Webservices zijn in principe uitstekend geschikt om applicaties geschreven in verschillende programmeertalen en draaiend op heterogene systemen met elkaar te laten communiceren. In de praktijk blijkt echter dat de interoperabiliteit van webservices geen vanzelfsprekendheid is. In dit artikel gaat Willem Koppenol in op interoperabiliteit tussen met name op Java en .NET gebaseerde webservices en bespreekt en passant de WS-I (Web Service Interoperability) Basic Profile standaard die door samenwerkende leveranciers in het leven is geroepen.

achtergrond

Webservice interoperabiliteit

Vooruitgang rond standaardisering

Webservices zijn gebaseerd op de uitwisseling van tekstuele berichten in XML-formaat. Dergelijke berichten kunnen in iedere taal en op ieder platform worden geproduceerd en gelezen. Daar komt nog bij dat ze deze berichten meestal uitwisselen via het alom tegenwoordige HTTP-protocol. In de praktijk blijkt echter dat de interoperabiliteit van webservices geen vanzelfsprekendheid is. Webservice standaarden als SOAP, WSDL en UDDI zijn in de loop der tijd geëvolueerd en door de verschillende toolkits anders geïnterpreteerd.

STANDAARDISERING Huidige interoperabiliteitsproblemen van webservices hebben meestal te maken met encoding types, data types, default waarden en namespaces. Voor het verdere succes van webservices als basis van een Service Oriented Architectecture (SOA) is een antwoord op interoperabiliteitsproblemen van wezenlijk belang. Diverse leveranciers waaronder Microsoft, Sun en IBM werken daartoe sinds enkele jaren samen in de Web Services Interoperability (WS-I) standaardiseringsorganisatie. Een eerste standaard van deze organisatie is het WS-I Basic Profile. Door de richtlijnen van deze standaard aan te houden worden, krijgen webservices de basis kenmerken voor interoperabiliteit. Andere aspecten van webservices zoals security, transacties en orchestratie zijn verschoven naar andere standaarden en worden in deze eerste standaard buiten beschouwing gehouden. De aandacht voor interoperabiliteit lijkt zijn vruchten al af te werpen. In de nieuwere versies van diverse webservice toolkits is veel meer aandacht voor interoperabiliteit dan in het verleden, zodat de problemen minder lijken te worden.

SOAP ENCODING Eén van de grootste problemen bij de interoperabiliteit van webservices heeft zijn oorzaak in het gebruik van de SOAP encoding stijl. De problemen van SOAP encoding worden in het kaderartikel (zie pagina 13) nader toegelicht. Kort gezegd beschrijft SOAP encoding hoe je de datatypes en structuren uit een bepaalde taal serialiseert tot een XML-bericht. Zolang je bij het deserialiseren gebruikt maakt van dezelfde taal en encoding library's is er niets aan de hand. Hoe de deserialisatie verloopt naar een andere taal met andere library's is echter met name voor complexe typen niet volledig gestandaardiseerd en dit is natuurlijk niet erg interoperabel. De problemen met SOAP Encoding hebben ertoe geleid dat deze SOAP stijl in het WS-I Basic Profile niet is toegestaan.

```
<soap:Envelope
     xmlns:mrns0="urn:xmethods-delayed-quotes"
     xmlns:soap="http://schemas.xmlsoap.org/
     soap/envelope/"
     xmlns:soapenc="http://schemas.xmlsoap.
     org/soap/encoding/"
     xmlns:xs="http://www.w3.org/2001/
     XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/
     XMLSchema-instance">
       <soap:Body soap:encodingStyle="http://
       schemas.xmlsoap.org/soap/encoding/">
```

```
        <mrns0:getQuote>
            <symbol xsi:type="xs:
string">IBM</symbol>
        </mrns0:getQuote>
     </soap:Body>
  </soap:Envelope>
```

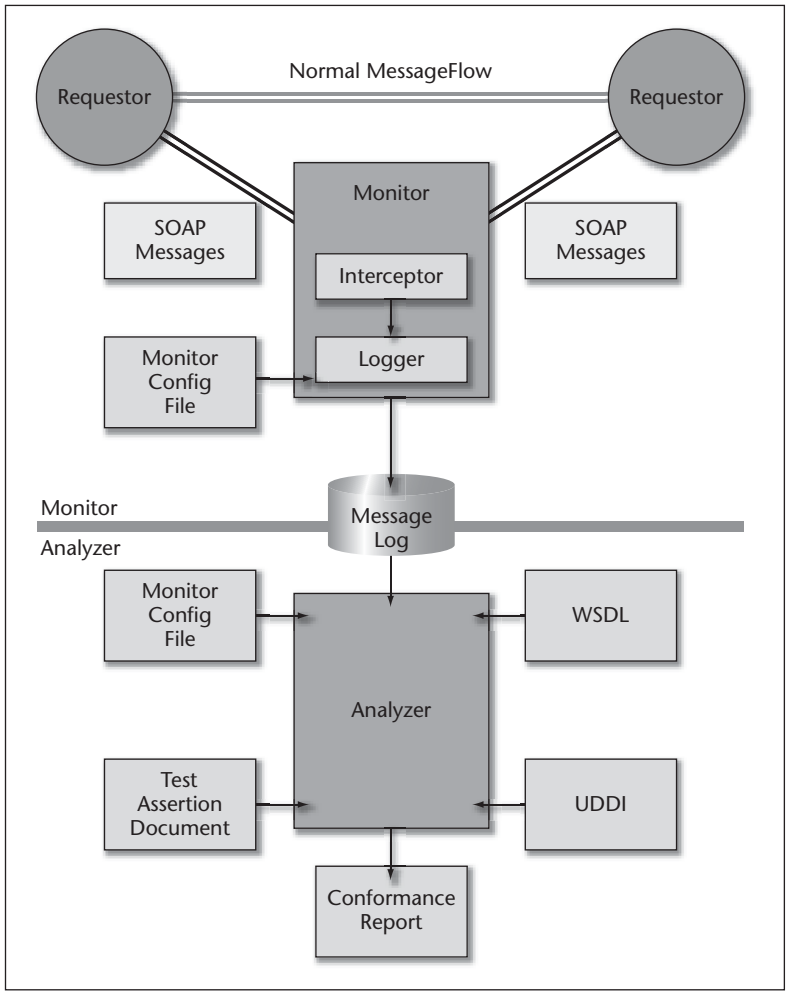
JAVA VERSUS .NET Overigens gebruik(t)en de meeste SOAP tools op het Java platform de SOAP Encoding stijl altijd als default. Onder die tools bevinden zich de oorspronkelijke Apache SOAP toolkit, haar opvolger, de op SAX gebaseerde Apache Axis implementatie, en IBM WebSphere. Deze toolkits maken bovendien standaard webservices waarin de SOAP berichten de vorm van een RPC (Remote Procedure Call) hebben. Een functienaam is dan het root element en de parameters zijn de subelementen. Visual Studio.NET gebruikt standaard echter de literal-stijl voor webservices:

```
<soap:Envelope
    xmlns:s0="http://ws.cdyne.com/"
    xmlns:soap="http://schemas.xmlsoap.org/
soap/envelope/"
    xmlns:xs="http://www.w3.org/2001/
XMLSchema">
    <soap:Body>
      <s0:GetQuote>
          <s0:StockSymbol>IBM</s0:StockSymbol>
            <s0:LicenseKey />
        </s0:GetQuote>
     </soap:Body>
  </soap:Envelope>
```

In deze stijl volgen de typedefinities in de SOAP berichten letterlijk een XML schema-definitie uit het WSDL-document. Voorts maakt Visual Studio.NET standaard webservices waarin de SOAP berichten de vorm van een document (XML-document in XML-schema omschreven) hebben. Tabel 1 geeft een overzicht van het gebruik van de verschillende webservice stijlen.

De toolkits in Java en .NET hanteren dus historisch gezien een andere default webservice stijl. De nieuwere versies van de toolkits kunnen ook andere stijlen vrij eenvoudig worden toegevoegd, zodat dit op zichzelf geen belemmering is voor interoperabiliteit.

BEGINNEN VANUIT WSDL Een WSDL document beschrijft de interface van een webservice waaronder de berichten die naar een webservice kunnen worden toegestuurd. Veel tegenwoordige IDE tools zijn in staat het WSDL document te genereren uit implementatiecode. Op zich is deze automatisering natuurlijk prachtig. Het zorgt voor een verhoging van de productiviteit. In de praktijk passen ook door de wol geverfde ontwikkelaars dit vaak toe. Zolang client en server op hetzelfde platform draaien en gebruik maken van dezelfde toolkits, zal dit niet leiden tot interoperabiliteitsproblemen. Het wordt anders in een heterogene omgeving waarbij client en server op een ander platform draaien en gebruik maken van andere toolkits. De client proxy zal door het client tool worden afgeleid van de door het server tool genereerde WSDL. In een dergelijke situatie is er sprake van een meervoudige mapping waarbij gemakkelijk informatie verloren kan gaan. Het is daarom beter het



FIGUUR 1. Testing tools architectuur

Webservice stijl	Praktijk gebruik
RPC/Encoded	Veel in Java omgevingen. Niet toegestaan in het WS-I Basic Profile
RPC/Literal	Zelden in de praktijk gebruikt Toegestaan in WS-I Basic Profile
Document/Encoded	Zelden in de praktijk gebruikt Niet toegestaan in het WS-I Basic Profile
Document/Literal	Standaard van .NET. De facto standaard voor Web Service interoperabiliteit

TABEL 1. Een overzicht van verschillende webservice-stijlen

Artifact: discovery
Assertion Result Summary:

Assertion ID	Passed	Failed	Warning	Not Applicade	Missing Input
WSI3001	0	0	0	0	x
WSI3002	0	0	0	0	x
WSI3003	0	0	0	0	x
WSI3004	0	0	0	0	x
WSI3006	0	0	0	0	x
WSI3007	0	0	0	0	x

FIGUUR 2. Resultaat van WS-I Assertions door de analyzer

WSDL als startpunt te gebruiken bij het ontwerp. Door met name eerst de data types te definiëren in XML Schema en vervolgens classes op basis daarvan te genereren wordt de interoperabiliteit van datatypen zeker gesteld. De tools hoeven niet overboord te worden gezet maar kunnen een WSDL raamwerk aanleveren waarin vervolgens schema's, messages en data binding worden aangebracht.

DATA TYPEN Het verschil in data typering tussen verschillende platformen geeft ook aanleiding tot interoperabiliteitsproblemen. De typen die het bij interoperabiliteitstesten meestal laten afweten zijn numerieke en datum-typen. Wanneer we ons beperken tot de verschillen tussen Java en .NET dan zijn er onder meer de volgende zaken om rekening mee te houden:

Numerieke typen

Decimale data types hebben last van taalafhankelijkheden als de grenzen worden opgezocht. `BigDecimal types` zijn daar een goed voorbeeld van. Decimale getallen met veel cijfers achter de komma, zijn een noodzakelijk onderdeel van financiële berekeningen in bijvoorbeeld bancaire applicaties, waar geen digits verloren mogen gaan. Op Java gebaseerde SOAP-implementaties zoals Axis gebruiken Java's `BigDecimal`, waarvan de precisie afhangt van het onderliggende besturingssysteem. `BigDecimal's` in .NET hebben echter een precisie tot 29 digits. De vraag is wat er met de extra digits gebeurt als Axis SOAP een `BigDecimal` van .NET ontvangt.

Datum typen

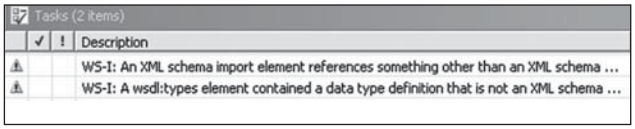
In Java zijn `java.util.Date` en `java.util.Calendar` reference types. In .NET is System.DateTime een value type. Een reference type kan de waarde null hebben, een value type niet. Als je een `Date` object in Java de waarde null geeft, zal .NET bij deserialisatie naar een `DateTime` object een exceptie optreden De exceptie kan worden voorkomen door het `Date` object te verpakken in een complex type en de waarde van het complex type op null te zetten. Ook blijkt uit de praktijk dat bij uitwisseling van datum en tijd typen tussen .NET en Java, de waardes in Java beter met een toepasselijke `compareTo()` methode kunnen worden vergeleken dan met ==. Met name in het milliseconden-bereik, treden anders inconsistenties op.

Namespaces

Ook bij het gebruik van namespaces moeten in verband met interoperabiliteit de nodige voorzorgen worden genomen. Wanneer bijvoorbeeld een tweetal webservices in .NET de namespaces URI's http://mydomain/accounting en http://mydomain/sales krijgen, en een webservice client in IBM WebSphere 5.1.2 wil deze webservices gebruiken, dan zullen beide namespaces bij de proxy-generatie uit het WSDL-document op hetzelfde Java package `mydomain` worden gemapt. Dit geeft problemen omdat de proxy klassen in dezelfde directory gaan conflicteren. Het gaat goed als we de namespaces van de webservices veranderen in http://accounting.mydomain en http://sales.mydomain. Nu krijgen beide webservices bij proxy generatie een eigen package. IBM WebSphere gebruikt namelijk voor de mapping van de namespace naar het package alleen het domein gedeelte van de namespace URI. Met namespaces zijn meer problemen die ook per toolkit kunnen verschillen. Het is in ieder geval aan te bevelen om waar mogelijk unieke namen te specificeren en geen defaults aan te laten staan.

WS-I RICHTLIJNEN Een goed begin voor interoperabiliteit is de richtlijnen te volgen van de WS-I organisatie. In het Basic Profile heeft de WS-I een reeks richtlijnen vastgelegd waarin staat beschreven, hoe bepaalde onderdelen van webservice specificaties zoals SOAP, WSDL en UDDI moeten worden gebruikt om interoperabele webservices te verkrijgen. In een richtlijn wordt door middel van kernwoorden als "MUST", "MUST NOT", "SHALL", "MAY" precies geformuleerd welke implementaties van onderdelen uit een specificatie zijn toegestaan. Een voorbeeld is hoe moet worden omgegaan met zogeheten SOAP trailers, zusterelementen van het `<soap:Body>` element. De interpretatie van dergelijke elementen is niet duidelijk omschreven in de SOAP-specificatie. Daarom zijn deze volgens richtlijn R1011 van Basic Profile verboden:

"R1011 A MESSAGE MUST NOT have any element children of soap:Envelope following the soap:Body element."



FIGUUR 3. WS-I conformance meldingen

```
INCORRECT:
<soap:Envelope xmlns:soap='http://schemas.
xmlsoap.org/soap/envelope/' >
  <soap: body>
     <p:Process xmlns:p='http://example.org/
     Operations' />
  </soap:Body>
  <m:Data xmlns:m='http://example.org/infor-
mation' >
  Here is some data with the message
  </m:Data>
</soap: Envelope>

CORRECT
<soap:Envelope xmlns:soap='http://schemas.
xmlsoap.org/soap/envelope/' >
  <soap: Body>
     <p:Process xmlns:p='http://example.org/
     Operations' />
        <m:Data xmlns:m='http://example.org/
        information' >
  Here is some data with the message
        </m:Data>
     </p:Process>
  </soap:Body>
</soap: Envelope>
```

WS-I TEST TOOLS Het WS-I levert naast papieren standaardisatie documenten ook een aantal test tools. Deze tools kunnen gebruikt worden om het berichten verkeer tussen webservices te analyseren. Zo kan het monitor tool als 'man in the middle' tussen client en webservice worden geplaatst en alle op en neer gaande berichten loggen. Het is sowieso al nuttig om via tracing te zien wat de precieze inhoud is van de geproduceerde SOAP berichten. De gelogde berichten zijn vervolgens input voor het analyzer tool dat een rapport produceert, waarin wordt opgesomd in hoeverre de webservice conformeert aan het WS-I Basic profile.

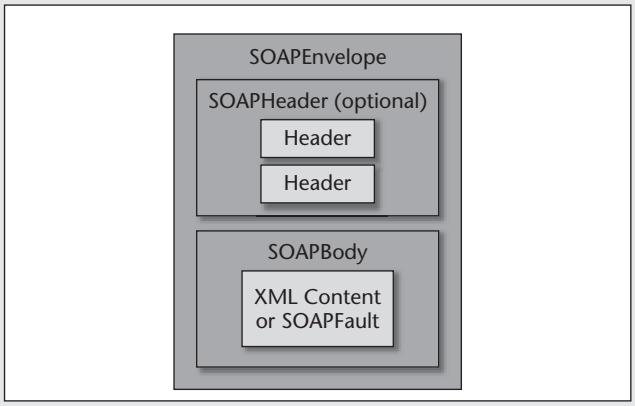
SLOTWOORD Interoperabiliteit van met name Java en .NET webservices gaat niet vanzelf, maar is met de nodige voorzorgsmaatregelen rond data types en namespaces zeker te realiseren. Met name door webservices te baseren op het gestandaardiseerde WS-I Basic Profile, waarin de SOAP encoding stijl is verboden, is interoperabiliteit mogelijk. Toolkit leveranciers besteden veel meer aandacht aan interoperabiliteitskwesties dan in de beginjaren van webservices het geval was, zodat er ook meer toolkit ondersteuning is bij het realiseren van interoperabiliteit. De webservice technologie met haar onderliggende standaards en specificaties is echter relatief nieuw en bovendien nog sterk in ontwikkeling. Interoperabiliteitsproblemen zijn dan ook nog wel te verwachten bij de nieuwere webservices standaards op het gebied van security, transacties en orchestratie. Hoopvol is daarbij dat de WS-I organisatie inmiddels ook een Attachments Profile en Simple SOAP Binding Profile standaard heeft geformuleerd en dat er druk wordt gewerkt aan een Basic Security Profile.

Het probleem met SOAP encoding

Het meeste uit de SOAP specificatie is een solide ondergrond voor webservices. SOAP encoding echter, ook wel genoemd section 5 encoding naar de sectie van de SOAP specificatie waarin deze is gedefinieerd, is een erfenis uit het verleden, waarvoor geen plaats is in de webservices van de toekomst. Dit wordt duidelijk als we de ontwikkeling van de SOAP en WSDL standaarden voor webservices nader bekijken, in historisch perspectief plaatsen en een voorbeeld beschouwen.

SOAP SOAP is één van hoekstenen van de webservices protocol stack. De SOAP specificatie formaliseert het gebruik van XML berichten voor gedistribueerde communicatie en definieert een uitbreidingsmodel, de representatie van protocol en applicatie fouten en richtlijnen voor het mappen van RPC aanroepen op SOAP berichten. SOAP berichten bestaan uit XML content in de SOAP body, verpakt in een SOAP envelope en eventueel voorzien van SOAP headers (zie Figuur 4).

DATAMODEL Toen webservices een aantal jaren geleden nog in hun kinderschoenen stonden, was het noodzakelijk een standaard te definiëren om algemene object georiënteerde programmeer constructies te vertalen naar XML. XML schema bestond nog niet, dus zagen de SOAP auteurs zich genoodzaakt een eigen SOAP data model te ontwerpen. Het object model wordt gerepresenteerd door een graph van untyped nodes die met pijlen met elkaar zijn verbonden. De nodes hebben waarden en de pijlen hebben labels. Als nodes uitgaande pijlen hebben zijn het samengestelde waarden, als ze alleen inkomende pijlen hebben zijn het simpele waarden. Het data model (zie Figuur 5) voor een `Point` object kan dus als volgt voorgesteld worden:



FIGUUR 4. Structuur van een SOAP bericht

```
class Point {
int x;
int y;
}
Point p = new Point(12,34);
```

Soms is het in een data model nodig om met behulp van id's meer dan eens te refereren aan hetzelfde object. Het SOAP data model ondersteunt dit doordat een node meer ingaande pijlen kan hebben, de zogeheten multirefs waarden. Ook circulaire referenties zijn geen probleem.

ENCODING SOAP encoding regels bepalen vervolgens hoe een data model instantie wordt geserialiseerd naar een SOAP bericht. De basis regels zijn simpel: uitgaande pijlen worden XML-elementen. Ze bevatten tekst als ze naar een eind node wijzen. Ze bevatten subelementen als ze naar een node wijzen die zelf uitgaande pijlen heeft. Als een object graph met multirefs volgens SOAP encoding wordt uitgeschreven, verwijzen id attributen naar objecten die al eerder uitgeschreven zijn.

```
<Point soapenv:encodingStyle="http://www.
w3.org/2003/05/soap-encoding">
    <x>12</x>
    <y>34</y>
</Point>
```

De SOAP encoding regels zijn krachtig, ondersteunen complexe object graphs en polymorphismen. SOAP encoding kan gemakkelijker een object model representeren dan een hiërarchische structuur als XML-schema. Om deze reden heeft het ook grote aantrekkingskracht. Het werd aan de SOAP implementaties overgelaten om hun eigen datastructuren te vertalen naar het SOAP data model.

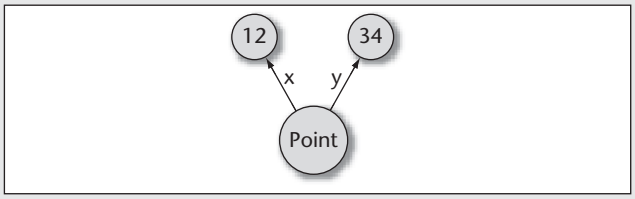
STIJLEN De SOAP body kan opgebouwd worden in RPC- of documentstijl. SOAP heeft zijn oorsprong in synchrone RPC (Remote Procedure Calls) over HTTP. Bij de RPC-stijl bevat de SOAP body slechts één element dat genoemd is naar de operatie waarop het SOAP bericht betrekking heeft. De parameters van de operatie worden gerepresenteerd door subelementen. Een aanroep van de functie:

```
float Distance(Point p1, Point p2)
{
  // apply the Pythagorean Theorem
}
Point one = new Point(10, 20);
Point two = new Point(100, 200);
float f = proxy.Distance(one, two);
```

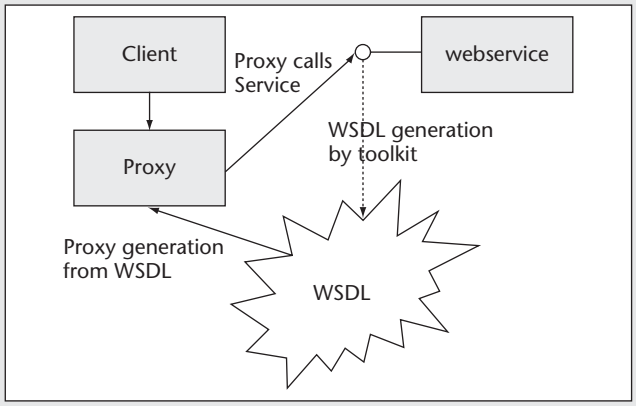
zal in de SOAP RPC stijl resulteren in het volgende SOAP-bericht:

```
<soap:Envelope xmlns:soap="http://schemas.
xmlsoap.org/soap/envelope/">
  <soap:body soap:encodingStyle="http://sche-
mas.xmlsoap.org/soap/encoding/">
    <ns:Distance xmlns:ns="http://www.
release.com/encoding/demo">
      <p1>
        <x>10</x>
        <y>20</y>
      </p1>
      <p2>
        <x>100</x>
        <y>200</y>
      </p2>
    </ns:Distance>
  </soap:Body>
</soap:Envelope>
```

De RPC-stijl is voornamelijk populair door het eenvoudige programmeermodel voor ontwikkelaars die gewend zijn aan Remote Procedure Calls. Later kwam men tot het inzicht dat ook een willekeurig XML document in de SOAP body kan worden verpakt en als SOAP-bericht kan worden verstuurd. De documentstijl kent geen speciale formatteringsregels. Een consequentie van de keuzevrijheid die de documentstijl biedt, is dat SOAP berichten die eraan voldoen, er exact hetzelfde kunnen uitzien als hun RPC-equivalent.



FIGUUR 5. SOAP's data model



FIGUUR 6. Het aanroepen van een webservice via een proxy

WSDL Naarmate webservices geaccepteerd raakten ontstond de behoefte het interface van SOAP-servermethoden te kunnen achterhalen, zodat op basis daarvan clients konden worden gebouwd. En zo ontstond een andere hoeksteen van de webservices protocol stack, de WDSL (Web Services Description Language). WSDL documenten beschrijven de interface van een webservice in XML. Op basis van WSDL kunnen tools een client proxy genereren. De client kan de webservice vervolgens via deze proxy aanroepen (zei Figuur 6).

De auteurs van WSDL realiseerden zich het belang van XML schema voor het beschrijven van types in SOAP berichten. Met XML schema kan immers de structuur en inhoud van een SOAP bericht gevalideerd worden. Bovendien heeft XML schema een rijk en uitbreidbaar type systeem. Het eerder besproken Point type en het input bericht voor de aanroep van de Distance functie zien er in XML schema als volgt uit:

```
<xsd:complexType name="Point">
        <xsd:sequence>
          <xsd:element name="x" type="xsd:
int" />
          <xsd:element name="y" type="xsd:
int" />
        </xsd:sequence>
</xsd:complexType>

<wsdl:message name="DistanceInput">
   <wsdl:part name="p1" type="tns:Point" />
   <wsdl:part name="p2" type="tns:Point" />
 </wsdl:message>
```

BINDING EN ENCODING Het <binding> element in WSDL beschrijft de koppeling met een concreet netwerk- en messaging-protocol (vaak SOAP over HTTP) en het formaat en encoding van de SOAP body. Hier wordt het eerder besproken onderscheid tussen de RPC- en documentstijl gemaakt. De default in WSDL is document binding. En hoewel de berichten in WSDL beschreven worden in XML schema, beschrijven deze schema's de opbouw van de berichten niet altijd letterlijk. Ten tijde van de inceptie van WSDL waren er namelijk al een aanzienlijk aantal toolkits op de markt waren die een SOAP data model via SOAP encoding naar XML formaat vertaalden. De ontwerpers van WSDL zagen ze zich genoodzaakt dit te omarmen. De oplossing die werd gekozen was om de berichten te definiëren in termen van XML schema, maar ook een verschillende encoding van de berichten toe te staan. Het use attribuut van het <soap:body> element heeft als mogelijke waarden literal en encoded. Bij *literal* specificeren de XML schema types in WSDL de exacte inhoud van de SOAP berichten. Bij *encoded* zijn het abstracte specificaties. De inhoud van de SOAP berichten wordt concreet nadat de SOAP encoding erop toegepast wordt.

```
<wsdl:binding name="DemoBinding" type="tns:
Demo">
    <soap:binding style="rpc"
    transport="http://schemas.xmlsoap/http"
    <wsdl:operation name="Distance">
      <soap:operation soapAction=""
      style="rpc" />
      <wsdl:input message="tns:DistanceInput">
        <soap:body namespace="http://demo"
        use="encoded" />
      </wsdl:input>
      <wsdl:output message="tns:
DistanceOutput">
        <soap:body namespace="http://demo"
        use="encoded" />
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```

SOAP encoding gaat echter over het vertalen een SOAP data model, een graph van untyped structures, naar XML. De vraag is dus hoe SOAP encoding kan worden toegepast op XML schema constructies die een door een hiërarchische structuur getypeerde reeks elementen vormen. Noch de SOAP-specificatie noch de

WSDL element	Beschrijving
<types>	Definities in XML schema van de gebruikte types
<message>	Eenrichtingsverkeer berichten met types uit <types>
<portType	Operaties gedefinieerd in termen van input en output berichten
<binding>	Koppeling van operaties met protocol en codering van SOAP body
<service>	Naam en url-adres van de webservice

TABEL 2. Hoofdelementen van een WSDL-document

WSDL specificatie geven hier antwoord op. En hier ligt de kern van het probleem van SOAP encoding.

Dat zien we als we kijken naar de aanroep van de eerder genoemde Distance functie, maar nu met twee keer hetzelfde object:

```
Point one = new Point(10, 20);
float f = proxy.Distance(one, one);
```

Bij het serialiseren van deze aanroep gebruikt de client het SOAP encoding schema voor "multi-reference accessors" en ziet het SOAP bericht er als volgt uit:

```
<soap:Envelope xmlns:soap="http://schemas.
xmlsoap.org/soap/envelope/">
  <soap:Body soap:encodingStyle="http://sche-
mas.xmlsoap.org/soap/encoding/">
    <ns:Distance xmlns:ns="http://www.
release.com/encoding/demo">
      <p1 HREF="#id1"/>
      <p2 HREF="#id1"/>
    </ns:Distance>
    <ns:Point id="id1" xmlns:ns="http://www.
release.com/encoding/demo">
      <x>10</x>
      <y>20</y>
    </ns:Point>
  </soap:Body>
</soap:Envelope>
```

Uit dit bericht is het duidelijk dat p1 en p2 niet overeenkomen met de XML schema definitie voor `Point` types. Dus hoewel in WSDL staat gespecificeerd dat het input bericht voor de `Distance` operatie een tweetal `Point` types moet bevatten ziet dit bericht er bij het gebruik van SOAP encoding anders uit. Het wordt nu erg moeilijk te garanderen dat de berichten die een webservice binnen krijgt correct zijn opgebouwd.

Hoewel het "multi-reference accessors" encoding mechanisme krachtig is, blijft het moeilijk om uit te drukken in XML schema, met als gevolg sterk verschillende implementaties per platform. Naar verwachting zal SOAP encoding daarom in de toekomst steeds meer naar de achtergrond verdwijnen. De SOAP 1.2 Specificatie maakt ondersteuning voor SOAP encoding optioneel. Een toolkit kan dus SOAP 1.2 compliant zijn zonder SOAP encoding te ondersteunen.

drs. Willem Koppenol is senior trainer en product specialist software development bij Twice IT Training (wkoppenol@twice.nl)

PATCHES Patches PATCHES Patches PATCHES Patches PATCHES

Artikelen over onderwerpen als software-ontwikkeling, Java, UML, eXteme Programming en nog veel meer vindt u in het Online Archief van Array Publications. Vaktijdschriften als Software Release, Java Magazine, Database Magazine en ons Oracle vakblad Optimize hebben hun artikelenarchief online gezet. Dankzij de heldere zoekstructuur vindt u snel wat u zoekt op www.release.nl.

Nieuwe versies OptimalJ, DevParter en Vantage

Compuware heeft van een aantal van zijn producten nieuwe versies op de markt gebracht voor ontwikkeling en beheer van Java-applicaties. De vernieuwingen zijn erop gericht applicaties op alle bekende Java-platformen sneller operationeel te krijgen en tegelijkertijd de efficiëntie van het beheer te vergroten.

OptimalJ is de modelgedreven, op patronen gebaseerde ontwikkelomgeving van Compuware. Versie 3.3 stelt Java-ontwikkelteams in staat om service georiënteerde applicaties sneller operationeel te krijgen. Zo heeft OptimalJ 3.3 meer automatiseringsmogelijkheden wat ontwikkeling versnelt. Ook is het hergebruik van data en functionaliteiten uit bestaande applicaties versneld en vergemakkelijkt wat verbinding met meer back ends mogelijk maakt.

DevPartner is een suite automatische test- en debuggingsmethodieken voor gebruik tijdens de ontwikkelfase van applicaties. DevPartner Java Edition 3.3 heeft nieuwe en verbeterde analysemogelijkheden om softwarefouten en bottlenecks van Java-applicaties al in de ontwikkelfase op te sporen en te verhelpen. De analyse kan onder meer op het geheugengebruik van applicaties automatisch en herhaald in kaart brengen. Hiermee is snel inzichtelijk welke delen van de applicatie teveel geheugen gebruiken en worden geheugenlekken in een applicatie automatisch in kaart gebracht.

Vantage is een analysemethode die prestaties van een complete IT-omgeving laat zien: van applicatie tot mainframe server. De Analyzer voor J2EE is hier een uitbreiding op. De nieuwe methodiek stelt in staat problemen met prestatie en beschikbaarheid van Java-applicaties snel te identificeren en te isoleren. Vantage Analyser let op prestatie-indicatoren als CPU, netwerk, SQL en geheugen. Vantage Analyser is speciaal bestemd voor complexe productieomgevingen en meet daarom gelijktijdig veel verschillende aspecten. Vantage Analyzer is makkelijk te installeren, vereist weinig J2EE kennis en is vooraf geconfigureerd om de noodzakelijke prestatie-indicatoren te verzamelen.