Roy Cornelissen

Data Centric Services met ADO.NET Data Services

De laatste tijd is de manier waarop we web applicaties bouwen sterk aan het veranderen. Traditioneel werden complete pagina's door server controls gerenderd en heen en weer gestuurd tussen server en client. Een dergelijke pagina bevatte structuur, opmaak en data. Tegenwoordig zien we steeds meer AJAX enabled websites, waarbij delen van de pagina worden ververst op basis van gebruikersacties. De browser wordt meer aan het werk gezet met Javascript en Dynamic HTML, of zelfs met gecompileerde user interfaces in Silverlight of Flash. De grootste driver hiervoor is de behoefte aan rijkere en meer geavanceerde gebruikersinteractie. Dit vergt een andere manier van werken en heeft geleid tot een nieuw type applicatie: de Rich Internet Application (RIA).

Een belangrijk element bij het ontwikkelen van RIA's is het uitwisselen van gegevens via het web. Met de komst van RIA's is de manier waarop deze data wordt geconsumeerd en gebruikt, veranderd. We hebben te maken met verschillende browsers, Javascript en web standaarden als Atom/RSS die een vlucht hebben genomen. Er is behoefte aan het kunnen scheiden van data, structuur en opmaak. Een geëigende methode om dat te realiseren is het bouwen van XML web services. Microsoft heeft de soorten services die zoal worden gebouwd, nog eens onder de loep genomen en heeft geconstateerd dat er eigenlijk behoefte is aan twee soorten interfaces: operation centric en data centric.

Er is behoefte aan 2 soorten interfaces: Operation Centric en Data Centric

Bij operation centric services draait het om operaties en de semantiek van operaties onderling. Voorbeelden hiervan zijn: het boeken van een vakantie, waarbij je een verzoek doet en verwacht dat er een aantal reserveringen wordt gemaakt, of een login mechanisme, waarbij je credentials opstuurt en verwacht dat ze worden gevalideerd en je een soort token terugkrijgt waarmee je verder kunt in de applicatie. Deze operation centric services zijn we al tamelijk gewoon te bouwen binnen onze service georiënteerde architecturen met standaarden als SOAP/XML en frameworks als WCF. Door Microsoft worden hiervoor de termen Activity Services of Process Services gebruikt.
In data centric services draait het om het adresseren, opvragen en manipuleren van gegevens. De operaties op die gegevens zijn veelal uniform (creëren, opvragen, muteren en verwijderen - CRUD) en datgene wat over de lijn wordt gestuurd, is alleen de data. Microsoft noemt dit type services ook wel Entity Services.

Vooral data centric services zijn gewild in RIA's omdat er behoefte is aan een standaard manier om gegevens helemaal tot in de presentatielaag (de browser) te krijgen, gescheiden van structuur en opmaak. De trend is daarom het gebruik van zogenaamde RESTful services. REST is een acroniem voor Representational State Transfer en staat voor een manier van denken waarbij gegevens (resources) in een bepaalde staat worden gecommuniceerd en kunnen worden aangewezen met unieke adressen: URI's. REST is geïntroduceerd in het proefschrift van Roy Fielding, een van de belangrijkste auteurs van het HTTP protocol. HTTP is zelf een goed voorbeeld van een implementatie van REST principes: resources (HTML documenten, plaatjes, Flash bestanden) worden aangeduid met URL´s. Ze worden gemanipuleerd met simpele opdrachten (HTTP verbs) als PUT (C), GET (R), POST (U) en DELETE (D).

Naast de behoefte aan dit type interfaces raakt ook het formaat waarin gegevens worden gecommuniceerd steeds meer gestandaardiseerd: Atom en RSS zijn veel gebruikte formaten in de wereld van blogs en mash-ups, JSON (JavaScript Object Notation) is bij uitstek geschikt voor AJAX clients. Praktijkvoorbeelden van RESTful, data centric services zijn Google Base en de Windows Live services zoals Live Photos.

ADO.NET Data Services

Het zelf bouwen van services met data centric interfaces en op basis van eerdergenoemde standaarden is goed mogelijk, maar is veel werk. Als je voor alle entiteiten in je gegevensdomein lees- en schrijfoperaties moet bouwen in een traditionele service, dan kan dat veel methods vergen en dus veel werk opleveren. Als de interface van data centric services uniform is en de formaten voor communicatie ook gedefinieerd zijn, dan moet het mogelijk zijn een framework te ontwikkelen dat het bouwen van zulke services gemakkelijk maakt. Daarnaast zou er een uniform programmeermodel moeten worden bedacht voor clients die deze services consumeren. Microsoft heeft deze handschoen opgepakt, wat heeft geresulteerd in ADO.NET Data Services. Dit project was eerder bekend onder de codenaam "Astoria". ADO.NET Data Services is een framework om gegevens te ontsluiten op een uniforme manier. Hierbij heeft Microsoft sterk gekeken naar de geldende standaarden op het web om niet zelf opnieuw het wiel uit te hoeven vinden. ADO.NET Data Services leunt daarom sterk op HTTP voor adressering en transport en Atom/Atom Publishing Protocol (APP) of JSON als gegevensformaat. Door hiervoor te kiezen krijg je

zaken als HTTP caching, proxies, security mechanismes, etc. cadeau. Daarnaast zijn de uitwisselingsformaten breed toepasbaar, wat de inzetbaarheid op verschillende platforms vergroot. Verderop in dit artikel zullen we zien dat ADO.NET Data Services leunt op nog meer "standaard" zaken die reeds voorhanden zijn. ADO.NET Data Services is namelijk gebouwd bovenop WCF, ASP.NET en LINQ technologie.

Astoria is een framework om gegevens te ontsluiten op een uniforme manier

Van SP1 voor .NET framework 3.5 is begin mei een beta-versie uitgebracht. In deze uitbreiding zitten zaken als ADO Entity Framework, ASP.NET extensions en ook ADO.NET Data Services. Tegelijkertijd met deze framework-update is ook een beta-versie van SP1 voor Visual Studio 2008 beschikbaar gekomen, waarin de tools en designers voor deze nieuwe zaken zitten.

Een interessant aspect aan de ontwikkeling van deze technologie is dat het Astoria team al vanaf het eerste prototype contact heeft gezocht met de .NET community. Op deze manier ontvingen ze al in een vroeg stadium feedback over de haalbaarheid en het nut van het project. Door een zeer open ontwerpproces - iedere belangrijke beslissing en overweging werd gedeeld op het team blog - kon de community invloed uitoefenen op het eindresultaat.

Aan de slag

Een ADO.NET Data Service bestaat uit twee delen: een vast deel voor het afhandelen van requests en serialisatie en een pluggable deel, de data source. In de meeste gevallen zal data zich in een database bevinden, en in een Microsoft only wereld zal dat SQL Server zijn. De wereld is echter niet altijd zo eenduidig, dus het kan voorkomen dat gegevens in andere bronnen staan dan relationele databases, bijv. flat files, XML of in memory. Voor het omgaan met gegevens, onafhankelijk van de bron, is in .NET 3.5 al een erg krachtige feature beschikbaar gekomen onder de naam LINQ (Language Integrated Query). Doordat een provider model wordt gebruikt, kunnen uiteenlopende bronnen worden benaderd via LINQ expressies. ADO.NET Data Services maakt ook gebruik van dit principe, door te leunen op LINQ en in het bijzonder de IQueryable interface voor het beschikbaar stellen van entiteiten uit de gegevensbron. Omdat IQueryable gericht is op het lezen van gegevens, is daaraan IUpdatable toegevoegd voor het manipuleren van de gegevens. Zolang een provider deze twee interfaces ondersteunt, maakt het niet uit waar je data zich bevindt, om er een ADO.NET Data Service van te maken.

Een ADO.NET Data Service bestaat uit twee delen: een voor het afhandelen van requests en serialisatie en een voor de data source

Voor gegevens in een relationele database is het ADO.NET Entity Framework bij uitstek geschikt. Met dit framework kun je snel en gemakkelijk de entiteiten in je gegevensmodel definiëren als .NET classes en daarbij een mapping aangeven op het onderliggende datamodel. Entity framework biedt op zijn beurt weer ondersteuning voor LINQ to SQL, waarbij LINQ queries binnen de .NET omgeving worden vertaald naar SQL queries op de database. Ook stored procedures worden ondersteund, voor het geval je geen prijs stelt op dynamische SQL queries vanuit je code. Wanneer je een Entity Framework model aanmaakt voor je database, kun je deze rechtstreeks gebruiken als bron voor een ADO.NET Data Service.

Na het installeren van SP1 is een nieuw item type beschikbaar gekomen in Visual Studio: de ADO.NET Data Service. Wanneer je een nieuw item van dit type creëert, krijg je een .SVC bestand met codebehind-file. Als je deze codebehind bekijkt, dan zul je zien dat de implementatieclass is afgeleid van de generic base class Data Service<T>. Het type T geeft aan wat het datatype is van je gegevensbron. Dit is een class met properties van het type IQueryable en IUpdateable, bijvoorbeeld een Entity Framework model.

Standaard zal de gegenereerde service nog geen data ontsluiten. Dit kun je regelen met policies, waarop we verderop in dit artikel terugkomen. Voor demonstratiedoeleinden roepen we config.SetEntitySetAccessRule("*", EntitySetRights.All) aan, om aan te geven dat alle entiteiten in ons model via de service opvraagbaar en muteerbaar zijn. Dit wordt sterk afgeraden voor productiecode. In listing 1 is een voorbeeld te zien van een Data Service voor de AdventureWorks database.

```
using System;
using System.Collections.Generic;
using System.Data.Services;
using System.Linq;
using System.ServiceModel.Web;
using System.Web;
using AdventureWorksModel;
using System.Linq.Expressions;

namespace AWWeb
{
  // Data service gebaseerd op AWEntities bron
  public class AWData : DataService<AWEntities>
  {
    public static void InitializeService
     (IDataServiceConfiguration config)
    {
      // Toegang tot alle entity sets toegestaan
      config.SetEntitySetAccessRule
        ("*", EntitySetRights.All);
    }
  }
}
```

Listing 1: Data service voorbeeld

Queries stellen

Als we de service aan ons model hebben gekoppeld en de access rules hebben gezet, kunnen we de service meteen testen door queries uit te voeren. Omdat alle data via URI's beschikbaar is en HTTP het protocol is, kunnen we de browser als client gebruiken. Wanneer we de service starten en met Internet Explorer er naartoe surfen, dan zien we allereerst een overzicht van alle toplevel-entiteiten die de service beschikbaar stelt, in XML formaat. Door de data browsen is vervolgens heel simpel: dit doe je door de naam van zo'n entiteit toe te voegen aan de URI in de adresbalk. Het basisformaat van een URI voor data services is:

```
http://host/vdir/<service>/<EntitySet>
 [(<Key>)
   [/<NavigationProperty>
     [(<Key>)/...]
   ]
 ]
```

In de AdventureWorks data service van listing 1 geeft http://server/AdventureWorksData.svc/Product ons alle entries in de Product tabel, http://server/AdventureWorksData.svc/Product(3) geeft ons het product met primary key 3. De naam "Product" in de URI is onze EntitySet en "3" is de Key.

Verder kun je de gehele hiërarchie van de database doorwandelen: http://server/AdventureWorksData.svc/Product(3)/WorkOrder geeft bijvoorbeeld alle work orders van product 3. "WorkOrder" is in dit voorbeeld een NavigationProperty. Navigation properties kun je zo diep nesten als de structuur van je model toelaat. Wanneer je een lijst van items opvraagt, zal je opvallen dat Internet Explorer de lijst presenteert als RSS feed, met de welbekende style sheet en de link "subscribe to this feed". Dat komt doordat het standaard formaat van een data service Atom/APP is.
Een enkel record in de tabel wordt door de service dan ook teruggegeven als een <entry> element zoals je die kent in Atom. Het formaat dat de data service teruggeeft kun je beïnvloeden door de Accept HTTP header in het request. Door "application/json" mee te geven krijg je de data in JSON formaat. Door gebruik te maken van het AJAX framework zal alle data die je vanuit Javascript opvraagt, in JSON formaat door de server worden geretourneerd.

Een lijst van items wordt gepresenteerd als RSS feed

Standaard zal de service een platte collectie van entiteiten teruggeven, gesorteerd op de primary key in de tabel. Microsoft heeft echter goed gekeken naar de bewerkingen die in veel applicaties nog op data sets worden gedaan voordat ze worden gepresenteerd. Denk hierbij aan sorteren, paging en het tonen van bepaalde detail-informatie in de objectgraaf. Hiervoor is een aantal argumenten beschikbaar, die je in de querystring van de URI kunt opgeven. Dit zijn:

- **$expand:** gebruik je om gerelateerde objecten ook in de resultaten terug te krijgen, die je anders in twee losse requests zou moeten ophalen, b.v. als je de ProductVendor van de producten ook meteen wilt hebben. De ProductVendor wordt dan een property van de Product objecten in de collectie.
- **$orderby:** beïnvloedt de sortering van de resultaten. Je kunt sorteren op properties van de objecten in de lijst. Meerdere velden worden gescheiden door komma's. Je kunt de sortering beïnvloeden met de asc (default) en desc opties.
- **$skip:** laat de service het opgegeven aantal rijen overslaan in de resultaatlijst. Dit is vooral handig voor paging, in combinatie met de $top optie.
- **$top:** geeft aan hoeveel rijen er moeten worden teruggegeven. Deze parameter werkt sterk samen met $orderby. Wanneer geen $orderby wordt opgegeven, wordt de primary key van de tabel gebruikt voor de sortering en wordt vervolgens $top toegepast.
- **$filter:** fungeert als de where clause in je query: hiermee geef je expressies op om de resultaatlijst te filteren op eigenschappen. De $filter property kent weer een eigen syntax, met operators en functies voor string-, datum-, mathematische en typebewerkingen, vergelijkbaar met wat je in de common language runtime ook vindt. Het gaat te ver voor dit artikel om de hele lijst te noemen, maar ze zijn te vinden in de documentatie.

De mogelijkheden die je in de querystring hebt, staan los van het formaat waarin de data wordt geretourneerd. Deze opties zijn dus altijd voorhanden. Het ADO.NET Data Services framework verzorgt de afhandeling van deze argumenten, dus zijn ze toepasbaar op iedere data service die je maakt, ongeacht de gegevensbron. Het framework vertaalt de URI naar een Linq expression tree en laat deze vervolgens los op de data. Dit brengt veel flexibiliteit met zich mee, omdat je de client kunt laten bepalen hoeveel gegevens moeten worden geretourneerd, terwijl je hier geen extra logica voor hoeft te bouwen in de service om dit allemaal te ondersteunen. Daarnaast wordt het netwerkverkeer beperkt, doordat de service deze faciliteiten standaard biedt.

Voorbeelden:

- Alle producten, met daarbij de ProductVendor relatie en de bijhorende Vendor entiteit ook meegenomen: http://server/AWData.svc/Product?$expand=ProductVendor/Vendor
- Uit alle producten, aflopend gesorteerd op naam, rij 30 t/m 40 (de vierde page): http://server/AWData.svc/Product?$skip=29&$top=10&$orderby=Name desc.
  Volgens de notatieregels voor URI's moet de spatie in bovenstaand voorbeeld worden vertaald naar "%20".
- Alle producten waarvan de naam (in lower case) de string 'nuts' bevat: http://server/AWData.svc/Product?$filter=contains(tolower(Name) ,'nuts')

Client programmeermodel

Naast een framework en tools om snel een data service te bouwen biedt SP1 voor Visual Studio 2008 ook een aantal libraries om clientcode voor data services mee te bouwen. ADO.NET Data Services is in eerste instantie ontwikkeld met het web in het achterhoofd, dus de ondersteuning voor clients is ook voornamelijk daarop gericht. Daarnaast integreren deze bouwblokken weer naadloos met bestaande technologie: het .NET framework en ASP.NET. Er zijn vier mogelijkheden:

- Een Javascript library die in combinatie met de ASP.NET AJAX Extensions een Javascript programmeermodel biedt voor het consumeren van data services vanuit een browser;
- De DataServiceDataSource control, die als data source kan dienen voor data aware ASP.NET controls, zoals de GridView. Hiermee kun je dus standaard ASP.NET controls koppelen aan iedere willekeurige ASP.NET Data Service;
- De DataServiceContext en DataServiceQuery objecten in de System.Data.Services.Client library, waarmee je vanuit iedere .NET applicatie een data service kunt consumeren, en dus ook vanuit Silverlight user interfaces;
- Een LINQ to Data Services provider, waarmee je in de client LINQ queries kunt bouwen om data uit data services op te vragen.

Voor de laatste twee opties is ook een tool beschikbaar waarmee je op basis van de service catalog classes kunt genereren die het eniteitenmodel in de client implementeren. Deze tool is vergelijkbaar met SvcUtil.exe voor WCF services en heet dan ook toepasselijk DataSvcUtil.exe. De gegenereerde code is LINQ enabled. In figuur 1 is te zien hoe een dergelijke LINQ query runtime wordt vertaald naar een ADO.NET Data Service URI.

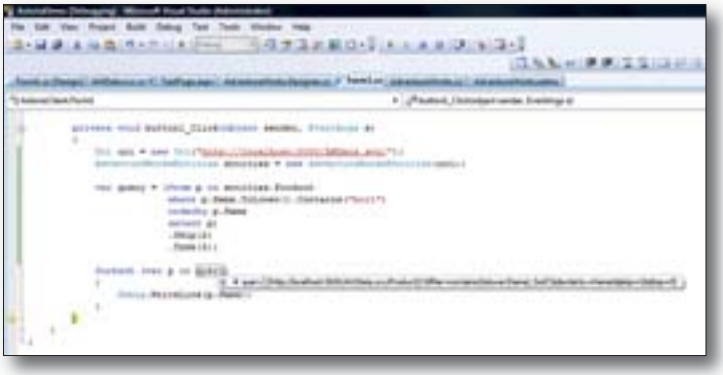

Fig. 1: Vertaling LINQ query naar ADO.NET Data Services URI

Beveiliging en extra controle

Tot zover hebben we het gehad over het beschikbaar stellen van data via een uniforme interface. Met een paar clicks en een regel code kun je een hele database aan het web hangen en voor iedereen beschikbaar stellen. De vraag rijst dan hoe we dit goed kunnen beveiligen of de gegevensset kunnen beïnvloeden op basis van een gebruikerscontext. Voor het beveiligen van services maakt ADO.NET Data Services gebruik van bestaande authenticatie-mechanismen: IIS biedt

ingebouwde Windows of Basic authentication, ASP.NET biedt Forms authentication en het Membership provider model en WCF bieden behaviors om services te beveiligen. HTTPS kan worden ingezet om verkeer te versleutelen.
In de data service zelf kun je met policies vrij precies bepalen welke delen van je entiteitenmodel beschikbaar zijn via de data service en welke operaties erop geoorloofd zijn. Dit doe je in de IntializeService method, die wordt gegenereerd, zodra je een ADO.NET Data Service aan je project toevoegt. Aangeven welke onderdelen op welke manier toegankelijk zijn doe je met de SetEntitySetAccessRule method van het config object dat je als parameter meekrijgt. Per onderdeel kan de toegestane wijze van toegang worden aangegeven: alles, read single, read multiple, write single, enz. Zo kun je delen van je model toegankelijk maken voor lezen en schrijven, terwijl andere delen slechts alleen lezen zijn, of kun je ervoor zorgen dat niet een complete set, maar alleen enkele items kunnen worden opgevraagd. De tweede parameter van SetEntitySetAccessRule is een enum van het type EntitySetRights, waarmee je de rechten kunt specificeren. Zie listing 2 voor een voorbeeld.

```
// Alles toegestaan op Product
config.SetEntitySetAccessRule("Product",
  EntitySetRights.All);

// ProductReviews mogen alleen worden toegevoegd
config.SetEntitySetAccessRule("ProductReview",
  EntitySetRights.WriteAppend);

// Uit WorkOrderDetail mogen alle records worden gelezen
config.SetEntitySetAccessRule("WorkOrderDetail",
  EntitySetRights.AllRead);
```

Listing 2: Entity set policies

Een derde mogelijkheid om controle uit te oefenen op de gegevens die worden gevraagd of bewerkt is de QueryInterceptor. Dit is een public method die wordt gedecoreerd met het QueryInterceptor attribuut en een return type Expression<Func<[entity type],bool>> heeft. In de constructor van het QueryInterceptor attribuut geef je de naam van een entity set op. Wanneer die entity set wordt geraakt, voor lezen én schrijven, dan zal het ADO.NET Data Services framework deze method altijd uitvoeren. In zo'n query interceptor kun je business logica toevoegen en een expressie opstellen die door de data service wordt toegevoegd aan de LINQ expression tree die wordt losgelaten op de data source. In de query interceptor method kun je van alles doen, van controle op gebruikersrollen tot het aanroepen van services.

Query Interceptor maakt controle mogelijk

Query interceptors kun je ook stapelen, zodat je meerdere interceptors achter elkaar kunt laten uitvoeren op een entity set. Listing 3 bevat een voorbeeld van een query interceptor voor de Product entity set, waarbij voor gebruikers die niet in de Administrators rol zitten, de producten met 'chain' in de naam niet worden teruggegeven. Deze interceptors worden ook uitgevoerd bij bewerkingen. Als je een product probeert te wijzigen dat door het toegevoegde filter verdwijnt uit de set, dan resulteert dit in een foutmelding met statuscode 404 - de HTTP status code voor Not Found. Het maakt ook niet uit via welke navigation properties je de betreffende entity set benadert: de interceptor wordt er altijd op losgelaten.

```
[QueryInterceptor("Product")]
public Expression<Func<Product, bool>> OnQueryProduct()
{
  if (HttpContext.Current.User.IsInRole("Administrator"))
  {
    // administrators mogen alles zien
    return (product) => true;
  }
  else
  {
    // chain producten zijn niet zichtbaar voor anderen
    return (product) =>
      !product.Name.ToLower().Contains("chain");
  }
}
```

Listing 3: Voorbeeld QueryInterceptor

Als je ook niet uit de voeten kunt met query interceptors, dan is er ten slotte nog de mogelijkheid om service-operations te schrijven. Deze kun je vergelijken met methods op een web service, waarbij de method ook toegankelijk is via de URI. ADO.NET Data Services doet de mapping van URI naar method en querystring parameters naar method parameters. Om te bepalen hoe de URI eruit ziet en op welke manier de parameters moeten worden vertaald, kun je de UriTemplate template class gebruiken.

In listing 4 zie je een voorbeeld van een method die alle producten van een bepaalde vendor geeft op basis van het account nummer van de vendor. Leesoperaties moet je markeren met een WebGet attribuut, methodes die ook wijzigingen uitvoeren markeer je met een WebInvoke attribuut. In deze service operations kun je natuurlijk alle business logica kwijt die je wilt, dus ook weer services aanroepen, security checks uitvoeren, etc. Omdat ADO.NET Data Services zorgt voor de afhandeling van je methods, kun je ook in deze gevallen de querystring operaties van het framework benutten, zoals $orderby, $filter of $top. Dit werkt zolang de set die je retourneert, dit ook ondersteunt. Een belangrijke restrictie van service operations is nog wel dat er alleen primitieve types als parameters kunnen worden gebruikt. Ze moeten namelijk op de URI kunnen worden meegegeven.

```
[WebGet(
UriTemplate=
 "/ProductsByAccountNumber?accountNumber={accountNumber}")]
public IQueryable<Product> ProductsByAccountNumber
  (string accountNumber)
{
  return from productIterator
  in this.CurrentDataSource.Product
  from vendorIterator
  in productIterator.ProductVendor
   where vendorIterator.Vendor.AccountNumber == account-
Number
  select productIterator;
}
```

Listing 4: Voorbeeld service operation

Concurrency, transacties en batches

Bij het zien van deze werkwijze rijst meteen ook de vraag hoe concurrency is geregeld. Als je data zo direct benaderbaar is, dan is het wenselijk om hier voorzieningen voor te hebben.
Als je niets regelt, dan geldt het "last update wins" principe: de laatste update die de service ontvangt, bepaalt de staat van een entiteit. Maar ADO.NET Data Services biedt ook ondersteuning voor optimistic concurrency control. Dit doen ze door weer heel slim te lenen van de HTTP specificatie: middels E-Tags. E-Tags worden in HTTP gebruikt voor precies datgene wat hier wordt beoogd: het bewaken van concurrency door een versie-indicatie toe te voegen aan de data.

Hiermee kunnen conditionele HTTP requests worden uitgevoerd, waarbij de E-Tag bepaalt of de bewerking plaatsvindt op de juiste versie van de data.
In het klassieke voorbeeld van Order/OrderRegel zul je nooit alleen een OrderRegel willen toevoegen aan je service, maar liever een complete Order met meerdere OrderRegels. ADO.NET Data Services ondersteunt hiervoor "deep inserts" en "deep updates". Dit houdt in dat je complete objectgrafen kunt opsturen naar de server, die in de juiste volgorde worden doorgevoerd in de gegevensbron. De juiste volgorde wordt afgeleid uit de semantiek van het onderliggende entiteitenmodel. Inherent daaraan is dat deze bewerkingen transactioneel moeten worden uitgevoerd. Voor data services die op ADO Entity Framework zijn gebaseerd wordt dit automatisch geregeld. Voor andere gegevensbronnen kan dit wat lastiger liggen en zul je daar zelf in moeten voorzien. Met name bij gegevensbronnen die gedistribueerd zijn kan dit uitdagingen opleveren. De TransactionScope class uit de System.Transactions namespace kan je hierbij helpen.
Naast deep inserts en updates kunnen ook batchverwerkingen worden opgestuurd. In plaats van een enkele order kunnen ook meerdere orders in één request naar een data service worden opgestuurd. Ook hier geldt dat transacties worden afgehandeld voor zover de onderliggende gegevensbron dit ondersteunt.

Bij het zien van deze werkwijze rijst meteen ook de vraag hoe concurrency is geregeld

En SOA dan?

We hebben gezien dat ADO.NET Data Services een krachtig en flexibel framework is om gegevens beschikbaar te stellen met een gestandaardiseerd protocol in een standaard gegevensformaat. ADO.NET Data Services richt zich met deze standaarden primair op het web, maar door de krachtige client library en de Linq-to-Data-Services-provider zijn ook andere typen applicaties prima te koppelen.
Tot op heden waren we in onze service oriented architecuren (SOA's) gewend om operation centric services te bouwen met een duidelijke semantiek en communicatie op basis van berichten. Operaties op business services representeren business acties en soms lopen zelfs complete processen over meerdere services heen. Nu komt daar zo'n vreemde eend in de bijt bij met een compleet andere manier van werken en een interface die de welbekende CRUD van weleer weer terugbrengt.
Zijn deze twee te verenigen? Ik denk dat door het beschikbaar komen van deze technologie, door de steeds grotere adoptie van REST-style-services en door de vlucht van RIA's er gaandeweg een mix zal ontstaan van deze twee stijlen. Hierdoor zullen traditionele SOA services op basis van SOAP steeds meer worden vermengd of aangevuld met hun RESTful tegenhangers. De keuze hierbij zou moeten gaan over het meest geschikte type interface: gaat het om data of gaat het om operaties, acties? Door de gelaagde opbouw van ADO.NET Data Services is het zelfs mogelijk om je entiteitenmodel te baseren op je bestaande set van services. Zolang je in je data service een objectlaag voor je entiteiten hebt die Linq enabled is, kun je onder water die services benaderen om de benodigde operaties uit te voeren. Als het je gaat om een getailorde interface voor RIA bovenop bestaande services, dan is een dergelijke combinatie een optie.
In een hybride oplossing zou het ophalen van gegevens via een ADO.NET Data Service kunnen verlopen, vanwege de krachtige opties die je cadeau krijgt voor filtering, sortering, paging en navigatie door data, terwijl bewerkingen veel eerder door operation centric services zouden kunnen worden afgehandeld.
Mijn verwachting is dat de tooling rondom ADO.NET Data Services na een eerste praktijkronde nog wat verder zal uitkristalliseren. Op dit moment staat de code generatie tool voor client side data contracten (DataSvcUtil.exe) bijvoorbeeld nog naast die voor WCF/SOAP interfaces (SvcUtil.exe). Er zal een unificatie moeten komen van deze twee modellen zodat datacontracten eventueel uitwisselbaar zijn tussen deze twee typen vanuit het perspectief van de client.

Conclusie

ADO.NET Data Services is een krachtige combinatie van framework en patterns waarbinnen bestaande standaarden op een slimme manier worden gecombineerd om data centric services te bouwen. REST, HTTP en Atom en zelfs LINQ zijn allemaal niet opnieuw bedacht, het heeft alleen ontbroken aan een gereedschapskist om goed en snel een dergelijk type service op te zetten. RIA's vormen het belangrijkste doelplatform voor deze technologie, maar het client framework is krachtig genoeg om ook andere typen applicaties te bedienen. Het maakt deel uit van SP1 voor .NET 3.5 (codenaam "Arrowhead"), die gepland staat voor november 2008. Tegelijk daarmee zal ook SP1 voor Visual Studio 2008 beschikbaar komen voor de bijhorende designers en IDE-integratie.
Het Astoria team gaat ondertussen verder met de ontwikkeling van ADO.NET Data Services. Een erg interessante ontwikkeling is het principe van Offline ADO.NET Data Services. Het idee hierachter is dat de gegevens in een data service "offline" kunnen worden genomen en zo geschikt worden voor clients in een "occasionaly connected" scenario. Wijzigingen worden lokaal opgeslagen en komen op de service terecht zodra een netwerkverbinding weer beschikbaar is. Ook hier vindt Microsoft het wiel niet opnieuw uit, maar maakt slim gebruik van het SyncFramework, dat ook onderdeel is van .NET 3.5 SP1. Dit idee bevindt zich nog in een prototype-fase, maar wordt inmiddels ook goed ontvangen door de community.
De beta versie van deze spullen is nu beschikbaar. Het is de moeite waard om er alvast mee te spelen, al was het maar om meteen kennis te maken met de slimme integratie met o.a. het ADO Entity Framework. Het bedrijf Veracity in Australie heeft het zelfs al aangedurfd om een productiesysteem te ontwikkelen met ADO.NET Data Services: FreeNatal. Dit is een open community applicatie waarin dossiers van zwangere vrouwen kunnen worden beheerd en uitgewisseld tussen verloskundigen en gynaecologen. De verwachting is dat de applicatie binnen enkele weken online zal gaan. Vermoedelijk zijn ze momenteel bezig om de applicatie te migreren van CTP bits naar SP1 Beta 1.
Nog een paar maandjes wachten tot deze nieuwe telg in de .NET familie ook het levenslicht ziet. Gelukkig hebben we alvast de echo's om naar te kijken.

Referenties

- ADO.NET Data Services home: http://astoria.mslivelabs.com
- Project Astoria Team Blog: http://blogs.msdn.com/astoriateam/
- FreeNatal: http://www.freenatal.org
- REST: http://en.wikipedia.org/wiki/Representational State Transfer

Roy Cornelissen

Roy Cornelissen werkt als IT architect bij Info Support voor de business unit Industrie. Hij heeft negen jaar ervaring in de ICT en heeft gewerkt aan uiteenlopende projecten. In zijn dagelijks werk past hij Microsoft technologie toe in service georiënteerde omgevingen. Hij is te bereiken via royc@infosupport.com.