Silverlight 4 en Data access technologieën

Gill Cleeren

Ruim 3 jaar geleden werd Silverlight aan de wereld voorgesteld als Microsofts RIA (Rich Internet Application) platform. Sindsdien is het aantal Silverlight implementaties exponentieel toegenomen. Niet alleen media-gerichte applicaties maar ook LOB (Line-Of-Business) applicaties worden vaker met Silverlight gebouwd. Deze laatste hebben data nodig. In dit artikel gaan we dieper in op de opties die Silverlight biedt om met data te werken.

Silverlight is een client-side platform: de applicatie wordt gedownload als een *.xap file en draait geheel op de client. Echter, in heel wat applicaties zal data nodig zijn. Denk bijvoorbeeld aan een webshop in Silverlight waarbij de product- en prijsinformatie beschikbaar moet zijn. We zouden in eerste instantie kunnen denken dat we met een client-side database kunnen gaan werken, waarin deze informatie zich bevindt. Silverlight heeft in de huidige versie geen ondersteuning voor databases.

Als we naar de beschikbare assemblies en namespaces kijken, zien we dat er ook geen ADO.NET ondersteuning is. Dus geen DataReader, geen LINQ-To-SQL etc. Daarnaast zou een client-side database ook geen oplossing zijn voor alle scenario's: denk maar aan de gevaren die zouden ontstaan als een database met gevoelige informatie als wachtwoorden naar de client zou worden overgebracht!

De oplossing kan gevonden worden in het gebruik van services. Deze communiceren met een database aan de server-side, liefst via een business layer en maken zo data beschikbaar. Deze services kunnen we dan aanspreken vanuit onze Silverlight applicaties.

Als we kijken naar de ondersteunde mogelijkheden, vinden we de traditionele technologieën terug zoals ASMX (standaard webservices), WCF en REST. Ook sockets kunnen gebruikt worden sinds Silverlight 2. Daarnaast is er een aantal specifieke implementaties ontstaan om data access in Silverlight te vergemakkelijken, met name WCF Data Services en WCF RIA Services.

WCF & ASMX

WCF (Windows Communication Foundation) werd geïntroduceerd met .NET 3.0 en is sindsdien de de facto standaard geworden voor het bouwen van services in .NET. Het biedt een uniforme manier aan om met verschillende APIs te werken, waaronder ASMX services, remoting etc. Het is dan ook begrijpelijk dat Silverlight een zeer uitgebreide ondersteuning biedt voor WCF.

Hoe werken we nu vanuit Silverlight met WCF? Het proces omvat een aantal stappen die telkens terugkomen:

- Definiëer de service (ServiceContract)
- Definiëer de methodes ondersteund door de service (OperationContract)
- Creëer een service reference van de Silverlight applicatie naar de service.
- Gebruik de service methodes in de client-side code

In codevoorbeeld 1 wordt een standaard WCF service getoond. Het ServiceContract atribute geeft aan dat dit de service class is en het OperationContract attribute geeft aan dat de methode waarop toegepast, beschikbaar is voor clients van de service.

```
[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements
(RequirementsMode = AspNetCompatibility-
RequirementsMode.Allowed)]
public class OrderService
{
[OperationContract]
        public List<Product> GetAll-
        Products()
        {
            using (var context = new
            NorthWindModelDataContext())
            {

                var result = (from o in
                context.Products
                              select o);
                return result.ToList();
            }
        }
    }
```

CODEVOORBEELD 1

Vanuit Silverlight kunnen we naar deze service nu een reference leggen. Net zoals bij andere projecttypes, zal Visual Studio een proxy bouwen. Deze kan gezien worden als een client-side kopie van de service, met dat verschil dat de methodes geen implementatie bevatten, maar enkel een call naar de server-side method, zoals te zien in codevoorbeeld 2.

```
public System.IAsyncResult BeginGetAll-
Products(System.AsyncCallback callback,
object asyncState)
{
object[] _args = new object[0];
System.IAsyncResult _result = base.
BeginInvoke("GetAllProducts", _args, call-
back, asyncState);
return _result;
}
```

CODEVOORBEELD 2

In de gegenereerde code wordt gebruik gemaakt van BeginInvoke(). De reden hiervoor is dat alle service communicatie vanuit Silverlight asynchroon gebeurt. Immers, zou dit niet het geval zijn, dan zou tijdens het ophalen van de data de browser blijven hangen, aangezien Silverlight in het browser process draait.

Dit asynchroon gedrag is dan ook terug te vinden bij de code die we zelf zullen schrijven wanneer we van deze service methode willen gebruik maken, zoals te zien in codevoorbeeld 3.

```
private void button1_Click(object sender,
RoutedEventArgs e)
```

```
{
OrderService.OrderServiceClient proxy =
new OrderService.OrderServiceClient();
       proxy.GetAllProductsCompleted
       += new EventHandler<OrderService.
       GetAllProductsCompletedEvent-
       Args>(proxy_GetAllProducts-
       Completed);
       proxy.GetAllProductsAsync();
}

void proxy_GetAllProductsCompleted(object
sender, OrderService.GetAllProducts-
CompletedEventArgs e)
{
ProductGrid.ItemsSource = e.Result;
}
```

CODEVOORBEELD 3

Om vanuit Silverlight code te kunnen aanspreken moet de WCF service correct zijn geconfigureerd. Visual Studio heeft Silverlight-enabled WCF Service template aan boord, die deze configuratie correct zet. Er wordt standaard gebruik gemaakt van binary message encoding, wat betekent dat data die over de lijn gaat, gecodeerd wordt. Dit is alleen nuttig voor een snellere data overdracht en is geen security mechanisme. De binding die gebruikt wordt is niet de standaard voor WCF, namelijk de wsHttpBinding, maar de basicHttpBinding. In codevoorbeeld 4 zien we hiervan het relevante deel.

```
<system.serviceModel>
    <behaviors>
      ...
    </behaviors>
    <bindings>
      <customBinding>
        <binding name="WCFService.Web.
        OrderService.customBinding0">
          <binaryMessageEncoding />
          <httpTransport />
        </binding>
      </customBinding>
    </bindings>
...
    <services>
      <service name="WCFService.Web.Order-
      Service">
        <endpoint address=""
        binding="customBinding" binding-
        Configuration="WCFService.Web.
        OrderService.customBinding0"
          contract="WCFService.Web.Order-
          Service" />
        ...
      </service>
    </services>
  </system.serviceModel>
```

CODEVOORBEELD 4

Waarom nu WCF kiezen als data access technologie binnen Silverlight? Allereerst is WCF alom verspreid en hebben veel developers er kennis van. Afgezien van het verplicht asynchroon model, is het ontwerpen van WCF services voor Silverlight niets anders dan voor andere technologieën. Bovendien is een aantal optimalisaties aanwezig als de binary message encoding, die de data overdracht versnellen. WCF biedt ook binnen Silverlight de mogelijkheid voor duplex communicatie aan de hand van de PollingDuplexHttpBinding en net.tcp binding.
WCF biedt ook vele mogelijkheden tot het beveiligen van de getransfereerde data onder de vorm van de wsHttpBinding. Helaas is deze niet ondersteund door Silverlight. Daarom moeten we voor security terugvallen op een combinatie van SSL (https) en message-based security. SSL zal ervoor zorgen dat de data die overgedragen wordt, geëncrypteerd wordt. Message-based security zorgt ervoor dat de service kan controleren of de service-call geauthenticeerd kan worden: in de message header moet een username/password combinatie meegegeven worden die door de service kan worden gecontroleerd. In codevoorbeeld 5 zien we de relevante configuratie code die hiervoor nodig is: we voegen een serviceCredentials element toe in de behavior en aan de binding een security element waarbij authenticationMode=UserNameOverTransport.

```
<system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="MyService-
        Behaviour">
          ...
          <serviceCredentials>
            <userNameAuthentication user-
            NamePasswordValidationMode=
            "Custom"
             customUserNamePassword-
             ValidatorType=
"MessageBasedSecurity.Web.UsernamePW-
Validator,
MessageBasedSecurity.Web" />
          </serviceCredentials>
        </behavior>
        ...
      </serviceBehaviors>
    </behaviors>
    <bindings>
      <customBinding>
        <binding name="MessageBased-
        Security.Web.MissileLaunchService.
        customBinding0">
          <security authenticationMode=
          "UserNameOverTransport">
            <secureConversationBootstrap
            />
          </security>
          ...
        </binding>
      </customBinding>
    </bindings>
    ...
  </system.serviceModel>
```

CODEVOORBEELD 5

ASMX services werken op dezelfde manier vanuit Silverlight, maar zijn in functionaliteiten wel beperkter. De enige reden om vandaag nog voor standaard web services te kiezen is wanneer er al een infrastructuur aanwezig is die is gebaseerd op ASMX.

REST

Met WCF kunnen we dus bijna elk service scenario aan. Echter, in sommige situaties is simpele overdracht van textuele informatie al voldoende. In deze gevallen kan WCF voor een zekere overhead zorgen: elke message is immers verpakt in een SOAP envelop, wat voor heel wat extra data en rekenwerk zorgt. In deze gevallen kan REST (REpresentational State Transfer) soms een betere oplossing bieden. Het REST protocol is tegenwoordig erg populair, niet in het minste omdat heel wat Web 2.0 applicaties, zoals YouTube, Flickr en Facebook, hun API openstellen via REST. Met REST is de overgebrachte data pure, eenvoudige XML of JSON (JavaScript Object Notation). Ook Silverlight kan perfect overweg met REST services. In meer detail kunnen we, net zoals bij WCF, een aantal algemene stappen beschrijven die telkens terugkomen bij het werken met REST in Silverlight:

- Bouw URL op waarmee moet worden verbonden
- Stuur een request naar deze URL
- Verwerk de response (XML of JSON)

Als voorbeeld gaan we verbinden met een REST service van de Flickr API. De volledige API kan bekeken worden op http://www.flickr.com/services/api/; we zullen de flickr.photos.search methode gebruiken, die ons laat zoeken binnen foto's op tag.
Zoals gezegd moeten we eerst een URL opbouwen waarmee we verbinden. Flickr's API bepaalt deze. Ook vereist Flickr het gebruik van een API key, deze kan gratis verkregen worden op de eerder vermelde site. In code geeft dit codevoorbeeld 6.

```
string api_key = "1234567812345678";//
TODO: replace with your own key
string searchUrl = "http://api.flickr.com/
services/rest/?method=flickr.photos.
search&api_key={0}&text={1}";
```

CODEVOORBEELD 6

Eenmaal deze URL opgebouwd, moeten we er een request naar sturen. Dit kan ofwel door middel van de WebClient class of met de HttpWebRequest class. De eerste zal voor de meeste gevallen volstaan, de HttpWebRequest geeft meer controle over de request die zal worden gestuurd. In codevoorbeeld 7 zien we hoe we het verzoek sturen. Deze code werkt, net zoals bij WCF, asynchroon.

```
private void LoadPhotosButton_Click(object
sender, RoutedEventArgs e)
{
WebClient client = new WebClient();
       client.DownloadStringCompleted +=
```

```
        new DownloadStringCompletedEvent-
        Handler(client_DownloadString-
        Completed);
        client.DownloadStringAsync(new
        Uri(string.Format(searchUrl,
        api_key, "Eiffel Tower")));
         }
```

CODEVOORBEELD 7

Het resultaat van deze call is pure XML. Deze XML kunnen we binnen Silverlight verwerken met XmlReader/XmlWriter, de XmlSerializer of LINQ-To-XML. De laatste is meteen ook de meest eenvoudige om mee te werken. In codevoorbeeld 8 lezen we het XML resultaat uit. Op zich zijn dit nog geen objecten en dus gaan we zelf instanties aanmaken van een zelf-gemaakte class, nl de FlickrImage class.

```
void client_DownloadStringCompleted(object
sender, DownloadStringCompletedEventArgs e)
{
XDocument xml = XDocument.Parse(e.Result);
 var photos = from results in xml.
 Descendants("photo")
  select new FlickrImage
   {
   ImageId = results.Attribute("id").
             Value.ToString(),
   FarmId = results.Attribute("farm").
             Value.ToString(),
   ServerId = results.Attribute("server").
             Value.ToString(),
    Secret = results.Attribute("secret").
             Value.ToString()
    };
PhotoListBox.ItemsSource = photos;
}
```

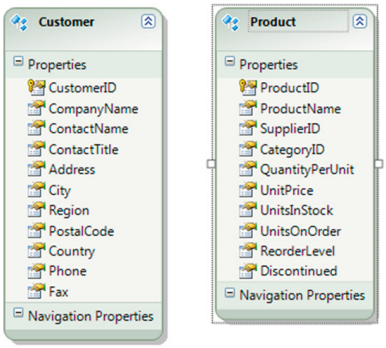
CODEVOORBEELD 8

Het nadeel aan het werken met REST is natuurlijk het ontbreken van een proxy class en daarmee ook het ontbreken van IntelliSense. Zoals in codevoorbeeld 8 te zien was, moeten we zelf de omschakeling maken van XML naar objecten.
REST is ook de basis voor een andere technologie, namelijk WCF Data Services. Deze lossen het laatst genoemde probleem op.

WCF Data Services

WCF Data Services, vroeger bekend onder de naam ADO.NET Data Services, is een server-side technologie die het mogelijk maakt om entiteiten van een model beschikbaar te maken over het web. Onderliggend wordt REST gebruikt om te communiceren.

Opdat de entiteiten beschikbaar gesteld kunnen worden, moet de data source de IQueryable interface implementeren. Dit kunnen we zelf doen op een model van eender welke data source ofwel gebruiken we als model het ADO.NET Entity Data Model (EDM). We gaan hier deze laatste gebruiken.
In de voorbeeldcode hebben we een zeer simpel ADO.NET Entity Data Model toegevoegd aan het web project (zie figuur 1).



Om nu deze beide entiteiten beschikbaar te stellen, voegen we een WCF Data Service toe. In deze service geven we aan dat deze entiteiten voor alle operaties beschikbaar zijn. In codevoorbeeld 9 wordt dit gedaan door de SetEntitySetAccessRule() methode.

```
    public class NorthWindService :
    DataService<NorthwindEntities>
    {
        public static void Initialize-
        Service(DataServiceConfiguration
        config)
        {
            config.SetEntitySetAccessRule
            ("Products", EntitySetRights.
            All);
            config.SetEntitySetAccessRule
            ("Customers", EntitySetRights.
            All);
            config.DataServiceBehavior.
            MaxProtocolVersion = Data-
            ServiceProtocolVersion.V2;
        }
    }
```

CODEVOORBEELD 9

Als we nu met een browser naar deze service navigeren, dan krijgen we een XML voorstelling van de data. Zo zal http://localhost:1234/Northwindservice.svc/ een overzicht geven van de beschikbare entiteiten, zoals te zien in de XML code van codevoorbeeld 10.

```
<?xml version="1.0" encoding="utf-8"
standalone="yes" ?>
<service>
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="Customers">
      <atom:title>Customers</atom:title>
    </collection>
    <collection href="Products">
      <atom:title>Products</atom:title>
    </collection>
  </workspace>
</service>
```

CODEVOORBEELD 10

We kunnen hier nu dieper op ingaan, om zo de entiteiten te bekijken en doorzoeken. Zo zal http://localhost:1234/Northwindservice.svc/Customers('ALFKI') ons de gegevens van Customer met ID ALFKI terug geven als XML.
Vanzelfsprekend kunnen we dit, net zoals in het voorbeeld met Flickr, met een WebClient gaan ondervragen en zelf binnen Silverlight omzetten naar instanties van zelf-gecreëerde classes. Dit heeft echter niet veel nut. Daarom bestaat er de WCF Data Services client library voor Silverlight. We kunnen client-side LINQ queries gaan schrijven die vertaald worden in een URL. De resulterende XML wordt vertaald in client-side entiteiten.
In codevoorbeeld 11 wordt gebruik gemaakt van de service. Zoals je ziet kunnen we nu volledige typed werken met de server-side data: we bouwen client-side een LINQ query op, deze wordt vertaald in een URL. Achterliggend wordt naar deze URL een request gestuurd, waarop XML (REST gebaseerd dus) terug komt. Deze wordt door het framework omgezet in objecten, die we in onze code kunnen verwerken.

```
NorthwindEntities context = new
NorthwindEntities(new Uri("NorthWind-
Service.svc", UriKind.
Relative));
ObservableCollection<Product> products-
Collection;

private void LoadProductsButton_Click
(object sender, RoutedEventArgs e)
{
var query = from p in context.Products
              select p;
DataServiceQuery<Product> dsq = (Data-
ServiceQuery<Product>)query;
       dsq.BeginExecute(ProductsLoaded
       Completed, dsq);
}

private void ProductsLoadedCompleted
(IAsyncResult asr)
{
DataServiceQuery<Product> dsq = (Data-
ServiceQuery<Product>)asr.AsyncState;
       productsCollection = new Observable
       Collection<Product>();

       foreach (var product in dsq.
       EndExecute(asr).ToList())
       {
          productsCollection.Add(product);
}
ProductGrid.ItemsSource = products-
Collection;
}
```

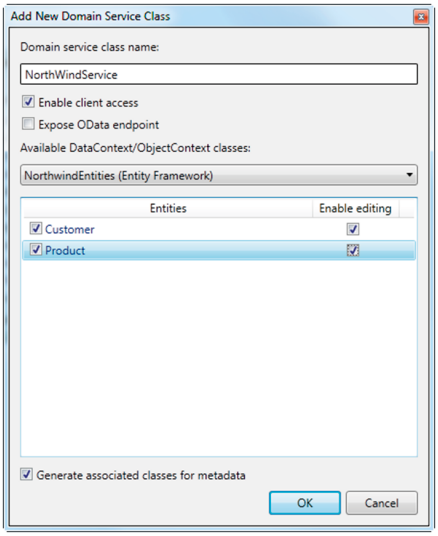
CODEVOORBEELD 11

Overigens: WCF Data Services zijn bruikbaar vanuit Silverlight maar daarnaast bestaat er ook een client library voor gebruik vanuit ASP.NET Ajax en standaard .NET.

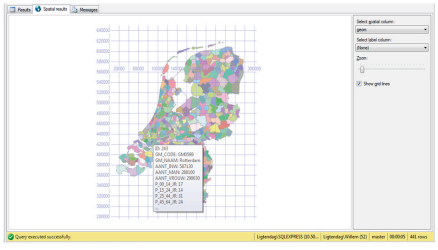
WCF RIA Services

De laatste technologie die we aanhalen is niet de minste. WCF RIA Services is een

framework speciaal geschreven voor Silverlight. Met RIA Services wil Microsoft een end-to-end verhaal aanbieden voor het bouwen van N-Tier applicaties met Silverlight. Het omvat een framework (een aantal assemblies zowel voor op de client- als op de server-side), tools (voornamelijk code-generatie) en services (bv. authentication). Met RIA services kan de Silverlight developer met hetzelfde gemak N-Tier, data-driven applicaties bouwen als hij dat met ASP.NET doet.
We zullen met een kort voorbeeld de belangrijkste punten aanhalen.
RIA Services is data-access neutral. Het werkt dus zowel met Entity Framework als NHibernate als eender welke andere data-access technologie. In dit voorbeeld gebruiken we hetzelfde Entity Model als voorheen. Boven het model wordt de RIA service dan gebouwd. Visual Studio heeft hiervoor een dialog window waarmee we kunnen aangeven welke entiteiten we willen opnemen. Er wordt optioneel al code in de service gegenereerd voor standaard CRUD operaties. De dialog is te zien op figuur 2. Deze server-side code kan vanzelfsprekend worden aangepast en uitgebreid.



Bij een Build-action wordt nu client-side code gegenereerd: er wordt als het ware een kopie van de server-side business code op de client gebouwd. Op figuur 3 zie je waar deze code te vinden is.



Door deze code-generatie kunnen we binnen onze Silverlight code eenvoudig validaties doen zonder dat hiervoor een round-trip naar de server nodig is (uiteraard niet voor alle types van validaties). De gegeneerde code bevat calls naar de server-side code en ook een kopie van entiteiten die op de server-side gebruikt worden.
Het grote voordeel is natuurlijk het hergebruik: we schrijven de business code eenmaal op de server-side en RIA services zorgt ervoor dat de client-side een up-to-date kopie krijgt. Verder bevat RIA services ook een aantal controls en tools die het bouwen van data-driven applicaties nog vereenvoudigt. Zo kunnen we nu binnen Silverlight applicaties gebruik maken van de Data Sources window. Deze geeft een overzicht van queries die beschikbaar zijn en met een eenvoudige drag naar het design window wordt meteen een DomainDataSource aangemaakt met bijhorende DataGrid. Het relevante deel van de code is te vinden in codevoorbeeld 12.

```
<riaControls:DomainDataSource Name=
"productDomainDataSource" QueryName=
"GetProductsQuery">
<riaControls:DomainDataSource.Domain-
Context>
          <my:NorthWindContext />

</riaControls:DomainDataSource.Domain-
Context>
</riaControls:DomainDataSource>
<sdk:DataGrid ItemsSource="{Binding Element
Name=productDomainDataSource, Path=Data}"
Name="productDataGrid">
<sdk:DataGrid.Columns>
                <sdk:DataGridTextColumn
                x:Name="categoryIDColumn"
                Binding="{Binding
                Path=CategoryID}" />
                ...
            </sdk:DataGrid.Columns>
        </sdk:DataGrid>
```

CODEVOORBEELD 12

Vanuit code kunnen we aansturen welke query moet worden uitgevoerd. Zo geven we in codevoorbeeld 13 aan dat data moet worden ingeladen. Hieruit blijkt nog steeds het asynchroon laden van de data.

```
private void LoadProductsButton_
Click(object sender, RoutedEventArgs e)
{
var context = new NorthWindContext();
productDataGrid.ItemsSource = context.
Products;

       context.Load(context.GetProducts
Query());
}
```

CODEVOORBEELD 13

WCF RIA Services biedt zoals gezegd nog veel meer en is zeker het overwegen waard bij het bouwen van enterprise-level applicaties met Silverlight. Merk op dat de Silverlight applicatie erg nauw verbonden is met de service zelf: hoewel de code verspreid is op 2 tiers, lijkt het toch alsof ze beiden deel uitmaken van 1 project. RIA Services zal daarnaast wel opengetrokken worden naar andere technologieën.

Conclusie

Wanneer we vandaag de vraag krijgen om een Silverlight LOB applicatie te bouwen, is de keuze voor de data-access technologie niet eenvoudig. Elke beschikbare technologie heeft zijn voor- en nadelen. Zo is WCF de beste keuze als we de services zelf willen bouwen, als we duplex communicatie moeten integreren of als vele andere platformen van dezelfde services gebruik moeten gaan maken. REST is dan weer een goede kandidaat wanneer simpele tekstuele informatie moet worden overgebracht. Het nadeel is het opbouwen van queries en het gebrek aan typed access. Dit wordt dan weer opgelost door WCF Data Services, wat als bijkomend voordeel heeft dat het met de client libraries vanuit zowat alle hoeken van .NET gebruikt kan worden. Tenslotte is WCF RIA Services de nieuwkomer. Met deze laatste krijgen we een end-to-end verhaal om met data te werken dat zich uitstrekt van controls en tools als code-generatie die toelaat business code eenmaal te schrijven en te hergebruiken op de client-side. Ongetwijfeld voer voor een lange architectuur discussie!

Gill Cleeren, is Microsoft Regional Director (www.theregion.com), MVP ASP.NET, INETA speaker bureau member en Silverlight Insider. Hij woont in België waar hij als .NET architect werkt bij Ordina Belgium. Als Microsoft Regional Director heeft Gill al heel wat sessies en trainingen gegeven over Silverlight, WPF en ASP.NET op conferenties en events waaronder TechDays en DevDays.