Dik Bijl
is werkzaam bij Microsoft

Van webservices naar SOA

DENKEN AAN SERVICES & MESSAGES OP VERLOREN AVONDEN EN IN FILES

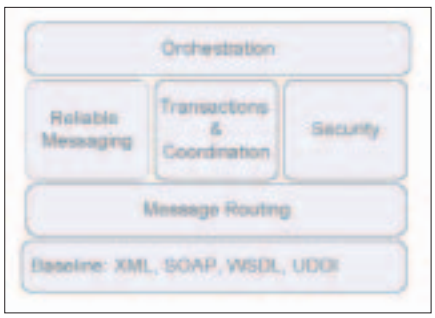
Om webservices in te zetten in bedrijfskritische toepassingen is er veel meer nodig: een robuuste en veilige infrastructuur, keiharde interoperabiliteitsgaranties en niet te vergeten een goede architectuur. Dit artikel geeft een introductie op de Service Oriented Architecture (SOA). Conclusie: we gaan weer een paradigma-shift tegemoet.

Geïnitieerd door Microsoft en getrokken door een Microsoft/IBM-samenwerkingsverband zijn webservices uitgegroeid van zomaar de volgende 'IT-hype' tot het breedst ondersteunde initiatief binnen de IT-industrie. Inmiddels is elke beetje ICT-er wel op de hoogte van het supertrio: SOAP, WSDL en UDDI.

Nu kunnen we – achteromkijkend – gerust zeggen: webservices werken technisch en zijn als concept aangeslagen in de markt: bij leveranciers, analisten en de klanten. Er zijn al tal van webservices-toepassingen - veelal in de proefsfeer, al zijn er zeker ook praktische toepassingen. Maar webservices zijn nog geen 'mainstream' voor bedrijfskritische toepassingen. Er zijn drie zaken nodig om webservices te tillen naar het niveau van mainstream. Pas dan kan het 'nirvana' van webservices tot zijn volle glorie komen: het produceren, coördineren en consumeren van webservices overal, op elk moment en met ieder device.

Wat zijn die drie zaken?

Ten eerste: een robuuste, operationele infrastructuur zodat webservices samengesteld kunnen worden en veilig, betrouwbaar, transactioneel en beheerd uitgevoerd kunnen worden. Volgens Don Box: 'The Microsoft Global XML Architecture (GXA) is a protocol framework designed to provide a consistent model for building infrastructure-level protocols for Web services and applications'. Box zegt ook: 'Global XML Web Services Architecture (GXA) is a composeable set of standards for web services. GXA builds on existing protocols (SOAP, WSDL, UDDI) and provides a more complete application architecture - adding additional building blocks as well as support for higher level protocols such as reliable messaging, security … '. Microsoft bouwt deze protocol stack (zie afbeelding 1) samen met anderen (zoals IBM, BEA, Verisign en SAP) en maakt dat de functionele webservices goed operationeel binnen een messaging-infrastructuur uitgevoerd kunnen worden. Een bespreking van GXA valt buiten de scope van dit artikel. Maar reken er op dat de Microsoft XML Web Services gerelateerde runtimes en tools de komende tijd flink uitgebreid gaan worden. De eerste tekenen zijn daarvan al zichtbaar in Microsoft's WSE-toolkit.



Afbeelding 1.

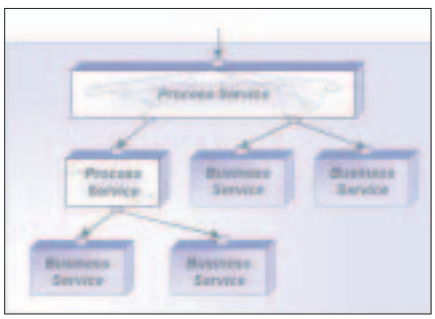
Ten tweede: interoperabiliteit. Geen enkele leverancier zal het webservices-spectrum volledig kunnen bezitten; ook Microsoft niet. Het is per definitie een coöperatie tussen implementaties van verschillende leveranciers. Immers, webservices heeft als doel alle heterogeniteit te overbruggen door afspraken over interoperabiliteit. Dat betekent dat iedere implementatie van een webservice (bij bedrijf A met webservices-technologie van leverancier B) in principe moet kunnen communiceren met willekeurig welke andere implementatie van een webservice (bij bedrijf C met webservices-technologie van leverancier D). Maar dat moet natuurlijk wel werken in de praktijk. De WS-I (Web Services Interoperability Organisation, www.ws-i.org) telt inmiddels 150 leden en houdt zich bezig met het controleren van de interoperabiliteit en op termijn wellicht het certificeren van webservice-producten.

Ten derde: gedegen architectuur. Bij dit laatste punt komt de Service Oriented-Architectuur (SOA) om de hoek kijken. Dit artikel gaat in op SOA maar ambieert niet meer dan een eerste kennismaking te zijn. Desalniettemin kan het een schok teweegbrengen. Het is wel even wat anders dan de gemiddelde ontwikkelaar vandaag met webservices in Visual Studio .NET doet. Zoals gezegd, de tech-

nologie en de tools zullen in de komende tijd sterk gaan veranderen. Maar het zijn vooral de architect, ontwerper en ontwikkelaar van SOA gebaseerde toepassingen die moeten mee veranderen. Heb je net de 'paradigma shift' van procedureel naar objectgeoriënteerd naar componentgebaseerd achter de rug, kun je je nu alweer gereed maken voor de volgende: services & messages. Het is misschien allemaal nog wat abstract, maar bedenk dat je het niet direct vanaf morgen overal hoeft toe te passen.Het advies is: groei er langzaam maar zeker in en ga – op verloren avonden of in de file – denken aan en in 'services & messages'.

Wat is SOA?

SOA is een architectuur waarin zelfstandige softwareservices met elkaar communiceren door het uitwisselen van berichten ('messages'). Er zijn twee basistypen softwareservices: de uitvoerende softwareservices (veelal: business services genoemd) en de organiserende en coördinerende services (veelal process services genoemd). De process services bepalen wie, wanneer, welke berichten verstuurd en beheren bijvoorbeeld inter-service-transacties. Process services kunnen genest zijn (zie afbeelding 2).



Afbeelding 2.

Merk op dat hier in algemene termen over softwareservices en messages gesproken wordt. SOA is ouder dan de webservices-technologie. Maar met webservices komt SOA pas goed tot zijn recht. In de rest van het artikel worden de algemene SOA-termen - softwareservices en messages - en de specifieke technologietermen – webservices en SOAP-messages – door elkaar heen gebruikt.
Dit is weliswaar een magazine voor developers en architecten, maar het is toch nuttig om eens te kijken naar de 'Business Value' van SOA en webservices. Waarom is er zoveel aandacht voor dit onderwerp? Drie woorden springen er daarbij uit: 'hergebruik', 'ontkoppeling' en 'alomtegenwoordigheid'.

Hergebruik

Een kernpunt van SOA is dat het eindelijk de belofte van hergebruik van code zal gaan inlossen. Hergebruik horen we in theorie natuurlijk al heel lang, maar is in de praktijk nooit echt van de grond gekomen. Vooral omdat hergebruik op een te 'fine grained'-niveau gepropageerd werd - hergebruik van procedures en later: objecten of componenten - en omdat hergebruik beperkt werd door de grenzen van technologie & implementatie. Een procedure uit een Cobol library kon niet eenvoudig worden aangesproken door een CORBA-object dat weer moeilijk aanspreekbaar was voor een COM-component. Bij SOA staan een veel grovere granulariteit (lees: module of zelfs applicatieniveau) en interoperabiliteit voorop volgens strikte industriestandaards, waarbij kennis van de specifieke implementatie (Cobol-procedure, Corba-object, COM, J2EE of NET componenten) niet nodig is.
Hoe gaat dat in de praktijk werken? Legacy-applicaties zullen deels als services 'gewrapped' worden; bijvoorbeeld door gebruik te maken van message brokering-technologie zoals Biztalk. Naar buiten toe worden het standaard webservices; intern vindt de vertaling plaats naar het interne, eigen formaat van de legacy-applicatie. Andere legacy-applicaties zullen uitgefaseerd en opnieuw gebouwd worden met de nieuwste technologie en direct als 'native' webservices. Het .NET-platform heeft XML en webservices 'at the core' en zal dus veel efficiëntere webservices opleveren, dan een gewrapte legacy-omgeving of een nieuwbouwomgeving, waarbij webservices meer een 'afterthought' is geweest (zoals bij J2EE). Nu daarmee het IT-landschap – hetzij door 'wrapping', hetzij door nieuwbouw - veranderd is van silo-achtige, gesloten end-to-end-applicaties in een set van 'coarse grained' services (halffabrikaten), worden de daadwerkelijke end-to-end-applicaties gemaakt door beschikbare services via orkestratie en coördinatie aan elkaar te knopen en te voorzien van de juiste message based en/of user interfaces. Dit leidt tot applicaties die vele malen sneller en goedkoper gemaakt (en afgebroken!) kunnen worden dan vandaag de dag; hetgeen sterk bijdraagt aan de 'agility' (wendbaarheid) van de ondernemingen.

Ontkoppeling

Maar er is meer met SOA aan de hand dan alleen hergebruik. SOA ontkoppelt en doet dat op minimaal twee manieren. Ten eerste, SOA ontkoppelt de specificatie (plus een daarbij behorende message based interface volgens de standaards) van de daadwerkelijke implementatie. De implementatie van de service is vrij: het mag in elke taal, op elk platform, volgens elk programmeermodel. Die vrijheid van keuze voor de implementatie kan iedere keer weer opnieuw gemaakt worden. Nooit meer een 'lock in' (behalve dan in services)! Het bevordert de concurrentie tussen leveranciers van webservices enabled runtimes en tools. En dat komt neer op meer kwaliteit tegen minder kosten.
De tweede ontkoppeling, en een met een enorme potentiële 'business value', is ontkoppeling van plaats en eigendom. Een bedrijf hoeft nieuwe, gewenste functionaliteit niet zelf te (laten) bouwen of te implementeren (in geval van een standaardpakket), maar kan de functionaliteit als softwareservice afnemen van een externe provider. Omgekeerd kan een bedrijf zijn eigen softwareservices inzetten ten behoeve van derden en daarmee extra inkomsten genereren. Tenslotte kan een bedrijf een deel van zijn bestaande services afbouwen en outsourcen naar derden, terwijl het de onderscheidende en kritische services zelf blijft voeren. Deze 'outsourcing' is dus op een veel fijner niveau dan de 'alles of niets'-situatie van vandaag de dag, waarbij de gehele IT-operatie wordt uitbesteed.

Alomtegenwoordigheid

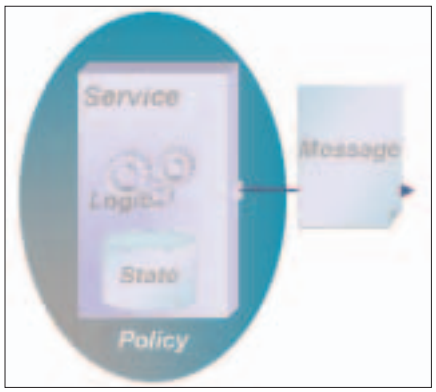
Webservices zullen alomtegenwoordig zijn. Enerzijds zullen vrijwel alle software-

leveranciers op de een of andere manier webservices ondersteunen en een steeds belangrijker positie geven in hun portfolio. Dus waar je ook kijkt: je kunt er niet meer omheen. Anderzijds zie je dat webservices algemeen toepasbaar zijn. Daar waar EDI en ebXML uitsluitend op B2B-scenario's gericht zijn, zijn webservices veel breder inzetbaar. Niet alleen voor B2B, maar ook voor interne applicatie-integratie (EAI) en integratie tussen mensen via devices (peer to peer). Al met al beloven SOA en webservices bij te dragen aan een verbeterde productiviteit van de business (door snellere toegang tot informatie en services), aan een wendbaarder business (door snellere IT-ondersteuning) en aan betere connectiviteit met klanten, partners en werknemers (overal, op elk moment, met ieder device); en dat tegen zo laag mogelijke kosten.

Services & Messages

Na deze 'business value'-motivatie is het de hoogste tijd eens wat dieper in SOA te duiken.

We beginnen met een antwoord op de vraag: wat is een softwareservice? Microsoft's Architecture Center (op www.microsoft.net/architecture) definieert het als volgt: 'services are network-capable units that implement logic, manage state, communicate via messages and are governed by policy' (afbeelding 3).



Afbeelding 3.

In het binnenste van een service zit de 'state'. Rondom de 'state' is de 'logic', ook wel 'business logic'. De state kan alleen benaderd en gemuteerd worden via die 'logic', die de consistentie van de state beschermt. Een mutatie wordt in de regel onder de bescherming van een transactie uitgevoerd, zodat de consistentie van de state gewaarborgd blijft. Dit is nog redelijk abstract, maar denk nu eens aan een applicatie, bijvoorbeeld een orderverwerkingssysteem met als 'state' een order database. Een (mutatie op een) order wordt alleen geaccepteerd als het aan de bedrijfsregels voldoet. Bijvoorbeeld het orderbedrag moet het totaal zijn van de bedragen in de orderregels verminderd met de standaardkorting. De granulariteit van een softwareservice is bij voorkeur grof; liefst op het niveau van de applicatie(module). Hoe grover de granulariteit, hoe groter de onafhankelijkheid en de zelfvoorziening van de service kan zijn. Dat wil zeggen, de service kan een complete interne transactie aan (zoals een nieuwe order) zonder daarbij afhankelijk te zijn van andere services. Dit is natuurlijk een algemene uitspraak. De praktijk is ingewikkelder: een te grof niveau van granulariteit kan hergebruik ook weer in de weg staan. Het vaststellen van het juiste niveau van granulariteit in specifieke situaties is bij uitstek het werk van de architect van de SOA-omgeving.

Die onafhankelijkheid en zelfvoorziening van services in een SOA is een groot goed: het bevordert 'loosely coupled'-applicaties en die verminderen de complexiteit en verhogen de flexibiliteit. Men kan veel sneller nieuwe applicaties maken c.q. bestaande applicaties wijzigen door de losse koppeling tussen de services. Het maken van service based-applicaties komt vooral neer op het orkestreren en coördineren van reeds bestaande services (ergo, hergebruik).

Messages

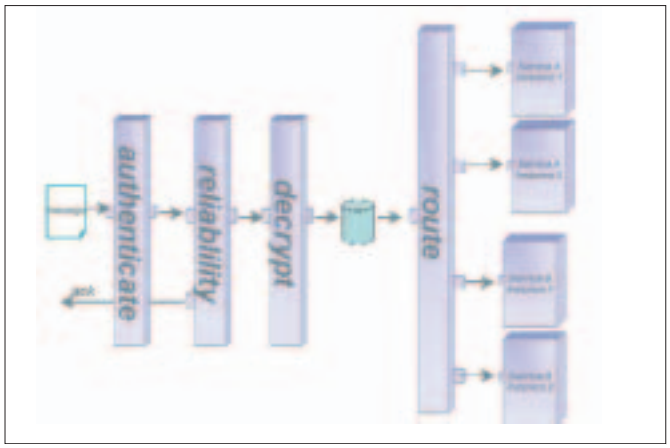
Services communiceren via messages; in het geval van webservices gaat het daarbij om SOAP-messages. De SOAP messages zijn volledig zelfbeschrijvend en beschrijven twee onderwerpen. Het functioneel inhoudelijke deel (bijvoorbeeld 'registreer order' met daarbij alle benodigde ordergegevens) en het operationele deel dat betrekking heeft op de messaging-infrastructuur. De messaging infrastructuur is een afgeleide van de GXA van afbeelding 1. Waarom is zo'n messaging-infrastructuur nodig? We maken een rondje langs de velden van afbeelding 1.

De messaging-infrastructuur moet rekening houden met eventueel afgesproken leveringsvoorwaarden: Service Level Agreements (SLA's) en Quality of Service (QoS). Denk aan het eerder besproken model, waarin de IT-organisatie selectief softwareservices outsourced. Als leverancier van bepaalde softwareservices heb je afspraken met je klanten; misschien wel met verschillende SLA-niveaus. Dat betekent bijvoorbeeld dat je sommige berichten sneller moet afhandelen dan andere. Daarvoor kan 'content based routing' nodig zijn of 'load balancing' van service-aanvragen met in elk geval verschillende service-instanties voor functioneel dezelfde service-aanvraag.

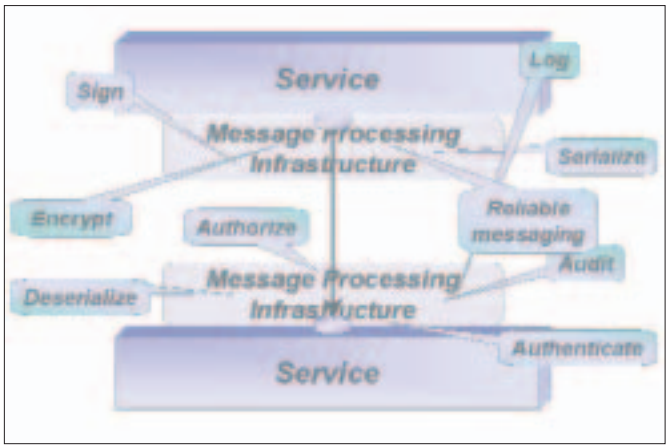
De messaging-infrastructuur moet een gezond wantrouwen hebben tegen elk bericht dat op hem afkomt. Immers we gebruiken voor de berichtuitwisseling – vooral tussen bedrijven - in de regel het publieke internet en dat is onveilig. Dit betekent dat de messaging-infrastructuur tal van zaken moet doen om het bericht – voordat het toegelaten wordt tot de service – te controleren en te valideren. Zijn het bericht en de zender authentiek? Is er onderweg niet met het bericht gesjoemeld (integriteit)? Is het bericht 'encrypted'? Het bericht is aan ons gericht, maar leveren wij de gevraagde service eigenlijk wel? Is de zender van het bericht geautoriseerd voor de gevraagde service? Is er sprake van criminele actie zoals een denial-of-service-aanval?

Een bericht kan onderdeel zijn van een business-transactie en zal als zodanig een transactie-id met zich meedragen. Dit betekent dat later mogelijk een verzoek komt tot een 'roll back' van een uitgevoerde service of dat er een compenserende service-aanvraag kan komen in het kader van de transactie. Ook zal een transactie uiteindelijk afgerond worden en daarover kan weer apart gecommuniceerd worden.

Bij bedrijfskritische toepassingen kan het van groot belang zijn dat 'reliable messaging' wordt toegepast. Dit wil zeg-



Afbeelding 4.



Afbeelding 5.

gen dat een verzonden bericht precies een keer aankomt. Daarvoor is extra communicatie nodig tussen zender en ontvanger. De ontvanger krijgt het bericht en stuurt een bevestiging, waarop de zender van het bericht een bevestiging over de bevestiging stuurt (en daarmee belooft het bericht niet nogmaals te verzenden). Misschien dat in het kader van de QoS het bericht alleen maar door de service verwerkt mag worden als zender en ontvanger (ook: van elkaar) weten dat het verzonden bericht is aangekomen. Overigens wordt die extra communicatie automatisch gegenereerd door de messaging infrastructuur zelf. De ontwikkelaar van de betreffende service interactie geeft slechts aan dat het bericht gegarandeerd moet worden afgeleverd.

Policy

Het laatste onderdeel van de definitie van een service sprak over 'policy'. Het voert te ver om in dit artikel uitgebreid in te gaan op 'policy', maar bedenk dat wat hiervoor is besproken over de security- en SLA- en reliability-aspecten in een policy (d.i. beleid/procedures) moet worden vastgelegd. Zo kan de policy van een service zijn, dat het alleen berichten accepteert die 'signed' zijn met een digitaal certificaat, binnenkomen over SSL en verstuurd zijn door zenders met een 'Gold Member' SLA-overeenkomst

Afbeelding 4 geeft een voorbeeld van een complexe messaging-infrastructuur pijplijn (alleen de ingaande lijn). Merk op dat de messaging-infrastructuur gedeeld wordt door verschillende services in plaats van dat iedere service zijn eigen infrastructuur heeft. Dat houdt in dat messages een aantal extra 'hops' met mogelijke tussentijdse 'store and forward' (queues) zullen ondergaan, voordat ze bij de uiteindelijke service terecht komen. Die 'intermediate hops' vallen allemaal onder het routing-mechanisme van de services-architectuur stack van afbeelding 1. Voor de terugweg geldt hetzelfde: berichten zullen gesigned moeten worden, encrypted, voorzien van transactie id's, bevestigt, etcetera. Afbeelding 5 geeft een ander beeld van de messaging-infrastructuur, waaruit blijkt dat zowel de zendende als de ontvangende kant van een message-interactie werk te verzetten heeft. De architect van een SOA-omgeving heeft flink wat te puzzelen. Maar zoals eerder gezegd: het hoeft morgen nog niet af!

Wat ook uit de figuur duidelijk naar voren komt is dat mainstream, bedrijfskritische webservices niet echt lijken op de simpele, directe request/reply webservices die vandaag de dag als 'Proof of Concept' met VS.NET gebouwd. En dat brengt ons bij het punt waarop SOA direct raakt aan de ontwerper/ontwikkelaar van webservice based toepassingen, zowel de 'consumerende' client als de 'leverende' server: het ontwikkelen voor asynchrone communicatie!

Asynchrone communicatie

In de hiervoor geschetste SOA-wereld met zijn messaging-infrastructuur past eigenlijk alleen maar het asynchrone programmeermodel. En dat terwijl vrijwel alle client-applicaties uitgaan van een blokkerend, synchroon programmeermodel. En zo hebben de meeste ontwikkelaars ook hun eerste schreden gezet op het webservices pad. Vanuit WSDL wordt een proxy gegenereerd en tegen die proxy wordt een synchrone call geschreven; neem het beroemde 'geef mij de huidige aandelenkoers van bedrijf X'-voorbeeld. Omdat de geschreven webservice call simpel is en meestal rechtstreeks – dat wil zeggen zonder intermediate hops - uitgevoerd wordt, komt het antwoord ook redelijk snel. Maar hoe complexer de infrastructuur en hoe meer outsourcing van services plaatsvindt, hoe onbepaalder de 'latency' wordt.

Dat vraagt om een andere aanpak, namelijk een waarin de request/reply rpc-communicatie verlaten wordt voor de 'one way' messagecommunicatie. De request/reply rpc wordt dan omgezet tot een request message heen, en een afzonderlijke reply-message terug. Dat is veel complexer programmeren. Immers de request en reply messages moeten op de een of andere manier aan elkaar gekoppeld worden (bijvoorbeeld via een message id) en iedere webservice consumer moet nu ook een webservice provider worden (en omgekeerd). Een webservice consumer moet een soort event handling/interruptie-mechanisme hebben om de reply message te verwerken en te synchroniseren met de client 'state'.Hoe dat er allemaal exact uit gaat zien, is nog niet duidelijk. Maar reken er op dat de Microsoft tools je daarbij zullen helpen. Dat betekent echter niet dat je niet moet beginnen te denken aan je nieuwe programmeermodel – services en asynchrone messages. Ik wens je daarvoor veel verloren avonden en zo nu en dan een kleine file.