Meteen naar de juiste plek

Zoeken, vinden en ordenen van semi-gestructureerde data

door Maarten Marx,
maartenmarx@uva.nl

Dit artikel is gebaseerd op de lezing van de auteur met dezelfde titel op het XML Holland congres 2008. Het bespreekt de fascinerende mogelijkheden die binnen handbereik komen als traditionele tekstdocumenten zoals notulen beschikbaar zijn in XML. Het expliciet maken in XML van de al aanwezige structuur biedt twee grote voordelen: voor zoeken en voor analyse.

Zoekmachines die met XML-documenten werken kunnen gebruikers makkelijk gericht naar de juiste plek(ken) binnen een document brengen. Dat levert vaak enorme tijdswinst op (de huidige veel gebruikte techniek hiervoor heet *Control F*). Analyse van bijvoorbeeld een jaargang notulen wordt kinderlijk eenvoudig met XPath en XSLT. Denk eens aan een simpele informatiebehoefte als *hoeveel agenda-onderwerpen zijn er het afgelopen jaar besproken?* of *hoe vaak heeft Jansen zich dit jaar weer afgemeld?*.
In het artikel wordt eerst het verschil tussen tekst- en datagebaseerde XML behandeld. Dan wordt ingegaan op gericht zoeken binnen XML. Ten slotte wordt het voorbeeld van de notulen uitgewerkt.

Data- en tekstgebaseerde XML

De eerste alinea van de W3C XML pagina rept al meteen van de twee gezichten van XML: een taal om gegevens uit te wisselen en een taal om tekstdocumenten mee op te maken. Deze twee geizchten komen ruwweg overeen met twee typen XML-bestanden:

- **tekst-gebaseerde XML**: markup bestaat veelal uit opmaakinstructies en semantische annotaties; de inhoud (#PCDATA) bestaat voornamelijk uit tekst. Zonder de markup is de inhoud nog begrijpelijk voor mensen.
- **data-gebaseerde XML**: de inhoud bestaat uit gegevens die zonder de markup niet of moeilijk te begrijpen zijn. De markup dient als container om data uit te wisselen.

Deze scheiding is ook terug te zien in de academische groepen die zich met XML bezighouden. Database (DB) groepen werken vrijwel uitsluitend met datagebaseerde XML. Information retrieval (IR) groepen richten zich juist op de tekstgebaseerde XML. Traditioneel is er weinig interactie tussen deze twee werelden.
Dat is jammer want misschien wel de interessantste XML-documenten zijn zowel data- als tekstgebaseerd. Een goed voorbeeld is de RSS standaard. RSS bestanden bevatten zowel strict gereguleerde datavelden (datum, tijd, URLs, en tegenwoordig ook geocodes) als vrije tekst (in de `title` en `description` elementen). De Lonely Planet reisgidsen zijn een ander mooi voorbeeld van gecombineerde data- en tekstgebaseerde documenten. Notulen van vergaderingen zijn vaak ook volgens een vaste manier gestructureerd en bevatten zowel vrije tekst als strict getypeerde gegevens.
Deze documenten met zowel tekst als data vragen om een eigen aanpak die technieken uit zowel de database als de information retrieval wereld combineert. We noemen drie specifieke punten:

1. **Speciale querytaal voor inhoud en structuur.** De XML-structuur nodigt uit tot het stellen van specifiekere zoekvragen dan standaard zoeksystemen als Google mogelijk maken. Een vraag over een verzameling notulen is bijvoorbeeld: *geef me de tekst van sprekers die het hebben over milieuvervuiling binnen agendaonderwerpen over luchthavens*. Deze zoekvraag is makkelijk te schrijven in een XPath-achtige taal:

```
//onderwerp[about(.,'luchthavens')]//
  sprekertekst[about(.,'milieu vervuiling')]//text()
```

   Hier is de `about()`-functie de IR analoog van XPath's `contains()`, maar met een lossere semantiek. Zo is `about(TEKST,'luchthaven')` ook waar als `TEKST` over luchthavens gaat maar in plaats van de string `luchthaven` termen als `vliegveld` of `Schiphol` bevat.
2. **Geordende lijst van antwoorden.** Gebruikers verwachten het meest relevante antwoord bovenaan. Het ordenen van antwoorden wordt echter een stuk moeilijker met complexe zoekvragen zoals die in punt 1.
3. **XML-DBMS met geïntegreerde DB- en IR-support.** De typische use-case bevat een continu groeiend, groot aantal, relatief kleine (minder dan 1Mb) XML-documenten. Hierop worden zowel complexe XQueries (data-analyse, data-mining, maken van overzichten) als inhoud en structuur zoekvragen gesteld. Bij de laatste verwachten we op relevantie geordende antwoorden.

Samenvattend: gecombineerde tekst- en datagebaseerde XML is razend interresant en dwingt XML-ontwikkelaars uit verschillende hoeken om samen te werken. Een academische setting waar dit gebeurt is de jaarlijkse INEX evaluatieronde (zie http://inex.is.informatik.uni-duisburg.de). Het Nederlandse DBMS MonetDB/XQuery met de PFTijah uitbreiding is een

XML-systeem met zowel een erg goede XQuery performance als een uitgebreide IR-zoekfunctionaliteit. Het voldoet goed aan de zojuist genoemde drie punten. Zie http://monetdb.cwi.nl/XQuery/ en http://dbappl.cs.utwente.nl/pftijah.

Meteen naar de juiste plek

XML-bestanden die zowel tekst als data bevatten vragen dus om een gecombineerde DB en IR aanpak. In deze sectie kijken we naar de IR kant. Standaard zoekmachines geven een lijst *documenten* als antwoord op een zoekvraag, en laten het vervolgens aan de gebruiker over om *binnen* die documenten de relevante informatie op te sporen. Er wordt soms schamperend gesproken over *document retrieval* in plaats van de gewenste *information retrieval*. Natuurlijk zou het fijn zijn als Google je ook meteen naar de juiste plek in een document brengt maar twee simpele beperkingen maken dit moeilijk. Ten eerste heeft een zoekmachine geen mogelijkheid om ankers in documenten van anderen aan te brengen. Verwijzen is daardoor eigenlijk niet goed mogelijk. Ten tweede is het lastig om willekeurige documenten in zinvolle antwoordeenheden op te delen.

In principe lost een document in XML-formaat beide problemen op. Het inzicht dat documenten in XML uitnodigen tot een meer fijnmazige manier van antwoord geven lag aan de basis van de eerder genoemde INEX workshops. Binnen INEX wordt onderzoek gedaan naar het maken van specifieke zoekmachines voor XML documenten. De INEX-"spelregels" kunnen ruwweg als volgt worden samengevat: (1) elk XML-element uit elk XML-document in de collectie is een mogelijk antwoord op een zoekvraag en (2) een antwoord is goed als het aan de volgende twee eisen voldoet:

A het antwoord is *relevant* (het bevredigt de informatiebehoefte van de gebruiker);
B het antwoord is *precies goed*; er staat niet teveel en niet te weinig in het teruggegeven element.

Eis A is standaard voor elke zoekmachine. Eis B is specifiek voor het zoeken binnen XML en maakt de XML-element zoektaak een stuk moeilijker dan ouderwetse document-retrieval. Het bouwen van een XML-element zoeksysteem is dus ook een stuk kostbaarder. Toch zijn er verschillende gevallen waarin die extra inspanning de moeite loont. Denk hierbij aan moeilijk doorzoekbare documenten (bijv. programmacode, video) en lange documenten die veel onderwerpen bevatten (notulen, jaarverslagen, handboeken).

Verschillende Nederlandse wetenschappers houden zich met XML-element retrieval bezig, waaronder Djoerd Hiemstra (Universiteit Twente), Jaap Kamps (Universiteit van Amsterdam) en Arjen de Vries (CWI Amsterdam). Hiemstra en de Vries zijn beide nauw betrokken bij de eerder genoemde XML-IR uitbreiding van MonetDB/XQuery.

Meer over INEX- en XML-zoeken is te vinden in het goed leesbare proefschrift van Börkur Sigurbjörnsson (http://borkur.sdf-eu.org/publications/details/sigurbjornsson2006phdthesis.html). Een andere ingang vormt het verslag van de laatste SIGIR workshop over *focussed retrieval* op http://doi.acm.org/10.1145/1480506.1480517.

Praktijkvoorbeeld: Handelingen der Staten Generaal

De verslagen van de vergaderingen in de Tweede Kamer zijn een prachtig voorbeeld van lange (vaak best saaie), zeer goed gestructureerde documenten die zowel data als tekst bevatten. Tot voor kort waren ze alleen in PDF beschikbaar. Nu al staan alle verslagen vanaf 1975 op het web, en vanaf 2010 zullen alle verslagen sinds 1814 voor iedereen beschikbaar zijn (http://www.statengeneraaldigitaal.nl). Aan deze mooie collectie kun je leuke onderzoeksvragen stellen. Een voorbeeld:

Sinds de verkiezingen van November 2006 staat het percentage vrouwelijke kamerleden op een recordhoogte die schommelt rond de 37%. Maar spreken de vrouwen verhoudingsgewijs ook net zo vaak in de Kamer?

Een voor de hand liggende manier om deze vraag te beantwoorden is om te tellen hoeveel betogen er sinds November 2006 zijn gehouden en hoeveel daarvan door vrouwen. Daarnaast zou je de spreek*tijd* kunnen benaderen door het aantal woorden te tellen. Dit is op basis van de verslagen in PDF natuurlijk een vreselijke klus.


De heer **Van Gerven** (SP): Ik heb gezien dat de amendementen inmiddels in Parlando staan.

De SP maakt zich zeer grote zorgen over de kwetsbare positie van kinderen. Punt is dat, wanneer ouders niet verzekerd zijn, zij zorg voor zichzelf, maar ook voor het kind mijden. Dit is niet in overeenstemming met het

Tweede Kamer Zorgverzekeringswet 2 april 2008 TK 71 71-4975

Figuur 1: Het onderste deel van een PDF-pagina van de Handelingen waarin de heer Van Gerven een uitspraak doet.

Maar stel dat we de verslagen in een XML-formaat hadden, dan is een simpele XPath 1.0 expressie voldoende:

```
let $sprekers := collection('HAN')//
notulen[@date ge 2006-11-30]//
                    spreker[@rol = 'parlementslid']
let $VrouwAandeel := count($sprekers[@geslacht=
'v']) div count($sprekers)
```

De XML-representatie, zoals die gebruikt wordt binnen www.polidocs.nl, van het stukje uit Figuur 1 staat hierna afgedrukt:

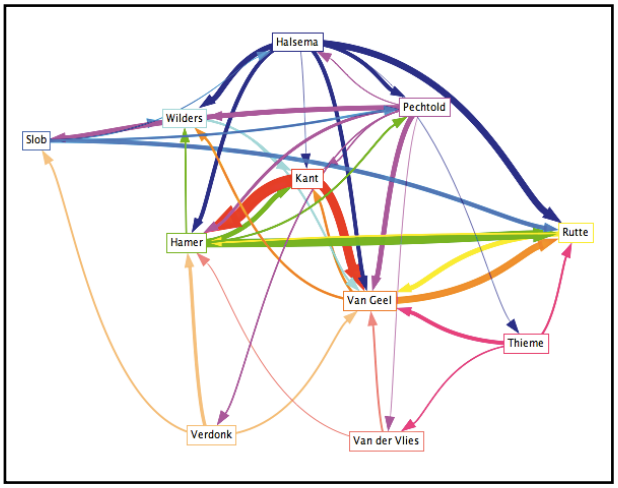
```
<spreker pagina="71-4975"
                    anker="568"
                    partij="SP"
                    naam="Van Gerven"
                    PDCid="03116"
                    rol='parlementslid'
                    geslacht='m'>
<p>Ik heb gezien dat de amendementen inmid-
dels in Parlando staan.</p>
</spreker>
```

In XPath 2.0 kan je in tegenstelling tot XPath 1.0 makkelijk woorden tellen met de `tokenize()` functie. Een andere manier om spreektijd te benaderen is natuurlijk om het aantal karakters te tellen. XPath 1.0's `string-length` functie kan dit prima, maar wil een string als argument, terwijl wij een verzameling knopen in `$sprekers` hebben zitten. Die kunnen we in XPath 2.0 als volgt aan elkaar plakken tot één lange string: `string-join(for $s in $sprekers return string($s))`.

Political Mashup

Binnen het PoliticalMashup project (zie http://www.politicalmashup.nl) wordt nu steeds meer politieke data omgezet naar XML. Er was behoefte aan een goede zoekmachine voor parlementaire data en die is gemaakt op basis van de XML-retrieval methodiek en is beschikbaar op http://www.polidocs.nl. Het beschikbaar zijn van politieke data in een eenvoudig te bewerken XML-formaat nodigt ook uit tot creatieve toepassingen die echt gebruik maken van de (XML-) structuur die in de data aanwezig is. De "aanvalsgraaf" in figuur 2, gemaakt door Rianne Kaptein, is daarvan een mooi voorbeeld. De visualisatie is gemaakt met Prefuse. Het invoer XML-bestand voor Prefuse is met XSLT 2.0 uit de XML-versie van het verslag van die dag gemaakt. Dat verslag in XML is beschikbaar op http://staff.science.uva.nl/~marx/politicalmashup/AB2008/AB2008.xml.

In de graaf is elke fractievoorzitter aangegeven als een knoop waaruit pijlen van 1 kleur vertrekken.



Figuur 2: Visualisatie van de interrupties tijdens de Algemene Beschouwingen in de Tweede Kamer op 17 september 2008.

Een pijl van persoon A naar persoon B betekent dat persoon A persoon B heeft onderbroken tijdens B's betoog op het spreekgestoelte. De dikte van de pijl wordt bepaald door het aantal interrupties. De kleuren zijn willekeurig gekozen en betekenen niets. Ze zijn alleen gebruikt om het diagram beter leesbaar te maken. Voorbeeld: Agnes Kant heeft twee personen aangevallen: Van Geel en Hamer. Dit zijn de twee dikste pijlen in het diagram. Kant heeft Van Geel 18 keer onderbroken en Hamer zelfs 24 keer. Er wijzen pijlen naar Kant vanuit Van Geel, Hamer, Halsema, Wilders en Pechtold. Kant is dus door deze vijf mensen geïnterrumpeerd. De dikste is vanuit Hamer die Kant 10 keer interrumpeert.

Kijkje onder de motorkap

Het stukje XSLT 2.0 code verderop in figuur 3 bouwt tabel 1 op, waarop de graaf in figuur 2 gebaseerd is. Om de code te begrijpen is enig inzicht in de structuur van de XML noodzakelijk. De verslagen van de vergaderingen van het parlement hebben een mooie hiërarchische structuur die als volgt te vangen is:

```
Notulen          → (onderwerp)+
onderwerp        → (blok)+
blok             → (spreker | stage-direction)+
spreker          → (p | stage-direction)+
p                → (#PCDATA | stage-direction)*
stage-direction  → (#PCDATA).
```

Figuur 1 bevat een voorbeeld van het element `spreker`. Een blok geeft een deel van het debat aan waarop één spreker achter het spreekgestoelte staat. De eerste spreker binnen een blok is altijd diegene achter het spreekgestoelte. De andere sprekers binnen dat blok interrumperen. Dit verklaart de logica van het XSLT-script. Het `stage-direction` element wordt in Nederland erg weinig gebruikt, maar in Duitsland en Vlaanderen des te meer. Voorbeelden van informatie hierin zijn *Gelach bij de SP, Geroffel op de bankjes* en *Heiterkeit.*

Conclusie

Er liggen veel mooie, zowel tekst- als datagebaseerde documenten te wachten om omgezet te worden naar XML om zo echt geopenbaard te worden. Daarnaast is de besproken technologie natuurlijk ook goed toepasbaar op nieuwe documenten. Hier is het de moeite waard om aan het begin van het publicatieproces een "XML-aftakking" in te bouwen naar een simpele, vooral semantisch georiënteerde DTD. Vaak kan dat onopgemerkt plaatsvinden. Hiermee kunnen niet alleen heel makkelijk mooie grafieken voor een jaarverslag gemaakt worden, het kan ook het begin zijn van een echt handig ontsloten archief.

Dr Maarten Marx is docent bij het Informatica Instituut van de Universiteit van Amsterdam. Met het hier beschreven onderzoek won hij de XML Award 2008. Zie ook: http://www.xmlholland2008.nl/programma/Meteen+naar+de+juiste+plek.aspx.

Wie interrumpeert wie? Input tabel voor de interruptiegraaf

Spreekgestoelte	Kant	Van Geel	Rutte	Hamer	Wilders	Slob	Halsema	Pechtold	Thieme	Van der Vlies	Verdonk
Kant		5	-	10	3	-	1	3	-	-	-
Van Geel	18		10	-	4	-	8	10	8	4	4
Rutte	-	14		17	-	9	13	-	4	-	-
Hamer	24	-	4		-	-	6	20	-	2	7
Wilders	-	5	-	3		2	11	6	-	-	-
Slob	-	-	-	-	-		-	10	-	-	4
Halsema	-	-	-	-	-	2		2	-	-	-
Pechtold	-	-	-	4	-	5	3		-	-	-
Thieme	-	-	-	-	-	-	1	-		-	-
Van der Vlies	-	-	-	-	-	-	-	1	3		-
Verdonk	-	-	-	-	-	-	-	3	-	-	

Tabel 1: Inputtabel voor interruptiegraaf uit figuur 2

```
<xsl:variable name ="kamerlidsprekers"
  select ="distinct-values for
       $b in $onderwerp//blok return
       $b //spreker[1] [@partij]/@naam)"/>
  <h3>Wie interrumpeert wie? Inputtabel voor de interruptiegraaf</h3>
<table valign="top" border='1' style="display: inline-table">
       <!-- header-->
       <tr>
          <th>Spreekgestoelte</th>
          <xsl:for-each select="$kamerlidsprekers">
             <th>
               <xsl:value-of select="."/>
             </th>
          </xsl:for-each>
       </tr>
       <!-- one row for each speaker -->
       <xsl:for-each select="$onderwerp//blok[.//spreker]">
          <xsl:variable name="thisblok" select="."/>
          <xsl:variable name="thisspreker" select="$thisblok//spreker[1]/@naam"/>
          <tr>
             <td align="center">
                <strong><xsl:value-of select="$thisspreker"/></strong>
             </td>
             <xsl:for-each select="$kamerlidsprekers">
                <xsl:variable name="interruptor" select="."/>
                <xsl:variable name='SprekerCount'
                    select="count($thisblok//spreker[@naam=$interruptor])"/>
                <td align="center">
                    <xsl:value-of select="if ($interrupt r=$thisspreker)
                       then ''
                       else
                          if ($SprekerCount=0
                          then '-'
                         else $SprekerCount"/>
                 </td>
          </xsl:for-each>
          </tr>
       </xsl:for-each>
</table>
```

Figuur 3: XSLT 2.0 code voor het omzetten van de informatie uit de handelingen tot tabel 1. Aangenomen is dat `$onderwerp` *een deep-copy van één element onderwerp bevat.*