Achtergrond

In onze eerdere artikelen in Software Release Magazine over de Service Oriented Architecture (SOA) is regelmatig de nadruk gelegd op de verwachte voordelen van een services architectuur. Helaas zijn er vooralsnog maar weinig voorbeelden aan te wijzen waar zo'n architectuur succesvol is gerealiseerd. Kennelijk is het bouwen ervan nog niet zo eenvoudig. In dit artikel zullen we ingaan op enkele aspecten die het realiseren van een services architectuur tot een redelijk hachelijke onderneming kunnen maken.

SOA: Schier Onhaalbare Architectuur?

Hindernissen op weg naar een services architectuur

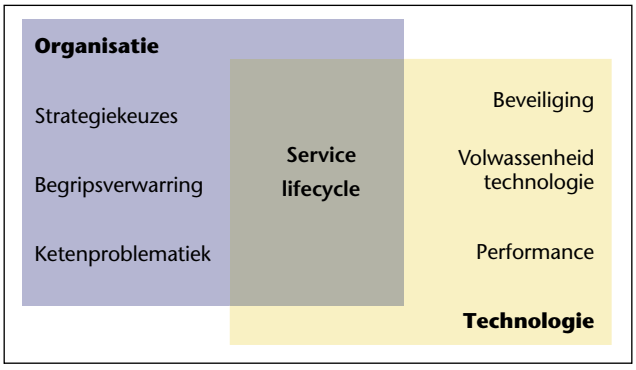
Aangezien een services architectuur net zoveel over organisatie gaat als over techniek, zullen we de hordes op weg naar een services architectuur ook verdelen in twee delen: organisatorische en technische. Op het vlak van de organisatie onderkennen wij de volgende belangrijke hordes voor succes met een services architectuur:

- Strategiekeuzes
- Begripsverwarring
- Ketenproblematiek

Voor het technische deel onderkennen wij de volgende hordes:

- Beveiliging
- Volwassenheid van de standaarden
- Performance

Daarnaast is er nog een horde die zowel betrekking hebben op de organisatorische als de technische kant: de service-lifecycle. Al deze factoren tezamen leveren het probleemveld op, zoals is weergegeven in figuur 1. We zullen al deze hordes op weg naar een service architectuur in het navolgende nader gaan bekijken.



FIGUUR 1. Problemen bij realisatie van een services architectuur per aandachtsgebied

STRATEGIEKEUZES Wellicht één van de belangrijkste voorwaarden om te komen tot een succesvolle realisatie van een servicegeoriënteerde IT is het bedenken en toepassen van een weldoordachte strategie om dit doel te bereiken. In het artikel 'De business drivers voor een SOA' uit Software Release Magazine nummer 1 van dit jaar worden hiertoe aanbevelingen gedaan. Een belangrijke boodschap in dit artikel was om de ambitie bij de implementatie van een services architectuur niet alleen af te stemmen op de doelen van de organisatie, maar zeker ook op de mogelijkheden binnen de organisatie om de bijbehorende verandering in goede banen te leiden. Klein beginnen maakt de kans op succes vele malen groter!

In de praktijk wil het vaak niet lukken om een dergelijke uitgebalanceerde strategie te implementeren. Veelal wordt getracht om meteen een services architectuur in de gehele organisatie door te voeren, omdat de te behalen voordelen het meest aantrekkelijk zijn en daardoor de business case aantrekkelijk lijkt. In de praktijk stranden dergelijke initiatieven echter vaak op de (te) grote complexiteit van de verandering.

Nog vaker komt het voor dat vanaf de (IT-)werkvloer wordt getracht om een services architectuur tot stand te brengen. Veelal beginnen deze initiatieven klein en zijn ze sterk technologiegedreven. Hoewel ze technisch vaak succesvol zijn, sneuvelen ze op de business case en op de bestaande, niet aangepaste, IT strategie.

De kunst is hier om de drang om het groots aan te

pakken te weerstaan en dus klein te starten, maar dan wel te kiezen voor een project waarin direct een pijnpunt van de organisatie kan worden getackeld en waarin bovendien de voordelen van services architectuur aantoonbaar zijn. Op deze wijze kan belangrijke ervaring worden opgedaan, draagvlak worden gecreëerd en de aanzet worden gegeven om services architectuur in een volgende fase breder in te steken. Het verdient dus absolute aanbeveling op zoek te gaan naar een services architectuur-project met een duidelijke *quick win*.

Een goed doordachte strategie en bijbehorende plannen zijn echter niet voldoende om een services architectuur ook daadwerkelijk succesvol te realiseren. Een volgende horde die minstens zo moeilijk te nemen is, is het zorgdragen voor een gemeenschappelijk beeld tijdens de realisatie en gebruik van een servicegeoriënteerde IT-huishouding.

Begripsverwarring Het spreken van dezelfde taal, en het voeren van een gemeenschappelijk vocabulaire is een noodzakelijke voorwaarde voor succes in ieder architectuurtraject. Wees vooral bewust van de zeer grote begripsverwarring die heerst rond het woord 'service'. Op directieniveau denkt men bij een 'service' vooral aan een product dat door een klant wordt afgenomen en waarmee geld verdiend wordt: een 'business service'. Een meer technisch georiënteerd persoon denkt bij een 'service' primair aan een webservice en is nauwelijks geïnteresseerd in de afnemer van de dienst. Zowel door de eindgebruiker als vanuit systeemontwikkeling wordt bij een 'service' primair gedacht aan een functie die een applicatie biedt aan de eindgebruiker; een 'end-user service'.
Deze begripsverwarring rond het begrip 'service' brengt een voortdurende stroom aan kleine en grote problemen met zich mee, zoals:

- Afstemming tussen opdrachtgever en uitvoerder verloopt moeizaam;
- Opdrachtgever kan zich niet vinden in voorgestelde oplossingen;
- Systeemontwikkelaars krijgen van bovenaf opdrachten mee waarmee ze in de gebruikersorganisatie niet uit de voeten kunnen;
- Eindgebruikers zijn ontevreden over geboden ondersteuning;
- Opdrachtgever bestempelt het project als mislukt omdat de business case niet gehaald wordt;

Indien deze begripsverwarring niet actief wordt aangepakt, zal de realisatie van een services architectuur een lange en moeizame weg worden. Zorg ervoor dat op een consistente wijze één visie wordt nagestreefd, of dat de verschillende visies op een beheerste wijze worden verenigd. Wordt dit niet goed opgepakt, dan kan er na een zwaar bevochten traject een hybride systeem staan dat aan geen van alle visies voldoende voldoet.

Ketenproblematiek Bij het leveren van een product of dienst zijn veelal meerdere leveranciers betrokken. In een optimaal functionerende servicegeoriënteerde IT-huishouding zouden eigenlijk ook al deze toeleveranciers hun bijdrage moeten leveren in de vorm van (deel)services. Hoewel dit vrij onschuldig klinkt, kleven aan deze uitspraak verstrekkende gevolgen. Vraagstukken waarover voorheen binnen de eigen organisatie kon worden besloten, dienen te worden afgestemd met verschillende ketenpartners. Dit begint al met het maken van technologische IT-keuzes en strekt zich uit tot meer organisatorische aspecten, zoals leverings- en garantievoorwaarden. Deze aspecten vatten we ook wel samen met de term *Quality of Service* (QoS).

Afgestemde technologiekeuzes zijn randvoorwaardelijk voor het gezamenlijk kunnen aanbieden van services, ongeacht hun aard en inhoud. Dergelijke keuzes zijn nodig om integratie mogelijk te maken en de benodigde integratie-inspanning te beperken. Dit betekent nog niet dat elk van de ketenpartners precies dezelfde productkeuzes dient te maken. Open standaarden (en een keuze hiervoor) kunnen hierbij een zeer voorname rol spelen, zo hebben we in eerdere artikelen gezien.

Afstemming van meer organisatorische aspecten als leverings- en garantievoorwaarden dient vooral om te kunnen komen tot een zo concurrerend mogelijke service. Immers in alle aspecten van de service geldt dat de kracht van de service wordt bepaald door de zwakste schakel. Indien één toeleverancier een maand garantie biedt en alle overige toeleveranciers een jaar dan zit op de gehele service een garantie van een maand. Indien één toeleverancier een servicewindow van 6*18 uur biedt en alle overige toeleveranciers van 7*24 uur dan geldt voor de gehele service een servicewindow van 6*18 uur.

Waar menige organisatie haar handen al meer dan vol heeft aan het beheersen van haar eigen IT, wordt met een volledig doorgevoerde service-georiënteerde IT de *span of control* uitgebreid tot de gehele productieketen. De beheersmatige problematiek die hiermee samenhangt, is enorm en kan nauwelijks onderschat worden. Behalve de eerder genoemde afstemming van keuzes is het ook van belang om duidelijke afspraken te maken over (deel)verantwoordelijkheden tussen de hoofdleverancier en de toeleveranciers en de randvoorwaarden waaronder de verschillende partijen deze verantwoordelijkheden wensen te aanvaarden.

Een alternatief is in dit geval om de afhankelijkheden met ketenpartners te isoleren achter (deel)services binnen de eigen service-georiënteerde IT. Op deze wijze kan de informatie-uitwisselingen met ketenpartners

ongestoord blijven en kan toch een servicegeoriënteerde IT worden gerealiseerd. Bedenk evenwel dat met dit alternatief een belangrijk deel van de te behalen voordelen van service-oriëntatie in het gedrang komen. Men wordt nu immers gedwongen om de bijdrage van de ketenpartners te repliceren binnen de eigen IT, en vanwege communicatievertragingen kan niet dezelfde quality of service geboden worden. Daarnaast blijft men volledig verantwoordelijk voor elke geleverde service.

SERVICE-LIFECYCLE Servicegeoriënteerde systemen brengen nieuwe eisen met zich mee aan de processen en tools om die systemen te ontwikkelen en te onderhouden. Alle disciplines komen daarbij aan de beurt, van bedrijfsmodellering tot implementatie en van projectmanagement tot infrastructuur. Daarbij is één principe leidend: we moeten uitgaan van een draaiend systeem waaraan onderdelen worden aangepast. Dit is een wezenlijk andere situatie dan wanneer het complete systeem kan worden ontwikkeld, getest en uitgerold. Dat de winkel open moet zijn, terwijl er een verbouwing plaatsvindt is doorgaans een eis. Het werken met systeemwanden, zodat ruimtes naar believen kunnen worden aangepast, is geen uitzondering meer, maar regel. Nu kunnen we ons de soms vermakelijke tegenstellingen tussen ontwikkeling en beheer, oftewel tussen flexibiliteit en continuïteit, nog permitteren. De tijd zal echter snel zijn aangebroken, dat flexibiliteit en continuïteit eensgezind gelijk op gaan. Dan zal blijken dat continuïteit een voorwaarde is voor flexibiliteit en flexibiliteit een voorwaarde voor continuïteit.

Wie op zoek gaat naar procesraamwerken die beide gebieden in een geïntegreerde, elkaar versterkende dynamiek omvatten, komt bedrogen uit. Wie tools wil aanschaffen om deze visie in de praktijk te brengen, komt vooralsnog meer presentaties tegen dan software. De processen en tools die we nu gebruiken, schieten tekort voor de ontwikkeling van servicegeoriënteerde systemen waarbij ook de service-lifecycle wordt ondersteund. Met deze processen en tools kan nooit die mate van kwaliteit en kostenefficiency worden bereikt die nodig is om op grote schaal servicegeoriënteerde systemen te kunnen toepassen.

Het is niet vreemd dat de partijen die als leiders worden gezien in de arena van *webservices enabled software*, IBM en Microsoft, zich intensief met deze problematiek bezighouden. Beide gaan er immers van uit dat servicegeoriënteerd systemen de toekomst hebben en dat een geïntegreerde benadering, van het modelleren van bedrijfsprocessen tot het technisch beheer met het continu *health-checking* van webservices, een pure noodzaak is. Wat bij Microsoft het *Dynamic Systems Initiative* (DSI) is gedoopt, lijkt zich in eerste instantie meer te richten op het *operations management* en dus het draaiend krijgen en houden van webservices. IBM concentreert zich met het Service Oriented Modeling and Architecture (SOMA) meer op het modelleren van servicegeoriënteerde systemen op basis van de eisen van de business. Ongetwijfeld zullen deze initiatieven de gehele levenscyclus van servicegeoriënteerde systemen gaan omvatten en zal de komende jaren het hier geschetste probleem tot het verleden behoren. Vanuit de service-lifecycle problematiek is de stap naar de meer technische problemen een kleine. Een technische uitdaging waar de meeste CTO's van zullen wakker liggen, is beveiliging.

BEVEILIGING Services architectuur gaat samen met een grotere nadruk op integratie over de keten heen (en dus ook buiten de organisatie). Bovendien kan een servicepotentieel op meerdere (en nog onbekende) plaatsen worden ingezet. Hierdoor dient beveiliging, nog meer dan voorheen, voldoende op de kaart te staan bij het realiseren van een services architectuur.

Het spreekt voor zich dat een open architectuur als een services architectuur moeilijker te beveiligen is dan een meer gesloten architectuur. In tegenstelling tot bijvoorbeeld bij componenten of objecten (denk aan OO) het geval is, weet een aanbieder van services vaak niet van tevoren welke consumers de service gaan gebruiken. Daarnaast is er natuurlijk het gegeven dat services worden ingezet in heterogene omgevingen, die daardoor verschillende authenticatie- en autorisatiemechanismen gebruiken. De moeilijkheid laat zich tevens voelen als op een andere wijze de zogenaamde *trust boundary's* van een domein overschreden worden, bijvoorbeeld indien een service communiceert met services of systemen die weliswaar op hetzelfde platform draaien, maar in een ander domein. Vraagstukken ten aanzien van beveiliging van services kunnen teruggebracht worden tot een beschouwing van vijf aspecten.

1. *Onweerlegbaarheid* (in vakjargon ook wel *non-repudiation* genoemd) heeft betrekking op maatregelen die borgen dat een entiteit niet kan ontkennen dat het een bericht heeft ontvangen of verstuurd.
2. *Vertrouwelijkheid* moet garanderen dat niet-geautoriseerde entiteiten berichten niet kunnen lezen.
3. *Autorisatie* bepaalt welke privileges beschikbaar zijn of worden gesteld voor een geauthentiseerde entiteit.
4. *Authenticatie* garandeert dat een entiteit (een persoon of een systeem) daadwerkelijk is wie of wat hij/het beweert te zijn.
5. *Integriteit* garandeert dat documenten of berichten niet gewijzigd zijn.

Oplossingen die invulling moeten geven aan bovengenoemde beveiligingsaspecten, vallen uiteen in oplos-

Lees verder op pagina 49.

Vervolg van pagina 8.

singen op het gebied van beveiliging van het netwerk en de communicatiekanalen (waarbij we kunnen denken aan het bekendste voorbeeld, Secure Sockets Layer ofwel SSL), en oplossingen op het gebied van beveiliging van het bericht zelf (encryptie, digitale handtekening). Het valt buiten de scope van dit artikel om alle oplossingen te bespreken (dit zullen we doen in een volgend artikel in de reeks over services architectuur). Een globaal overzicht van beveiligingsaspecten en de daarbij behorende oplossingsrichtingen vindt u in tabel 1.

Aspect	Oplossingsrichting
Onweerlegbaarheid	Digitale handtekeningen (S/MIME)
Vertrouwelijkheid	Netwerkbeveiliging, encryptie, SSL
Autorisatie	Netwerkbeveliging, digitale certificaten
Authenticatie	Netwerkbeveliging, digitale certificaten
Integriteit	Netwerkbeveiliging, digitale handtekeningen, SSL

TABEL 1. Beveiligingsaspecten en oplossingsrichtingen

Het goede nieuws is dat de standaarden rondom beveiligingsaspecten van services volop in ontwikkeling zijn. Binnen de zogenaamde WS-* standaarden zijn er een aantal die specifiek op beveiliging gericht zijn, waarvan WS-Security en WS-SecureConversation het verste zijn qua ontwikkeling en volwassenheid.

Het punt van volwassenheid brengt meteen het slechte nieuws: er is op dit moment nog geen standaardvoorstel voor service-beveiliging dat het ook daadwerkelijk tot standaard heeft geschopt. De diverse leveranciers - Microsoft voorop met haar WSE (Web Service Enhancements) toolkit - bieden wel in meerdere of mindere mate ondersteuning voor de WS-* beveiligingsstandaarden, maar het is de vraag welke standaard wel, en welke niet algemeen geaccepteerd gaat worden. Bovendien lijkt het er op dat er behoorlijk wat redundantie zit in wat de markt te bieden heeft op het gebied van beveiliging van services (bijvoorbeeld SSL versus WS-SecureConversation), zodat de keuze voor een juiste invulling van security op zijn zachtst gezegd een uitdaging is.

De volgende technische uitdaging is met de verwijzing naar de WS-* standaarden al genoemd: de *volwassenheid* van de standaarden.

VOLWASSENHEID VAN STANDAARDEN In eerdere artikelen in onze reeks over services architectuur hebben we al een aantal problemen ten aanzien van de volwassenheid van de technologie de revue laten passeren. Zo hebben we gezien dat er nog geen eenduidige standaard is voor het maken van procesmodellen voor webservices (ook wel orkestratie of choreografie genoemd), dat ontwikkelaars zich nieuwe, lastige concepten moeten eigen maken, en dat zeker niet alle webservices geschikt zijn voor gebruik binnen een services architectuur, ook weer vanwege ambiguïteit in de standaarden voor de services.

Een ander punt dat ook nog steeds onvoldoende is ingevuld, zijn de zogenaamde semantische standaarden. De standaarden om systemen technisch met elkaar te koppelen, via services op een loosely coupled, asynchrone manier met gebruik van service contracten, worden steeds volwassener. De problemen bij veel integratiepogingen en koppelingen doen zich voor op het semantische vlak: dat services met elkaar kunnen *binden*, wil nog niet zeggen dat ze elkaar begrijpen en dus kunnen samenwerken. Ook op technisch niveau kan er 'begripsverwarring' zijn. XML biedt hier *geen* oplossing voor, ondanks dat het vaak wordt aangemerkt met de term *lingua franca*. XML is echter eerder een alfabet dan een vocabulaire zoals in een eerder artikel reeds is aangegeven, dus voor het 'begrijpen' van services en de berichten die ze uitwisselen, zijn we aangewezen op standaarden die binnen een organisatie worden afgesproken, of tussen partners in een marktsegment. Bekende voorbeelden hiervan zijn ebXML, HL7 en LIPS/IMS.

> Klein beginnen maakt de kans op succes vele malen groter

Hoewel de standaarden voor de technische koppeling dus steeds beter en volwassener worden, zijn de zogenaamde verticale standaarden voor webservices nog in volle ontwikkeling.

PERFORMANCE De services-architectuur steunt voor wat betreft de technische implementatie zwaar op XML. De eXtensible Markup Languauge is een zeer flexibele en breed toepasbare standaard, maar de andere kant van de medaille is, dat het ook zeer breedsprakig (*verbose*) is. Om XML ook leesbaar te maken voor mensen (een van de andere voordelen van XML!), is het vaak nodig om lange(re) namen te gebruiken voor XML-tags, en om andere (meta-)informatie op te nemen in het bericht. Om iets vergelijkbaars qua informatie over de lijn te sturen, verbruikt een binair protocol beduidend minder

bandbreedte dan XML, wat voelbaar en merkbaar is in de performanceverschillen tussen beide.

Vanzelfsprekend is performance dan ook een aandachtspunt bij het ontwerpen en toepassen van services. Er zijn verschillende oplossingsrichtingen die (naast elkaar) gebruikt kunnen worden om de performance-uitdaging van op XML gebaseerde services het hoofd te bieden.

- Een van de eerste oplossingsrichtingen die gehanteerd moet worden, is het gebruik van *document-style webservices* in plaats van RPC-style. In nr. 8/2004 van Software Release Magazine is reeds aangegeven dat document-style webservices beter bij concepten van service oriëntatie als loose coupling, asynchroniciteit en servicecontracten passen, maar dit type webservices biedt daarnaast als voordeel dat er over het algemeen een kleinere hoeveelheid berichten wordt verstuurd, dan het geval is bij RPC-style services.
- Daarnaast is het belangrijk dat de juiste mate van korreligheid (*granularity*) wordt nagestreefd bij het ontwerpen van services. Hoe grofmaziger de services, hoe minder berichten er over en weer worden uitgewisseld, wat een positief effect heeft op de overall performance.
- Een derde oplossingsrichting die de laatste tijd veel in ontwikkeling is, betreft het toepassen van allerlei *compressietechnieken*, die de grootte van berichten ook moet doen afnemen. Eén van de mogelijkheden is om dit op het niveau van het http-transport te doen, bijvoorbeeld door gebruik te maken van veelgebruikte tools als gzip. Daarnaast kan compressie worden toegepast op het bericht zelf, waarbij er zowel bij de consumer als bij de provider extra code moet worden toegevoegd om het bericht te comprimeren of te decomprimeren.
- Een vierde oplossingsrichting is de zogenaamde *virtualisatie van SOAP-berichten*. Deze techniek wordt vooral toegepast bij het versturen van bijvoorbeeld bestanden of andere (zeer) grote berichten. Bij virtualisatie wordt dan uitsluitend een referentie naar het grote bestand of bericht verstuurd in het SOAP-bericht, en wordt aangegeven waar het betreffende bestand staat. Met de verkregen referentie kan de consumer dan naar deze opslagservice, om het betreffende bestand te raadplegen. Het spreekt voor zich dat deze vorm van virtualisatie vooral wordt toegepast in scenario's waarbij het niet altijd volstrekt duidelijk is dat de (zware) inhoud van het bericht ook daadwerkelijk op dat moment nodig is. We kunnen hierbij bijvoorbeeld denken aan een service waarmee men kan samenwerken aan een bepaald document, en waarover SOAP-berichten worden uitgewisseld.

Er is enige tijd de roep geweest om binaire XML, waarbij de XML-tags automatisch versleuteld zouden worden naar een minder breedsprakig, binair formaat. De angst om de simpliciteit en openheid (sommigen vreesden de ontwikkeling van een gesloten binair XML-formaat waarmee ontwikkelaars afhankelijk zouden worden van een leverancier) van op tekst gebaseerde XML te verliezen en het feit dat er redelijke alternatieven zijn (zie boven) heeft deze roep om binaire XML wat doen verstommen.

> Zorg ervoor dat op een consistente wijze één visie wordt nagestreefd

Conclusie Na het lezen van de hordes die in dit artikel zijn beschreven zal het iedereen duidelijk zijn dat het pad naar een servicegeoriënteerde architectuur vol hindernissen is. Het goede nieuws is echter dat voor deze problemen oplossingen voorhanden of in ontwikkeling zijn. De risico's zijn grotendeels terug te voeren tot het kunnen weerstaan van de verleiding om het te groots aan te willen pakken en het omgaan met een nieuwe technologie. Deze risico's zijn zo oud als de informatietechnologie zelf en er is al veel ervaring met het omgaan met deze problemen. Het is derhalve goed mogelijk om met de geschetste problemen om te gaan en de realisatie van een services architectuur tot een succes te maken waarin ook daadwerkelijk (veel van) de door de analisten geschetste voordelen worden gehaald.

Hierbij geldt met nadruk dat een gewaarschuwd mens voor twee telt. Wees ten hoogste alert voor het optreden van de problemen die in dit artikel geschetst zijn. Richt binnen uw (project)organisatie een goed risicomanagementsproces in dat er op gericht is om problemen in een vroegtijdig stadium te signaleren en te bestrijden. Op die wijze zijn een hoop problemen goed te beheersen. De punten die wij hier genoemd hebben, zouden dan ook in geen enkele risicolijst voor een services architectuur mogen ontbreken.

Anton van Weel is managing consultant bij Capgemini, en is enterprise architect. Loek Bakker is als senior consultant werkzaam bij Capgemini, en is gespecialiseerd in architectuur en integratievraagstukken. Ad Strack van Schijndel is managing consultant bij Capgemini en werkzaam als procescoach voor innovatieve softwareontwikkeltrajecten.