Complexiteit van Rich Internet Applications

WCF RIA Services maken ontwikkelaars productiever

Wouter Overmeer en Sander Schutten

Internetapplicaties met rijke gebruikersinterface nemen steeds meer in aantal toe, of ze nu zijn gebouwd met AJAX, Adobe Flex, Microsoft Silverlight of een andere technologie. Een Rich Internet Application (RIA) is een applicatie die draait in de browser en waarvan de functionaliteit en gebruikerservaring niet of nauwelijks onderdoet voor die van een desktopapplicatie.

Kenmerkend aan de architectuur van een Rich Internet Application is dat de presentatielogica op de client en applicatielogica op de server draait. Het ontwikkelen van Rich Internet Applications voelt hierdoor onnatuurlijk en druist in tegen wat we gewend zijn. Ineens krijgen we te maken met asynchrone communicatie, wachten op antwoord van de server, validatie op client én server, conflicten, concurrency, offline scenario's, authenticatie enzovoorts.

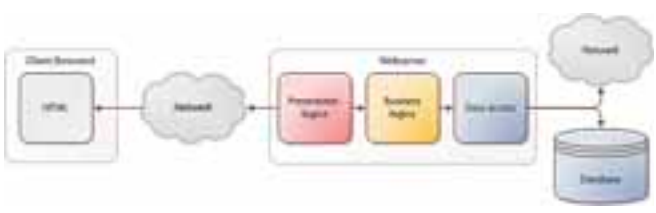
Van 2-tier naar 3-tier

WCF RIA Services hebben als doel het ontwikkelen van Rich Internet Applications te vergemakkelijken. Hieraan ten grondslag ligt de gedachte om client en server als twee helften van een applicatie te zien. Dit is het best te illustreren aan de hand van een voorbeeld. Figuur 1 toont de logische architectuur van een traditionele ASP.NET webapplicatie. Een traditionele ASP.NET applicatie is in de meeste gevallen een 2-tier applicatie, namelijk de webapplicatie en dataopslag. In een dergelijke applicatie bevindt alle logica zich op dezelfde plek, namelijk de webapplicatie. De communicatie tussen de logische delen vindt plaats via directe method calls.

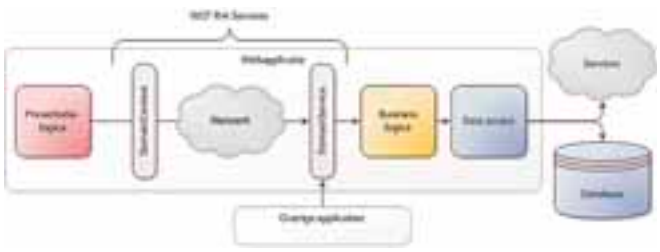
In het geval van Rich Internet Applications verschuift de presentatielogica naar de client (browser) voor een betere gebruikerservaring. Dit model wordt een 3-tier model genoemd, omdat er een extra fysieke laag is toegevoegd waar ook uitvoerbare delen van de applicatie draaien. Communicatie met de andere logische delen van de applicatie kan niet meer verlopen via directe method calls en is dus ingewikkelder. Het resultaat is dat een servicelaag moet worden ontwikkeld en onderhouden. Bij het bouwen van de servicelaag moet je zelf rekening houden met de nieuwe problemen die eerder in dit artikel zijn beschreven, zoals authenticatie en validatie.

De WCF RIA Services architectuur

WCF RIA Services beschouwen een Rich Internet Application als een geheel (zie Figuur 2), zoals je gewend bent bij het ontwikkelen van ASP.Net applicaties. Dit doen WCF RIA Services door de brug tussen client en server transparant te maken. Voor jou als ontwikkelaar vervaagt hiermee de grens tussen client en server, waardoor je het gevoel hebt een 2-tier applicatie te ontwikkelen. De kern van WCF RIA Services zijn de DomainService en DomainContext.
Een DomainService is een WCF RIA Services service die je op de server definieert en waarin je de businesslogica van de applicatie schrijft. WCF RIA Services zorgen ervoor dat de gebouwde functionaliteit wordt gerepliceerd naar de client als de DomainContext. Dit stelt je in staat om op de client via de DomainContext gebruik te maken van businesslogica die feitelijk op de server wordt uitgevoerd. WCF RIA Services zorgen voor de communicatie tussen de DomainContext (client) en DomainService (server).
Zoals de naam al doet vermoeden maken WCF RIA Services gebruik van WCF. Dit zorgt ervoor dat WCF RIA Services be-



FIGUUR 1: TRADITIONELE ASP.NET WEBAPPLICATIE.



FIGUUR 2: WCF RIA SERVICES TOEGEPAST OP RICH INTERNET APPLICATION.

FIGUUR 3: ADVENTUREWORKS WEBSHOP.

schikken over de kracht van WCF, zoals interoperabiliteit door gebruik van open standaarden, ondersteuning voor ASP.NET authenticatiemechanismen en herkenbaarheid voor ontwikkelaars, maar dan zonder de complexiteit van WCF.

DomainService

De DomainService zal vaak gebruik maken van een data-access technologie. Standaard ondersteunen WCF RIA Services verschillende data-access technologieën zoals Linq-to-SQL, Linq-to-Entities of NHibernate, maar het is ook mogelijk je eigen data-access technologie te gebruiken.
In de DomainService definieer je CRUD- of custom-operaties. De Read-operaties retourneren geen objecten, maar een IQueriable resultaat dat je met Linq kunt aanpassen, ook op de client. Zo kun je een query die alle producten uit een tabel teruggeeft op de client aanpassen, zodat alleen de producten met een bepaalde naam worden geretourneerd. De Linq-query wordt door WCF RIA Services geserialiseerd en naar de DomainService gestuurd om daar te worden uitgevoerd.

> WCF RIA Services beschikken over de kracht van WCF, maar dan zonder de complexiteit ervan.

Naast de DomainService zul je vaak een bijbehorend metadatabestand gebruiken. In het metadatabestand definieer je metadata classes, voor de objecten die de service aanbiedt, die validatieregels en attributen voor weergave beschrijven. De validatieregels pas je toe met behulp van attributen en lopen uiteen van maximale veldlengte, waardebereik, regular expressions of custom validatieregels. De validatieregels worden eveneens gerepliceerd naar de DomainContext en worden zowel op de client als op de server getoetst.

DomainContext

De DomainContext bevat verzamelingen van de verschillende IQueriable-objecten die worden aangeboden door de DomainService. Als er een Create-, Update- of Delete-operatie aanwezig is voor een IQueriable-object, dan is het ook mogelijk om het object aan te maken, te wijzigen of te verwijderen.
Het bewerken van objecten is batchgewijs. Dit wil zeggen dat alle wijzigingen aan objecten, zoals het aanmaken, updaten of verwijderen, worden opgespaard door de DomainContext totdat de SubmitChanges() methode wordt aangeroepen. Op dat moment bepaalt de DomainContext welke objecten zijn gewijzigd en alleen die objecten worden verstuurd naar de DomainService. Het opslaan van wijzigingen is transactioneel. Als er een fout optreedt bij het opslaan van de wijzigingen van een object in de transactie, worden alle wijzigingen in de transactie ongedaan gemaakt.

Starten met WCF RIA Services

Om WCF RIA Services te gebruiken download en installeer je de laatste versie WCF RIA Services van de RIA Services homepage. Op het moment van schrijven is dit RC2 voor Visual Studio 2010. Er is ook een versie voor Visual Studio 2008 beschikbaar, maar deze wordt niet actief doorontwikkeld en loopt iets achter op de RC2.
Hoewel WCF RIA Services in principe platformonafhankelijk zijn, zullen we in het voorbeeld in dit artikel gebruik maken van Silverlight. De reden hiervoor is dat bij de ontwikkeling van de eerste versie van WCF RIA Services de nadruk ligt op Silverlight. De ondersteuning en functionaliteit van de Silverlight-componenten is daarmee het grootst. Daarbij is Silverlight bij uitstek geschikt om de kracht van WCF RIA Services te laten zien.

AdventureWorks webshop

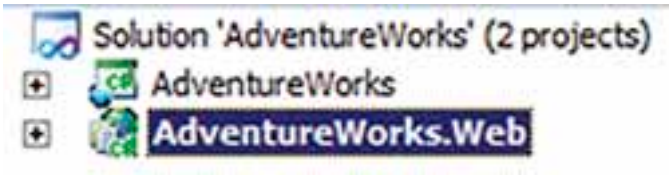
Aan de hand van een voorbeeldapplicatie op basis van de Microsoft AdventureWorks database kunnen we de kracht en het gemak van de WCF RIA Services demonstreren. De voorbeeldapplicatie die je met dit artikel gaat bouwen is een eenvoudig webshop-concept met filters, een productoverzicht, productdetails en validaties.

In Figuur 3 is de AdventureWorks webshop afgebeeld. Nu beschrijven we hoe de applicatie is opgezet en welke stappen je kunt doorlopen om zelf te experimenteren met WCF RIA Services en Silverlight. De opbouw van de webshop is als volgt: aan de linkerzijde staan de productcatalogus-selectie en een productfilter, in het midden staat het datagrid met de producten en een productpager en aan de rechterkant staat de detailweergave van een geselecteerd product.
De webshop is gebaseerd op het Silverlight Business Application project-template. Dit template bevat standaard een hoop Silverlight features en ziet er direct gelikt uit.

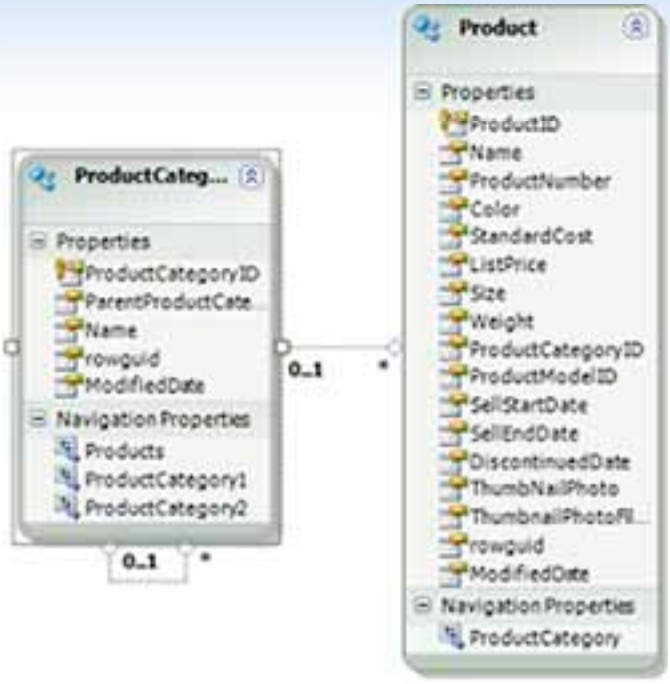
De oplossing

Bij het aanmaken van een Business Application solution worden automatisch twee projecten aangemaakt (zie Figuur 4). Het AdventureWorks project is het clientproject en bevat onder andere de Silverlight-pagina's in de views map. Het AdventureWorks.Web project is het serverproject en bevat onder andere de DomainService en validatieregels.

De volgende stap is het toevoegen van de DomainService aan de solution. Hiervoor is eerst een datamodel nodig. Voeg een ADO.NET Entity Data Model toe aan het serverproject. Verwijs naar de AdventureWorks database en selecteer de tabellen en views die



FIGUUR 4: SOLUTION-STRUCTUUR.



FIGUUR 5: HET DATAMODEL.

je in de applicatie wilt gebruiken. In dit geval gebruiken we de Product en ProductCategory tabellen.

Op basis van het datamodel kunnen we een DomainService class voor de applicatie genereren. Voeg hievoor een nieuw item toe aan het AdventureWorks.Web project en kies voor Domain Service Class. Belangrijk is wel om het AdventureWorks.Web project eerst een keer te builden, anders is het datamodel nog niet beschikbaar als DataContext.

Vink bij het aanmaken van de DomainService 'Enable client access' en 'Generate associated classes for metadata' aan. De 'Enable client access optie' zorgt ervoor dat de DomainService vanuit het clientproject kan worden benaderd. De gegenereerde metadata classes worden gebruikt om validatieregels toe te voegen.
Na het nogmaals builden van de solution verschijnt er een Generated_Code map in het clientproject (indien 'show all files' op het project is aangezet). Hierin staat de DomainContext die gegenereerd is op basis van de DomainService en die we kunnen gebruiken in het clientproject.

Productcategorieën laden

De DataContext kan zowel vanuit code als in XAML via databinding worden gebruikt. In de webshop wordt de data voor de ComboBoxen via code geladen en voor alle overige componenten via databinding. In Codevoorbeeld 1 en Codevoorbeeld 2 staan de benodigde code en XAML-definitie om de hoofdcategorieën te laden. Via de DomainContext worden alle hoofdcategorieën uit de SQL database opgehaald en deze worden via de LoadOperation in de ComboBox geladen. Via de Load-operatie wordt de data asynchroon opgehaald en getoond. Merk op dat de Linq-query die op de server wordt uitgevoerd op de client wordt aangepast.

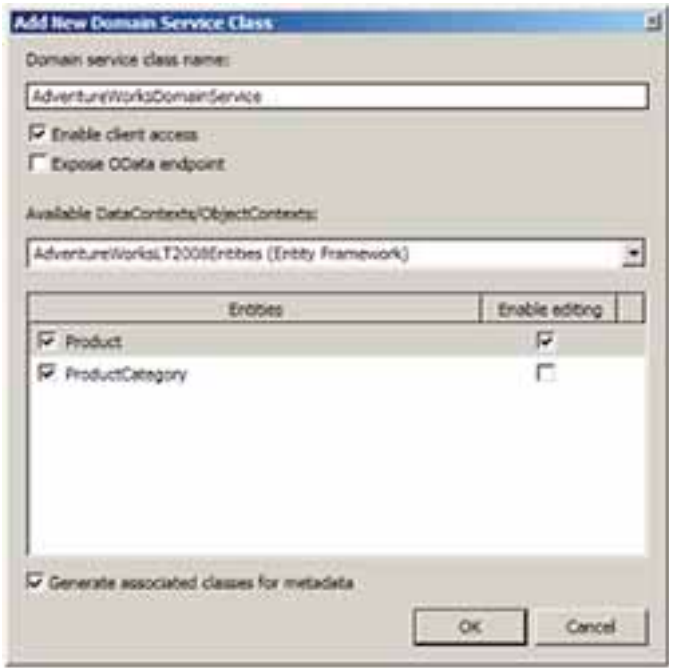
```
public Home()
{
  InitializeComponent();

  this.Title = ApplicationStrings.HomePageTitle;
  AdventureWorks.Web.AdventureWorksDomainContext adventureWorks
  Context =
new AdventureWorksDomainContext();
  EntityQuery<ProductCategory> rootProductCategoriesQuery =
adventureWorksContext.GetProductCategoriesQuery();
  rootProductCategoriesQuery =
from pc in rootProductCategoriesQuery
    where pc.ParentProductCategoryID == null
    select pc;
  adventureWorksContext.Load<ProductCategory>(rootProductCategorie
  sQuery,
RootProductCategoriesLoaded, null);
}

void RootProductCategoriesLoaded(LoadOperation<ProductCategory>
loadOperation)
{
  rootProductCategoriesComboBox.ItemsSource = loadOperation.Entities;
}
```

CODEVOORBEELD 1: CODE VOOR LADEN VAN DATA VOOR COMBOBOX.

```
<ComboBox Name="rootProductCategoriesComboBox" Width="120"
Height="23">
  <ComboBox.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding Name}" />
    </DataTemplate>
  </ComboBox.ItemTemplate>
</ComboBox>
```

CODEVOORBEELD 2: COMBOBOX DEFINITIE.

Nu de hoofdcategorieën zijn geladen kan de tweede ComboBox gevuld worden met de subcategorieën. De producten in de AdventureWorks database vallen allemaal onder een van de subcategorieën. Standaard is er op de DomainService geen operatie om producten aan de hand van een hoofdcategorieID op te halen. Deze operatie kan echter eenvoudig worden toegevoegd. De eerste stap is om een kopie te maken van de bestaande GetProducts-operatie in het AdventureWorksDomainService.cs bestand in het serverproject. Geef de nieuwe operatie een andere naam en een parameter van het type 'int' voor het ID van de categorie. Vervolgens voeg je een Linq-query toe om de producten uit de ObjectContext te selecteren aan de hand van het categorie ID. Zie Codevoorbeeld 3 voor de code van de query.



FIGUUR 6: TOEVOEGEN VAN EEN DOMAIN SERVICE.

```
public IQueryable<ProductCategory> GetProductsCategories(int root-
ProductCategoryId)
{
  return
from pc in this.ObjectContext.ProductCategories
where pc.ParentProductCategoryID == rootProductCategoryId
select pc;
}
```

CODEVOORBEELD 3: GETPRODUCTSCATEGORIES QUERY.

Nadat de solution opnieuw is gecompileerd, is de nieuwe query beschikbaar op de DomainContext in het client project. Door het SelectionChanged event op de rootProductCategoriesComboBox te implementeren kan de subProductCategoriesComboBox worden gevuld (zie Codevoorbeeld 4). Maak ook hier gebruik van Load-operatie om de data voor de ComboBox asynchroon op te halen.

```
private void rootProductCategoriesComboBox _ SelectionChanged(
object sender, SelectionChangedEventArgs e)
{
  ProductCategory cat = ((ComboBox)sender).SelectedItem as Product-
  Category;

  if (cat != null)
  {
    AdventureWorksDomainContext context =
 new AdventureWorksDomainContext();
EntityQuery<ProductCategory> rootProductCategoriesQuery =
context.GetProductsCategoriesQuery(cat.ProductCategoryID);
    context.Load<ProductCategory>(
rootProductCategoriesQuery, SubProductCategoriesLoaded, null);
  }
}
```

CODEVOORBEELD 4: SUBCATEGORIEEN LADEN.

Producten uit een categorie tonen

De ComboBoxen zijn beide gevuld en de DomainDataSource en het DataGrid kunnen worden toegevoegd voor het tonen van de producten uit de geselecteerde categorie. De DomainDataSource is een onzichtbaar user control dat wordt geleverd met WCF RIA Services. Met de DomainDataSource is het mogelijk

> De DomainDataSource is een onzichtbare user control dat wordt meegeleverd met WCF RIA Services.

in XAML, op een declaratieve manier, gegevens op te halen van de DomainService zonder dat er een regel code geschreven hoeft te worden. De DomainDataSource en DataGrid worden volledig in XAML gedefinieerd zoals in Codevoorbeeld 5 en Codevoorbeeld 6 is te zien.

```
<riaControls:DomainDataSource AutoLoad="True" x:Name="productDomain-
DataSource"
QueryName="GetProductsQuery">
  <riaControls:DomainDataSource.SortDescriptors>
    <riaControls:SortDescriptor PropertyPath="Name" />
  </riaControls:DomainDataSource.SortDescriptors>
  <riaControls:DomainDataSource.DomainContext>
    <my:AdventureWorksDomainContext />
  </riaControls:DomainDataSource.DomainContext>
  <riaControls:DomainDataSource.FilterDescriptors>
    <riaControls:FilterDescriptor IgnoredValue="Null" Operator=
    "IsEqualTo"
      PropertyPath="ProductCategoryID"
Value="{Binding ElementName=subProductCategoriesComboBox, Path=
SelectedItem.ProductCategoryID}" />
  </riaControls:DomainDataSource.FilterDescriptors>
</riaControls:DomainDataSource>
```

CODEVOORBEELD 5: DOMAINDATASOURCE DEFINITIE.

```
<sdk:DataGrid AutoGenerateColumns="False" x:Name="productDataGrid"
ItemsSource="{Binding ElementName=productDomainDataSource,
Path=Data}">
  <sdk:DataGrid.Columns>
    <sdk:DataGridTextColumn Header="Name" Width="Auto"
Binding="{Binding Path=Name}"/>
    <sdk:DataGridTextColumn Header="List Price" Width="Auto"
Binding="{Binding Path=ListPrice}"/>
  </sdk:DataGrid.Columns>
</sdk:DataGrid>
```

CODEVOORBEELD 6: DATAGRID DEFINITIE.

De FilterDescriptor in de DomainDataSource zorgt ervoor dat de producten worden gefilterd op basis van de gekozen subcategorie. Door de IgnoredValue te zetten worden er geen producten geladen bij het laden van de pagina als er nog geen categorie gekozen is. Aan de DomainDataSource kunnen meerdere controls worden gekoppeld, zoals de BusyIndicator en de DataPager. Deze zorgen met slechts twee regels XAML (Codevoorbeeld 7) voor een professionele indruk van de applicatie, namelijk een progressbar wanneer data wordt geladen en een pager voor het navigeren door grote aantallen producten.

```
<toolkit:BusyIndicator
IsBusy="{Binding ElementName=productDomainDataSource,
Path=IsBusy}"/>
<sdk:DataPager HorizontalAlignment="Left" VerticalAlignment=
```

```
"Bottom" PageSize="10"
Source="{Binding ElementName=productDomainDataSource,
Path=Data}"/>
```

CODEVOORBEELD 7: BUSYINDICATOR EN DATAPAGER.

Met behulp van de DataForm-control kan een detailweergave van het in de DataGrid geselecteerde product worden getoond. Codevoorbeeld 8 toont de XAML van de DataForm-control. Via de ItemsSource wordt de DataForm gekoppeld aan de DomainDataSource. In dit voorbeeld worden de kolommen automatisch gegenereerd. Het is ook mogelijk zelf kolommen te specificeren door het implementeren van de EditTemplate van de DataForm-control.

```
<dataFormToolkit:DataForm AutoGenerateFields="True" AutoEdit=
"True" AutoCommit="True"
ItemsSource="{Binding Path=Data, ElementName=productDomainData-
Source}" />
```

CODEVOORBEELD 8: PRODUCTDETAILS.

Validaties

Zoals eerder beschreven bieden RIA Services ondersteuning voor validatie en kunnen de validatieregels via het metadatabestand behorende bij de DomainService worden aangepast. In Codevoorbeeld 9 staan twee voorbeelden van validatieregels: een Required-attribuut voor een verplicht veld en een CustomValidator-attribuut voor custom validatie die in een aparte class wordt gedefinieerd.

```
[CustomValidation(typeof(CustomValidators), "IsValidProductNumber")]
public string ProductNumber { get; set; }

[Required(ErrorMessage = "You must enter a name.")]
public string Name { get; set; }
```

CODEVOORBEELD 9: VALIDATIEREGELS.

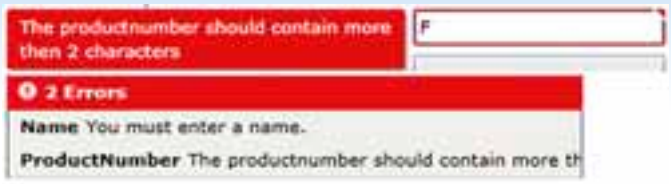
In Codevoorbeeld 10 staat de definitie van de IsValidProductNumber validatieregel. Bij het maken van de custom validatieclass is het belangrijk het bestand als Shared aan te merken. Op die manier wordt het bestand met de validatieregel automatisch gekopieerd naar het clientproject bij een build. Het bestand markeer je als Shared door dit op te nemen in de naam van het bestand, bijvoorbeeld CustomValidators.Shared.cs. In Figuur 7 staat het resultaat van de eerder gedefinieerde validatieregels wanneer je de productgegevens probeert te wijzigen.

```
public static ValidationResult IsValidProductNumber(
object value, ValidationContext context)
{
  string[] memberNames = new string[] { context.MemberName };
  if (((string)value).Length > 2)
  {
    return ValidationResult.Success;
  }
  else
  {
    return new ValidationResult(
"The productnumber should contain more than 2 characters",
memberNames);
  }
}
```

CODEVOORBEELD 10: CUSTOM VALIDATIEREGEL.

Voordelen op een rijtje

WCF RIA Services helpen je als ontwikkelaar om productiever te zijn bij het bouwen van Rich Internet Applications. De belangrijkste voordelen van WCF RIA Services zijn:



FIGUUR 7: VALIDATIEMELDINGEN.

- **Interoperabiliteit**; zoals de naam al doet vermoeden maakt WCF RIA Services gebruik van WCF. Onderwater is een DomainService een REST-service die ook door andere applicaties en platformen aangeroepen kan worden.
- **Productiviteit**; het bouwen van een goede servicelaag die voorziet in alle functionaliteit zoals authenticatie, validatie, concurrency, etc. is veel werk. WCF RIA Services maakt dit heel eenvoudig.
- **Consistentie**; door WCF RIA Services te gebruiken voor al je applicaties creëer je consistentie en herkenbaarheid in je applicatie.
- **Onderhoudbaarheid**; met WCF RIA Services hoef je geen volledige service laag te bouwen die ook nog onderhouden moet worden.
- **DataBinding support**; Met de DomainDataSource control en support van verschillende Silverlight controls zoals de DataGrid en Pager controls kun je snel en eenvoudig data-driven applicaties bouwen met paging, filtering, sorting en grouping.

Links

- WCF RIA Services home page
  http://www.silverlight.net/getstarted/riaservices/
- Silverlight homepage
  http://www.silverlight.net/getstarted/
- Nikhil Kothari's blog
  http://www.nikhilk.net/
- Brad Abrams' blog
  http://blogs.msdn.com/BradA/
- AdventureWorks Database
  http://msftdbprodsamples.codeplex.com/

Wouter Overmeer, is Consultant bij Avanade en houdt zich bezig met SharePoint- en custom .NET development.

Sander Schutten, is Senior Consultant bij Avanade. Hij specialiseert zich in rich application development op basis van Silverlight en WPF.