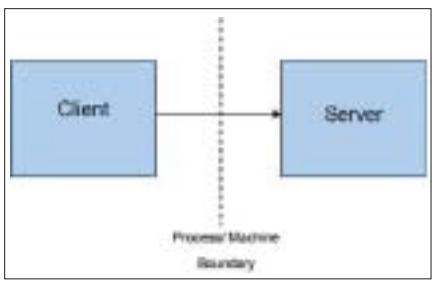
Anko Duizer
is trainer en consultant bij Class-A te Woerden
http://www.class-a-.nl

XML Web Services of .NET Remoting?

WANNEER GEBRUIK JE WELKE TECHNOLOGIE VOOR DE AANROEP VAN REMOTE SERVICES

In dit artikel worden twee technologieën behandeld waarmee een "remote" service kan worden aangesproken: XML Web Services & .NET Remoting. Er wordt kort beschreven hoe de twee technologieën in code kunnen worden toegepast. Daarnaast wordt ingegaan op de vraag "Wanneer gebruik ik welke technologie?".

In het .NET Framework is er de mogelijkheid om vanaf het ene proces een service (methode) aan te roepen binnen een ander proces, waarbij de processen eventueel over twee of meerdere machines verspreid kunnen zijn. In deze situatie wordt veel gesproken over een remote call of een remote procedure call . De situatie is weergegeven in afbeelding 1.



Afbeelding 1. Remoting overzicht

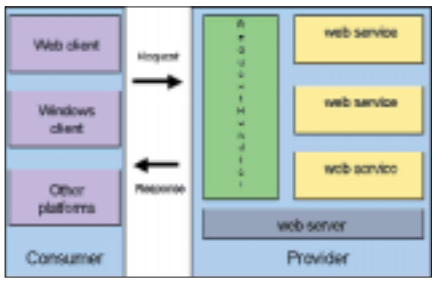
In afbeelding 1 roept een client een server aan, waarbij de server op een andere machine, danwel binnen een ander proces draait. Binnen het .NET Framework zijn er drie mogelijke opties om deze remote situatie te verwezenlijken:

1. DCOM;
2. XML Web Services;
3. .NET Remoting.

In dit artikel ga ik in op hoe een call verwezenlijkt kan worden met XML Web Services in .NET, en hoe een vergelijkbare remote call uitgevoerd kan worden op basis van .NET Remoting. Deze twee technologieën worden met elkaar vergeleken. DCOM wordt hierbij buiten beschouwing gelaten.

XML Web Services

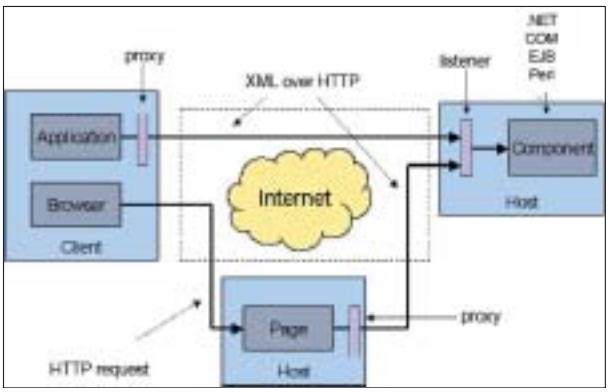
Wanneer er gesproken wordt over XML Web Services, gebeurt dat meestal in de context van integratie en het aanbieden van programeerbare services op het inter/intranet. Toch kan de web services-technologie binnen .NET ook worden gebruikt om eenvoudigweg een remote object aan te spreken. XML Web Services zijn binnen het framework sterk geïntegreerd en Visual Studio biedt uitgebreide ondersteuning om XML Web Services te bouwen en consumeren.
In de wereld van XML Web Services worden de client en server veelal Consumer en Provider genoemd. In afbeelding 2 is de basisarchitectuur van de XML Web Services binnen .NET weergegeven. Wanneer de consumer en provider op een aparte machine draaien toont de figuur in afbeelding 2 duidelijke overeenkomsten met die in afbeelding 1.



Afbeelding 2. XML Web Service-architectuur (1)

Wanneer de consumer wordt vervangen door de client, en de provider door de server hebben we de remote-situatie te pakken. Het standaardprotocol dat wordt gesproken tussen de client en de server is SOAP (Simple Object Access Protocol). Binnen het .NET Framework is het mogelijk om eenvoudigweg een proxy te generen op basis van de beschrijving

van een XML Web Service, de WSDL-file. De proxy kan worden gebruikt binnen de client code en op die manier wordt de code via de proxy doorgegeven naar de provider. Hier wordt de call in eerste instantie afgehandeld door de Internet Information Server. Daarna komen we vervolgens in de "managed world" terecht waar de werkelijke service code staat. Het een en ander is weergegeven in afbeelding 3.



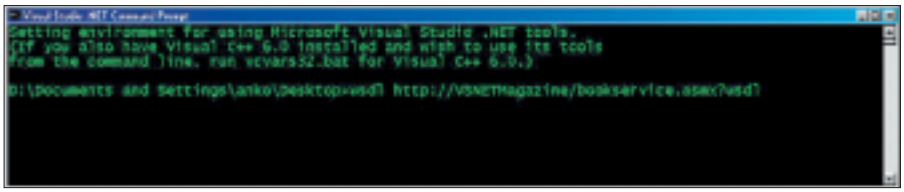
Afbeelding 3. XML Web Service-architectuur (2)

In theorie maakt het niet uit op welk platform of in welke taal je de consumer en/of de provider implementeert. Natuurlijk is het mogelijk om zowel de consumer als de provider in .NET te implementeren met bijvoorbeeld C# als taal. In de voorbeeldcode van afbeelding 4 vind je de eenvoudige code om een XML Web Service in .NET te implementeren.
Wanneer de code wordt geschreven in een file met de extensie ".asmx" onstaat

```
using System;
using System.Web;
using System.Web.Services;

namespace VSNETMagazine
{
  public class BookService : WebService
  {
    [WebMethod]
    public int OrderBook(string isbn, int amount)
    {
       // implement logic
       return 0;
    }
  }
}
```

Afbeelding 4. Voorbeeldcode van een eenvoudige XML Web Service



Afbeelding 5. WSDL.EXE

er een XML Web Service. De code kan nog eenvoudiger worden wanneer er geen gebruik gemaakt hoeft te worden van state. De `BookService` class overerft van de `WebService` class. Dit is alleen noodzakelijk wanneer men gebruik wil maken van de ASP.NET classes om state te bewaren zoals `Session` en `Application`. In principe zijn XML Web Services "stateless", maar met het gebruik van de ASP.NET objecten is het toch mogelijk om state te bewaren over meerdere methode-aanroepen.
Wanneer deze XML Web Service remote gebruikt moet gaan worden kan er via een command-line tool of via Visual Studio.NET een proxy worden gegenereerd in één van de volgende drie .NET talen: C#, VB.NET & JScript. Afbeelding 5 bevat een voorbeeldscherm van de aanroep om een proxy class te genereren.
Deze tool wordt op de achtergrond ook gebruikt door Visual Studio.NET om een proxy te genereren bij het maken van een web reference. Deze proxy kan vervolgens worden gebruikt binnen een project dat dient als client. Dit hoeft niet per se een webproject te zijn, maar kan bijvoorbeeld ook een windows-applicatie of een windows service zijn. Mogelijke client code kan er uit zien zoals in afbeelding 6.

Op basis van de code in afbeelding 6 is het dus eenvoudig om een remote-situatie te creëren met het .NET Framework en XML Web Services.

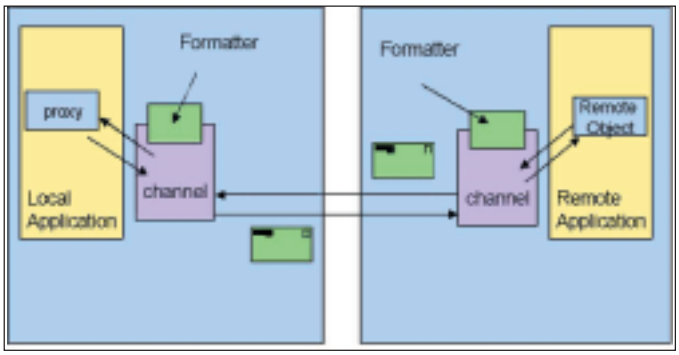
.NET Remoting

Een alternatief voor het gebruik van XML Web Services is .NET Remoting. Eigenlijk is .NET Remoting de werkelijke vervanging van DCOM. In z'n algemeenheid zou je kunnen zeggen dat .NET Remoting gemakkelijker uit te breiden en aan te passen is dan XML Web Services. .NET Remoting is beter geschikt voor inter/intranet-situaties dan DCOM. Om het goed te implemente-

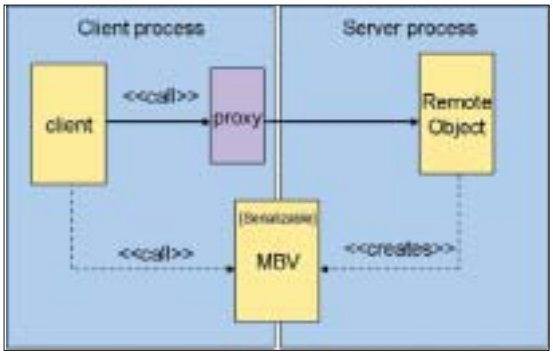
```
using System;
using System.Web;
using System.Web.Services;

namespace VSNETMagazine
{
 public class Client
 {
  public Client()
  { }
  public void AddSomeBook()
  {
  BookService _bs = new
    BookService();
  int _result = _bs.
    OrderBook("111-111-1123",10);
    }
  }
}
```

Afbeelding 6. Voorbeeldcode van een Web Service client



Afbeelding 7. .NET Remoting-architectuur



Afbeelding 8. Marshalling

ren is er echter diepgaande kennis noodzaklijk van de .NET Remoting-architectuur en –technologie; het is minder transparant dan DCOM.

In afbeelding 7 is op hoofdlijn de .NET Remoting-architectuur weergegeven. Ook hier wordt er gebruik gemaakt van een proxy object dat een afspiegeling is van het werkelijke object. Dit object wordt runtime gegenereerd zodat de client code er gebruik van kan maken. In werkelijkheid worden er twee proxies in memory geladen: “transparent proxy” en een “real proxy”. De “transparent proxy” overerft van de “real proxy” en zorgt ervoor dat de interface voor de client eruit ziet als het werkelijke server object.
Zodra een methode wordt aangeroepen op de proxy dan wordt er een boodschap opgebouwd die wordt geformatteerd in XML of een binair formaat. Vervolgens wordt het bericht door een channel overgezonden naar de server. Dit is of een Tcp of een HTTP channel. Aan de server kant gaat het bericht de omgekeerde weg. De formatter en channel kunnen naar behoefte worden geconfigureerd. Dit geeft de volgende vier mogelijke combinaties:

Protocol	Formaat
TCP	Binair
TCP	XML
HTTP	Binair
HTTP	XML

De formatter en het channel worden een “sink” genoemd. Het is mogelijk om de .NET Remoting-architectuur uit te breiden door het implementeren van een eigen sink, bijvoorbeeld om security te controleren. Op dit punt is de .NET-architectuur dus volledig uitbreidbaar.

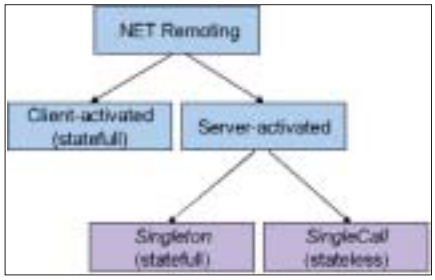
Type van objecten

Binnen .NET zijn in het kader van .NET Remoting drie soorten objecten te onderscheiden:
1. *Not-marshaled objects*. Objecten die op geen enkele manier remote kunnen worden aangeroepen. Dit is de default.
2. *Marshal By Value objects*. Objecten die geserializeerd kunnen worden en op die manier naar een remote client kunnen worden gestuurd.
3. *Marshal By Reference objects*. Objecten die écht remote aangesproken kunnen worden via de .NET Remoting-infrastructuur.

De “Marshal By Reference” objecten zijn het meest interessant in het kader van .NET Remoting. “Marshal By Value” objecten kunnen echter ook worden toepast voor remoting wanneer een object instantie in z’n geheel naar een ander proces moet worden verzonden. Dit is weergegeven in afbeelding 8.

Soorten .NET Remoting

Wanneer we uitgaan van “Marshal by Reference” objecten zijn er binnen het .NET Framework verschillende configuratiemogelijkheden. Naast de mogelijkheid van protocol- en formaatconfiguratie is het ook mogelijk te bepalen hoe het server object geïnstantieerd gaat worden. Dit heeft invloed op de levensduur van een server object en de mogelijkheid om state te bewaren. In afbeelding 9 zijn de mogelijkheden schematisch weergegeven.



Afbeelding 9. .NET Remoting-mogelijkheden

In feite zijn er dus twee instantiemogelijkheden: client-activated of server-activated. In het geval van client-activated hebben we eigenlijk te maken met een “normaal” object-instantie model. Iedere client krijgt een eigen instantie, bepaalt de state van dit object en hoe lang de instantie gebruikt gaat worden.
In het geval van server-activation moet er worden gekozen uit twee modes: *Sing-*

```
using System;
namespace VSNETMagazine
{
  public class BookService : System.MarshalByRefObject
  {
   public int OrderBook(string isbn, int amount)
   {
     // implement logic
     return 0;
   }
  }
}
```

Afbeelding 10. Voorbeeldcode van een Remotable class

```
<configuration>
  <system.runtime.remoting>
   <application name="SimpleServer">
     <service>
       <activated type="BookService, BookService" />
     </service>
     <channels>
       <channel ref="tcp server" port="9000" />
     </channels>
   </application>
  </system.runtime.remoting>
</configuration>
```

Afbeelding 11. Voorbeeldcode van een server configuratie file

leton of *SingleCall*. Wanneer gebruik wordt gemaakt van *Singleton* dan wordt er één object instantie aangemaakt die gebruikt kan worden over meerdere clients. Deze instantie kan state bewaren. Wanneer gekozen wordt voor de *SingleCall* mode dan wordt er voor iedere methode-call een object instantie aangemaakt en wordt de call uitgevoerd. Dit is een stateless programmeermodel.
Bij een keuze voor client-activated of de mode *Singleton* dient er rekening te worden gehouden met "lifetime-management". Oftewel de vraag "hoe lang leeft een object instantie?" moet worden beantwoord. .NET Remoting heeft hiervoor een standaard-infrastructuur gebaseerd op zogenaamde "leases". Het voert te ver om deze structuur volledig te behandelen in dit artikel, het is echter wel belangrijk om te realiseren dat het gebruik van .NET Remoting implicaties heeft voor het ontwerp.

Configuratie

De configuratie van de remoting infrastructuur kan worden gedaan met een XML configuratie file of in code. Het heeft de voorkeur om dit in een configuratie file te doen omdat dit tijdens deployment de flexibiliteit geeft om de configuratie aan te passen zonder een codewijziging door te hoeven voeren. In de voorbeeldcode in afbeelding 10 is de eerder beschreven XML Web Service geschikt gemaakt voor .NET Remoting.
Zoals zichtbaar is in de voorbeeldcode in afbeelding 10 is het werkelijke verschil dat we overerven van de class: *MarshalByRefObject*. Verder is het *[WebMethod]* attribuut niet meer noodzakelijk, het mag overigens wel worden gebruikt. Om deze class remote beschikbaar te stellen hebben we een host-process nodig. In het geval van een XML Web Service regelt de Internet Information Server dit. Bij .NET Remoting moeten we dit zelf regelen. Dit kan een willekeurige .NET-applicatie zijn. Het meest voor de hand liggend is een Windows Service. Deze host application moet een channel openen en luisteren op een specifiek port-nummer. Dit moet allemaal worden geconfigureerd in code of in een configuratie file. De voorbeeldcode in afbeelding 11 laat een mogelijke configuratie file zien.
In deze configuratie file wordt een server configuratie beschreven voor een client-activated instantie. Dit is herkenbaar aan het keyword "activated". Verder wordt gebruikt gemaakt van een Tcp channel en wordt er geluisterd op port nummer 9000. Wanneer gebruik wordt gemaakt van een Tcp channel, zal er default binair worden geformatteerd. Deze configuratie file moet worden ingelezen door de host-applicatie. Een voorbeeld hiervan is te vinden in de voorbeeldcode in afbeelding 12.

Het enige wat deze applicatie doet is het verzorgen van een process en het inlezen van de juiste .NET Remoting configuratie. Voor de client is het natuurlijk ook noodzakelijk om de configuratie in te lezen, hiervoor kan opnieuw worden gekozen uit de code-variant of een configuratie file. Een voorbeeld configuratie file is weergegeven in de voorbeeldcode in afbeelding 13. Een mogelijke client is weergegeven in de voorbeeldcode in afbeelding 14. In deze code wordt eveneens de .NET Remoting configuratie ingelezen. Daarna ziet de code eruit alsof de class lokaal wordt geïnstantieerd en aangeroepen. Het werkelijke remoting werk gebeurt achter de schermen door de proxy.

Wanneer welke technologie?

Er zijn dus binnen het .NET Framework meerdere mogelijkheden om een remote

```
using System;
using System.Runtime.Remoting;

namespace classa
{
  class SimpleServer
  {
   static void Main(string[] args)
   {
     RemotingConfiguration.Configure("Server.exe.config");

     Console.WriteLine("Press return to exit");
     Console.ReadLine();
   }
  }
}
```

Afbeelding 12. Voorbeeldcode van een .NET Remoting host server

```
<configuration>
  <system.runtime.remoting>
   <application name="SimpleClient">
     <client url="tcp://localhost:9000/SimpleServer">
       <activated type="BookService, BookService"/>
     </client>
     <channels>
       <channel ref="tcp client" />
     </channels>
   </application>
  </system.runtime.remoting>
</configuration>
```

Afbeelding 13. Voorbeeldcode van een client configuratie file

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Activation;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

namespace classa
{
  class SimpleClient
  {
   static void Main(string[] args)
   {
      RemotingConfiguration.Configure("Client.exe.config");
      BookService obj = new BookService();
      Console.WriteLine(obj.AddBook("1-9-0",10);
   }
  }
}
```

Afbeelding 14. Voorbeeldcode van een .NET Remoting client

	XML Web Service	.NET Remoting
Performance	Matig, door de vele lagen en vertalingen en het onderliggende formaat en protocol is de performance in zijn algemeenheid beperkt.	Uitstekend, te tunen op basis van protocol en message formaat
State	In principe stateless, echter in combinatie met ASP.NET is het mogelijk om state te bewaren	Mogelijkheid tot statefull en stateless calls
Platform/ OS beschikbaarheid	Meerdere platformen en operating systems. Uitstekende integratiemogelijkheden	Alleen beschikbaar in combinatie met een .NET runtime
Standaard	Ja	Nee
Uitbreidbaarheid	Aan de servererkant is het mogelijk dit te realiseren op basis van http-modules	Goede mogelijkheden op basis van eigen gebouwde sinks
Ontwikkelinspanning	Minimaal, veel wordt geregeld door het .NET Framework/ Visual Studio.NET	In de code/configuratie file moet rekening worden gehouden met .NET Remoting. Het is niet transparant
Bruikbaar in combinatie met firewall	Ja	Ja

procedure call te verwezenlijken. Maar wanneer gebruik je welke technologie? In de tabel op deze pagina is een vergelijking opgenomen van de twee in dit artikel beschreven mogelijkheden.
Volgens de tabel kan er worden geconcludeerd dat, wanneer performance belangrijk is en de client en de server beiden gebaseerd zijn op het .NET Framework, .NET Remoting de meest voor de hand liggende keuze is. Wanneer echter integratie met andere interne of externe systemen een rol gaat spelen, zijn XML Web Services de meest logische keuze. In de praktijk wordt binnen het eigen netwerk vaak gekozen voor .NET Remoting, en naar buiten toe (bijvoorbeeld het Internet) voor XML Web Services.

Voor beide technologieën is een plaats

Binnen het .NET Framework zijn er meerdere uitstekende manieren om een remote procedure call te verwezenlijken. Het is aan de developer/ontwerper om uiteindelijk de keuze te maken voor de juiste technologie. Dit hoeft niet beperkt te zijn tot één, voor beiden technologieën is een plaats.