Gijs de Jong is senior consultant bij Microsoft Services

Programmeren met Indigo

GEÜNIFICEERD FRAMEWORK VOOR SERVICEGEORIËNTEERDE APPLICATIES OP HET WINDOWS-PLATFORM

Indigo is een geünificeerd framework voor het bouwen van servicegeoriënteerde applicaties op het .NET Framework. Het biedt een interoperabele communicatie-infrastructuur voor het bouwen van veilige, robuuste en transactionele services. Indigo is een integraal onderdeel van Windows "Longhorn" en zal ook ondersteund worden op Windows XP en Windows Server 2003. In dit artikel wordt de problematiek beschreven waar Indigo oplossingen voor biedt. Daarna volgt een introductie op de architectuur en een uiteenzetting van het programmeermodel. Dit artikel richt zich met name op de Indigo-infrastructuur en gaat in mindere mate in op Service Oriented Architecture (SOA) of WS*-protocollen.

Tijdens de PDC in oktober 2003 is Indigo voor het eerst publiekelijk geïntroduceerd. Dit betrof een zeer vroege 'technology preview', waarin een beeld werd geschetst van wat de markt de komende jaren op het gebied van webservices van Microsoft mag verwachten. Conceptueel is er niet veel veranderd, maar de concrete architectuur en het programmeermodel hebben aardige metamorfosen ondergaan. Dit artikel is gebaseerd op een Microsoft-interne versie van Indigo. Tijdens het schrijven van dit artikel was de publieke pre-bètaversie namelijk nog niet beschikbaar. De mogelijkheid bestaat dus dat er verschillen zijn met de eerste publieke release.

Waarom hebben we Indigo nodig?

Wanneer een nieuwe technologie wordt geïntroduceerd, vraagt men zich altijd als eerste af 'waarom?'. De problematiek die Indigo aanpakt is onder te verdelen in drie categorieën: unificatie, interoperabiliteit en serviceoriëntatie.

Unificatie

Er zijn vandaag de dag verschillende technologieën voor het bouwen van gedistribueerde applicaties. Afhankelijk van de randvoorwaarden van de te bouwen applicatie zal hierin een keuze gemaakt moeten worden. Tabel 1 bevat een overzicht van de verschillende technologieën en belangrijkste karakteristieken.

Deze technologieën zijn door de jaren los van elkaar ontwikkeld en zijn daarom niet volledig op elkaar afgestemd. Dit heeft als gevolg dat de ontwikkelaar verschillende programmeermodellen moet beheersen. Een aantal technologieën beschikt zelfs over verschillende API's. Een voorbeeld hiervan is MSMQ met een .NET-, COM- en Win32-API. Indigo unificeert al deze programmeermodellen, technologieën en bundelt alle krachtige karakteristieken tot een eenduidig programmeermodel.

Technologie	Karakteristieken
System.Messaging	Berichtgeoriënteerd
Enterprise Services	Attribuutgebaseerd programmeermodel
.NET Remoting	Samenstelbaarheid en uitbreidbaarheid
ASMX	Servicegeoriënteerd en Interopabiliteit
Web Service Enhancements (WSE)	WS*-specificaties

Tabel 1. Huidige technologieën

Interopabiliteit

Vandaag de dag staan applicaties nog zelden volledig op zichzelf. Een computersysteem bestaat vaak uit een samenstelling van verschillende softwarecomponenten die met elkaar moeten communiceren over een netwerk. De integratie van deze componenten is een uitdagende aangelegenheid, zeker wanneer het verschillende platformen betreft.

Alle communicatie met Indigo-services is volledig gebaseerd op platformonafhankelijke industriestandaarden. Een aantal van deze gestandaardiseerde specificaties, zoals XML, XML Schema, SOAP en WSDL, kennen we al geruime tijd. Indigo maakt naast deze specificaties ook intensief gebruik van additionele webservice WS*-specificaties. Deze specificaties breiden het SOAP-protocol uit met zaken zoals security, transacties en reliable messaging. Hiermee wordt het mogelijk om veilige, betrouwbare en platform/technologieonafhankelijke gedistribueerde systemen te bouwen. Een aantal van deze specificaties is nu al te gebruiken door ASP.NET Web Services aan te vullen met Web Services Enhancements (WSE).

Er is in Indigo veel aandacht besteed aan integratie met de bestaande Microsoft-technologieën. Zo is het bijvoorbeeld mogelijk Enterprise Services-componenten als Indigo-services beschikbaar te stellen. Op deze manier kunnen investeringen in bestaande oplossingen gewoon blijven bestaan, terwijl nieuwe services gebruik kunnen maken van de faciliteiten van Indigo.

Serviceoriëntatie

Serviceoriëntatie is een architectuuraanpak voor het ontwerpen van gedistribueerde applicaties. Services abstraheren de onderliggende implementatie en communiceren op basis van berichten. Het contract van deze services en schema van de berichten wordt beschreven door gestandaardiseerde platformonafhankelijke specificaties. Services zijn autonoom, ze dienen onafhankelijk uitgerold en beheerd te worden. Deze aanpak vereenvoudigt integratie en evolutie van applicaties over tijd.

Behavior	Omschrijving
Instancing	Bepaalt instantiatie van een service (PerCall, PrivateSession, SharedSession, Singleton)
Error Handling	Bepaalt hoe het servicemodel omgaat met excepties
Concurrency	Bepaalt of er wel of niet meer requests gelijktijdig kunnen worden afgehandeld (Multiple, Re-entrant, Single)
Throttling	Bepaalt de maximale hoeveelheid gelijktijdige requests, connecties, service-instanties en wachtende berichten

Tabel 2. Behaviors

Indigo biedt de volledige infrastructuur voor het bouwen van schaalbare, interoperabele en betrouwbare servicegeoriënteerde systemen.

Indigo-architectuur

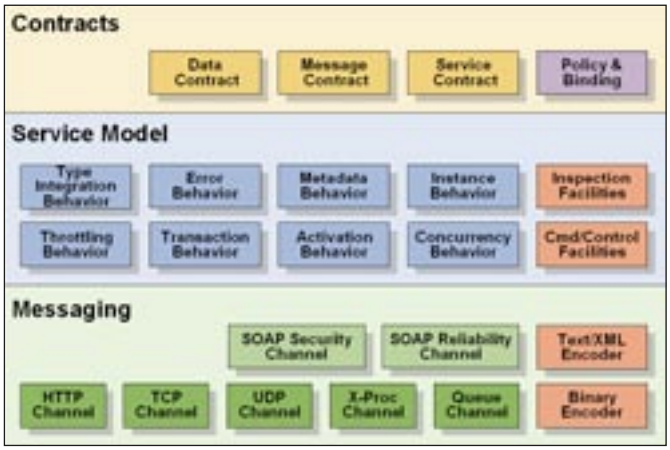
Indigo is gebaseerd op een gelaagd architectuurmodel. Op het laagste niveau bevindt zich de messaging-infrastructuur voor SOAP-berichtenuitwisseling over verschillende protocollen. Op de messaging-laag is het servicemodel gebouwd. Het servicemodel maakt de vertaling van low level berichtenuitwisseling naar de wereld van CLR-classes en methodes. Op het servicemodel bevindt zich de contractlaag. Door middel van deze laag worden service-, message- en datacontracten gedefinieerd die de externe representatie van een service weergeven. Ontwikkelaars kunnen op elk niveau van de architectuur insteken, afhankelijk van de vereisten van de te bouwen service.

Indigo bewerkstelligt unificatie door een samenstelbare architectuur. Alle onderdelen binnen de architectuur zijn modulair opgezet en kunnen samengesteld worden tot een geïntegreerd geheel. Deze modulaire opzet heeft ook een positieve invloed op performance. Een eenvoudige Indigo-service is licht van gewicht en zeer snel. Door een service uit te breiden met additionele infrastructurele zaken, zoals transacties, zal de overhead toenemen. Er worden dus alleen maar performancekosten 'betaald' voor de functionaliteit die de service ook daadwerkelijk gebruikt.

Messaging

De messaging-laag is de fundering van de Indigo-architectuur. Alle Indigo-services zullen direct of indirect van deze laag gebruik maken. Messaging is, zoals de naam al zegt, een berichtgeoriënteerd model. Dit is vergelijkbaar met MSMQ zoals we dat vandaag de dag kennen. De belangrijkste concepten binnen de messaging-laag zijn messages, channels en encoders. Binnen Indigo is een message een CLR-type dat een SOAP-bericht vertegenwoordigt. Het bericht bevat een *payload* in de vorm van de *Body* en eventuele infrastructurele elementen in de vorm van *Headers*. Indigo gebruikt standaard SOAP 1.2, maar door configuratie en/of policy kan ook aangegeven worden dat SOAP 1.1 gebruikt moet worden.

Channels hebben als in- en output Indigo-messages die ze moeten verwerken. Verwerken kan letterlijk alles betekenen. Zo zijn er bijvoorbeeld transportchannels, zoals HTTP, TCP en Named Pipes, die bedoeld zijn voor het versturen van het bericht. Een ander type channel is het security channel. Dit channel is verantwoordelijk voor het toevoegen/valideren van de beveiliging aan/van de SOAP-message. Channels kunnen ook aan elkaar gekoppeld worden. Dit is vergelijkbaar met sinks in .NET Remoting. Doordat channels aan elkaar gekoppeld kunnen worden ontstaat een zeer flexibele architectuur. Het security channel is namelijk net zo goed voor een HTTP-transportchannel te plaatsen als voor een TCP-channel. Dit geldt ook voor het reliability channel. Dit is mogelijk omdat security en reliability op SOAP-niveau worden bepaald en niet afhankelijk zijn van de mogelijkheden en beperkingen van het transportprotocol. Het channel-mechanisme is ook een uitstekende manier om interceptie uit te voeren. Het is mogelijk om op elk punt in de hele *stack* aan channels een eigen gebouwd channel te plaatsen.



Afbeelding 1. Indigo-architectuur

Binnen de messaging-laag heeft de programmeur te maken met objectinstanties van het messagetype. Wanneer dit object geserialiseerd moet worden naar een SOAP-bericht kunnen verschillende typen encoders gebruikt worden. Vanuit interopabiliteitsoogpunt ligt het gebruik van de Text/XML-encoder voor de hand. In sommige scenario's is het echter wenselijk een ander type encoder te gebruiken. Wanneer bijvoorbeeld op een zo efficiënt mogelijke manier tussen twee Indigo-services gecommuniceerd moet worden, levert het versturen van binary data een betere performance op. Indigo is volledig gebouwd op basis van haar eigen architectuurprincipes. Dat maakt het relatief eenvoudig zelf implementaties te maken van nagenoeg alle onderdelen binnen de architectuur, zoals een eigen transportchannel of encoder.

Servicemodel

De messaging-laag biedt de ontwikkelaar alle vrijheid en controle over het versturen en ontvangen van SOAP-berichten. Deze low level controle is echter niet altijd noodzakelijk. Ontwikkelaars zijn over het algemeen veel vertrouwder met het programmeren via classes en methodes dan met berichten. Het servicemodele biedt een hoger niveau van abstractie en slaat de brug tussen de wereld van berichten en die van CLR-classes en -methodes.

Zoals channels centraal staan in de messaging-laag staan *behaviors* centraal in de service runtime-laag. Behaviors bepalen het (infrastructurele) gedrag van een service. Dit is net zoals met channels op een samenstelbare manier opgezet, waardoor meer behaviors te combineren zijn. Het meest essentiële behavior is het 'Type Integration' behavior. Dit behavior maakt de vertaling van berichten naar classes en methodes. Op basis van een binnenkomend bericht wordt de desbetreffende methode op een class aangeroepen. De service runtime biedt ook support voor WSDL en andere aan metadata gerelateerde zaken. Voorbeelden van andere behaviors staan in tabel 2.

Het servicemodel biedt een declaratief programmeermodel. Hierbij kunnen alle behaviors via attributen aan een service worden toegekend. Tevens biedt het servicemodel support voor WSDL en andere aan metadata gerelateerde zaken.

Contracts

Contracts beschrijven de externe representatie van een service. Hierin wordt bijvoorbeeld vastgelegd wat voor operaties een service ondersteunt en wat voor bericht- en datastructuren geaccepteerd worden. Bij object remoting-technologieën zoals .NET Remoting wordt het contract impliciet gedefinieerd. De interface van het lokale object wordt ook meteen het contract voor externe applicaties. Indigo hanteert een duidelijke scheiding tussen de lokale OO-representatie en het externe servicegeoriënteerde contract. Voor alle aspecten die een externe representatie hebben zal dit expliciet aangegeven moeten worden. Dit wordt ook wel een Opt-Inmodel genoemd. Op deze manier wordt het veel eenvoudiger om

de evolutie van een service onder controle te houden.
Indigo onderkent drie typen contracten: data-, message- en servicecontracten.

- Het datacontract is een structureel contract waarmee aangegeven wordt hoe een CLR-object geserializeerd moet worden door de XmlFormatter. De XmlFormatter kan voor elk CLR-type een valide XSD Schema creëren.
- Het messagecontract is een specialisatie van het datacontract en biedt controle over de externe representatie van het SOAP-bericht. Alle aspecten van het SOAP data- en processingmodel worden hierin onderkend.
- Het servicecontract beschrijft de interface van een service en bepaalt het message exchange pattern. De externe representatie van het servicecontract komt overeen met de WSDL PortType-definitie.

Hosting

Zoals elke .NET-applicatie wordt een Indigo-service uitgevoerd binnen een CLR AppDomain. De verwerking vindt plaats op het moment dat een bericht arriveert. Het CLR AppDomain zal uiteraard binnen een host-proces moeten draaien. Indigo ondersteunt de volgende typen hosts:

- .EXE (bijvoorbeeld Windows Forms of Console-applicatie)
- Internet Information Services (IIS)
- Avalon (nieuwe presentatie subsysteem van Windows)
- Windows Activation Service (nieuwe service hosting-omgeving)
- Windows Service
- COM+

Indigo Programmeermodel

Nu we enigszins een beeld hebben van hoe Indigo is opgebouwd, wordt het tijd om het programmeermodel verder onder de loep te nemen. De beste manier hiervoor is om gewoon een eenvoudige service en corresponderende client te bouwen.

ServiceContract

We beginnen met het definiëren van het servicecontract van de service. Dit doen we door een CLR-interface te definiëren en een ServiceContract-attribuut op de interface te plaatsen. Elke operatie die we in het servicecontract willen opnemen zullen we expliciet moeten onderkennen door het OperationContract-attribuut op de methode op te nemen.

Bij het definiëren van operaties kan onderscheid gemaakt worden in twee programmeerstijlen; parameters of berichten. De parameterstijl is vergelijkbaar met een standaard methodedefinitie. Er kunnen nul of meer input/output-parameters zijn en optioneel een retourwaarde. *SubmitOrder1* geeft de parameterstijl weer. Dit is vergelijkbaar met ASP.NET webservices waar de onderliggende berichtenuitwisseling volledig wordt geabstraheerd.
Bij berichten kan er nog onderscheid gemaakt worden tussen sterk en zwak getypeerde berichten. Door het *Message*-object als enige parameter mee te geven, zoals in *SubmitOrder2*, heb je rechtstreeks toegang tot het (zwak getypeerde) bericht. Voor sterk getypeerde berichten, zoals in *SubmitOrder*, zal een MessageContract moeten worden gedefinieerd. Hier volgt straks meer informatie over.

Een ander aspect dat opvalt aan codevoorbeeld 1 is dat de *SubmitOrder*-operatie gespecificeerd is als *IsOneWay*. Op deze manier wordt aangegeven dat het een asynchrone operatie betreft. Dit resulteert in een WSDL-definitie waarin alleen een 'in'-bericht is gedefinieerd en geen 'out'. Met andere woorden, de aanroepende partij hoeft niet te wachten totdat het bericht verwerkt is. Zodra het SOAP-bericht in de lokale *transmit-buffer* op de client is weggeschreven, zal de code-executie op de client continueren. Alle andere operaties in codevoorbeeld 1 zijn request/response-interacties. Indigo ondersteunt ook zogenaamde duplex channels. Bij duplex channels wordt een dialoog opgezet tussen een client en een service, waarbij een of meer ongecorreleerde *OneWay*-berichten uitgewisseld worden.

MessageContract

Door het gebruik van het MessageContract-attribuut kan controle uitgeoefend worden op het SOAP-bericht. Hier wordt een onderscheid gemaakt tussen Headers en een Body. Deze kunnen gedefinieerd worden door het MessageHeader- en MessageBody-attribuut.

DataContract

Voor het Order-object dat we in de OrderMessage meesturen, zullen we expliciet moeten aangeven hoe het object geserializeerd moet worden. Dit doen we met het DataContract-attribuut. Door het DataMember-attribuut te plaatsen op fields en properties, kunnen we bepalen hoe het datacontract (XSD Schema) er uit moet komen te zien.

De serializer houdt bij het serializeren van het object geen rekening met de access-modifiers (public, private, et cetera.) maar alleen met de datacontract-attributen. Hierdoor kan heel expliciet onderscheid gemaakt worden tussen de interne OO-vertegenwoordiging en het publieke datacontract. Bijvoorbeeld *Date* zal

```
using System;
using System.Servicemodel;

[ServiceContract (Name = "OrderContract")]
public interface IOrderContract
{
// [OperationContract]
// void SubmitOrder1 (int orderId, double amount); // Parameter stijl

// [OperationContract]
// void SubmitOrder2 (Message message); // Bericht stijl (zwak getypeerd)

   [OperationContract (IsOneWay = true)]
   void SubmitOrder (OrderMessage message); // Bericht stijl (sterk getypeerd)
}
```

Codevoorbeeld 1.
Voorbeeld ServiceContract

```
[MessageContract(Action = "http://submitorderaction")]
public class OrderMessage
{
   [MessageHeader (
       Namespace = "http://namespace",
       Name = "MyHeader",
       MustUnderstand = false,
       Relay = false,
       Actor = "http://actor",
       Position = 2)]
   Public string MyHeader;

   [MessageBody]
   public Order Order;
}
```

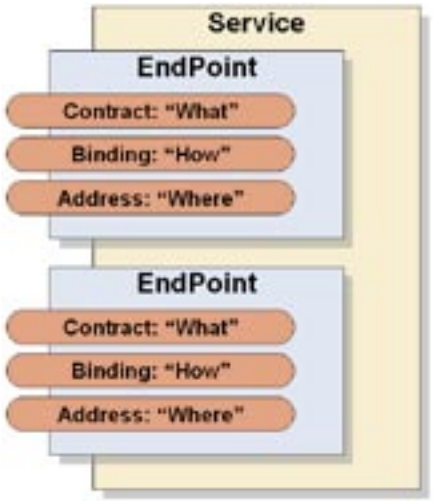
Codevoorbeeld 2.
Voorbeeld MessageContract

```
[DataContract (Name = "Order")]
public class Order
{
   public DateTime Date;

   [DataMember (Name = "Id")]
   public int _id;

   [DataMember]
   public double Amount;
}
```

Codevoorbeeld 3.
Voorbeeld DataContract



Afbeelding 2. Service en EndPoints

wel te zien zijn als publiek field van Order, maar zal niet in het datacontract naar voren komen. Het datacontract biedt ook een fraai versiemechanisme. Voor elke datamember kan aangegeven worden of deze optioneel is of niet. Op deze manier kan heel expliciet bepaald worden in hoeverre verschillende versies compatibel met elkaar zijn.

Services en EndPoints

De verschillende type contracten bieden de mogelijkheid heel nauwkeurig en expliciet controle uit te oefenen op de representatie van een service naar de buitenwereld. In de volgende stap kunnen we de werkelijke functionaliteit aan de service toevoegen. Zoals codevoorbeeld 4 laat zien, is dit vrij eenvoudig te realiseren door *IOrderContract* te implementeren in een eigen gedefinieerde class. Om de service nu ook aanroepbaar te maken zullen we een of meer zogenaamde Endpoints moeten definiëren. Een Endpoint bevat een adres en implementeert een servicecontract en een specifieke *binding*. Een *binding* bepaalt hoe het Endpoint te benaderen valt. Bijvoorbeeld een transport-binding geeft aan welk transportprotocol gebruikt moet worden. Encoding, security en reliability zijn echter ook typen bindings.

Een Endpoint kan declaratief aan een service worden toegekend door het ServiceEndpoint-attribuut; zie codevoorbeeld 4.

In codevoorbeeld 4 wordt bepaald dat de OrderService toegankelijk is over HTTP op port 80 en adres *'http://localhost/orderservice'*. Als we dezelfde service ook over TCP toegankelijk willen maken, is het mogelijk een tweede Endpoint te definiëren. Indigo ondersteunt naast HTTP en TCP ook Named Pipes voor cross-process-communicatie op dezelfde machine.

Naast het definiëren van een ServiceEndPoint hebben we in codevoorbeeld 4 ook twee behaviors aan de service toegekend met het ServiceBehavior-attribuut. De *InstanceMode.PerCall* geeft aan dat voor elk bericht een nieuwe service-instantie gecreëerd moet worden. Met de *ConcurrencyMode.Multiple* wordt bepaald dat de service multi-threaded is. Meer requests kunnen dan gelijktijdige afgehandeld worden.

In codevoorbeeld 4 hebben we declaratief aangegeven over welk protocol en op welk adres de service te benaderen is. Dit zou ook op een imperatieve wijze bewerkstelligd kunnen worden. Beide manieren zijn wel door code op te geven, maar zijn na het compileren niet meer te wijzigen. Uiteraard faciliteert Indigo ook uitgebreide configuratie. Relevante settings zullen na het compileren en uitrollen van een service nog steeds te wijzigen zijn. Een goed voorbeeld hiervan is natuurlijk een EndPoint van een service.

```
[HttpTransportBinding (Port = 80)]
[TextEncodingBinding]
interface IHttpBinding { } // HTTP transport + Text/XML Encoding

[ServiceEndpoint(Address= "http://localhost/orderservice",
  Binding=typeof(IHttpBinding),
  Contract=typeof(IOrderContract)]
[ServiceBehavior(InstanceMode = InstanceMode.PerCall,
  ConcurrencyMode = ConcurrencyMode.Multiple)]

public class OrderService:IOrderContract
{
  void SubmitOrder (OrderMessage message)
  {
    String myHeader = message.MyHeader; // haal Header uit het bericht

    Int id = message.Order.Id; // Haal OrderId uit de order
    Double amount = message.Order.Amount;

    // Code ...
  }
}
```

Codevoorbeeld 4.
Service-implementatie

```
<%@ service class="OrderService" %>
```

Codevoorbeeld 5.
IIS hosting in een .svc-bestand

```
ServiceHost<OrderService> host = new ServiceHost<OrderService>();
host.Open();

Console.WriteLine("De service is nu aanroepbaar ...");
Console.ReadLine(); // Houd de console applicatie open

host.Close();
```

Codevoorbeeld 6.
Service Hosting

```
Svcutil.exe http://localhost/orderservice/orderservice.svc
```

Codevoorbeeld 7.
Proxygeneratie voor een service met IIS als host

```
OrderContract channel =
ChannelFactory.CreateChannel<OrderContract>(); // Creëer een channel
                                               // object
OrderMessage message = new OrderMessage();

Order order = new Order();
order.Id = 10;
order.Amount = 150;

message.MyHeader = "header tekst …";
message.Order = order;

channel.SubmitOrder (message); // Roep de service aan en geef de
                               // message mee
```

Codevoorbeeld 8.
Aanroepen service

Service Hosting

Nu de service volledig is geïmplementeerd, gaan we de servicehost creëren. Zoals in de architectuursectie al is beschreven, zijn er verschillende manieren om een Indigo-service te hosten. Om IIS als host te laten fungeren, maken we allereerst een bestand aan met .de svc-extensie in een virtual directory. In dit bestand moet aangeven worden in welke class de service is geïmplementeerd.

Service Client

Om de service vanaf een client aan te kunnen roepen zullen we eerst een proxy moeten creëren. Zoals ASP.NET Web Services de WSDL.exe-tool kent, zo heeft Indigo SvcUtil.exe. Om voor de in IIS gehoste service een proxy te genereren, wordt de svcutil-tool aangeroepen volgens de manier die staat in codevoorbeeld 7.

SvcUtil creëert een bestand met de service Proxy code. Dit bestand moet vervolgens aan een Visual Studio .NET-project toegevoegd worden. In codevoorbeeld 8 staat hoe de OrderService vervolgens aangeroepen kan worden.

De versie van Indigo die voor dit artikel gebruikt is, biedt nog geen integratie met Visual Studio .NET. Daarom moet de SvcUtil.exe commandline-tool gebruikt worden. De finale versie van Indigo zal uiteraard volledig integreren met Visual Studio .NET.

Verder dan webservices

Indigo is een geünificeerd framework voor het bouwen van servicegeoriënteerde applicaties op het Windows-platform. Dit gaat veel verder dan webservices zoals we die vandaag de dag kennen. Indigo biedt een volledige communicatie-infrastructuur voor allerlei typen applicaties, inclusief de toekomstige generatie Microsoft-producten.
Indigo is een veelomvattende technologie. In dit artikel is slechts een introductie gegeven op de meest essentiële onderdelen van de architectuur en het programmeermodel. In volgende edities van het .NET Magazine mag je gerichtere en diepgaandere artikelen verwachten, onder andere op het gebied van security, reliable messaging en transacties.

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

Nuttige Internetadressen

http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnlong/html/introindigov1-0.asp
http://msdn.microsoft.com/theshow/episode046/default.asp
http://msdn.microsoft.com/Longhorn/understanding/pillars/Indigo/default.aspx
http://msdn.microsoft.com/webservices/
http://msdn.microsoft.com/webservices/community/blogs/