Wat is er mis met XML?

Marc de Graauw
eerder gepubliceerd in <!ELEMENT Jaargang 14, nummer 2, juli 2008

Was XML maar dood. Dan hadden we de kans gehad iets beters ervoor in de plaats te creëren. Helaas is XML inmiddels diep verweven in de hele ICT wereld, en moeten we allemaal leven met de diepbedroevende reeks fouten en blunders die in de familie van XML specificaties zitten.

Han Schouten is zo vriendelijk geweest mijn bescheiden verslag van het XML 2007 congres getiteld "XML is dood"[1]. te bespreken. Han concludeert "Marc lijkt met het badwater ook het kindje te willen wegspoelen". Op een punt snijdt de kritiek van Han echter wel degelijk hout: "Marc ... beweert dingen die hij op geen enkele wijze onderbouwt" en "Marc citeert vooral anderen...". (Het was ook een congresverslag, toch?)

Er volgt ook een uitdaging: "Tenzij je een hekel hebt aan punthaakjes, is er in feite *niets* mis met XML", "misschien is XML gewoon wel grotendeels af" en "Geef mij een concreet voorbeeld!". Vooruit. Daar gaan we. Wat is er mis met XML?

1. XML

De "kale" XML 1.0 specificatie zelf is heel redelijk – "mostly harmless", zouden we kunnen zeggen. Toch kunnen heel basale dingen niet met XML: XML in XML inpakken en binaire data in XML inpakken. Als we dat toch proberen, lopen we het risico dat het zaakje in elkaar stort door conflicterende encodings, ongeldige tekens, dubbele XML declarations of dubbele ID's. Dat dwingt ons om wanneer we iets willen doen wat kinderlijk eenvoudig zou moeten zijn: een plaatje of een XML document in een XML document stoppen we gedwongen worden een inefficiënte (converteren naar Base64), krakkemikkige (Soap With Attachments) of complexe (MTOM) oplossing te gebruiken. XML is hier dus veel te simpel.

2. Zoek de Namespace

Bij de volgende spec – Namespaces – wordt het erger. Namespaces zijn in theorie optioneel, in de praktijk niet. Wellicht als je XML alleen in de eigen keuken gebruikt, met zelfgebakken schema's en eigenbereide stylesheets, dat je ervoor kunt kiezen Namespaces links te laten liggen. Wanneer je – zoals ik – in een serieuze omgeving werkt waar gegevens uitgewisseld worden tussen verschillende organisaties, en je niet alle wielen zelf opnieuw uit gaat vinden, maar aansluit bij bestaande vocabulaires als UBL[2] (zaken) of HL7v3[3] (gezondheidszorg), zijn Namespaces – helaas – een noodzakelijk kwaad. Ik moet er niet aan denken hoeveel vocabulaires niet <name> of <id> definiëren, en zou niet weten hoe dit zonder Namespaces op te lossen. Namespaces dus.

Namespaces zijn werkelijk een van de slechtst mogelijke specs die de XML familie kent. Na 10 jaar is de XML gemeenschap er nog steeds niet uit of de URI die een Namespace definieert naar een resource moet wijzen (b.v. een Schema of RDF bestand, of een HTML pagina) die iets zinnigs over die Namespace zegt of niet. Verwarring alom bij de (nieuwe) gebruiker, iedereen associeert een URI immers met een webpagina. Na deze hobbel is er de verkorte notatie – prefix – van een Namespace. Er is nooit een poging gedaan deze te registreren, zoals bijvoorbeeld Media Types. Gevolg: de prefix van XML Schema kan (en deze komen allemaal regelmatig voor) xs: zijn, of xsd:, of wxs: of zelfs iets zelfbedachts als kul:, en erger, we kunnen prima xs: gebruiken als prefix voor XSLT elementen. Iedere programmeur kan dus niets zinnigs zeggen over een element, zonder eerst de prefix op te zoeken bij de namespace declaratie, daar de hele betreffende URI te lezen (wie weet de URI van XML Schema uit het hoofd?) en te kijken of het nu écht XML Schema is waar het over gaat.

---

[1] <!ELEMENT, Jaargang 14, nummer 1, april 2008
[2] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl
[3] http://www.hl7.org/v3ballot/html/welcome/environment/index.htm

"Tim and I were both on the old W3C XML Working Group that designed the Namespaces spec -- in retrospect, we got too far in front of implementors' requirements and delivered a spec to solve problems someone might have some day in the future, instead of problems people actually had at the time."[4]

David Megginson

Alsof deze *mess* nog niet erg genoeg is, is het mogelijk namespaces overal in een document te declareren. Zeker als een default Namespace gebruikt wordt (dus een Namespace zonder prefix) is het zoeken naar een Namespace-declaratie in een hooiberg. En zeg eens eerlijk – wie weet nu op het eerste gezicht of een attribuut in een Namespace valt of niet? Ooit een XPath of XSLT fragment ingetikt, nul nodes als resultaat gekregen en na heel lang zoeken er achter gekomen dat er ergens een namespace te weinig of verkeerd gedeclareerd stond?

Sommigen zullen zonder twijfel van mening zijn dat dit weliswaar allemaal kan, maar dat dit het resultaat is van een slecht documentontwerp, en dat bij een goed ontwerp prima in orde is. Een goede spec nodigt ons uit om de dingen op een heldere, leesbare wijze op te zetten. Namespaces nodigt ons uit er een zooitje van te maken, en is dus geen goede spec.

Het wordt echter nog erger. Kijk naar het volgende fragment:

```
<myns:greeting>Hello world</myns:greeting>
```

Prima. Dat hebben we dus nodig. Als "myns" goed gedeclareerd is, niets mis mee. Maar nu het volgende:

```
<element name="purchaseOrder" type="po:PurchaseOrderType"/>
```

Iedere programmeur, ongeacht welke achtergrond, heeft bij de tekst "po:PurchaseOrderType"[5] dezelfde intuïtie: Ha! Dit staat tussen aanhalingstekens! Het is een literal – ik, als programmeur, hoef me niet te bemoeien met de inhoud hiervan. Helaas niet zo bij namespaces. Tegen alle conventies in is wat er tussen aanhalingstekens staat, significant. Het kan nog erger[6]:

```
<faultcode>SOAP-ENV:MustUnderstand</faultcode>
```

Jawel, de inhoud van een element is gekwalificeerd met een namespace! Hoe moet je dan weten of "SOAP-ENV" deel is van de platte tekst die men over wil brengen, of dat het een Namespace prefix is, en geen deel van de tekst? Mag "SOAP-ENV" dan dus niet in de tekst voorkomen? Antwoord: dat kun je niet weten. Giswerk. En dat alleen omdat de Namespace spec onvoldoende duidelijk is over wat nu precies wel en niet met een Namespace gekwalificeerd kan worden.

3. Schema? What Schema!

Laten we Namespaces maar even rusten. Verder met XML Schema. Kijk eens naar het volgende fragment, in RelaxNG Compact Notation:

```
element addressBook {
  element card { attribute contact_type { "business" | "personal"  } ,
    (
      ( element name { text }, element email { text } )
```

[4]

[5] XML Schema Part 0: Primer, http://www.w3.org/TR/xmlschema-0/

[6] Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/2000/NOTE-SOAP-20000508

```
      |
      ( element name { text }, element phone { text } ) ) ,
      element adres { text } ? ,
      element * { text } ?
  }*
}
```

Glashelder, niet? Een adreskaart bevat een contacttype, een naam en een emailadres of telefoonnummer, optioneel een adres, en mogelijk extra elementen. In XML Schema wordt exact ditzelfde fragment:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema                                    xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="addressBook">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="card">
          <xs:complexType>
            <xs:sequence>
              <xs:choice>
                <xs:sequence>
                  <xs:element name="name" type="xs:string"/>
                  <xs:element name="email" type="xs:string"/>
                </xs:sequence>
                <xs:sequence>
                  <xs:element name="name" type="xs:string"/>
                  <xs:element name="phone" type="xs:string"/>
                </xs:sequence>
              </xs:choice>
              <xs:element minOccurs="0" name="adres" type="xs:string"/>
              <xs:any processContents="skip"/>
            </xs:sequence>
            <xs:attribute name="contact_type" use="required">
              <xs:simpleType>
                <xs:restriction base="xs:token">
                  <xs:enumeration value="business"/>
                  <xs:enumeration value="personal"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

U ziet het goed. Zelfs een eenvoudig schemaatje wordt een enorme lap tekst. Deze is wellicht nog te overzien: maar echte Schema's bijvoorbeeld die uit de eerder genoemde UBL of HL7v3 zijn alleen voor de echte ingewijde leesbaar zonder speciale tooling. Daar gaat dus het uniforme kennismodelleringsinstrument: in plaats van een leesbaar Schema zijn we weer afhankelijk van een tool.

"...it's time to stop sweeping it under the rug. W3C XML Schemas (XSD) suck. They are hard to read, hard to write, hard to understand, have interoperability problems, and are unable to describe lots of things you want to do all the time in XML."[7]

[7] http://www.tbray.org/ongoing/When/200x/2006/11/27/Choose-Relax

Tim Bray

Maar het wordt erger: het bovenstaande XML Schema fragment mag helemaal niet. Een simpele constructie als: een adreskaart bevat naam en email, of naam en telefoonnummer kan helemaal niet in XML Schema uitgedrukt worden! (Je kunt schrijven: naam en (email of telefoonnummer), maar dat werkt niet meer bij langere constructies). Dit is geen theoretische beperking: bij een poging een XML Schema te maken voor een klant die een XML-formaat gebruikte dat stamde van voor de tijd van XML Schema, bleek dit bestaande XML formaat helemaal niet in XML Schema uit te drukken te zijn!

Ook de laatste constructie in het voorbeeld: na het adres mogen andere, niet nader gedefinieerde elementen (<xs:any>) volgen, is illegaal! Omdat het adres optioneel is, "weet" XML Schema niet meer of een adres in een document het "adres" is, of een deel van de onbekende content! Deze gevreesde regel, de "Unique Particle Attribution" heeft er toe geleid dat bij het eerder genoemde UBL voorstellen om voorwaartse compatibiliteit van berichten te ondersteunen met XML Schema zijn verlaten. Nu volgt UBL de veel eenvoudigere lijn van ondersteuning hiervan met XSLT. Tenslotte zijn er nog de "co-occurrence constraints": een eenvoudige regel als "het contact_type 'business' is, moet het emailadres gevuld zijn" is helemaal niet uit te drukken in XML Schema.

En denk niet dat het alleen de critici van XML Schema zijn die dit als feilen beschouwen: in XML Schema 1.1 worden precies deze zaken veranderd. Aangezien XML Schema 1.1 nog volop in ontwikkeling is mogen we alleen maar hopen dat dit soort essentiële feilen nog binnen 10 jaar na het verschijnen van XML Schema 1.0 daadwerkelijk gerepareerd zijn... Er zijn uiteraard alternatieven: RelaxNG[8] (een ISO standaard) en Schematron[9] (dito). Jammer genoeg werken de meeste tools, bijvoorbeeld die van Microsoft, en de Web Services specificaties nagenoeg exclusief met XML Schema. Hinken op krukken, dus.

4. Wilt u hier maar even tekenen?...

Maar weer even terug naar XML zelf? Een van de basale problemen met XML is natuurlijk de mogelijkheid een stukje tekst op te slaan in een attribuut of een element:

```
<auteur>
    <naam>Marc</naam>
</auteur>
```

```
<auteur naam="Marc"/>
```

Met de energie die het XML-wijze deel van de wereld al gestoken heeft in de discussie welke van de twee bovenstaande voorbeelden "beter" is hadden we heel wat ernstigere problemen op kunnen lossen. Maar het probleem is niet alleen elementen-versus-attributen. XML zit propvol met dergelijke redundante, overbodige verdubbelingen. Kijk eens naar:

```
<mdg:auteur naam="Marc" xmlns:mdg="http://www.marcdegraauw.com/voorbeeld/"/>
```

versus:

```
<auteur xmlns="http://www.marcdegraauw.com/voorbeeld/" naam="Marc"></auteur>
```

Volgens de XML theorie zijn deze twee regels volkomen gelijk. De expliciete eind-tag is gelijk aan de impliciete, de volgorde van attributen is niet significant in XML en een default namespace heeft

[8] http://relaxng.org/
[9] http://www.schematron.com/

hetzelfde effect als een met prefixes gedeclareerde. Prima toch? Vrijheid, blijheid, iedereen mag doen wat 'ie wil, en het is toch precies hetzelfde?

Jazeker, totdat je het document wilt voorzien van een digitale handtekening. De eerste string octets is heel anders dan de tweede. Wanneer je dus het eerste voorbeeld tekent, met de handtekening naar een ontvanger stuurt, wiens XML programmatuur er vrolijk het tweede van maakt (dat mag ook, volgens de XML theorie) *is de handtekening niet meer geldig*.

> "XML is an inherently unstable and therefore unsignable data format."[10]
>
> Peter Gutmann

Er is een oplossing voor bedacht: XML Canonicalization. Daarmee wordt ieder XML document eerst in een canonieke vorm gebracht, waarna de handtekening wel valideert. Maar eerlijk, XML Canonicalization is eerder een hack dan een oplossing, en een hack die in de eerste plaats compleet overbodig had moeten zijn. XML Canonicalization is een bekende en vaak desastreuze performance bottleneck voor applicaties die handtekeningen nodig hebben, tot 96% (!) van de tijd die nodig is om de DOM te lezen, canoniek te maken en de handtekening te berekenen[11].

Wie kent een ander dataformaat dat zoveel redundantie kent als XML? Dit is een van de fundamentele problemen van XML. De SGML-erfenis maakt het natuurlijk niet makkelijk, maar de kern van het probleem is dat XML *teveel wil zijn voor iedereen*. De "one size fits all" benadering waarmee XML zowel documentgeöriënteerde problemen als datageöriënteerde problemen op wil lossen, is de reden dat XML vaak zo'n kromme oplossing is.

5. DOM, DOMmer, DOMmest

Het wordt nog erger bij de DOM en programmeren met XML.

> "I wrote a bunch of code to process arbitrary incoming XML, and I found it irritating, time-consuming, and error-prone."[12]
>
> Tim Bray

Het begint natuurlijk met het simpele voorbeeld, wat iedereen die ooit met de DOM geprogrammeerd heeft zal herkennen:

```
<doc>
    <naam>piet</naam>
</doc>
```

Wanneer we met de DOM het eerste element van <doc> ophalen en afdrukken, zien we niets – het fragment kent immers *drie* nodes onder <doc>: newline plus tab, <naam>, newline. Het eerste "child" is dus een witregel, niet de <naam>. Hoewel dit voorbeeld uitermate simpel is, geeft het het wezenlijke probleem van de DOM goed weer: de DOM werkt voor nieuwe programmeurs *tegenintuïtief*.

> "And of all the things XML has brought to the table, about the only one I can think of that is worse than namespaces is DOM. It's ugly, inconsistent, memory-intensive, slow, thoroughly despised by users, and frankly just hideous."[13]

[10] http://www.cs.auckland.ac.nz/~pgut001/pubs/xmlsec.txt
[11] http://www.w3.org/2007/xmlsec/ws/papers/06-zhang-ximpleware/
[12] http://www.tbray.org/ongoing/When/200x/2003/03/16/XML-Prog
[13] http://lists.w3.org/Archives/Public/www-tag/2008Jun/0020.html

Elliotte Rusty Harold

Hieronder ligt een veel fundamenteler probleem. Programmeurs werken (uiteraard) graag met objecten uit de eigen programmeertaal. (En dat is meestal een objectgeöriënteerde taal. Van de 10 meestgebruikte programmeertalen, zijn er 7 ofwel objectgeöriënteerd, ofwel voorzien van uitgebreide objectgeöriënteerde uitbreidingen.[14]) XML, en met name de DOM, passen slecht op objecten. Meestal is een hele slag nodig om met veel getElementsByTagName statements de XML boom in te lezen in lokale objecten. Precies de reden waarom programmeren met de DOM steeds meer vervangen wordt door XML API's die goed passen bij het lokale idioom. Het populaire ElementTree[15] bijvoorbeeld maakt het eenvoudig een XML boom in te lezen en te behandelen als typische Python structuren: dictionaries en lists. Dat maakt het bewerken van de XML voor de Python-programmeur tot een veel natuurlijker, "Pythonesque", proces dan werken met de DOM. Ook de Microsoft-taal LINQ[16] die inmiddels deels in het .NET Framework 3.5 zit, had als een van de motivatoren een eenvoudiger manier om met XML te werken.

6. Over JSON en struisvogelpolitiek

Op precies dit pijnpunt richt JSON[17] (JavaScript Object Notation) zich ook: JSON documenten zijn in wezen geserializeerde JavaScript objecten – hoewel er op zich heel weinig Javascript specifieks aan JSON is, JSON past heel makkelijk op andere programmeertalen.

Een voorbeeld (ontleend aan Simon Willison)[18]:

```
person = {
  "name": "Simon Willison",
  "age": 25,
  "height": 1.68,
  "urls": [
    "http://simonwillison.net/",
    "http://www.flickr.com/photos/simon/",
    "http://simon.incutio.com/"
  ]
}
```

Dit heeft in al zijn eenvoud geen uitleg nodig. Dit past precies op een object, maakt niet uit of dat in C++, Java, C#, Python of Perl moet gebeuren. JSON kent niet de desastreuze redundantie van XML. JSON is korter (zet het bovenstaande voorbeeld maar een om in XML en tel de bytes...). JSON past natuurlijk op programma-objecten. En deze eenvoud is heel belangrijk. Uiteraard kan iemand met de nodige informatieanalytische achtergrond en de nodige programmeertalen in de rugzak met XML wel een oplossing bouwen. Maar het probleem is dat *er te weinig van zijn*. Er is simpelweg onvoldoende IT-personeel om ieder project te bemannen met alleen de hoost gekwalificeerde medewerkers. Daarnaast is XML niet alleen een taal voor hardcore ontwikkelaars. Ook webontwerpers, met een achtergrond als grafisch ontwerper, programmeren er vaak wat bij in Javascript. Daarom is het zo belangrijk dat een taal, bijvoorbeeld XML, het leven van de programmeur simpel maakt. XML doet dat niet.

[14] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
[15] http://effbot.org/zone/element-index.htm
[16] http://en.wikipedia.org/wiki/Language_Integrated_Query
[17] http://www.json.org/
[18] http://simonwillison.net/2006/Dec/20/json/

JSON is uiteraard ook niet de oplossing voor alles. JSON heeft (nog) geen schemataal – dat maakt het bij voorbaat ongeschikt voor heel veel grootschalige problemen – zoals gegevensuitwisseling in de gezondheidszorg, mijn vakgebied.

Maar we moeten de kritiek van Douglas Crockford en JSON serieus nemen. De jippie-mentaliteit waarmee velen XML omarmen getuigt van een wereldvreemde navelstaarderij. XML zit boordevol problemen. Ernstige, serieuze problemen, waar de opkomst van JSON een teken van is. Wanneer de reactie op kritiek is de kop in het zand te steken, en de problemen te ontkennen, is XML binnenkort echt dood. Het grote manco van de afgelopen 10 jaar dat we XML hebben, is dat alle problemen die hierboven genoemd zijn, al het grootste deel van die 10 jaar bekend zijn, en dat er 10 jaar lang niets of nauwelijks iets mee gedaan is.

XML is niet dood als documentformaat – de wereld zit propvol met XML documenten en berichten, en het worden er iedere dag meer en meer. XML is wel dood als veelbelovende technologie, als familie van specs die zich ontwikkelt naar de toekomst, en zich verbetert zonder zich onnodig te compliceren.

Passed on! XML is no more! It has ceased to be! It's expired and gone to meet 'is maker! It's a stiff! Bereft of life, it rests in peace! If you hadn't nailed it to the perch it'd be pushing up the daisies! It's metabolic processes are now history! It's off the twig! It's kicked the bucket, it's shuffled off it's mortal coil, run down the curtain and joined the bleedin' choir invisibile!! Ex-ML!![19]

[19] http://www.youtube.com/watch?v=hTSAFcLXqYY