Webservices maken met JDeveloper

Ondersteuning op het 9i platform

Dit artikel is het tweede deel in een serie over webservices. In het eerste deel, dat in de vorige editie van Optimize (april 2002) verscheen onder de titel 'Een evolutionaire stap voorwaarts' is een definitie gegeven van een webservice en is ingegaan op de toepassingen van webservices in een viertal scenario's. In dit tweede en tevens laatste deel wordt uitgebreid ingegaan op de ondersteuning voor het ontwikkelen en beschikbaar stellen van webservices op het Oracle9i platform en zal in meer detail naar de standaards SOAP en WSDL gekeken worden.

In het voorbeeld uit het eerste artikel van deze serie werd voor de besproken webservice geen gebruik gemaakt van SOAP en WSDL maar alleen van XML over HTTP. In dit artikel zal duidelijk worden welke voordelen SOAP en WSDL bieden. Hierbij zullen we onder andere stap voor stap laten zien hoe een eenvoudige webservice ontwikkeld kan worden met behulp van Oracle9i JDeveloper. Daartoe gebruik gemaakt van Oracle9i JDeveloper Release Candidate 2. In afbeelding 1 is aangegeven hoe het aanroepen, beschrijven en registreren van webservices op basis van de SOAP, WSDL en UDDI standaards plaatsvindt. De provider heeft een aantal webservices geïmplementeerd en beschreven in WSDL. De WSDL beschrijvingen zijn gepubliceerd in een UDDI Registry. De SOAP Server zorgt voor de afhandeling van de SOAP aanroepen die via de HTTP Server binnenkomen. De consumer "ontdekt" de webservice in de UDDI Registry. Op basis van de WSDL definitie van de webservice wordt een Web Service Client ontwikkeld die met behulp van de SOAP Client API de webservice kan aanroepen. De applicatie roept de Web Service Client aan om de Web Service te gebruiken.

Eisen van provider

Voor een bedrijf dat webservices wil aanbieden - de provider - is het belangrijk dat investeringen die reeds gedaan zijn benut kunnen worden. Dit is het geval indien het mogelijk is om bestaande programmatuur (functionaliteit) als webservices beschikbaar te stellen. En, indien er gebruik kan worden gemaakt van aanwezige platform- en programmeertaalkennis voor het ontwikkelen van webservices. Daarnaast moet een provider beschikken over een platform dat betrouwbaarheid, performance en schaalbaarheid biedt.

Eisen van consumer

Voor de consumer - het bedrijf dat webservices wil afnemen - is het van belang dat op een eenvoudige manier gebruik kan worden gemaakt van webservices. Daarnaast wil uiteraard ook de consumer investeringen die al gedaan zijn benutten door een programmeertaal en platform naar keuze te kunnen gebruiken. Tenslotte kan het voor een consumer interessant zijn om op een geavanceerde manier gebruik te kunnen maken van webservices. Voorbeelden hiervan zijn:

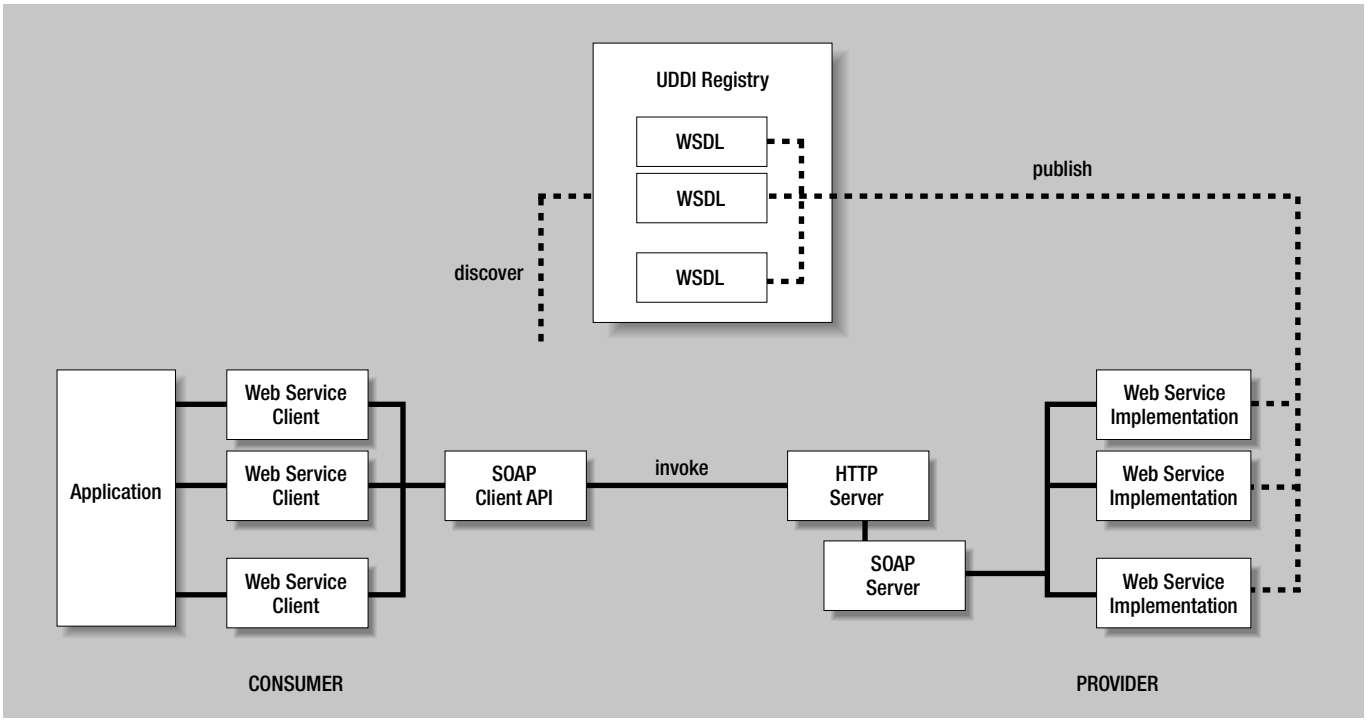
- **Samengestelde webservices**
  Het achter elkaar aanroepen van meerdere webservices. Bijvoorbeeld het opvragen van de koers van een aandeel in Amerikaanse Dollars en het opvragen van de wisselkoers van de Amerikaanse Dollar naar de Euro.

- **Conditioneel gebruik**
  Het op basis van een conditie aanroepen van een bepaalde webservice. Bijvoorbeeld het opvragen van de koers van een aandeel bij NASDAQ of AEX afhankelijk van de beurs waar het aandeel genoteerd is.

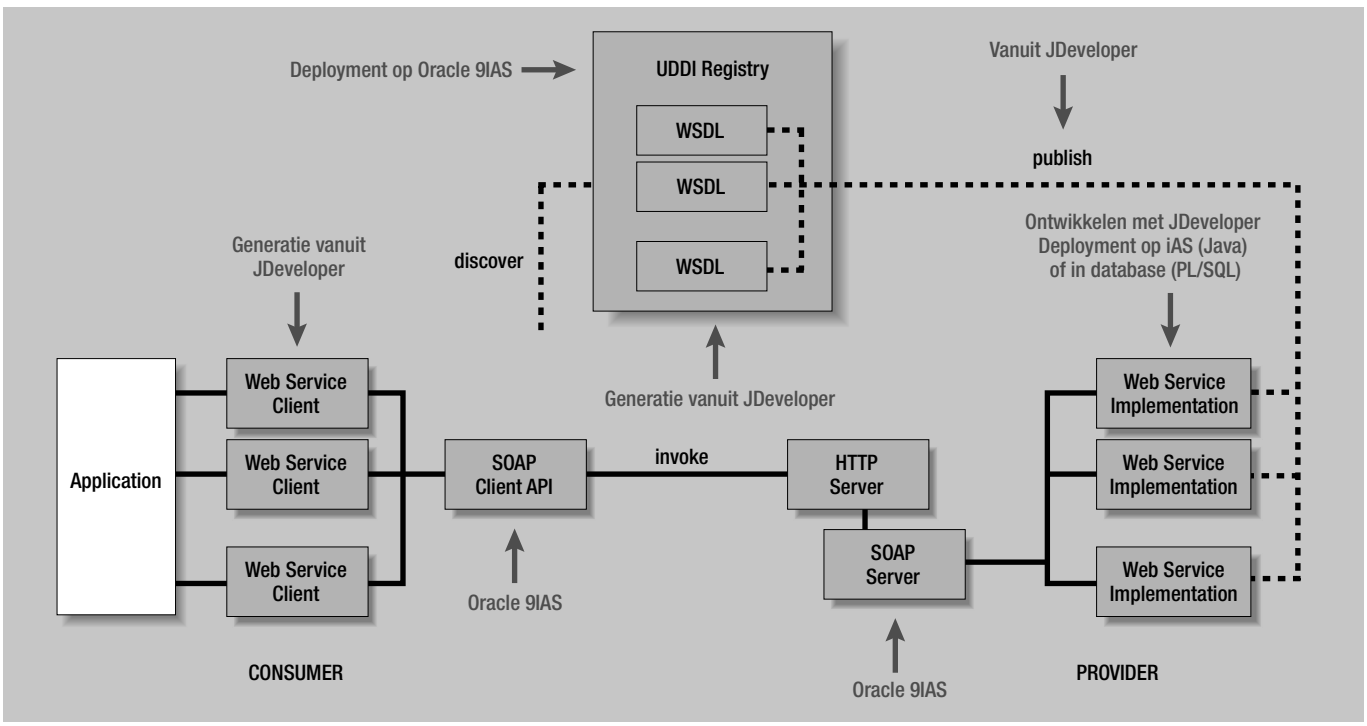
- **Terugvallen op alternatieve webservices**
  Het aanroepen van een alternatieve webservice indien een webservice niet bereikt kan worden. Bijvoorbeeld het opvragen van fileberichten bij een andere provider.

Oracle9i platform

Het Oracle9i platform biedt uitgebreide ondersteuning voor het ontwikkelen en beschikbaar stellen van webservices. In afbeelding 2 is aangegeven welke architectuurcomponenten uit afbeelding 1 door het Oracle9i platform ingevuld worden.



Afbeelding 1. Architectuur van webservices op basis van SOAP, WSDL en UDDI



Afbeelding 2. Invulling architectuurcomponenten door Oracle9i platform

Provider

Voor een provider van webservices biedt de Oracle9i Application Server (Oracle9iAS) een betrouwbare, performante en schaalbare HTTP en SOAP Server. Bestaande Java classes en Enterprise Java Beans (EJB's) op Oracle9iAS en PL/SQL programmatuur in de database kunnen met behulp van JDeveloper op een eenvoudig manier beschikbaar worden gesteld als webservices. Uiteraard kunnen ook nieuwe webservices ontwikkeld worden met JDeveloper, waarbij eveneens een keuze kan worden gemaakt tussen Java classes, EJB's of PL/SQL. Voor het beschikbaar

stellen zijn twee "smaken" (deployment platforms) beschikbaar: Oracle9iAS release 2 / Apache 2.2 SOAP Server en Oracle J2EE Web Services. De laatste biedt betere performance en schaalbaarheid en verdient daarom de voorkeur. Vanuit JDeveloper kan een WSDL beschrijving van een webservice gegenereerd worden en gepubliceerd worden in een UDDI Registry. Oracle9iAS release 2 bevat een UDDI Registry. Tenslotte zal het beheer van de beschikbaar gestelde webservices kunnen plaatsvinden met behulp van een nieuwe versie van Oracle Enterprise Manager.

Consumer

Een consumer van webservices kan met behulp van JDeveloper een Java Web Service Client genereren op basis van een WSDL beschrijving van een webservice. De Web Service Client maakt gebruik van de met Oracle9iAS meegeleverde SOAP Client API om de webservice aan te roepen.

Als we kijken naar de hierboven uiteen gezette eisen en wensen van providers en consumers van webservices kunnen we vaststellen dat het Oracle9i platform voor providers een volledige invulling biedt. Immers, Oracle9iAS is een schaalbare, performante en betrouwbare applicatie server waarop bestaande en nieuw ontwikkelde PL/SQL en Java programmatuur (de twee belangrijkste programmeertalen voor Oracle ontwikkelaars) op een eenvoudige manier als webservices beschikbaar kan worden gesteld. Voor consumers is de invulling van het Oracle9i platform minder volledig. JDeveloper kan alleen een Java Web Service Client genereren. Hoewel dit een zeer voor de handliggende keuze is, betekent dit dat het voor PL/SQL ontwikkelaars minder makkelijk is om van webservices gebruik te maken. Daarnaast moet het eerder beschreven "geavanceerde gebruik" van webservices (nog) zelf geprogrammeerd worden. Oracle schetst hiervoor wel toekomstige functionaliteit waarbij generatie en implementatie met behulp van onder andere Activity Modelling (een UML modelleringtechniek, ondersteund door JDeveloper voor het modelleren van bedrijfsprocessen) en Oracle Workflow plaatsvindt.

SOAP

Zoals al eerder gezegd is er in het voorbeeld dat gegeven is in het eerste artikel geen gebruik gemaakt van SOAP maar alleen van XML over HTTP. Hierbij is voor de aanroep van de webservice een XML document ontworpen waarin de aanroep met daarin alle data (parameters) kan worden aangegeven. En is er een XML document ontworpen waarin de data (return waarden) van het resultaat van de aanroep kan worden bevat. Dit is vergelijkbaar met de manier waarop "traditionele" business-to-business (B2B) integratie op basis van XML over HTTP plaatsvindt: de XML berichten (vocabulary) worden eerst ontwikkeld, waarna de programmatuur wordt ontwikkeld om deze berichten te versturen, te ontvangen en te verwerken. Bij het gebruik van SOAP wordt deze aanpak als het ware omgedraaid en grotendeels geautomatiseerd: eerst wordt de programmatuur ontwikkeld, waarna SOAP als een "black box" voor de vertaling naar en van XML zorgt bij de aanroep van een webservice. Hieronder wordt verder uitgewerkt wat de stappen zijn die de programmatuur van de provider en consumer moeten uitvoeren bij gebruik van XML over HTTP en welke functionaliteit SOAP biedt.

Consumer

De consumer implementeert bij het gebruik van XML over HTTP voor de aanroep van een webservice de volgende stappen:

- stel het XML document voor de aanroep samen
- verstuur het XML document (over HTTP) en ontvang het antwoord
- valideer het als antwoord ontvangen XML document
- haal de data uit het XML document

De SOAP Client API automatiseert deze vier stappen. Hierbij worden de parameters van de aanroep van de webservice op een standaard manier in XML "gecodeerd" *(encoded)*. De Web Service Client schermt de benodigde aanroepen van de SOAP Client API af voor de ontwikkelaar van de applicatie waar vanuit gebruik wordt gemaakt van de door de webservice geboden functionaliteit. Voor deze ontwikkelaar is het feit dat van een webservice in het spel is niet zichtbaar.

Met behulp van JDeveloper kan de Web Service Client gegenereerd worden op basis van de WSDL beschrijving van de webservice. Als provider van de webservice is het dus belangrijk om de WSDL beschrijving te publiceren. De aangewezen manier om dit te doen is in een UDDI Registry, maar dit is niet noodzakelijk. De WSDL beschrijving kan ook op een (normale) website gepubliceerd worden. Een bekende website waarop een groot aantal webservices gepubliceerd zijn is www.xmethods.com. JDeveloper kan een WSDL beschrijving genereren op basis van de implementatie van de webservice.

Provider

Voor het afhandelen van de aanroep van een webservice implementeert de provider bij gebruik van XML over HTTP de volgende stappen:

- ontvang het XML document met de aanroep van de webservice
- valideer het ontvangen XML document
- haal de data (parameters) uit het XML document
- voer de implementatie van de webservice uit
- stel het XML document samen voor het antwoord
- verstuur het antwoord

De SOAP Server automatiseert deze zes stappen. Op basis van een *descriptor* van de webservice en de WSDL beschrijving is de

SOAP Server in staat om de op de standaard wijze gecodeerde data uit de aanroep te halen en de implementatie van de webservice uit te voeren. Het antwoord wordt weer op dezelfde standaard manier in een XML gecodeerd en vervolgens teruggestuurd naar de consumer. Met behulp van JDeveloper kunnen de descriptor en de WSDL beschrijving van de webservice gegenereerd worden en de webservice beschikbaar worden gesteld *(deployed)*.

Stap voor stap ontwikkeling van een eenvoudige webservice

Als voorbeeld zullen we met behulp van JDeveloper een eenvoudige webservice ontwikkelen en beschikbaar stellen. De webservice is geïmplementeerd in de Java class uit afbeelding 3.

```
package com.acme;

public class ProductInfo {

  public static int getAvailability(int productID, int quantity) {
    ...
  }

  public static float getCurrentPrice(int productID) {
    ...
  }

  public static String[] getDescriptions(int productID) {
    ...
  }
}
```

Afbeelding 3. Java class waarin de webservice geïmplementeerd is

Met 'getAvailability' kan de beschikbaarheid van een bepaalde hoeveelheid van een bepaald product worden opgevraagd. De mogelijke waarden die worden teruggeven zijn: 1 (artikel op voorraad), 2 (onvoldoende voorraad), 3 (in bestelling), 4 (uit assortiment) of 5 (vervangen). De actuele prijs van een product kan worden opgevraagd met 'getCurrentPrice'. Tenslotte kan met 'getDescriptions' een aantal verschillende beschrijvingen van een bepaald product worden opgevraagd.

Web Service Publishing Wizard

Als eerste stap zullen we de deployment descriptor en de WSDL beschrijving van de webservice genereren met behulp van de Web Service Publishing Wizard.

In de eerste stap wordt de class geselecteerd waarin de webservices zijn geïmplementeerd. Dit is de class uit afbeelding 3. Daarnaast wordt een naam gegeven aan de webservice en wordt het deployment platform geselecteerd: Oracle9iAS Release 2 / Apache SOAP 2.2 Server of Oracle J2EE Web Services. In dit voorbeeld maken we gebruik van de laatste.

Afbeelding 4. Web Service Publishing Wizard - Stap 1

Afbeelding 5. Web Service Publishing Wizard - Stap 2

Afbeelding 6. Web Service Publishing Wizard - Stap 3

De methods uit de geselecteerd class die als componenten van de webservice beschikbaar moeten worden gesteld worden in de tweede stap van de wizard aangevinkt. Wij kiezen voor alle drie de methods. Tevens geven we aan dat we voor deze webservices geen state hoeven vast te houden. Dat wil zeggen dat iedere aanroep van de webservices gezien kan worden als een onafhankelijk aanroep. Er hoeft geen informatie vastgehouden te worden tussen achtereenvolgende aanroepen door een bepaalde consumer.

In de derde, en laatste, stap wordt aangegeven waar de gegenereerde WSDL beschrijving moeten worden neergezet en wat de URL is waar de webservice op kan worden aangeroepen (endpoint). Dit laatste wordt aangegeven door een eerder aangemaakte connectie naar een applicatie server of SOAP server (afhankelijk van het geselecteerde deployment platform) te selecteren. Tenslotte wordt een namespace gegeven die gebruikt wordt in de gegenereerde WSDL beschrijving. De gegenereerde WSDL beschrijving van de webservices is te zien in afbeelding 7.

WSDL

De WSDL beschrijving bestaat uit vijf delen. Allereerst worden eventuele complexe datatypen die als parameter of return waarde gebruikt worden gedefinieerd in het <types> element.

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<!—Generated by the Oracle9i JDeveloper Web Services WSDL Generator—>
<!—Date Created: Sat Apr 27 21:41:08 CEST 2002—>
<definitions
   name="ProductInfo"
   targetNamespace="http://xmlns.acme.com/ws"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://xmlns.acme.com/ws"
   xmlns:ns1="http://xmlns.acme.com/ws/schema">
   <types>
      <schema
         targetNamespace="http://xmlns.acme.com/ws/schema"
         xmlns="http://www.w3.org/2001/XMLSchema"
         xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
         <complexType name="ArrayOfstring" xmlns:wsdl=
         "http://schemas.xmlsoap.org/wsdl/">
            <complexContent>
               <restriction base="SOAP-ENC:Array">
                  <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType=
                  "xsd:string[]"/>
               </restriction>
            </complexContent>
         </complexType>
      </schema>
   </types>
   <message name="getAvailability0Request">
      <part name="productID" type="xsd:int"/>
      <part name="quantity" type="xsd:int"/>
   </message>
   <message name="getAvailability0Response">
      <part name="return" type="xsd:int"/>
   </message>
   <message name="getCurrentPrice1Request">
      <part name="productID" type="xsd:int"/>
   </message>
   <message name="getCurrentPrice1Response">
      <part name="return" type="xsd:float"/>
   </message>
   <message name="getDescriptions2Request">
      <part name="productID" type="xsd:int"/>
   </message>
   <message name="getDescriptions2Response">
      <part name="return" type="ns1:ArrayOfstring"/>
   </message>
   <portType name="ProductInfoPortType">
      <operation name="getAvailability">
         <input name="getAvailability0Request"
message="tns:getAvailability0Request"/>
         <output name="getAvailability0Response"
message="tns:getAvailability0Response"/>
      </operation>
      <operation name="getCurrentPrice">
         <input name="getCurrentPrice1Request"
message="tns:getCurrentPrice1Request"/>
         <output name="getCurrentPrice1Response"
message="tns:getCurrentPrice1Response"/>
      </operation>
      <operation name="getDescriptions">
         <input name="getDescriptions2Request"
message="tns:getDescriptions2Request"/>
         <output name="getDescriptions2Response"
message="tns:getDescriptions2Response"/>
      </operation>
   </portType>
   <binding name="ProductInfoBinding" type="tns:ProductInfoPortType">
      <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="getAvailability">
         <soap:operation soapAction="" style="rpc"/>
         <input name="getAvailability0Request">
            <soap:body use="encoded" namespace="com.acme.ProductInfo"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </input>
         <output name="getAvailability0Response">
            <soap:body use="encoded" namespace="com.acme.ProductInfo"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </output>
      </operation>
      <operation name="getCurrentPrice">
         <soap:operation soapAction="" style="rpc"/>
         <input name="getCurrentPrice1Request">
            <soap:body use="encoded" namespace="com.acme.ProductInfo"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </input>
         <output name="getCurrentPrice1Response">
            <soap:body use="encoded" namespace="com.acme.ProductInfo"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </output>
      </operation>
      <operation name="getDescriptions">
         <soap:operation soapAction="" style="rpc"/>
         <input name="getDescriptions2Request">
            <soap:body use="encoded" namespace="com.acme.ProductInfo"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </input>
         <output name="getDescriptions2Response">
            <soap:body use="encoded" namespace="com.acme.ProductInfo"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </output>
      </operation>
   </binding>
   <service name="ProductInfo">
      <port name="ProductInfoPort" binding="tns:ProductInfoBinding">
         <soap:address
location="http://quatsun2.kantoor.office:8989/WebServices/
com.acme.ProductInfo"/>
      </port>
   </service>
</definitions>
```

Afbeelding 7. Gegenereerde WSDL beschrijving

Hiervoor wordt gebruik gemaakt van de XML Schema standaard. In ons voorbeeld is dit het geval voor 'getDescriptions' die een array van strings retourneert. Voor deze return waarde is het type 'ArrayOfstring' gedefinieerd. Voor de primitieve datatypen van Java (int, float, et cetera), bijhorende "wrapper" classes (Integer, Float, et cetera) en de String class hoeft in de WSDL beschrijving geen type gedefinieerd te worden. Indien een instantie

28: Advertentie Wildher

Afbeelding 8.
Beschikbaar stellen van de webservice

van een andere (eigen) class als parameter of return waarde wordt gebruikt moet hiervoor een XML Schema type definitie worden gegeven in de WSDL beschrijving. De huidige versie van JDeveloper kan deze type definities echter niet genereren. Dit betekent dat dan handmatig de WSDL beschrijving moet worden aangepast. Hiervoor is kennis nodig van WSDL, SOAP en XML Schema.

In het tweede deel van de WSDL beschrijving wordt in voor iedere component van de webservice een bericht *(message)* gedefinieerd voor de aanroep *(request)* en een bericht voor het resultaat *(response)*. Hierbij worden de parameters en return waarden als onderdelen *(parts)* van het bericht gedefinieerd. Bijvoorbeeld het bericht 'getDescriptions2Response' met een onderdeel (return waarde) van het complexe datatype 'Array-Ofstring' (gedefinieerd in het eerste deel van de WSDL beschrijving).

De componenten van de webservice worden in het derde deel gedefinieerd in het element '<portType>'. Voor iedere component *(operation)* wordt verwezen naar de in het tweede deel gedefinieerde request en response berichten. In het vierde deel van de WSDL beschrijving wordt in het '<binding>' element voor iedere operation gedefinieerd in het derde deel de "binding" met het SOAP protocol aangegeven. Tenslotte wordt in het '<service>' element aangegeven op welke URL de webservice beschikbaar is gesteld.

Afbeelding 9.
Web Service Stub/Skeleton Wizard - Stap 1

Afbeelding 10.
Web Service Stub/Skeleton Wizard - Stap 2

Deployment

De webservice kan beschikbaar gesteld (deployed) worden via een deployment profile. In afbeelding 8 wordt dit gedaan via het rechter-muisknop-menu van het deployment profile.

Web Service Stub / Skeleton Wizard

Voor het genereren van de Web Service Client biedt JDeveloper de Web Service Stub/Skeleton Wizard. Deze wizard bestaat uit twee stappen. In de eerste stap wordt de WSDL beschrijving aangegeven waarvoor de Web Service Client moet worden gegenereerd. In de tweede stap wordt de package naam en class naam voor de Web Service Client class aangegeven.

In afbeelding 11 is een deel van de gegenereerde Web Service Client class getoond. We zien hier de 'getAvailability' component. De code van deze method bestaat uit een reeks aanroepen van de SOAP Client API om het XML request document op te bouwen met daarin de parameters van de aanroep. Vervolgens wordt de webservice aangeroepen en wordt, indien er geen fout wordt geconstateerd, de return waarde uit het XML response document gehaald en teruggegeven. Een applicatieontwikkelaar kan deze

```
public Integer getAvailability(Integer productID, Integer quantity)
throws Exception
  {
    Integer returnVal = null;

    URL endpointURL = new URL(endpoint);
    Call call = new Call();
    call.setSOAPTransport(m_httpConnection);
    call.setTargetObjectURI("com.acme.ProductInfo");
    call.setMethodName("getAvailability");
    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
    Vector params = new Vector();
    params.addElement(new Parameter("productID", Integer.class,
productID, null));
    params.addElement(new Parameter("quantity", Integer.class, quanti-
ty, null));
    call.setParams(params);

    Response response = call.invoke(endpointURL, "");

    if (!response.generatedFault())
    {
      Parameter result = response.getReturnValue();
      returnVal = (Integer)result.getValue();
    }
      else
      {
        Fault fault = response.getFault();
        throw new SOAPException(fault.getFaultCode(),
fault.getFaultString());
      }

    return returnVal;
  }
```

Afbeelding 11. Deel van gegenereerde Web Service Client class

```
package com.acme;

public class Tester
{
  public static void main(String[] args)
  {
    ProductInfoStub stub = null;
    try {
      stub = new ProductInfoStub();
      System.out.println(stub.getAvailability(Integer.valueOf("121")

,Integer.valueOf("10")));
    }
    catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Afbeelding 12. Gebruik van webservice vanuit applicatie

```
package com.acme;

public class ProductAvailability
{
  static final int IN_STOCK = 1;
  static final int INSUFFICIENT_STOCK = 2;
  static final int BACKORDER = 3;
  static final int OBSOLETE = 4;
  static final int REPLACED = 5;

  public int availability;
  public int stock;
  public int days;
  public int productID;

  public ProductAvailability(int availability
                             ,int stock
                             ,int days
                             ,int productID)
  {
    this.availability = availability;
    this.stock = stock;
    this.days = days;
    this.productID = productID;
  }
}
```

Afbeelding 13. Class 'ProductAvailability'

'getAvailability' method gebruiken zonder enige kennis van SOAP te hebben. In afbeelding 12 wordt dit geïllustreerd. Hier wordt de beschikbaarheid van 10 stuks van product '121' opgevraagd.

Zoals al eerder opgemerkt kan de huidige versie van JDeveloper (release candidate 2) geen WSDL beschrijving genereren voor methods die een instantie van een "eigen" class als parameter of return waarde gebruiken. Stel dat we de 'getAvailability' method in de class 'ProductInfo' uitgebreidere informatie zouden willen laten teruggeven om aan te geven wat de beschikbaarheid van een bepaald product is. We zouden hiervoor de class 'ProductAvailability' uit afbeelding 13 kunnen gebruiken. De member 'availability', 'stock', 'days' en 'productID' bevatten respectievelijk de code die de beschikbaarheid aangeeft, het aantal dat op voorraad is, het verwachte aantal dagen tot aanvulling van de voorraad en de identificatie van het vervangende product.
Als we de method 'getAvailability' in class 'ProductInfo' een instantie van deze class willen laten teruggeven zal in stap 2 van de Web Service Publishing Wizard de method niet geselecteerd kunnen worden als component van de webservice (zie afbeelding 14). Het feit dat JDeveloper de benodigde files niet kan genereren betekent niet dat deze method niet als component van de webservice beschikbaar kan worden gesteld. Met de juiste kennis van WSDL, SOAP en XML Schema kunnen de deployment descriptor en WSDL beschrijving handmatig worden aangepast om zodoende deze component alsnog beschikbaar te stellen.

Afbeelding 14. Method 'getAvailability' kan niet meer geselecteerd worden als webservice component

Conclusie

Het Oracle9i platform biedt een brede ondersteuning voor het ontwikkelen en beschikbaar stellen van webservices. JDeveloper maakt het ontwikkelen van webservices en het gebruikmaken van webservices erg gemakkelijk. Hierbij moet wel gerealiseerd worden dat het hier nieuwe technologie betreft met de bijbehorende valkuilen en dat in de praktijk niet alles met wizards gerealiseerd kan worden. Daarom is het aan te raden om ervoor te zorgen dat u bij het uitvoeren van webservices projecten over de juiste kennis en ervaring kunt beschikken.

Erwin Groenendal en Jan Vissers
Jan Vissers is Senior Consultant bij Cumquat Information Technology. Hij heeft ruim 7 jaar ervaring met Oracle technologie en richt zich voornamelijk op J2EE toepassingen op het Oracle platform. (e-mail: jan.vissers@cumquat.nl).
Erwin Groenendal is Technisch Directeur van Cumquat Information Technology. Hij heeft tien jaar ervaring met Oracle technologie. E-mail: erwin.groenendal@cumquat.nl. Cumquat richt zich op de toepassing van de nieuwste Oracle e-business technologie, XML en Java voor het realiseren van Internet- en enterprise applicaties, B2B- en A2A integratieoplossingen, web services en portals.