Bart Wullems

Stop het gevecht tegen de tools

HET BOUWEN VAN N-TIER-APPLICATIES

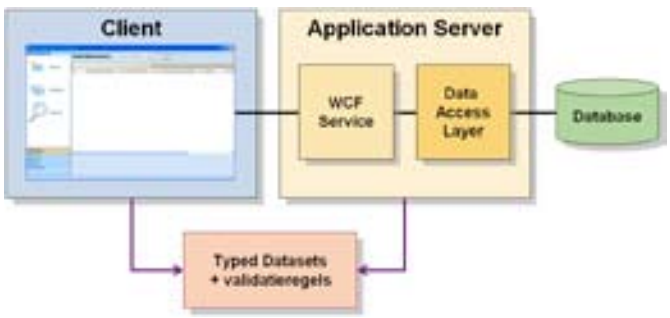
Van het nut van N-tier-applicaties hoeft niemand meer overtuigd te worden. Ze zijn onderhoudsvriendelijk, eenvoudig aanpasbaar en hebben een losse koppeling, waardoor ze ideaal zijn om een softwarearchitectuur te structureren. In dit artikel gaan we dieper in op de nieuwe mogelijkheden die Visual Studio 2008 ons biedt om op een betere manier N-tier-applicaties te bouwen.

Veel bedrijven ontwikkelden N-tier-applicaties met behulp van Visual Studio 2005. Toch waren hier enkele nadelen aan verbonden, waardoor soms een gevecht moest worden gevoerd tegen de beschikbare tools (zoals de dataset designer en de webservice proxy generator). Met het verschijnen van Visual Studio 2008 komt hier eindelijk verandering in. Visual Studio 2008 ondersteunt nu standaard WCF (Windows Communication Foundation). Ook de integratie is sterk verbeterd en levert een veel aangenamere ontwikkelervaring.
Het gebruik van meerdere lagen in applicaties is niet nieuw. Al jaren proberen ontwikkelaars hun code op te splitsen om zo tot een beter beheerbaar geheel te komen. Een architectuur zoals beschreven in afbeelding 1 vind je dan ook veelvuldig terug. Hierbij heb je een client-toepassing die communiceert met een back-end database via een applicatieserver. Deze client kan om het even wat zijn, een windows forms-applicatie, WPF-applicatie, een web-client, het maakt niet uit. Het feit is dat er data gelezen en geschreven worden van en naar een back-end database. Op de applicatieserver vinden we twee zaken terug. Enerzijds een WCF-service die de communicatie mogelijk maakt tussen de client en de applicatieserver en anderzijds een hele lading data access-code die weet hoe hij data moet lezen, schrijven en updaten in de back-end database. Veelal wordt er hier nog een extra layer geplaatst tussen de WCF service en de Data Access Layer. Dat kan om verschillende redenen gebeuren, schaalbaarheid, centralisatie van security, enzovoort. Voor dit artikel maken we gebruik van een iets vereenvoudigd model. In de praktijk zal je dit meestal uitbreiden met nog wat extra tussenliggende logica.

Pijnpunten in Visual Studio 2005

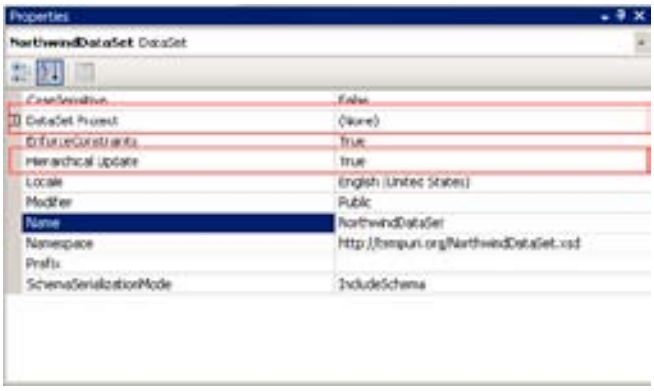
Enkele van de pijnpunten bij het bouwen van een dergelijke architectuur in Visual Studio 2005 kunnen we situeren rond de designers voor het helpen genereren van de data access-layer. Daarnaast was ook het genereren van proxies op de client om deze te laten communiceren met de WCF-service geen eenvoudige klus. Laten we eerst eens dieper ingaan op het pijnpunt rond de data access-layer. Visual Studio 2005 biedt een zeer goede ervaring voor het gebruik van Typed DataSets en het databinden met onder meer forms. Je hebt een designer die jou de keuze laat welke tabellen je wilt gebruiken in jouw toepassing. Een hoop code wordt dan voor je gegenereerd. Die code bestaat algemeen gesproken uit twee delen.

- Typed Dataset: deze bevat de structuur van jouw data in code waar je typische zaken zoals validatie wenst te plaatsen. Ze beschrijft de data, of het nu gaat over orders, customers, enzovoort.
- Table Adapters: deze bevatten een hele hoop andere code die nodig is voor alles wat bij data access komt kijken. De Table Adapter weet hoe alle SQL-commands uitgevoerd moeten worden, bevat de connection strings, enzovoort.



Afbeelding 1. Vereenvoudigde N-tier-architectuur

De DataSet Designer genereert dit alles voor je, wat op zich een zeer handige en gemakkelijke functionaliteit is, maar - en hier duiken de eerste problemen op - de tools stoppen zowel de Table Adapters als de Typed DataSets in hetzelfde project. Dat brengt enkele nadelen met zich mee. Alle data access-logica is toegankelijk voor elke klasse die naar de datasets refereert, gevoelige informatie zoals connection strings worden dan ook vrijgegeven. Veelal willen we toch onze dataset met wat eigen bijgeschreven logica (voorbeeld validatie) hergebruiken, zowel op de client- als op de serverzijde. Als je dat wil doen, dan moet je naar dat ene bestand verwijzen in je client-code, met als gevolg dat al jouw data access-logica ook op de client beschikbaar is, iets wat je absoluut probeert te vermijden. Een alternatief is - en hier begin je te vechten tegen de tools - om zelf een deel van de door de DataSet Designer gegenereerde code uit die ene file te plukken en deze handmatig te refactoren en in een aparte dll te plaatsen. Een tijdrovend proces dat bij elke wijziging moet worden herhaald.
Een ander pijnpunt vinden we op de clientzijde. Voor het communiceren met de WCF-service maakt het programmeermodel in Visual Studio 2005 gebruik van references. Door het toevoegen van een reference wordt een proxy gegenereerd op de client die alle objecten gaat genereren die de service gebruikt. Op de client gebruik je deze proxy dan om te communiceren met de service. In Visual Studio 2005 krijg je, op het moment dat je een referentie legt naar een service, alle types die deze service gebruikt opnieuw gegenereerd op de clientzijde. Voor veel scenario's (bijvoorbeeld een SOA-architectuur) is dit prima, maar wanneer je jouw eigen geschreven validatiecode in de Typed Dataset opnieuw wenst te gebruiken op de client, dan verlies je



Afbeelding 2. Nieuwe eigenschappen van de DataSet Designer

deze validaties doordat deze informatie niet mee gepubliceerd wordt in de WSDL van de service. De validatiecode wordt dus niet mee gegenereerd op de client. Het enige alternatief is via de command-line(met behulp van svcutil) de reference gaan vastleggen. Visual Studio gaat hierdoor bij het genereren van de proxy eerst kijken of een bepaald object reeds terug te vinden is in de gespecifieerde reference. Zo ja, dan zal hij in plaats van het object opnieuw aan te maken, een verwijzing leggen naar het bestaande object. Alhoewel dit het gedrag is dat we wensen, zijn we dan al niet meer met Visual Studio bezig en dat proberen we natuurlijk zoveel mogelijk te vermijden.
Deze pijnpunten zijn in Visual Studio 2008 aangepakt en opgelost. We demonstreren deze functionaliteit aan de hand van een demo. We hebben een solution die uit drie delen bestaat:

- Een windows form-client
- Een WCF-service host
- Een Northwind-data access layer

```
public partial class OrdersDataTable
{
   public override void EndInit()
   {
      //add validation event
      this.ColumnChanged += new
      DataColumnChangeEventHandler(OrdersDataTable_ColumnChanged);
      base.EndInit();
   }

   private void OrdersDataTable_ColumnChanged(object sender,
   DataColumnChangeEventArgs e)
   {
      if (e.Column.ColumnName == this.OrderDateColumn.ColumnName |
      e.Column.ColumnName == this.ShippedDateColumn.ColumnName)
         ValidateDates(e.Row as OrdersRow);
   }

   private void ValidateDates(OrdersRow row)
   {
      if (row.OrderDate > row.ShippedDate)
      {
         row.SetColumnError(this.OrderDateColumn,
         "Can't ship before it's been ordered");
         row.SetColumnError(this.ShippedDateColumn,
         "Can't ship before it's been ordered");
      }
      else
      {
         row.SetColumnError(this.OrderDateColumn, "");
         row.SetColumnError(this.ShippedDateColumn, "");
      }
   }
}
```

Codevoorbeeld 1.

Data access-laag aanmaken

Laten we beginnen met het aanmaken van een data access-laag door het toevoegen van een nieuwe class library met de hoogst originele naam DAL. We voegen een nieuw DataSet-item (NorthwindDataSet.xsd) toe, specificeren onze connectiegegevens, kiezen de juiste tabellen en voltooien de wizard zoals we dat al gewoon waren in Visual Studio 2005. Tot zover niets nieuws onder de zon. Op dit moment zitten we in de DataSet Designer waar we een orders- en customer-tabel hebben. Herinner je hier opnieuw het pijnpunt, waarbij zowel de DataSet als de data access-code worden gegenereerd in hetzelfde project? Wat willen we nu opgelost zien in dit scenario? We willen dat alle data access-code in de data layer blijft, maar dat de DataSets eruit gehaald worden en in een ander project worden geplaatst. Hoe krijgen we dat nu gedaan?

We voegen een nieuw project toe (opnieuw een class library) NorthwindEntities. Hier willen we dat de Typed Datasets uiteindelijk terechtkomen. In Visual Studio 2005 moesten we hiervoor handmatig de genereerde xsd-file openen, de code eruit knippen en in het nieuwe project plakken. In Visual Studio 2008 kan dit veel eenvoudiger. We klikken met de rechtermuisknop op de dataset en kiezen 'Properties'. We vinden hier een nieuwe property, namelijk de DataSet Project-property; zie afbeelding 2. Het enige dat we nog moeten doen, is deze property openklappen en het project kiezen waar de DataSets moeten terechtkomen. We kiezen hier Northwind-Entities, we bewaren de DataSet en klaar! Automatisch worden de juiste referenties gelegd, de code wordt opnieuw gegenereerd in het juiste project en we kunnen aan de slag. De DataSet Designer weet nu dat we willen dat de DataSet zelf in een ander project persisteert en doet hiervoor al het noodzakelijke. Als we nu in de Dataset Designer het View-Code-commando selecteren (via de rechtermuisknop) worden we automatisch naar het juiste project gebracht en wordt daar een nieuwe file aan toegevoegd. In dit bestand kunnen we dan onze eigen validatiecode toevoegen zonder dat een gevecht tegen de tools noodzakelijk is. We kiezen hier voor een eenvoudig voorbeeld, waarbij we controleren of de verzendingsdatum niet voor de besteldatum ligt; zie codevoorbeeld 1. Als laatste stap in de opbouw van onze data access-laag maken we nog een bijkomende klasse NorthwindManager aan die als API zal dienen voor het communiceren met de Table Adapters; zie codevoorbeeld 2.

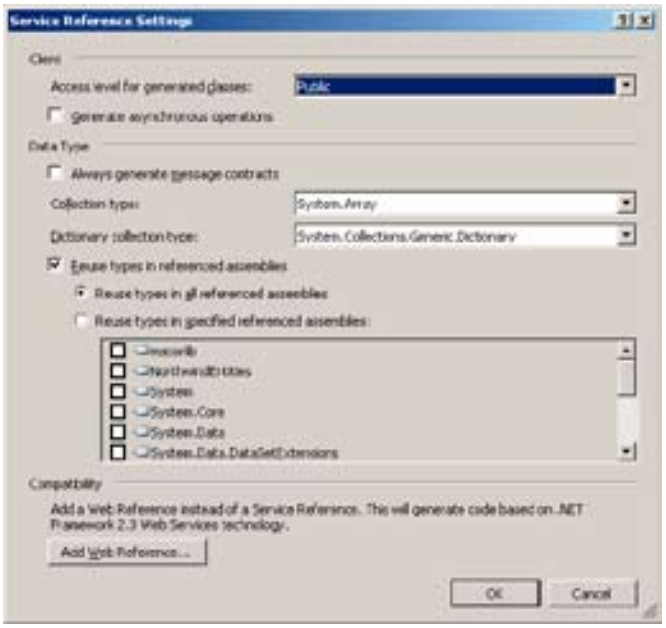
Aanmaken van de WCF-service

Laten we nu overgaan tot het aanmaken van onze WCF-service. We starten met het toevoegen van een nieuw WCF-webservice-project aan onze solution. Deze creëert een WCF-service die gehost wordt in IIS.

Opmerking: WCF moet zeker geen webbased service zijn. In Visual Studio 2008 bestaat er ook een nieuwe template voor service- libraries. Gebruikmakende van deze template kan Visual Studio services automatisch voor jou gaan hosten. Daarnaast is er ook nog een WCFTestClient die het testen en debuggen van WCF-services moet vereenvoudigen. Hier gaan we verder niet op in.

```
public class NorthwindMgr
{
   public static OrdersDataTable GetOrders()
   {
      OrdersTableAdapter ordersAdapter = new OrdersTableAdapter();
      return ordersAdapter.GetData();
   }
}
```

Codevoorbeeld 2.



Afbeelding 3. Bijkomende instellingen bij het leggen van een service-reference

Als ons serviceproject is aangemaakt, voegen we referenties toe naar zowel onze DAL als naar ons NortwindEntities-project. We vervangen de voorbeeld servicecontractinterface door codevoorbeeld 3 en passen ook de servicecontractimplementatie aan naar de code in codevoorbeeld 4.

Aanmaken van de WCF-client

Nu onze service klaar is, hebben we nog een client nodig om deze te consumeren. Op deze client willen we twee dingen doen:
een referentie leggen naar de zonet gemaakte WCF-service en het opnieuw gebruiken van de validatiecode die we geschreven hebben en die zich in het NorthwindEntities-project bevindt.

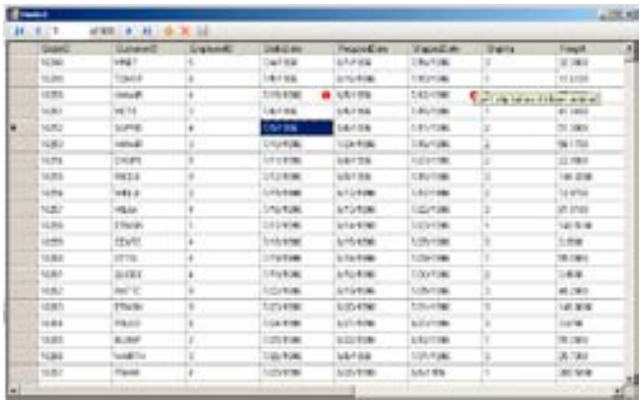
We voegen een nieuw Winforms-project toe aan onze solution. Als het project is aangemaakt, voegen we een verwijzing toe naar de WCF-service door simpelweg rechts te klikken op het project en voor 'Add Service Reference...' te kiezen. De Add Web Reference- mogelijkheid, die aanwezig was in 2005, is verdwenen en is nu ook beschikbaar onder de 'Add Service Reference...' actie. Een dialoogvenster verschijnt. Voor bestaande webservices (zowel ASMX als WCF) kun je hier de url ingeven en op 'go' klikken. Voor services binnen jouw solution is het echter voldoende op 'Discover' te klikken en alle services binnen de huidige solution worden getoond.
Iets wat ik onder de aandacht wil brengen is het scherm Service

```
[ServiceContract]
public interface IService
{
  [OperationContract]
  OrdersDataTable GetOrders();
}
```

Codevoorbeeld 3.

```
public class Service : IService
{
  //service implementatie
  public OrdersDataTable GetOrders()
  {
    return DAL.NorthwindMgr.GetOrders();
  }
}
```

Codevoorbeeld 4.



Afbeelding 4. Eindresultaat van onze demo

Reference Settings; zie afbeelding 3. Dat kun je laden door op 'Advanced...' te klikken op het scherm Add Service Reference. In dit scherm kun je een aantal aspecten van jouw WCF-service aanpassen. Ook nadat je een service-reference hebt gelegd, is het mogelijk dit scherm opnieuw op te roepen. In Visual Studio 2005 moest je hiervoor via de command-line de SDK-tools gebruiken. In Visual Studio 2008 is dat niet langer nodig en kun je de service-reference aanpassen zonder dat je de Visual Studio IDE moet verlaten.

Proxytype reuse

Een van de belangrijkste settings op het Service Reference Settings-venster is de "Reuse types in referenced assemblies". Deze setting verbergt een van de krachtigste toevoegingen in Visual Studio 2008. Herinner je het probleem dat de service bij het toevoegen van de reference een proxy gaat genereren? Op de client wordt dus een NorthwindDataset gegenereerd en deze is niet identiek aan onze NorthwindDataSet in onze businessentities-klasse. Zo ontbreekt al onze validatielogica op de client. Voor een SOA-architectuur is dit gewenst gedrag. Zolang de client een object kan serialiseren in hetzelfde formaat is dit oké. Wij willen echter niet de 'valse' NorthwindDataSet gebruiken, want dat zou betekenen dat we twee versies van onze validatielogica moeten beheren. Proxytype reuse kan ons hierbij helpen. Wat gaat dat doen? Aangezien we in onze solution zowel de client als de server controleren, kunnen we dezelfde onderliggende DataSet gaan hergebruiken, inclusief onze zelfgeschreven validatielogica. Omdat we op onze client naar NorthwindEntities refereren, zal Visual Studio in plaats van een nieuwe Typed DataSet te genereren - zoals het was in versie 2005 - de dataset in NorthwindEntities hergebruiken. De tool detecteert dus dat de DataSet die gebruikt wordt in de service ook beschikbaar is via een andere dll waar hij naar refereert. Hij gaat die dan ook gebruiken in plaats van een nieuwe aan te maken. Om deze demo te voltooien, gaan we naar Datasources en daar vinden we de twee tabellen (orders en customers) terug. We selecteren de

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
  }

  private void Form1_Load(object sender, EventArgs e)
  {
    ServiceReference.ServiceClient proxy =
    new ServiceReference.ServiceClient();
    this.northwindDataSet.Orders.Merge(proxy.GetOrders());
  }
}
```

Codevoorbeeld 5.

orderstabel en slepen deze op onze form. Een gridview wordt aangemaakt en alle databinding-settings staan ook net zoals we al gewoon waren uit Visual Studio 2005. Het enige dat we nog moeten doen, is de call naar de service schrijven; zie codevoorbeeld 5.

Nadat we op F5 hebben gedrukt, kunnen we onze toepassing eindelijk in actie zien. We hebben een 3-tierapplicatie die data ophaalt via een WCF-service. Wanneer we de datagrid veranderen en de verzendingsdatum verplaatsen naar een datum voor de besteldatum, zien we dat de grid de fout detecteert en hier netjes de foutboodschap toont; zie afbeelding 4.

Conclusie

We hebben kort gedemonstreerd hoe het mogelijk is om snel en eenvoudig een N-tier-applicatie te maken met Visual Studio 2008, waarbij we maximaal de nieuwe mogelijkheden van WCF en data access inzetten. Veel meer nieuws is toegevoegd aan Visual Studio 2008, maar we laten het aan de lezer zelf dit alles verder te ontdekken.

Bart Wullems is als softwarearchitect werkzaam bij Ordina Belgium (www.ordina.be). Hij houdt zich voornamelijk bezig met het uitbouwen van SOA-architecturen op het .NET-platform. Daarnaast focust hij zich vooral op het stroomlijnen van Application Lifecycle Management met behulp van Team Foundation Server. Bart is te bereiken via bart.wullems@ordina.be.

Referenties

What's new in VS2008?: http://msdn2.microsoft.com/en-us/library/bb386063.aspx
Application Architecture: An N-Tier approach: http://www.15seconds.com/issue/011023.htm
Introduction to Building WCF Services: http://msdn2.microsoft.com/en-us/library/aa480190.aspx