**Meegezogen door de opkomst van webservices, wint de Service Oriented Architecture (SOA) aan populariteit. In nummer 5 van Software Release Magazine gaf Loek Bakker in zijn artikel aan dat op SOAP gebaseerde webservices op dit moment de belangrijkste enabler zijn van een SOA. SOAP kent meerdere implementaties, maar niet alle implementaties sluiten even goed aan bij de concepten van service-oriëntatie. Ditmaal legt de auteur uit waarom de ene stijl van webservices beter binnen service-oriëntatie past dan de andere stijl.**



# Service-oriëntatie met stijl

## *SOAP styles in een SOA*

De Service Oriented Architecture (SOA) is de laatste jaren aan een opmars bezig. Hoewel zeker niet nieuw (het idee van Service Oriëntatie was er al ver voordat we ooit van webservices en XML hadden gehoord), is er de laatste jaren een stijgende interesse voor de SOA. Gartner was in 1996 de eerste die sprak van een Service Oriented Architecture, en mede door de duw in de rug die het krijgt door de adoptie van webservices, is er de laatste tijd veel aandacht voor. Als architectuur voor de toekomst krijgt de SOA de nodige attentie van meer dan alleen de IT wereld. Gartner voorspelt zelfs dat in 2008 zo'n zestig procent van alle organisaties een SOA als richtlijn hanteren voor het ontwikkelen van applicaties en bedrijfsprocessen.

**Principes** Service Oriëntatie is een benadering in de (digitale) architectuur die IT middelen als locatie-onafhankelijke diensten op het netwerk beschouwt en waarvan de interface-beschrijvingen gepubliceerd en herleid (*discovered*) kunnen worden. De diensten binnen Service Oriëntatie zijn duidelijk gedefinieerd (*well-defined*), op zichzelf staand (*self-contained*) en zijn onafhankelijk van de context of de status van andere diensten.

Bij Service Oriëntatie staan drie concepten centraal:
- ontkoppeling (*loose coupling*);
- service-contracten;
- asynchroniciteit.

Via de juiste invulling van deze concepten, moet de abstractie tussen implementatie en interface die Service Oriëntatie beoogt worden bereikt.

**Ontkoppeling** De mate van koppeling (*coupling*) beschrijft de hoeveelheid gedeelde kennis die nodig is tussen een aanbieder en een aanvrager voor de uitwisseling van gegevens in een gedistribueerde omgeving. Hoe strakker gekoppeld systemen zijn, hoe meer gedeelde kennis de systemen bezitten. Het ene systeem heeft een grote afhankelijkheid van de werking en het gedrag van het andere systeem, wat meteen ook betekent dat een wijziging in het ene systeem vaak leidt tot de noodzaak om het andere systeem ook te wijzigen. Het adagium bij ontkoppeling is: deel de interface, niet de klasse (*class*).

Bij een ontkoppeld systeem is de gedeelde kennis minimaal en is kennis deze vastgelegd in een zogenaamd service-contract.

**Service-contracten** Een service-contract is feitelijk niet meer dan een interface-afspraak (een contract) tussen twee partijen over het versturen en verwerken van data. Zoals hierboven beschreven, wordt in het service-contract de gedeelde kennis tussen aanbieder en aanvrager vastgelegd en wordt hiermee een ontkoppeling beoogd. Een service-contract echter is noch een noodzakelijke, noch een afdoende voorwaarde voor ontkoppeling. Ook op contractniveau kan er namelijk sprake zijn van strakke koppeling, bijvoorbeeld omdat het contract beschreven is in een proprietary formaat. De sleutel van ontkoppeling en service-contracten is gelegen in het toepassen van open standaarden, die universeel zijn en door meerdere platformen begrepen en ondersteund worden. Bij webservices wordt hierom meestal gebruik gemaakt van de W3C standaard WSDL (Web Service Description Language) om het service-contract te definiëren.

**Asynchroniciteit** In principe moet een SOA altijd uitgaan van het principe van asynchrone com-

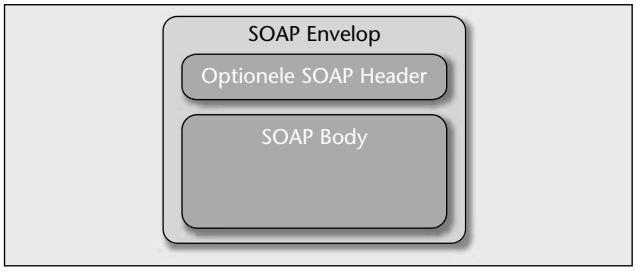
municatie. Bij een asynchrone uitwisseling voert het eindpunt niet direct de gewenste taak uit, maar geeft deze hooguit een bevestiging van de aanvraag aan de aanvrager. De taak wordt op een later tijdstip uitgevoerd, en daarna wordt het resultaat teruggegeven aan de aanvrager. Het kan zelfs zo zijn dat het eindpunt in het geheel geen bevestiging teruggeeft. Omdat eindpunten niet direct taken uitvoeren, wachten noch aanvrager, noch aanbieder op een antwoord. Ze sturen berichten, en kunnen vervolgens andere taken uitvoeren.

Asynchroniciteit is een min of meer noodzakelijke voorwaarde voor ontkoppeling tussen aanvrager en aanbieder, en is de basis voor de wijze waarop bedrijfsprocessen verlopen. Aangezien een SOA net zoveel gaat over bedrijfsprocessen als over technologie, lijkt het logisch om processen zoveel mogelijk asynchroon te laten verlopen. Bovendien betekent asynchrone communicatie dat de zender en de ontvanger niet op hetzelfde moment "in de lucht" hoeven te zijn. Dit maakt de koppeling minder gevoelig voor trage netwerken, *downtime* en piekbelasting van één van beide systemen.

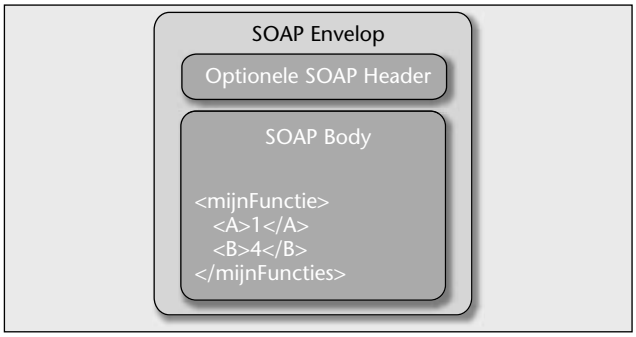
**De rol van SOAP** WSDL is een standaard waarmee een service-contract wordt gemaakt. WSDL is ontworpen voor gebruik met SOAP, waarbij SOAP het messaging transport is voor WSDL. Het bijzondere aan SOAP is dat het de eerste universeel geaccepteerde interface is, en in die hoedanigheid is het momenteel de belangrijkste enabler van de Service Oriented Architecture.

De WSDL standaard is echter een vrij brede standaard, waardoor je er veel kanten mee op kunt. SOAP webservices kunnen hierdoor op meerdere manieren geïmplementeerd worden, via verschillende SOAP stijlen die beide onderdeel uitmaken van de SOAP 1.1 specificatie. Als SOAP stijlen onderscheiden we de *document-oriented* stijl en de *RPC-oriented* stijl.

**Verschillende SOAP stijlen** De termen *document-oriented* en *RPC-oriented* zijn een artefact van de WSDL-specificatie. Diep weggestopt in de WSDL-specificatie zit een beschrijving van één van de meest wezenlijke onderdelen van de WSDL-standaard, namelijk de parameter die beschrijft of de SOAP implementatie *RPC georiënteerd* is, of *document* georiënteerd[1]. Een standaard




**Figuur 1. Een standaard SOAP-bericht**




**Figuur 2. De volgorde van de onderdelen is gelijk aan de volgorde van de parameters**

SOAP bericht ziet er uit zoals in Figuur 1. Een SOAP envelop bestaat uit minimaal een SOAP *body*, en een optionele SOAP *header*. Het verschil tussen *RPC* en *document* stijl is de manier waarop de *body* van het SOAP bericht wordt ingevuld.

*RPC stijl*

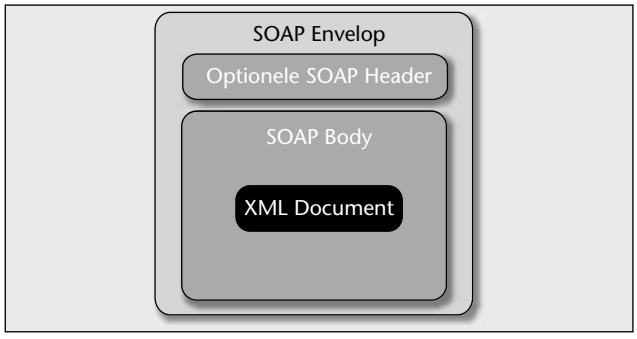
De RPC stijl kent berichten die parameters bevatten en returnwaarden teruggeven. RPC staat voor Remote Procedure Call, een term die bekend is binnen gedistribueerde omgevingen. Bij *RPC* stijl is ieder onderdeel van de *body* een parameter of een returnwaarde, en ieder onderdeel is opgenomen in een omvattend element binnen de *body* van het bericht. De naam van het omvattende element (*wrapper element*) is identiek aan de naam van de operatie, en de naam van ieder berichtonderdeel is identiek aan de naam van de bijbehorende parameter van de aanroep. De volgorde van de onderdelen is gelijk aan de volgorde van de parameters (zie figuur 2). Wat er bij RPC stijl feitelijk gebeurt, is dat het begrip van de context van de communicatie in de aanbieder én in de aanvrager gecodeerd moet zijn. De context is dus opgenomen in de code.

De RPC stijl was vooral in de beginjaren van SOAP populair, wat niet verwonderlijk is omdat webservices een groot deel van hun oorsprong vinden in het vinden van een oplossing voor het doen van remote calls naar componenten die op een ander platform of achter een firewall draaien. SOAP-berichten worden overwegend wel doorgelaten door de meeste firewalls, vooral ook omdat een standaardprotocol als HTTP gebruikt wordt, waarvan de firewall-poort 80 meestal openstaat. Daarnaast ondersteunt praktisch ieder platform XML.

De *RPC* stijl heeft de volgende kenmerken:

- Gebruik van methods, parameters en return waarden, op de wijze zoals dit is vastgelegd in de SOAP berichtstructuur.

1 Om precies te zijn, is dit het style attribuut van het element soap:operation, zie sectie 3.4 van de WSDL specificatie, http://www.w3.org/TR/wsdl




**FIGUUR 3. De document stijl gaat uit van een SOAP-bericht dat een XML document bevat.**

- Request/response uitwisseling: meestal wordt de *RPC* stijl toegepast voor een aanroep en een teruggave van een procedure of method.
- Meestal beperkt tot implementatie via HTTP, gezien de request/response kenmerken.

In de praktijk zien we dat met name veel op Java gebaseerde SOAP-implementaties gebruik maken van de *RPC* stijl, hoewel we de laatste tijd een tendens richting toepassing van de *document* stijl zien. Ook .NET Remoting in de HTTP-variant maakt gebruik van de *RPC* stijl. De overige Microsoft SOAP-implementaties (waaronder webservices in het .NET framework en de SOAP toolkit voor COM componenten) zijn gebaseerd op de *document* stijl.

*Document stijl*

De *document* stijl gaat uit van een SOAP-bericht dat een XML *document* bevat. In het geval van een document stijl zijn er geen omvattende elementen (zogenaamde *wrapper elements*), maar staat het enige berichtonderdeel, het XML document, direct onder het *body* element van het SOAP bericht (zie figuur 3).

Bij de *document* stijl zijn de berichten over het algemeen op zichzelf staand (*self-contained*), waarmee bedoeld wordt dat de context van het bericht in het document of bericht beschreven is, in plaats van in de code van de aanbieder en de aanvrager. Anders gezegd bevatten de berichten meestal een bepaalde mate van *state* van de uitwisseling. De kenmerken van de *document* stijl zijn:

- De SOAP body bevat een op zichzelf staand XML document.
- Uitwisseling vindt meestal plaats op basis van doorsturen van het bericht; er wordt doorgaans geen antwoord op de aanroep verwacht. De communicatie is over het algemeen asynchroon van aard.
- Implementatie via HTTP, SMTP, FTP of andere internetprotocollen; niet beperkt tot HTTP.

In tegenstelling tot RPC stijl interfaces, wordt bij *document* stijl vaak gebruik gemaakt van XSD schema definities om het XML document te beschrijven dat in het SOAP-bericht wordt verstuurd. XSD staat voor XML Schema Definition, en is de open standaard voor het beschrijven van de structuur en de data typen in een XML document[2]. De WSDL wordt bij *document* stijl gebruikt om de SOAP-specifieke details van de uitwisseling te beschrijven, zoals transportprotocol en eindpuntidentificatie. Maar wat het SOAP bericht verder vooral doet, is het "verpakken" van het XML bericht en dit versturen naar het eindpunt, zonder dat er kennis is van de inhoud of betekenis van het XML bericht.

**STIJLEN IN RELATIE TOT SOA** Indien we de hierboven beschreven kenmerken van de beide stijlen mappen naar de concepten van Service Oriëntatie, dan zien we het volgende:

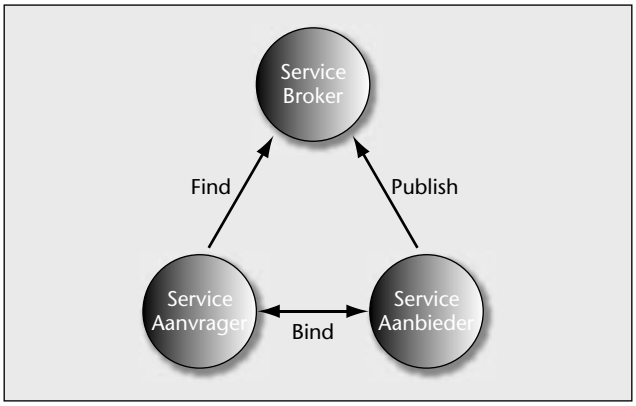
*Ontkoppeling*

RPC stijl is gebaseerd op parameters van het onderliggende component, dus meer afhankelijk van (technische) implementatie. Daardoor zijn op *RPC* stijl gebaseerde webservices minder ontkoppeld dan het geval is bij *document* stijl, waar er geen afhankelijkheid is van de onderliggende implementatie. In het traditionele plaatje van een Service Oriented Architecture wordt de rol van Service Broker onderscheiden (zie figuur 4).

In een situatie zoals bij *RPC* stijl webservices lijkt de rol van Service Broker overbodig. Webservices worden bij *RPC* stijl vooral ingezet als een verbeterde manier waarop aanvragers kunnen communiceren met aanbieders. De SOA echter vraagt om een andere aanpak: in een SOA mag een aanvrager nooit zodanig geprogrammeerd zijn, dat deze rechtstreeks communiceert met de service aanbieder. Iedere wijziging in de aanbieder leidt in een dergelijk scenario namelijk tot een noodzakelijke wijziging in de aanvrager. Hiermee zijn aanvrager en aanbieder verre van ontkoppeld. In een SOA is de rol van de Service Broker juist belangrijk voor de gewenste ontkoppeling. Daarnaast verliest de *RPC* stijl een ander belangrijk aspect voor ontkoppeling uit het oog, namelijk dat in principe de aanvrager bepaalt hoe de applica-

> Het adagium bij ontkoppeling is: deel de interface, niet de klasse

2 Zie ook http://www.w3.org/XML/Schema




**FIGUUR 4. In het traditionele plaatje van een SOA wordt de rol van Service Broker onderscheiden**

tie zich gedraagt, en niet de aanbieder. De aanvrager bepaalt hoe hij de dienst geleverd wil krijgen door de aanbieder. Bij *RPC* stijl webservices werkt dit meestal omgekeerd: de aanbieder bepaalt hoe de betreffende service gebruikt kan en mag worden.

*Service-contracten*

Ook ten aanzien van service-contracten zijn er grote verschillen. We hebben eerder gezien dat bij webservices de WSDL dienst doet als service-contract. Naast het service-contract zijn er natuurlijk ook andere afspraken tussen de communicerende partijen, zoals syntax, semantiek en processturing (routering van berichten). Deze onderdelen van het contract worden bij *RPC* stijl allemaal vastgelegd in de WSDL, bij *document* stijl is dat veelal in een XSD en meestal niet in de WSDL. Door de koppeling van de WSDL aan de implementatie-specifieke kenmerken zoals we die zien bij *RPC*, bevat het contract bij *RPC* stijl meer gedeelde kennis, en zal het contract meer aan verandering onderhevig zijn dan het geval is bij *document* stijl.

Bij *document* stijl webservices worden uitsluitend die gegevens in de WSDL vastgelegd die nodig zijn om tot communicatie te komen, bij *RPC* stijl interfaces wordt er naast deze informatie, ook vastgelegd welke gegevens er worden uitgewisseld, hoe deze gecodeerd zijn, van welk type ze zijn, et cetera.

Wat we bij document stijl zien, is dat uitsluitend de "verpakking" van het XML document via SOAP geregeld wordt, maar dat de rest vastgelegd in (onafhankelijk) extern XSD schema. Veel ontwikkelaars zijn in eerste instantie geneigd tot het creëren van één service-contract voor één component (dit is iets wat wordt aangemoedigd door allerlei *toolkits* waarmee eenvoudig een WSDL gegenereerd kan worden op basis van een bestaande component). Wat dan snel vergeten wordt, is dat op deze manier een nauwe koppeling wordt bewerkstelligd tussen interface en implementatie: een wijziging in de implementatie leidt ook hier tot de noodzaak om ook de interface te wijzigen.

*Asynchroniciteit*

*RPC* stijl is gebaseerd op het principe van request/response. Hoewel dit Message Exchange Pattern (MEP) ook asynchroon toegepast kan worden, zien we in de praktijk meestal een synchrone toepassing, over het algemeen via het HTTP protocol. De reden dat meestal voor synchroon via HTTP wordt gekozen, is het feit dat synchrone aanroepen eenvoudig te programmeren zijn, en dat gebruik gemaakt kan worden van transport-level correlatiemechanismen om vraag en antwoord te combineren.

Bij *document* stijl zien we dat, omdat de XML documenten op zichzelf staan en context bevatten, uitwisseling niet volgens een request/response scenario verloopt, maar eerder via eenrichtingsverkeer (bijvoorbeeld via publish-subscribe). XML documenten worden vaak door meerdere services gerouteerd. Omdat de documenten op zichzelf staan en dus onafhankelijk van verdere context gebruikt en geïnterpreteerd kunnen worden, is het bovendien eenvoudiger om een asynchroon programmeermodel te hanteren voor *document* stijl webservices. Alle informatie voor correlatie en routering staat immers in het document.

**CONCLUSIE** Op SOAP gebaseerde webservices zijn de drijvende kracht achter de recente opmars van Service Oriëntatie en de SOA. SOAP webservices zijn er in meerdere smaken, en indien de kenmerken van de verschillende stijlen gekoppeld worden aan de principes en concepten van Service Oriëntatie, dan blijken webservices die gebaseerd zijn op de *document* stijl beter aan te sluiten bij een SOA dan webservices gebaseerd op een *RPC* stijl. *Document* stijl gebaseerde services ondersteunen beter de principes van asynchroniciteit en ontkoppeling, en de wijze waarop invulling wordt gegeven aan service-contracten sluit beter aan bij een SOA dan het geval is bij *RPC* stijl. Indien een organisatie webservices ziet als een eerste stap richting de adoptie van een SOA[3], dan doet zij er verstandig aan deze webservices te baseren op de *document* stijl, en de *RPC* stijl uitsluitend toe te passen als het enige doel van de webservice is het via XML ontsluiten van een bepaalde component.

3 IBM ziet de implementatie van webservices als een potentieel startpunt voor de implementatie van een SOA, http://www-306.ibm.com/software/info/openenvironment/soa/soa-adoption.html

*Loek Bakker is als senior consultant werkzaam bij Capgemini, en is gespecialiseerd in architectuur en integratievraagstukken.*