What's in a name? In het geval van Java API's voor XML niet bijzonder veel. Het volstaat om een door gebrek aan inspiratie bijna mysterieuze prefix 'JAX' te vervolledigen met zowat elke letter uit het Latijnse alfabet. Met dit recept vult Sun tegenwoordig de Java libraries aan met de regelmaat van de klok. Op zoek naar de zin en onzin van deze nieuwe acroniemen voor de Java ontwikkelaar begeleidt Marc Portier, Outerthought, ons in dit tweede artikel uit de JaxPack reeks naar de Java API's voor het gebruik van SOAP.

JAX-M - SOAP voor Javanen

De JaxPack - Java API's voor verwerking van XML (2)

Er zal ongetwijfeld een goede verklaring zijn (minstens een elfseptemberexcuus[1]) maar de betrokken leveranciers pakken niet echt uit met de verkoopresultaten van de boeiende reeks aan XML messaging- en applicatie-integratie producten die ze het laatste jaar op de markt hebben gebracht. (MS Biztalk server, SilverStream Extend Composer, Webmethods…) Nochtans staat de onderliggende technische redenering als een huis. XML Messaging realiseert (meer dan beloven dus) de ideale 'losse koppeling' die voor oplossing van de problematiek rond de applicatie-integratie broodnodig is. De ontkoppeling die XML messaging meebrengt is tweevoudig. Enerzijds breekt het de noodzaak tot gezamenlijke beschikbaarheid voor de aanvrager en de leverancier van een bepaalde elektronische dienst (ontkoppeling in beheer). Het meer klassieke vraag-antwoord ('request-response') model wordt vervangen door het 'asynchroon' model wat genoegzaam bekend is van message-oriented middleware (MoM) producten als MQSeries en MSMQ. Anderzijds laat het ook toe -gesteund door de XML component- dat technisch zeer diverse platformen met elkaar XML boodschappen gaan uitwisselen (ontkoppeling van technisch platform). Droom maar hardop: een eigenhandig geschreven VBA functie in MS-Excel haalt over Internet gegevens op van een embedded Linux systeem waar bijvoorbeeld een leuke Perl-Python oplossing draait. Die laatste levert uiteraard de gevraagde gegevens even netjes aan een Java-MIDP applicatie op een palmtop[2].

XML Messaging Voor de ontwikkeling van dergelijke boeiende toepassingen over heterogene systemen heen worden we dus minder gedwongen om dat te gaan doen in Java. Anders gesteld: de XML fundering van webservices maakt Java minder prominent als 'de facto' oplossing voor het cross platform probleem. Ontwikkel je dit soort dienst toch in Java dan merk je snel dat het huidige Javaplatform je een aanzienlijk gevulde gereedschapskist ter beschikking stelt. Enigszins classificerend komen we op volgende drie mogelijke aanpakken van XML Messaging met behulp van Java:

> Webservices: het verpakken van je inputparameters en je returnwaarden in een XML kleedje

- De 'rol-ze-zelf' aanpak met behulp van JAXP (mogelijk JAXB[3]) gecombineerd met wat je maar wilt uit java.net.*
- De slimme herbruiker die terugvalt op XML messages over JMS.
- De open standaard freak die SOAP via JAXM of JAXRPC verkiest.

Rol ze zelf Hoewel de laatste keuze voor standaarden z'n natuurlijke toegevoegde waarde heeft, hoeft het misschien niet altijd zo complex te zijn. Het is vast

1 Elfseptemberexcuus verdient als zelfstandig naamwoord zijn plaats in de Van Dale. Allicht gaat het dan iets betekenen als een even eenvoudige als algemeen geldende oorzaak voor zoveel onheil dat alleen al het nadenken over het mogelijk valideren ervan zonder meer als immoreel wordt afgedaan.

2 MIDP (Mobile Information Device Profile, zie ttp://java.sun.com/products/midp/) is een set van Java API's voor de ontwikkeling van applicaties voor wireless en palmtop toestellen.

3 JAXB werd behandeld in het eerste artikel uit deze reeks. (verschenen in Software Release Magazine nr. 8, december 2001)

een ontnuchtering voor een ieder die in deze branche nog graag eens op de next-hype-wave meesurft, maar ook deze keer zijn er geen fundamentele nieuwe gedachten of uitvindingen. Geen wereldschokkend nieuwe ideeën of inzichten dus die de 'webservices' mogelijk moeten maken. Het gaat eerder om een logisch samenvoegen van een aantal stromingen binnen het huidige webdevelopment. Of zo je wilt, de formele neerslag (men doet de moeite standaards neer te leggen) van het 'voortschrijdend inzicht' dat leeft in de industrie. In het beste geval kun je de evolutie naar webservices nog het best enige luiheid toedichten. Na het invoeren van de XML/XSLT aanpak voor websites, ingegeven door de luie reflex om niet op elke pagina (ook al is het een asp of jsp) de huisstijl te moeten toepassen (en om de zoveel tijd wijzigen) komen we er nu op uit dat die hele XSLT zwik toch ook niet zó makkelijk is en worden plots vele voordelen gevonden in het doorsturen van de naakte XML. Want ontdaan van zijn standaardiseringsjasje gaat het in het hele webservices verhaal slechts hierover: het verpakken van je inputparameters en je returnwaarden in een XML kleedje.

Iedereen die dus op een blauwe maandag z'n eerste PHP-, ASP-, JSP- of aanverwante stappen heeft gezet, kan eigenlijk gelijk aan zijn eerste webservice beginnen. Je mikt alle HTML tags uit je code, en je introduceert een (beperkte) set van zinvolle XML markup die je datastructuur weet te communiceren. Voor de communicatie van de gebeurlijke input parameters kiezen we dan om die te verpakken in het klassieke name=value URL formaat van de gekende HTTP 'GET' of 'POST' boodschappen. Het grote voordeel hiervan is dat je webservice ook met een zeer eenvoudige HTML form kan getest worden. Gevolg gevend aan de natuurlijke esthetiek van de symmetrie zal een weinig meer complexe multipart-post (zie RFC 1867) de XML zowel in als uit ondersteunen. Voor de echte doeners: het Java Activation Framework (JAF, http://java.sun.com/products/javabeans/glasgow/jaf.html) dat ook de MIME handling voor de JavaMail API implementeert, levert de nodige API's en implementaties om dat soort messages op een vlotte manier terug uit te pakken. Het grootste voordeel van de symmetrische aanpak is alvast dat het op een vlottere manier toelaat verschillende webservices te laten samenwerken. De ene kan de andere aanroepen, en in dat proces kunnen ze focussen op het interpreteren en genereren van nette XML. De decodering van alle XML-wereld-vreemde URL escaping verdwijnt dan naar de geschiedenisboekjes.

MULTI-DEVICE CONSUMPTIE Sinds de 'Servlet - Filters' (beschikbaar sinds de nieuwe Servlet 2.3 specificatie) beschikken we trouwens over de ideale gastheren voor deze gescheiden verantwoordelijkheid. Een zeer eenvoudig filter kan de URL-escaped input variabelen van de HTML form in een klein XML stroompje verpakken dat naar de echte XML verwerkende servlet (de eigen niet-standaard webservice) wordt gestuurd. Eenmaal deze keuze gemaakt, kan ons filter ook op de uitgaande weg de geretourneerde XML nog behandelen. Via een XSLT proces genereert die dan de eindgebruikersvriendelijke HTML. Het zo gerealiseerde patroon dient zowat de basis van de toekomstige webapplicatie te zijn: ze scheidt de verantwoordelijkheden die klassiek op onze webtiers leven: 'HTML formattering', 'Form behandeling' en het daarvan te scheiden: 'Via HTTP Wereldwijd beschikbaar stellen van de applicatie'. In een webservices context waar machines in plaats van mensen de eindgebruikers van je dienst zien, kan het eerste deelaspect achterwege gelaten door in ons patroon de filter te bypassen. Uiteraard, zijn we door het inpluggen van een andere filter, die de XML output anders formatteert, weer wat beter gewapend in (immer?) afwachting van doorbrekende multi-device consumptie.

> Net als alle zuster API's in de J2EE bundel is de JMS een zogenaamde 'insulation' API

Meteen merken we dat er ook een positieve keuze kan zijn om deze eerste strategie aan te hangen. Door onze ziel niet aan SOAP te verkopen hoeven onze XSLT processen geen rekening te houden met de eigenheden die de SOAP envelope met zich meebrengt. Al moet dat door de rigoureuze namespace scheidingen in de SOAP spec eigenlijk vooral een oefening voor de XSLT-klas worden. Veel belangrijker argumenten worden dan: de behouden controle over het gehele proces, en de kans om net iets beter performance te tunen. De logica kan ook die van de ongebondenheid zijn. Wat dat betreft is de SOAP envelope al even vreemd aan onze applicatie als het met de http/html zat. Het behandelen of verpakken van SOAP boodschappen is dan ook een aspect dat los staat van het 'ontkoppeld (XML over HTTP) aanbieden' van je dienst. En het leuke van standaards blijft dan ook 'dat er zoveel zijn om uit te kiezen'.

> SOAP is populair en zal dat vast blijven omdat het zo weinig om het lijf heeft

De totale ongebondenheid hoeven we alvast niet na te streven, ook in deze 'eigen' aanpak kunnen we nog even in de standaardenwinkel langs: XSLT is al aangehaald, de andere gratuite aanrader is dan het gebruik van DTD of

XML Schema om de blauwdrukjes van de geproduceerd en verwachte in- en output documenten te vangen. Teruggrijpend naar deel één van deze reeks artikelen (JAXB) kunnen we ook tijdens development ons voordeel halen uit dit soort contracten tussen zender en ontvanger. Het JAXB codegeneratie proces kan uitgaande van de aangeleverde schema's de nodige Java classes genereren die automatisch en op zeer performante wijze van en naar XML representaties kunnen worden omgezet.

JMS OP VISITE De tot hier besproken oplossing zweert natuurlijk wel nog bij een request-response architectuur. Om ook de aangehaalde 'ontkoppeling in beheer' te kunnen realiseren dringt een asynchroon (messaging) framework zich op. De MoM ondersteuning is al een tijd geleden de Java wereld binnengeslopen onder de vorm van de eerste JMS specificatie. Pas sinds de komst van de message driven beans in de laatste update van de overkoepelende J2EE specificatie is het ook eindelijk voldoende in de spotlights gekomen. Een effect dat op natuurlijke wijze versterkt wordt door de komst van JMS implementaties afkomstig van de grote marktspelers in de MoM wereld: Tibco, MQSeries... Net als alle zuster API's in de J2EE bundel is de JMS een zogenaamde 'insulation' API. Ze schermt het gebruik van om het even welke (nou ja…JMS compliant dan) 'messaging' middleware af van de specifieke implementatie. Dezelfde applicatie kan dus met een Ti-Ta-Tovenaar handenklap van de ene implementatie naar de andere worden geporteerd. Het stilzetten van de tijd is ook hier bijzonder handig.

NODELOZE OVERHEAD Centraal in de JMS API staat de 'message' welke zowat dienst doet als de eenheid van communicatie, ze worden naar een 'topic' of een 'queue' gezonden en daar opgepikt door de geregistreerde 'listeners' of 'consumers'. De inhoud (ook wel payload) van de 'message' wordt door een aantal gespecialiseerde versies (subclasses) eenvoudig toegankelijk gemaakt: StreamMessage en ByteMessage voor rauwe, binaire data, ObjectMessage voor geserialiseerde objecten en verder het vaak gebruikte MapMessage voor heldere naam-waarde tabellen. Tenslotte is er het zeer vrije TextMessage voor leesbare, vaak specifiek gecodeerde tekst. Die laatste is natuurlijk de ideale gastheer voor XML boodschappen. In afwachting dat de API er ook expliciet een subclasse (bijvoorbeeld XMLMessage) voor voorziet is de eenvoudigste techniek alvast je XML output naar een StringOutputBuffer te schrijven en daarna de zo bekomen String als 'text' af te geven aan de TextMessage. Aan de andere zijde zal die inhoud verpakt in een StringInputStream de ideale input zijn voor je XML parser. Dit voor eens en altijd in je eigen 'mypackage.XMLMessage' subclasse (van TextMessage) coderen is een fijne oefening die bij de leidende implementaties in JMS land dan ook al voor je gebeurd is. De meeste ervan hebben trouwens in dit verhaal zó kort op de bal gespeeld dat hun implementatie dateert van de pre-JAXP API. Het gevolg daarvan is dat je eigen applicatie zowat verplicht wordt dezelfde parser te gebruiken als die bijgeleverd in hun oplossing. Verder dient het ook opgemerkt dat elk van deze implementaties een honderd procent DOM aanpak neemt om de XML boodschap te construeren en ontrafelen. Voor het gros van de asynchrone applicaties die met XML hun voordeel gaan doen is dit aanvaardbaar. Soms is de DOM tussenstap echter nodeloze overhead. Als de XML boodschap rechtstreeks in een voor SAX geoptimaliseerde XSLT engine wordt binnengeloodst bijvoorbeeld. In die gevallen vallen we natuurlijk makkelijk terug op de eerder besproken techniek om op het niveau van TextMessage de XMLString op te vissen en daarna zelf te gaan behandelen. Ook de recyclage van het JAXB idee wordt dan weer een optie.

> De applicatie kan met een Ti-Ta-Tovenaar handenklap worden geporteerd

Tenslotte is het belangrijk op te merken dat de eerder vermelde 'onafhankelijkheid van de implementatie' bij JMS niet zonder meer een zegen is. De MoM wereld mist op dit moment nog een standaardisering omtrent het zogenaamde 'wire protocol'. Vermoedelijk ervaren alle messaging vendors de aard en inhoud van hun boodschappen als het cruciale en dus te beschermen intellectuele eigendom van hun implementaties. Het onzalige neveneffect is dat twee verschillende implementaties niet met elkaar kunnen samenwerken. De beoogde 'ontkoppeling van platform' wordt dus gebroken doordat alle communicerende partijen over dezelfde JMS-implementatie dienen te beschikken. Deze tweede voorgestelde oplossing blijft de meest pragmatische (en vandaag enige) aanpak om XML messaging op betrouwbare gronden asynchroon te realiseren, maar ze houdt helaas geen stand in een 'vrije standaarden' -Internet- omgeving.

ECHTE ZEEP We mogen dus zowat ons hele voorgeslacht voor hun geboorteplanning bedanken dat we het huidige SOAP[4] tijdperk van zo nabij mogen meemaken. Zo lijkt het toch als je blind alle verhalen mag geloven. En toegegeven, in volle dot-com hadden we vast nog wel

4 SOAP stond voor Simple Object Access Protocol. Sommigen zagen het eerder als SOA-Protocol waar SOA (toevallig?) het theoretisch onderliggende model is waarop de SOAP-UDDI combinatie is gebaseerd. SOA zelf staat voor Service Oriented Architecture. Om alle naamsverwarring uit te sluiten zal SOAP vanaf versie 1.2 ophouden een letterwoord te zijn, en dus niets anders betekenen dan SOAP.

dat ene open oog hiervoor een beetje verder dichtgeknepen. De pijnlijke waarheid rond SOAP komt dan ook verbazend snel tot openbaring. De vlotheid waarmee de standaard het levenslicht zag, en dus akkoorden werden bereikt tussen niets steeds even wederzijds vriendelijke partijen, had ons dan ook eerder moeten doen speuren naar het angeltje. SOAP is zo populair en zal dat vast terecht nog wel blijven omdat het zo weinig om het lijf heeft. De standaarden rond SOAP werden zo vlot[5] bereikt omdat niemand van de betrokken partijen echt zijn ziel moest prijsgeven om alles op papier te krijgen. Als Java developer merk je het meteen als je met de JAXM API aan de slag gaat.

> SOAP zal vanaf versie 1.2 ophouden een acroniem te zijn, en dus niets anders betekenen dan soap

Verbazend eenvoudig De API is eigenlijk verrassend eenvoudig, en doet nog het meest aan de socket ondersteuning (java.net.*) denken om op een laag niveau genetwerkte applicaties te bouwen. Enerzijds zijn er klare begrippen als 'Endpoint' en 'Connection' om aan te geven waar we de boodschappen heen sturen. Anderzijds is er een eenvoudige 'SOAPMessage' die vlotte assemblage en correcte plaatsing in de SOAP Envelope ondersteunt. Verbazend eenvoudig en ook meteen als dusdanig bruikbaar. Maar niet zo eenvoudig als je het wel zou willen. Punt is namelijk dat de SOAP specificatie enkel vertelt wat er allemaal (niet) kan en (niet) mag in een SOAP Envelope, maar niet echt 'hoe' je al die vrijheid dient te gebruiken om je dienst op een zinvolle manier beschikbaar te stellen. De realiteit blijft dat de leveranciers van meer full-featured platformen (MS is met z'n GXA, "Global XML Webservices Architecture" oefening al aardig op weg) vermoedelijk een niet evident (en niet breed aanvaard) gebruik zullen voorstellen voor met name de velden in de SOAP-Header. Daar vinden we straks hoogstwaarschijnlijk directieven terug voor de ondersteuning van de op dit moment duidelijk ontbrekende (maar dan ook orthogonale) aspecten in het gedistribueerde verhaal dat SOAP uitdraagt: transactiebeheer, authentisering, uitgewerkte routering.

Of het nu ingegeven werd door de politieke beslissing bij Sun (die hun inzet bij de ebXML werkgroep willen recycleren) dan wel een toonbeeld moet zijn van de luciditeit eigen aan de JAXM designers doet er weinig toe; de specificatie ondersteunt alvast een toekomst waarin verschillende 'manieren om SOAP te gebruiken' naast elkaar kunnen bestaan. Ze kunnen namelijk onder de vorm van zogenaamde 'Profiles' in de JAXM implementatie geplugd worden. De JAXM 'MessageFactory' zal dan afhankelijk van die profile namelijk een andere specifieke versie van de 'SOAPMessage' afleveren die de extra data encodering correct zal afhandelen. JAXM wordt zo de immer uitbreidbare API om XML messaging volgens de SOAP standaard te gaan realiseren. Ze blijft echter op een encoderingsniveau dat niet waarmaakt dat SOAP effectief een alternatief wordt voor de DCE, CORBA-IIOP, RMI of DCOM dezer wereld.

Oplossingen Deze klassieke oplossingen voor gedistribueerde computing ondersteunen een meer top-down approach die de encoderingsdetails voor ons verbergen. De start is een formele beschrijving (vaak in IDL, Interface Definition Language) van het dienstencontract dat dan door middel van codegeneratie aanleiding geeft tot de nodige communicatie code voor gebruik aan beide (client en server) zijden. Die classes (de client-'stub', en server-'tie') verbergen de communicatie-implementatie en gedragen zich naar de rest van de applicatie als proxy of wrapper van de in het contract beschreven dienst. Dit proces is precies wat JAX-RPC aanbrengt voor SOAP. Meer over deze en de andere, jongere JAX-API's die specifiek voor de webservices ondersteuning zijn uitgedacht in het volgende deel van deze reeks.

Marc Portier
e-mail: mpo@outerthought.org

Marc Portier is mede-oprichter van Outerthought, een klein en spitstechnologisch Java & XML competentiecentrum. Als Java-advocaat van het eerste uur volgt hij van nabij de nieuwe API's en richtingen (zoals J2EE) die de taal en het platform sinds de begindagen in '96 heeft genomen. Daarbij gaat zijn interesse vooral uit naar de Internet en XML gerelateerde onderwerpen. Outerthought helpt bij de beslissingen rond en de introductie van deze Java-XML technologieën als werkmiddelen voor applicatieontwikkeling bij zijn klanten.

5 Omtrent de kwaliteit van SOAP als specificatie is er tegenwoordig nogal wat gaande. Vooral vanuit de hoek van de HTTP dogmatici die zweren bij het REST-model (Representational State Transfer, zie: Roy Fielding en http://internet.conveyor.com/RESTwiki/moin.cgi) worden nogal wat argumenten geleverd om eerder de besproken 'rol ze zelf' technieken te gaan uitwerken. (Meer over de discussie op: http://www.xml.com/pub/a/2002/02/20/rest.html. De auteur van het artikel, Paul Prescod, wordt beschouwd als een autoriteit op XML gebied).