De technologieën en tools die expliciet bedoeld zijn om servicegeoriënteerde applicaties te maken staan nog in de kinderschoenen. Eén van de technologieën die zich aandient is Indigo, de codenaam voor Microsoft's framework voor het bouwen van servicegeoriënteerde applicaties. In dit artikel blikt Willem Koppenol alvast vooruit op Indigo dat zal verschijnen als onderdeel van de volgende release van het Windows-besturingsssysteem (codenaam Longhorn) die in 2006 wordt verwacht.

Vooruitblik op Indigo

Technologie voor servicegeoriënteerde applicaties

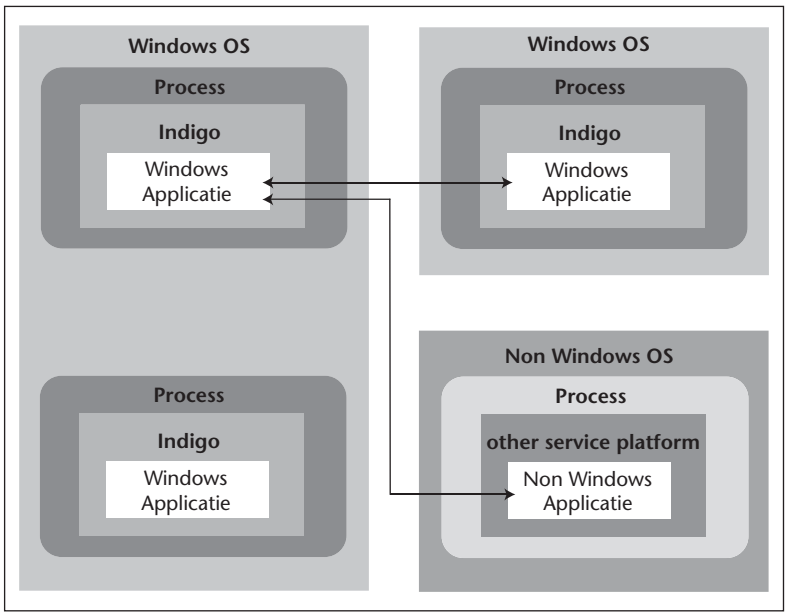
Objectoriëntatie is tegenwoordig het toonaangevende paradigma bij het ontwerpen en bouwen van applicaties. Objectoriëntatie zit verweven in moderne platforms als .NET en Java en wordt ondersteund door tools. Bij het ontwerpen en bouwen van de communicatie tussen applicaties zijn objecten echter minder succesvol gebleken. Voor de communicatie in gedistribueerde applicaties wordt service-oriëntatie steeds meer gezien als leidend ontwerpparadigma. Bij service-oriëntatie communiceren applicaties door via services met goedgedefinieerde interfaces berichten uit te wisselen. Het denken in services als fundamentale softwarebouwstenen is van recente datum.

INDIGO IN HET KORT Indigo is een onderdeel van WinFX, het framework voor het ontwikkelen van Microsoft Windows managed code. Naast Indigo maken ook andere technologieën, zoals Avalon voor het ontwikkelen van user interfaces hier deel van uit. Indigo is al beschikbaar in een pre-releaseversie. Je hebt er een besturingssysteem voor nodig dat door Indigo wordt ondersteund (Windows XP of Windows 2003), het .NET Framework 2.0 en een ontwikkelomgeving zoals Visual Studio 2005 (verschijnen beide in 2005) en het Windows Application Pack (WAP). Indigo bestaat uit een aantal classes die in de Common Language Runtime (CLR) draaien. Een centrale rol in Indigo speelt een drietal contracten: het servicecontract, het datacontract en het message-contract. Deze worden door middel van .NET-attributen aan classes gekoppeld.

SERVICE-ORIËNTATIE Indigo biedt expliciete ondersteuning voor het creëren van servicegeoriënteerde applicaties. Deze applicaties communiceren met elkaar door services te consumeren en aan te bieden. De services hebben expliciete XML-interfaces en draaien autonoom. Een service en de clients van de service kennen beide het interface maar zijn verder geheel onafhankelijk. De clients en de service wisselen informatie uit die staat beschreven in een XML-schema. Bij voormalige gedistribueerde objecttechnologieën, zoals Distributed COM (DCOM), was het gebruikelijk hele classes uit te wisselen. Bij service-oriëntatie is dit ongewenst vanwege de koppeling die tot gevolg heeft. Een leidraad bij het ontwerp van Indigo is verder dat de grenzen van services expliciet beschreven worden. Bij DCOM was het verbergen van distributie juist een doel. Ten overvloede zij nog opgemerkt dat objectoriëntatie binnen services en clients nog steeds gebruikt zal worden. De communicatie naar buiten verloopt via services.

INTEROPERABILITEIT Aangezien service en clients alleen een XML-interface delen kunnen ze geschreven zijn in een andere programmeertaal, draaien op een verschillend operating systeem of in een andere runtime omgeving gebruiken zoals de Common Language Runtime (CLR) of de Java Virtual Machine (JVM). Zolang ze dezelfde interface hanteren kunnen ze met elkaar communiceren.

Het fundamentele communicatiemechanisme in Indigo is SOAP. Verder committeert Indigo zich aan de webservice-standaarden die momenteel worden gespecificeerd zoals WS-Security, WS-Addressing en WS-Policy. De servicegeoriënteerde applicaties die met Indigo worden gemaakt kunnen zodoende communiceren met applicaties op niet-Windows systemen, zoals Java EE applicatieservers, die zich aan deze standaarden conformeren.



FIGUUR 1. Communicatie tussen Indigo-applicaties op Windows en andere plafformen.

UNIFICATIE VAN PROGRAMMEERMODELLEN

Naast service-oriëntatie en interoperabiliteit, betekent Indigo ook een unificatie van verschillende Windows-programmeermodellen voor gedistribueerde systemen. In de eerste releases van het .NET Framework heeft men vele keuzes voor het bouwen van gedistribueerde functionaliteit. Voor webservices wordt meestal ASP.NET (ASMX) gebruikt, eventueel aangevuld met de Web Services Enhancements (WSE). De WSE zijn als uitbreiding op .NET te downloaden en implementeren de nieuwste WS-specificaties op het gebied van webservices. .NET Remoting is een mogelijke optie bij het koppelen van twee .NET Framework-applicaties. Het gebruik van Enterprise Services, de .NET Framework opvolger van COM+, ligt voor de hand in situaties waar applicaties behoefte hebben aan gedistribueerde transacties. En tenslotte is er nog Microsoft Message Queuing (MSMQ) voor de creatie van applicaties die asynchroon berichten uitwisselen. Al met al een hele reeks technologieën, die weliswaar alle bestaansrecht hebben, maar ook verwarring veroorzaken onder ontwikkelaars. De vraag is immers wanneer welke technologie het beste ingezet kan worden. Indigo adresseert alle voornoemde problemen en verandert het keuzespectrum. Het antwoord op de vraag welke technologie het beste kan worden ingezet, luidt dan steevast: Indigo.

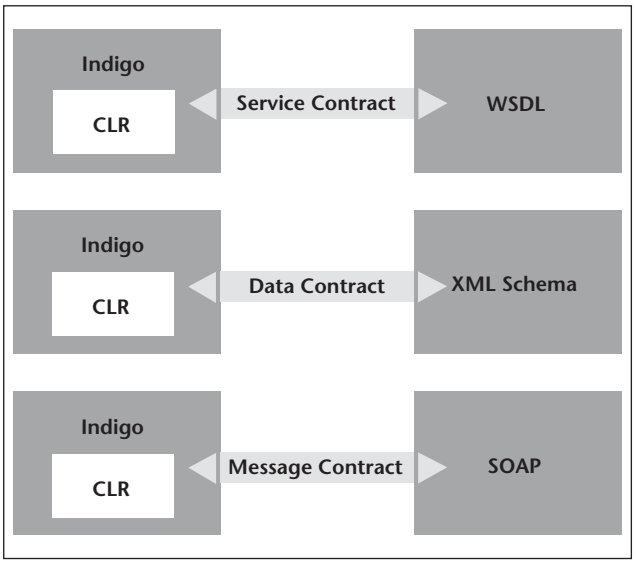
INDIGO SERVICES Een Indigo service bestaat uit vier elementen:

1) Een service class, geïmplementeerd in een .NET taal zoals C# of VB.NET
2) Minimaal één servicecontract eventueel aangevuld met data- en message-contracten
3) Een host-omgeving, applicatiedomein en het proces, waarin de service draait
4) Eén of meer endpoints die door clients worden gebruikt om de service te benaderen

Voor een goed begrip van Indigo zullen deze concepten nu nader worden toegelicht.

SERVICE CLASS Een Indigo service class is een gewone class maar met een paar toevoegingen in de vorm van .NET attributen. Deze attributen specificeren één of meer contracten die de class implementeert. Iedere Indigo service class implementeert minimaal één servicecontract, waarin de methoden staan die de service beschikbaar stelt. Attributen zijn een vorm van declaratief programmeren in het .NET Framework. Ze worden sinds de eerste release voor verschillende zaken gebruikt. Het zijn aanduidingen bij classes en methoden die aspecten van het gedrag van de class of methode wijzigen. Zo wordt het `[WebMethod]` attribuut in ASP.NET webservices gebruikt om aan te geven welke methoden clients kunnen aanroepen. In Indigo wordt een hele reeks van attributen gebruikt om Indigo-services te definiëren en controleren.

Ook een datacontract, dat de data waarmee de methoden werken specificeert, kan onderdeel zijn van de service. Evenals een message-contract dat de precieze inhoud van SOAP-berichten bepaalt.



FIGUUR 2. Verschillende contract typen van een Indigo service.

SERVICECONTRACT Een servicecontract beschrijft welke operaties door Indigo-clients kunnen aanroepen. Een servicecontract wordt gedefinieerd door een class of interface te markeren met het `[ServiceContract]` attribuut. Een service class kan zelf voorzien zijn van het `[ServiceContract]`attribuut of de service class kan een interface met dit attribuut implementeren.

Iedere operatie in de service class of het interface die clients kunnen aanroepen moet verder het [OperationContract] attribuut krijgen. Operaties die dit attribuut ontberen zijn uitgesloten van het servicecontract en kunnen dus niet door Indigo-clients worden aangeroepen. Een voorbeeld van een servicecontract dat geïmplementeerd wordt door de CarDealer class is:

```
  using System.ServiceModel;

  [ServiceContract]
  class CarDealer {
   [OperationContract]
   private Car buyCar(string brand , int
type, int price){
       Car car = new Car(brand, type,
price);
     ...
     return car;
   }
   [OperationContract]
   private int sellCar(Car car){
     ...
     return price;
   }
    public int calcTax(Car car, int
perc){
      ...
      return tax;
   }
    public int calcLease(Car car, int
term, int km){
      ...
      return leaseprice
   }
  }
```

In de CarDealer class zijn de buyCar en sellCar operaties beide voorzien van het [OperationContract] attribuut en dus door Indigo-clients aan te roepen. Dit geldt echter niet voor calcTax en calcLease.

In Indigo komen een tweetal abstracties samen: services en objecten. Beide abstracties gebruiken contracten, in het geval van objecten impliciet, om aan te geven wat in de buitenwereld zichtbaar is. Een class definieert, met behulp van de private en public keywords, de operaties die andere objecten in dezelfde applicatie kunnen aanroepen. In de CarDealer class zijn dat calcTax en calcLease. De buyCar en sellCar operaties zijn private en onzichtbaar voor andere objecten. Het objectcontract en het servicecontract zijn geheel verschillend en private methoden kunnen dus best door Indigo-clients worden aangeroepen.

Het is ook mogelijk servicecontracten te specificeren door uit te gaan van het interface-type. De attributen [ServiceContract] en [OperationContract] worden dan in het interface opgenomen. Een serviceclass kan vervolgens dit interface implementeren. Het gebruik van expliciete interfaces is iets gecompliceerder, maar ook flexibeler. Zo kan een class meer interfaces implementeren en dus ook meer servicecontracten. Via verschillende endpoints kunnen deze verschillende services dan beschikbaar worden gesteld aan verschillende clients.

Het [OperationContract] attribuut kan ook gebruikt worden om one-way operaties te beschrijven waarbij de service geen bericht terug stuurt. Ook interacties waarbij beide kanten fungeren als client en server, en operaties van elkaar aanroepen zijn mogelijk. In dit laatste geval spreekt men van een duplex contract. Dit soort zaken, waartoe ook het aangaan van een sessie met een Indigo-service behoort, wordt geregeld door de attributen parameters mee te geven.

DATACONTRACT Een datacontract beschrijft de data die bij de service-operaties wordt uitgewisseld. Een servicecontract betekent ook één of andere vorm van datacontract omdat de operaties uit het servicecontract gebruik maken van data. Soms is het datacontract impliciet gedefinieerd bij het servicecontract omdat de parameters en de returnwaarden van de operaties de uitgewisselde data volledig beschrijven. Bij de uitwisseling van simpele typen zoals integers is er geen behoefte aan een apart datacontract.

Services kunnen echter ook complexere typen uitwisselen, zoals classes. In dergelijke gevallen is een expliciet datacontract nodig. Een datacontract specificeert hoe complexe typen worden geserialiseerd zodat ze over het netwerk verstuurd kunnen worden.

Een datacontract wordt gedefinieerd door het [DataContract] attribuut voor de class naam te zetten. Het [DataMember] attribuut bij velden van deze class geeft aan dat deze velden in het serialisatieproces moeten worden meegenomen. Een simpel voorbeeld is:

```
[DataContract]
class Car {
 [DataMember] public string Brand;
 [DataMember] public string Type;
 [DataMember] private int Price;
 int public age;
}
```

Als een object van het type Car nu als parameter wordt meegegeven een methode die met [OperationContract] is gemarkeerd, zullen alleen de velden gemarkeerd met [DataMember], Brand, Type en Price, worden doorgegeven. De public en

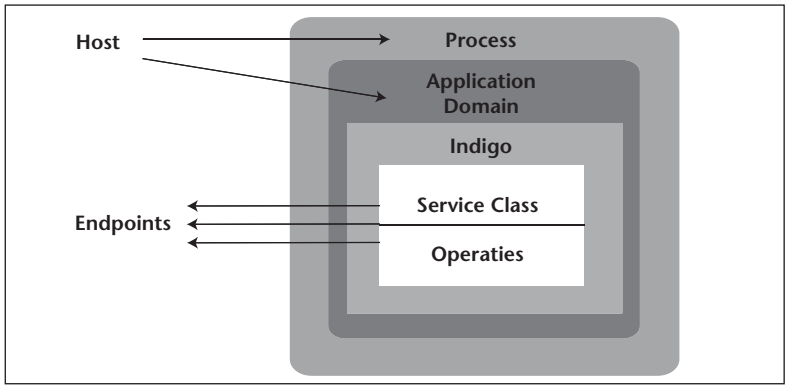
`private` keywords spelen hierbij geen rol.

Tenslotte moet nog worden opgemerkt dat geen enkele operatie of veld automatisch in het service- of datacontract wordt opgenomen. Indigo-services zijn gebaseerd op expliciete interfaces. Alleen operaties en velden voorafgegaan door respectievelijk `[OperationContract]` en `[DataMember]` zijn onderdeel van het service en data contract.

MESSAGE-CONTRACT Een message-contract beschrijft de precieze structuur van een SOAP-bericht. Het bepaalt met name of informatie in de header of in de body van het SOAP-bericht terecht komt. Dit kan van belang zijn om interoperabiliteit met andere systemen te realiseren. Met een message-contract kun je voldoen aan de precieze verwachtingen die een ander systeem aan de structuur van het SOAP-bericht stelt. Een message-contract wordt gespecificeerd door een class het `[MessageContract]` attribuut te geven. De details van het message-contract worden ingevuld door velden in de class te voorzien van de `[MessageBody]` en `[MessageHeader]` attributen. Een eenvoudig voorbeeld is:

```
[MessageContract]
public class CarPartRequest
{
    [MessageHeader]
    public string partNo;
    [MessageBody]
    public CarPart carPart;
}
```

De class definieert een typed message waarin de datatypes precies zijn aangegeven. Deze message kan nu als parameter bij service-operaties worden gebruikt. Het messagecontract geeft nauwe controle over het mappen van de data op de berichtstructuur. De `MessageHeader` en `MessageBody` attributen kennen nog verschillende parameters om nadere details van de message structuur te regelen. Het is overigens ook mogelijk een service te maken die met willekeurige, untyped messages werkt.



FIGUUR 3. Het hosten van een Indigo-service.

Desgewenst kan een ontwikkelaar zelfs op laag niveau toegang krijgen tot de messages en zelf communicatiekanalen openen.

HOSTING Indigo-serviceclasses worden typisch gecompileerd tot .NET-library's, ook wel assembly's genaamd. De service wordt actief als deze library in een Windows-proces in applicatiedomein wordt geladen. Er zijn twee opties voor het hosten van Indigo-servicelibrary's. Je kunt de service hosten in Internet Information Server (IIS). Of je kunt een willekeurig, zelf te bepalen, applicatiedomein en proces als host kiezen. We zullen beide beschrijven.

HOSTING IN IIS Het hosten van Indigo-services in IIS, is vergelijkbaar met het hosten van de 'oude' ASMX-webservices in IIS. In beide gevallen wordt er gebruik gemaakt van virtual directory's, die een alias vormen voor echte directory's in het Windows file-system.

In het voorbeeld van de `CarDealer` service kunnen we de service class compileren tot bijvoorbeeld de `cardealer.dll` library. Deze library plaatsen we vervolgens in een virtual directory vergezeld van een tekstbestand met de extensie `.svc` (voor service). IIS ziet de virtual directory en leest uit het `.svc` bestand dat het geacht wordt deze service te hosten. Voor de `CarDealer` service heet het bestand typisch `cardealer.svc` en heeft de volgende inhoud:

```
%@Service language=C# class="CarDealer" %
```

Als ook nog een endpoint wordt gedefinieerd zal bij een aanroep door een client van één van de operaties van de `CarDealer` service, automatisch een instantie van de class worden aangemaakt om de operatie uit te voeren. Deze instantie zal gaan draaien in het applicatiedomein waarin het standaard IIS-proces voorziet. Een voordeel van deze manier van hosten is dat de service automatisch wordt gestart en dat IIS voorziet in procesrecycling in geval van niet afgehandelde excepties.

HOSTING IN EEN WILLEKEURIG PROCES Het gebruik van IIS als host voor een Indigo-service is een eenvoudige oplossing. Soms hebben applicaties echter de behoefte om services vanuit hun eigen proces aan te bieden. Dit kan gerealiseerd worden met behulp van de `ServiceHost`-class. Deze class accepteert een willekeurige serviceclass als parameter van zijn constructor. De onderstaande voorbeeld code creëert een proces waarin de `CarDealer` service wordt gehost:

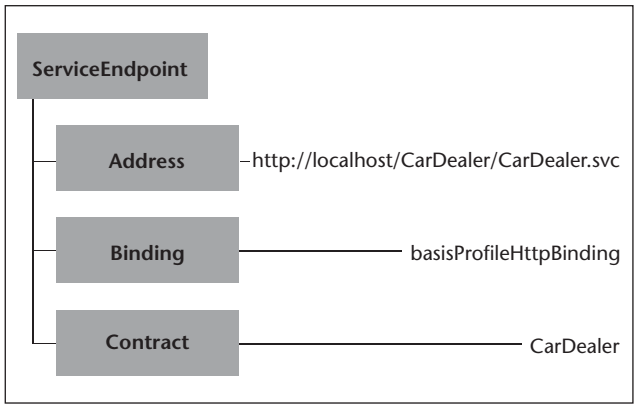
```
using System.ServiceModel;

public class CarDealerHost
{
```

```
  public static void Main()
  {
    ServiceHost<CarDealer> c =
      new ServiceHost<CarDealer>();
    c.Open();
    Console.Writeline("Press ENTER to end
service");
    Console.Readline();
  }
}
```

Bij het uitvoeren van de `Main` methode van `CarDealerHost` wordt een proces gestart waarin de `CarDealer` service gaat draaien. Na de creatie van de ServiceHost instantie en het aanroepen van de Open methode, is de service beschikbaar voor clients. De `ServiceHost<T>` class is een geparameteriseerde class, een generic, waaraan je tussen de tekens < en > de parameter meegeeft. Generics zijn een nieuw verschijnsel in versie 2.0 van het .NET-framework. Zolang het proces draait is de service beschikbaar voor clients en dit wordt in dit geval gerealiseerd door `Main` te laten wachten op input.

ENDPOINTS Een Indigo-service moet ook één of meer endpoints hebben om benaderd te kunnen worden. Alle communicatie met een Indigo-service verloopt via de endpoints van de service. Ieder endpoint wordt beschreven door drie elementen:

1) Een adres dat aangeeft waar het endpoint kan worden gevonden.
2) Een binding die aangeeft hoe een client met het endpoint kan communiceren
3) Een contract dat de operaties aangeeft die kunnen worden aangeroepen



FIGUUR 4. De drie elementen van een endpoint.

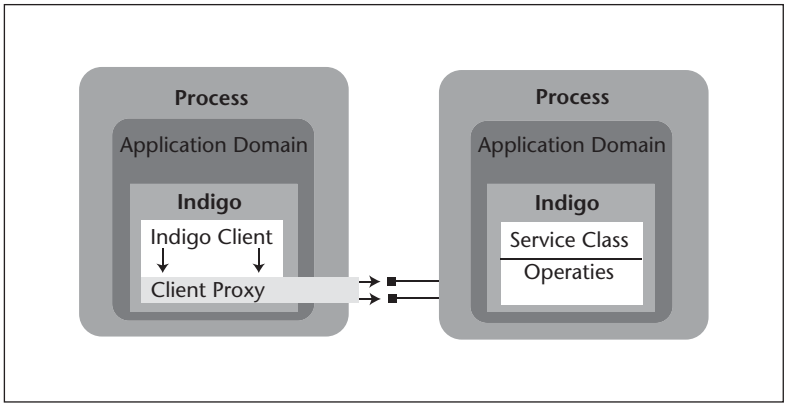
Een adres bestaat uit een URL die een machine op het Internet aangeeft plus een adresaanduiding van de service binnen die machine (de virtuele directory en het `.svc` file). Een binding bepaalt welk protocol kan worden gebruikt voor het benaderen van de service. Als een service zowel via SOAP over HTTP als via SOAP over TCP kan worden benaderd, zijn twee endpoints met een verschillende binding nodig. Ook andere zaken zoals of de communicatie betrouwbaar is en welk security-mechanisme wordt gebruikt zijn onderdeel van de binding. Om het gebruik van bindings te vergemakkelijken komt Indigo met een reeks voorgedefinieerde bindings die veel voorkomen. Hieronder bevinden zich de `BasicProfileHttpBinding` (SOAP over HTTP volgens de WS-I (Web Service Interoperability) standaard en de `BasicProfileHttpsBinding` (SOAP over HTTPS (Secure Sockets). De default binding is `BasicProfileHttpBinding.` Deze wordt gebruikt als de binding niet expliciet is aangegeven.

Een contract geeft aan welk service contract wordt aangeboden via dit endpoint. Een class die voorzien is van het `[ServiceContract]` attribuut en die geen expliciete interfaces implementeert kan slechts één servicecontract aanbieden. Alle endpoints gebruiken dan hetzelfde contract. Als de class echter verschillende interfaces implementeert die het `[ServiceContract]`-attribuut hebben, zal het aangeboden contract per endpoint kunnen verschillen.

Endpoints worden niet met attributen beschreven. Het is mogelijk endpoints programmatisch op te geven, maar meestal worden ze in een configuratiebestand beschreven. In geval van IIS hosted services is dat `web.config.` Services die in een willekeurig proces worden gehost zullen het configuratiebestand, `app.config`, van dit proces gebruiken. Het voordeel van het gebruik van een configuratiebestand is dat de settings zonder hercompilatie van code kunnen worden aangepast. Een voorbeeld van een deel van een configuratiebestand is:

```
<configuration>
  <system.serviceModel>
    <services>
      <service serviceType="CarDealer">
            <endpoint
               address="http://localhost/
CarDealer/CarDealer.svc"
               bindingType="basicProfileHttpB
inding" />
               contractType="CarDealer"
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

INDIGO-CLIENTS Indigo-clients kunnen Indigo-services benaderen via een proxy class die de remote service simuleert. Deze proxy kan uit de service worden gegenereerd door het `svcutil` tool. De client neemt de proxy op het client-proces en roept de locale service-operaties op de proxy aan. Onder water worden deze



FIGUUR 5. Indigo client

aanroepen door de genenereerde proxy-code vertaalt in berichten aan de operaties in de Indigo-service. Eén en ander wordt geïllustreerd in figuur 5.

Het tool `svcutil` kan ook zorgen voor de generatie van WSDL- en XSD-metadata. Deze metadata kan dan als input dienen voor tools die voor clients op andere platforms proxy's genereren. Een beter alternatief is nog het service-ontwerp te beginnen met de interface en type-beschrijvingen in WSDL en XSD. Niet alle classes kunnen namelijk vertaald worden in WSDL en XSD. Door de laatste als uitgangspunt te nemen worden problemen op dit gebied vermeden.

SLOTWOORD In dit artikel konden alleen de basisaspecten van Indigo op basis van de pre-release worden besproken. Deze basisaspecten kenmerken zich door een grote eenvoud. Indigo biedt ook ondersteuning voor complexere zaken zoals security, transacties, concurrency en messaging. Veel van deze zaken net als de basisaspecten zijn toegankelijk via declaratieve programmering met attributen. De besproken zaken zijn mogelijk nog aan verandering onderhevig voor de definitieve release.

Hoewel het nog wel even duurt voordat Indigo werkelijkheid wordt, is nu al wel duidelijk dat de Indigo-technologie een behoorlijke impact zal hebben. Indigo biedt veel ondersteuning voor de creatie van service-georiënteerde applicaties. Indigo biedt een unificatie van het programmeermodel voor gedistribueerde applicaties. Het wordt het fundament voor alle communicatie tussen Windows applicaties onderling en met niet-Windows applicaties. Indigo voert interoperabilteit hoog in het vaandel, maar voor de realisatie daarvan is het nodig dat ook andere partijen zich committeren aan de standaarden. De recente toenadering tussen Sun en Microsoft is daarbij hoopgevend.

drs. Willem Koppenol is senior trainer en productspecialist software development bij Twice IT Training (e-mail: wkoppenol@twice.nl)