months		1 or 3	15%	3	0,45	
2.2	Number of point/patch releases in past 12 months	Sometimes Jira-issues contain patches. But not integrated in release	0 or > 6	15%	1	0,15	
2.3	Number of opened bugs for the last 6 months	131	100 - 500	10%	3	0,30	
2.4	Number of bugs fixed in last 6 months (compared to # of bugs opened)	61	45% - 60%	30%	3	0,90	
2.5	Number of P1/critical bugs opened	mostly in client-code, bugs seem not to be that critical …	1 - 5	10%	4	0,40	
2.6	Average bug age for P1 in last 6		> 4 weeks	20%	1	0,20	

[10] [10] http://inventarisoss.smals.be/nl/160-RCH.html

	months						
3	**Security**			0%	**4,60**	**0,00**	
3.1	Number of security vulnerabilities in the last 6 months that are moderately to extremely critical		0	40%	5	2,00	
3.2	Number of security vulnerabilities still open (unpatched)		0	40%	5	2,00	
3.3	Is there a dedicated information (web page, wiki, etc) for security?		Yes	20%	3	0,60	
4	**Performance**			15%	**4,00**	**0,60**	
4.1	Performance Testing and Benchmark Reports available		Yes, with good results	50%	5	2,50	
4.2	Performance Tuning & Configuration		Yes, Some	50%	3	1,50	
5	**Scalability**			0%	**4,00**	**0,00**	
5.1	Reference deployment		Yes, with publication of user's size	50%	5	2,50	
5.2	Designed for scalability		Yes, some	50%	3	1,50	
6	**Architecture**			0%	**4,20**	**0,00**	
6.1	Are there any 3rd party Plug-ins		2 to 5	20%	3	0,60	
6.2	Public API / External Service		Yes, extensive	30%	5	1,50	
6.3	Enable/disable features through configuration		Yes, maybe compile time	20%	3	0,60	
6.4	Standards		Latest industry standards	30%	5	1,50	
7	**Support**			15%	**5,00**	**0,75**	
7.1	Average volume of general mailing list in the last 6 months		> 720 msg per month	50%	5	2,50	
7.2	Quality of professional support		Installation + troubleshooting + integration / customization support	25%	5	1,25	
7.3	Assessed paid support		>5	25%	5	1,25	
8	**Documentation**			5%	**4,40**	**0,22**	
8.1	Existence of various documents.	Lots of documentation but sometimes hard to find or to place it in context	Install/deploy, user, admin, upgrading guides available in several formats	60%	4	2,40	
8.2	User contribution framework		People are allowed to contribute, and it is edited / filtered by experts	40%	5	2,00	
9	**Adoption**			14%	**2,80**	**0,39**	

9.1	How many books does amazon.com gives in the Books / Advanced Search query: "subject:computer and title:component name"		[1 – 3)	15%	2	0,30	
9.2	Reference deployment		Yes, with publication of user's size	50%	5	2,50	
9.3	Total number of downloads	Hard so say, no numbers available		35%	0	0,00	
10	**Community**			5%	**4,50**	**0,23**	
10.1	Average volume of general mailing list in the last 6 months		> 720 msg per month	25%	5	1,25	
10.2	Number of unique code contributors in the last 6 month		[10 – 20)	25%	3	0,75	
10.3	Age		> 3 years	20%	5	1,00	
10.4	Status		Mature, stable	30%	5	1,50	
11	**Professionalism**			0%	**3,60**	**0,00**	
11.1	Project Driver	Apache project	Independent foundation supported by corporations (apache / OSDL style)	30%	5	1,50	
11.2	Difficulty to enter the core developer team		Rather difficult, must contribute accepted patches for some time	20%	3	0,60	
11.3	Leading team		2 to 5	20%	3	0,60	
11.4	Roadmap		Existing road-map without planning.	30%	3	0,90	
12	**License**			0%	**0,00**	**0,00**	
12.1	Protection against proprietary forks (GPL preferred)		Very permissive like BSD or Apache licenses.	0%	1	0,00	
12.2	Permissiveness preferred		Very permissive like BSD or Apache licenses.	0%	5	0,00	
12.3	Multiple licenses?		Only open source license	0%	5	0,00	
12.4	Limitations of community or free edition		No different flavors or flavors offer same functionality. Commercial license offers e.g. services, GPL protection, …	0%	5	0,00	

6.2. Testscenario

Als testopstelling werd gebruik gemaakt van:

- Standaardlaptop: HP EliteBook 8730w; 3 GB RAM; Windows XP SP3
- Apache Solr 1.4.0 met Jetty als Java-container
- Java 1.6
- Eclipse IDE 3.4 voor de ontwikkeling van de connector met Daisy
- DaisyCMS 2.1 als content-repository op een aparte server. Deze had beschikbare inhoud die werd gebruikt in dit prototype.