Gijs de Jong is senior consultant bij Microsoft Services

# Webservices in Visual Studio 2005

## GESTANDAARDISEERDE EN BREED TOEGANKELIJKE WEBSERVICES BOUWEN

**Visual Studio 2005 is de opvolger van Visual Studio .NET 2003. Visual Studio 2005 en het .NET Framework 2.0 bieden over de gehele linie veel nieuwe functionaliteit. Ook webservices in Visual Studio 2005 bevatten de nodige nieuwe features en verbeterpunten, onder meer op het gebied van interoperabiliteit, uitbreidbaarheid en productiviteit. In dit artikel komen de belangrijkste nieuwe aspecten aan de orde op basis van bèta 1. Ter afsluiting behandel ik in het kort wat je na Visual Studio 2005 op het gebied van webservices kunt verwachten.**

De kracht van webservices is met name interoperabiliteit tussen verschillende platformen. Interoperabiliteit wordt bewerkstelligd door de standaardspecificaties en protocollen waarop webservices zijn gebaseerd. Deze specificaties zijn flexibel van aard, waardoor al snel ruimte ontstaat voor verschillende interpretaties. Dit komt de interoperabiliteit uiteraard niet ten goede. Interoperabiliteitsproblemen zijn ontstaan door bijvoorbeeld SOAP *Message Style* (Document versus Rpc) en *Serialization Format* (Literal versus Encoded).

### Webservices volgens WS-I Basic Profile 1.0

De Web Services Interopability Organisation (WS-I) heeft daarom een specificatie in het leven geroepen met de naam Basic Profile 1.0 (BP). BP bevat regels waaraan webservice-ontwikkelaars zich moeten houden om interoperabiliteit te kunnen garanderen. Een van de regels die het BP bijvoorbeeld stelt is dat SOAP Encoding niet gebruikt mag worden. Regels alleen zijn niet voldoende, de WS-I heeft ook tools ontwikkeld die webservices valideren voor deze regels. Microsoft Patterns & Practices biedt tevens een white paper waarin staat beschreven hoe volgens BP 1.0 webservices ontwikkeld kunnen worden met Visual Studio .NET 2003. Webservices in Visual Studio 2005 zijn standaard conform BP 1.0. Het afdwingen van conformiteit kan bepaald worden door een nieuwe property (*ConformanceClaims)* op het *WebServiceBinding*-attribuut. Codevoorbeeld 1 laat zien hoe dit geïmplementeerd kan worden.

Er treedt een exceptie op wanneer een webservice niet BP-conform is, terwijl dit door de *ConformanceClaims*-property wel wordt aangegeven. Dit gebeurt op het moment dat de webservice voor de eerste keer aangeroepen wordt, of wanneer de WSDL van de webservice opgevraagd wordt. Als de *ConformanceClaims*-property niet aanwezig is en de webservice niet BP-conform is, treedt geen exceptie op. Er verschijnt dan slechts een waarschuwende tekst op de webservice-testpagina. In deze tekst wordt beschreven aan welke regel(s) van de BP niet wordt voldaan en eventueel hoe dit kan worden verholpen. Vanwege specifieke vereisten kan het zo zijn dat een webservice moet worden ontwikkeld die niet conform is aan BP 1.0. In dat geval kunnen de waarschuwingsteksten onderdrukt worden door het web.config-configuratiebestand aan te passen; zie codevoorbeeld 2.

Webservice proxy-classes die door Visual Studio 2005 gegenereerd worden, zijn standaard ook conform aan BP 1.0. Als de webservice waarvoor de proxy-class gegenereerd wordt niet conform BP is, worden waarschuwingen getoond. Desondanks zal de proxy-class wel gewoon gecreëerd worden, indien mogelijk.

De meeste nieuwe features rondom het genereren van webservice proxy-classes worden in bèta 1 van Visual Studio 2005 nog niet volledig ondersteund vanuit de IDE. In de commandline-tool *wsdl.exe* zijn deze al wel geïmplementeerd. De definitieve versie van Visual Studio 2005 ondersteunt de functionaliteit uit *wsdl.exe* ook in de IDE.

### Ondersteuning van SOAP 1.2

Visual Studio 2005 biedt ondersteuning voor SOAP 1.2. Standaard worden webservices in SOAP 1.1 en 1.2 aangeboden. Mocht je deze configuratie willen wijzigen dan kan dat door het web.config-bestand aan te passen. Codevoorbeeld 3 toont hoe SOAP 1.1 verwijderd kan worden.
Ook kan bij het genereren van webservice proxy-classes aan *wsdl.exe* de */protocol*-switch meegegeven worden om aan te geven welke SOAP-versie gebruikt moet worden. Geldige waarden zijn SOAP11 en SOAP12. Programmatisch kunnen webservice-clients de SOAP-versie bepalen door het zetten van de *SoapVersion*-property van de proxy-class. Codevoorbeeld 4 illustreert hoe SOAP-versie 1.2 gedefinieerd kan worden.

```
[WebServiceBinding(ConformanceClaims=WsiClaims.BP10)]
public class EmployeeService : System.Web.Services.WebService
{
   [WebMethod]
   public string GetEmployeeName()
   {
      return "Gijs de Jong";
   }
}
```

Codevoorbeeld 1.

```
<configuration>
   <system.web>
      <webServices>
         <conformanceWarnings>
            <remove name='BP1.0'/>
         </conformanceWarnings>
      </webServices>
   </system.web>
</configuration>
```

Codevoorbeeld 2.

## Event-based asynchroon programmeermodel

Visual Studio 2005 biedt een nieuw programmeermodel voor het asynchroon aanroepen van webservices. Het "Asynchronous Pattern for Components" is onderdeel van het .NET Framework 2.0 en niet specifiek voor webservices. Proxy- classes die gegenereerd worden met de Visual Studio 2005-versie van *wsdl.exe* maken intensief gebruik van dit nieuwe pattern. In dit artikel zal slechts worden ingegaan op de elementaire functionaliteit die voor webservices van toepassing zijn.

Eerdere versies van Visual Studio bevatten al een programmeermodel voor het asynchroon aanroepen van webservices. In dit model wordt voor elke methode automatisch een asynchrone variant gegenereerd: *Begin<MethodeNaam>* en *End<MethodeNaam>*. Als je na het beëindigen van de asynchrone methode een notificatie (met een eventueel resultaat) wil krijgen, moet er een *CallBack* meegegeven worden aan de *Begin<MethodeNaam>*-methode. Ondanks het feit dat dit model het relatief eenvoudig maakt webservices asynchroon aan te roepen is er nog steeds een aspect dat de ontwikkelaar zelf moet verzorgen. De *CallBack* die meegegeven wordt aan de *Begin<MethodeNaam>*-methode wordt op een andere thread uitgevoerd dan waarop *Begin<MethodeNaam>* origineel op aangeroepen was. Met name in applicaties met een GUI kan dit voor additionele complexiteit zorgen. Omdat de *CallBack* wordt uitgevoerd op een thread uit de .NET ThreadPool kan deze niet zomaar GUI-componenten benaderen. De ontwikkelaar moet er zelf voor zorgen dat de desbetreffende code op de UI-thread wordt uitgevoerd. Dit is mogelijk door de *Invoke*-methode van de desbetreffende control aan te roepen.

Het nieuwe model neemt de thread-synchronisatiecomplexiteit weg bij de ontwikkelaar door de introductie van een eenvoudig event-gebaseerd model. Voor elke methode op de proxy-class wordt nu ook een asynchrone variant van de methodes en bijbehorende Events, Delegates en EventArgs gegenereerd. Voor het aanroepen van de *<MethodeNaam>Async*-methode hoeft de ontwikkelaar slechts een EventHandler te registeren voor het *<MethodeNaam>Completed* event. Wanneer de methode klaar is zal de EventHandler automatisch aangeroepen worden. Dit gebeurt op dezelfde thread als waarop *<MethodeName>Async* initieel was aangeroepen. Vanuit deze EventHandler is het dus mogelijk rechtstreeks GUI-componenten te benaderen. De eventuele return-waarde en outputparameters worden aan de EventHandler meegegeven in de vorm van de *<MethodeNaam>CompletedEventArgs*-parameter.

Ook is het mogelijk de asynchrone webservice-aanroepen voortijdig af te breken door de *CancelAsync*-methode op de proxy-class aan te roepen.

## Databinding

GUI-applicaties maken vaak gebruik van webservices voor het ophalen van data. Het vullen van UI-controls met de data kan sterk vereenvoudigd worden door gebruik te maken van databinding. In GUI-applicaties worden data vaak opgehaald met webservices om UI-controls op het scherm te vullen. Databinding vereenvoudigt het vullen van deze controls. Databinding is echter alleen mogelijk met Properties en niet met Fields. Aangezien Visual Studio .NET 2003 (en de bijbehorende versie van *wsdl.exe*) Fields genereert, was Databinding standaard niet mogelijk. Visual Studio 2005 genereert proxy-classes met Properties in plaats van Fields, waardoor Databinding standaard wel mogelijk is zonder het schrijven van additionele code. Mocht het vanwege compatibiliteitsreden toch vereist zijn om proxy-classes met Fields te genereren, dan kan dat door de */Fields*-switch mee te geven aan *wsdl.exe*.

## Type Sharing

Visual Studio 2005 maakt het mogelijk objecten te delen tussen verscheidene webservice proxy-classes. Dit is een zeer praktische nieuwe feature waardoor veel handmatige acties tot het verleden behoren. Om een beter idee te krijgen van de toegevoegde waarde zal ik de huidige problematiek met Visial Studio .NET 2003 eerst toelichten. Voor het uitwisselen van data met webservices worden vaak eigen gedefinieerde types (User Defined Types) gebruikt. Een zelfde type zou je mogelijkerwijs in meer webservices willen hergebruiken. Codevoorbeeld 6 illustreert hoe het *Employee*-object zowel in *EmployeeService* als in *VacationService* gebruikt kan worden.

```
<configuration>
  <system.web>
    <webServices>
      <protocols>
        <remove name="HttpSoap1.1"/>
      </protocols>
    /webServices>
  </system.web>
</configuration>
```

**Codevoorbeeld 3.**

```
EmployeeService empService = new EmployeeService();
empService.SoapVersion = SoapProtocolVersion.Soap12;
```

**Codevoorbeeld 4.**

```
private void button1_Click(object sender, EventArgs e)
{
  // Instantieer een proxy voor de Service
  EmployeeService empService = new EmployeeService();

  // Registreer een EventHandler, deze wordt aangeroepen
  // wanneer de Web service aanroep is geretourneerd
  empService.GetEmployeeNameCompleted += new
  GetEmployeeNameCompletedEventHandler(empService_GetEmployeeNameCompleted);

  // Roep de asynchrone variant van de methode aan
  empService.GetEmployeeNameAsync();
}

void empService_GetEmployeeNameCompleted(object sender,
GetEmployeeNameCompletedEventArgs args)
{
  // Vul de textbox met het resultaat van de Web service aanroep
  textBox1.Text = args.Result;
}
```

**Codevoorbeeld 5.**

Nu de webservices gebouwd zijn, kunnen hiervoor de proxy-classes gegenereerd worden. Normaal gesproken zou je nu in Visual Studio .NET 2003 een 'Add Web Reference' uitvoeren voor beide services. Omdat we het *Employee*-type over beide services heen willen gebruiken, zal de CLR-namespace voor beide proxies gelijk moeten zijn. Wanneer je dit probeert te bewerkstelligen, zie je dat Visual Studio .NET 2003 het niet toelaat om twee proxy-classes te genereren met dezelfde namespace. Dit gedrag is te voorkomen door *wsdl.exe* met de */namespace*-switch te gebruiken. *Wsdl.exe* zal voor beide services een keer opgestart moeten worden (met dezelfde */namespace*-parameter). Voor beide keren wordt er een webservice proxy-bestand gegenereerd. Omdat in beide services het *Employee*-type wordt gebruikt, genereert *wsdl.exe* in beide proxy-bestanden een definitie voor *Employee*. Nadat de twee bestanden aan het Visual Studio .NET 2003 Project zijn toegevoegd, kan er gecompileerd worden. Tijdens de compilatieactie zal een foutmelding optreden, omdat de definitie voor *Employee* al binnen de desbetreffende namespace aanwezig is. Om dit probleem te verhelpen moet de tweede definitie dus handmatig uit het proxy-bestand verwijderd worden. Dit heeft in grotere projecten met meer webservices uiteraard een negatieve impact op de productiviteit en is zeer foutgevoelig. De *wsdl.exe*-versie van Visual

Studio 2005 bevat een nieuwe commandline-switch: */sharetypes*. Deze switch zorgt ervoor dat er maar één definitie van een object wordt gegenereerd. Alle services waarvoor types gedeeld moeten worden zullen op de commandline meegegeven moeten worden; zie codevoorbeeld 7.

De webservice-client zou er uiteindelijk uit kunnen zien als de code in codevoorbeeld 8. Een instantie van het type *Employee* wordt opgevraagd aan de *EmployeeService* en wordt vervolgens doorgegeven aan de *VacationService*.

## Uitbreidbaarheid

Visual Studio 2005 en het .NET Framework 2.0 bieden een rijke infrastructuur voor het bouwen en consumeren van webservices. Er kunnen zich echter altijd uitzonderlijke situaties voordoen waarbij de bestaande functionaliteit niet toerijkend is en deze dus verder uitgebreid moet kunnen worden. Visual Studio 2005 bevat de mogelijkheid volledige controle uit te oefenen over de generatie van webservices en proxy-classes. Deze functionaliteit was voor een groot gedeelte al in eerdere versies aanwezig - *SoapExtensionReflector* en *SoapExtensionImporter* - maar was helaas alleen bedoeld voor intern gebruik door het .NET Framework. In Visual Studio 2005 wordt deze functionaliteit ook ondersteund voor eigen toepassingen.

Bij het genereren van de webservice kan de ontwikkelaar controle uitoefenen over hoe een object geserialiseerd moet worden en hoe het schema eruit moet zien. Custom serialisatie kan worden bewerkstelligd door de *IXmlSerializable*-interface te implementeren. Bij de serialisatie van het desbetreffende object zal de *IXmlSerializable.WriteXml*-methode aangeroepen worden. Hierin kan de ontwikkelaar een eigen implementatie schrijven die de serialisatie bepaalt. Hetzelfde geldt voor deserialisatie waarbij *IXmlSerializable.ReadXml* wordt aangeroepen. Voor de controle over de generatie van het schema van een specifiek object moet het nieuwe *XmlSchemaProvider*-attribuut worden gebruikt. Dit attribuut moet op een class geplaatst worden en verwacht een stringwaarde die vertelt welke methode aangeroepen moet worden voor het ophalen van het schema.

```
public class Employee
{
  public int Id;
  public string Name;
}

public class EmployeeService : System.Web.Services.WebService
{
  [WebMethod]
  public Employee GetEmployee(int id)
  {
    // Haal Employee gegevens uit de database ...

    Employee emp = new Employee();
    emp.Id = id;
    emp.Name = "Gijs de Jong";

    return emp;
  }
}

public class VacationService : System.Web.Services.WebService
{
  [WebMethod]
  public void RegisterVacation(Employee employee,
    System.DateTime from, System.DateTime to)
  {
    // Valideer vakantie aanvraag en sla eventueel op in de database...
  }
}
```

**Codevoorbeeld 6.**

```
wsdl /sharetypes http://localhost/VacationService.asmx
http://localhost/EmployeeService.asmx
```

**Codevoorbeeld 7.**

De *SchemaImporterExtension*-class biedt de mogelijkheid controle uit te oefenen over het genereren van de proxy-class voor het consumeren van een webservice. Bij het schrijven van een eigen extensie zal afgeleid moeten worden van de *SchemaImporterExtension*-class. Deze eigen extensie wordt gedurende het proxy-generatieproces aangeroepen. Hierdoor ontstaat de mogelijkheid om de gegenereerde code te beïnvloeden. Om ervoor te zorgen dat de extensie ook daadwerkelijk wordt aangeroepen, moet er een configuratie-element in de *machine.config* worden opgenomen; zie codevoorbeeld 9.

## Webservices op de desktop

De *Web Service Description Language* (WSDL) onderkent vier zogenaamde 'Message Exchange Patterns' (MEP's). Een MEP beschrijft het uitwisselingpatroon van SOAP-berichten tussen een service en een client. De vier patronen zijn:

1. OneWay – de client verstuurt een SOAP-bericht naar de service.
2. Request/Response – de client verstuurt een SOAP-bericht naar de service en ontvangt een bericht terug.
3. Notification – de service verstuurt een SOAP-bericht naar de client.
4. Solicit/Response – de service verstuurt een SOAP-bericht naar de client en ontvangt een bericht terug.

De eerste twee patronen zijn tot op heden verreweg het meest gebruikelijk. Bij de laatste twee patronen worden de rollen omgedraaid. De client fungeert als service en de service fungeert als client. Dit betekent dat er op de desktop een webserver, bijvoorbeeld Internet Information Services, aanwezig moet zijn om de SOAP-berichten te kunnen ontvangen. Het .NET Framework 2.0 kan een lichtgewicht webserver hosten binnen de clientapplicatie. Dit wordt mogelijk gemaakt door System.Net.HttpListener. HttpListener is een object binnen het .NET Framework dat http-requests kan afhandelen. *HttpListener* is gebaseerd op Http.sys, een kernel-mode 'http-listener', en is beschikbaar op Windows 2003 en Windows XP Service Pack 2.

Met het kunnen aanbieden van webservices op de desktop komt een heel nieuw scala aan mogelijkheden vrij. Het wordt nu bijvoorbeeld een stuk eenvoudiger om complexe asynchrone applicaties te ontwikkelen. Het opstarten van een asynchroon proces op een server is vrij eenvoudig te realiseren door de client een OneWay-bericht te laten versturen. De complexiteit ontstaat pas wanneer de client na verloop van tijd een resultaat van de service terugverwacht. Door gebruik te maken van *HttpListener* kan de service het resultaat naar de client terugsturen wanneer deze klaar is. Op deze manier hoeft de client de service niet met een bepaalde interval te raadplegen (pollen) of het resultaat al beschikbaar is.

## Webservices van morgen

Webservices van vandaag de dag richten zich puur op het uitwisselen van data in de vorm van SOAP-berichten. Voor complexe applicaties is alleen het uitwisselen van data vaak onvoldoende. Applicaties vereisen in veel gevallen eigenschappen zoals security, transacties, reliable messaging en discovery. De WS-* specificaties is een set van specificaties die het SOAP-protocol verder verrijkt met bovengenoemde karakteristieken. Technisch wordt dit bewerkstelligd door het toevoegen van gestandaardiseerde SOAP Headers. De specificaties beschrijven het formaat van de desbetreffende headers en hoe deze geïnterpreteerd en verwerkt dienen te worden. De kracht van deze aanpak is dat er gestandaardiseerd wordt op SOAP-berichtni-

```
// Instantieer proxies voor de Employee en Vacation services
EmployeeService empService = new EmployeeService();
VacationService vacService = new VacationService();

// Haal de Employee op
Employee employee = empService.GetEmployee(37);

// Geef het Employee object mee aan de Vacation service
vacService.RegisterVacation(employee, new DateTime(2004, 10, 1), new
DateTime(2004, 10, 8));
```

**Codevoorbeeld 8.**

```
<configuration>
  <system.xml.serialization>
    <schemaImporterExtensions>
      <add name="Name" type="Type, Assembly" />
    </schemaImporterExtensions>
  </system.xml.serialization>
</configuration>
```

**Codevoorbeeld 9.**

veau. Op deze manier worden webservices dus platform-, runtime- en transportkanaalonafhankelijk.

Een aantal van deze specificaties is vandaag de dag al te gebruiken door standaard .NET Web Services verder uit te breiden met Web Services Enhancements 2.0 (WSE). WSE is de concrete implementatie van een aantal van de WS-* specificaties. Het is een add-on op Visual Studio en is bedoeld voor 'early adopters' van deze nieuwe webservice-protocollen. De langetermijnvisie van Microsoft op het gebied van webservices wordt geconcretiseerd in 'Indigo'. Indigo is de codenaam voor een nieuw geünificeerd framework voor het bouwen van service georiënteerde applicaties in Windows. Hoewel Indigo volledig gebaseerd is op webserviceprotocollen is het veel meer dan XML webservices zoals we die vandaag de dag kennen. Indigo brengt de concepten van een aantal bestaande technologieën, zoals MSMQ, Enterprise Services, .NET Remoting en ASP.NET Web Services samen. Het biedt een uniforme infrastructuur en programmeermodel voor het bouwen van gedistribueerde en servicegeoriënteerde applicaties op basis van WS-* protocollen.

## Eenvoudiger webservices ontwikkelen

Hoewel Indigo nog wel een tijdje op zich laat wachten kunnen we nu al een goede basis leggen met webservices in Visual Studio 2005. Door webservices te ontwikkelen conform Basic Profile 1.0 vereenvoudigen we interoperabiliteit met bestaande en toekomstige webservice-implementaties. Naast de ondersteuning van standaarden zoals het Basic Profile en SOAP 1.2 biedt Visual Studio .NET 2005 een heel scala aan nieuwe functionaliteit, waardoor de ontwikkelaar nog eenvoudiger robuuste webservices kan ontwikkelen.

**Gijs de Jong** is senior consultant bij Microsoft Services. Zijn e-mailadres is gijsdj@microsoft.com.

**Nuttige internetadressen**

Web Services Developer Center: http://msdn.microsoft.com/webservices
Web Services Interoperability Organization: http://www.ws-i.org
Building Interoperable Web Services: WS-I Basic Profile 1.0: http://msdn.microsoft.com/library/en-us/dnsvcinter/html/wsi-bp_msdn_landingpage.asp
Longhorn Developer Center Indigo: http://msdn.microsoft.com/Longhorn/understanding/pillars/indigo
Patterns en practices: http://www.microsoft.com/resources/practices/application/services.mspx