Een 'sales configurator' bouwen met Silverlight

Combinatie van Silverlight en RIA services werkt goed

René Titulaer, Danny op het Veld, Marcel Poppe en Wilbert Janssen

Bij Océ is een project gestart voor de ontwikkeling van een Sales Configurator. De Sales Configurator zal worden gebruikt door vertegenwoordigers om producten te verkopen. Het op maat samenstellen van een product is niet eenvoudig vanwege verschillende varianten met verschillende opties. Corporate IT besluit om de configurator in Silverlight te bouwen en doet hier zijn verhaal.

In het huidige proces kan het voorkomen dat een vertegenwoordiger een product wil verkopen die technisch niet kan worden geleverd omdat de verschillende onderdelen niet met elkaar samenwerken. De vertegenwoordiger moet dan terug naar de klant om de aanbieding te herzien. De nieuwe Sales Configurator moet dit proces verbeteren. De Sales Configurator moet ook verschillende contract typen ondersteunen zoals lease, verkoop en huur en de bijbehorende prijzen bepalen. Vanwege deze eisen en wensen heeft de Sales Configurator ook een sterke koppeling met de back office.

Toen IT werd gevraagd om de Sales Configurator te implementeren was de eerste vraag natuurlijk op basis van welke technologie we dit zouden doen. Het was duidelijk dat we dit met .Net zouden doen omdat dit de standaard is binnen Océ voor maatwerk applicaties. We moesten ook kiezen of het een web applicatie zou worden, een Windows fat cliënt, een Windows thin cliënt of een Silverlight applicatie. Al vrij snel ging de voorkeur uit naar Silverlight. Silverlight geeft je de voordelen van een Windows applicatie en van een web applicatie. De voordelen van een Windows applicatie omdat Silverlight een rijke gebruikerservaring geeft vanwege een grote hoeveelheid aan geavanceerde controls en omdat je gewoon in .Net kunt ontwikkelen (en geen gebruik hoeft te maken van HTML en javascript). De voordelen van een webapplicatie omdat je een eenvoudig deployment model hebt. De Silverlight applicatie kan gewoon op de web server gedeployed worden.

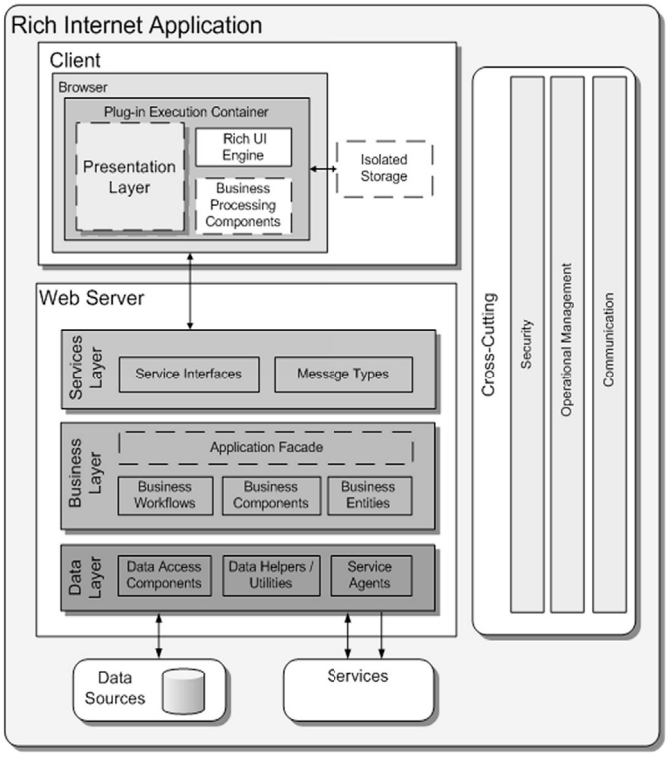
De voordelen voor Silverlight waren dus duidelijk, er was echter twijfel of het geen risico was om in deze nieuwe technologie te ontwikkelen. Toen we de keuze moesten maken was Silverlight 3 de meest recente beschikbare versie. Het feit dat er al een derde release was klinkt wel als een volwassen product maar de versies volgde elkaar wel erg snel op wat toch weer een minder goed gevoel gaf. Zo was er ook al weer een Silverlight 4 beta beschikbaar toen we de keuze moesten maken (in maart is deze officieel gereleased). Een andere vraag die we hadden was of er in de markt wel voldoende kennis was van deze technologie. Wij bij Océ hadden deze kennis niet dus we hadden behoefte aan externe kennis. Ook dit viel niet mee. Veel ontwikkelaars experimenteren wel met Silverlight maar echte ervaring is er niet. De mensen die ervaring hadden met Silverlight hadden dit dan weer vaak op grafisch gebied en niet zo zeer ervaring met het ontwikkelen van business applicaties. Bij het maken van de keuze werden we geholpen doordat enthousiaste ontwikkelaars binnen het team inmiddels waren begonnen met het implementeren van een aantal schermen van het beschikbare storyboard. Het was verbazingwekkend hoe snel zij waren om functionaliteit te implementeren. Veel sneller dan met ASP.Net het geval zou zijn geweest. Dit heeft ons uiteindelijk doen besluiten om toch voor Silverlight te kiezen.

Met een team van 4 ontwikkelaars zijn we begonnen met het opstellen van de architectuur en het maken van prototypes Belangrijke vragen die beantwoord moesten worden waren: voor welke architectuur kiezen we? Gaan we RIA Services gebruiken? Gaan we gelijk over op Silverlight 4? Gaan we gelijk over op Visual Studio 2010? Welke data access framework gaan we gebruiken? Gaan we Blend gebruiken? Gaan we MVVM gebruiken?
Na het beschrijven van de architectuur en de patterns die we zouden gaan gebruiken zijn we begonnen met de implementatie. Het uiteindelijk technisch team bestond uit 8 personen. In de rest van het artikel zal ik de architectuur uitdagingen verder beschrijven.

Architectuur

Voor de architectuur hebben we als uitgangspunt de Microsoft RIA (Rich Internet Applications) Architecture guide genomen (http://apparch.codeplex.com/releases/view/19799). Deze guide is gemaakt door het Microsoft Patterns and Practices team. Dit team heeft voor vrijwel alle mogelijke architecturen een dergelijke guide gemaakt, zo bestaan er naast de RIA Architecture guideline ook guidelines voor bijvoorbeeld Web of Mobile. Deze guides be-

schrijven in detail hoe je technische uitdagingen te lijf kunt gaan en welke Microsoft technologie je het beste kunt gebruiken.



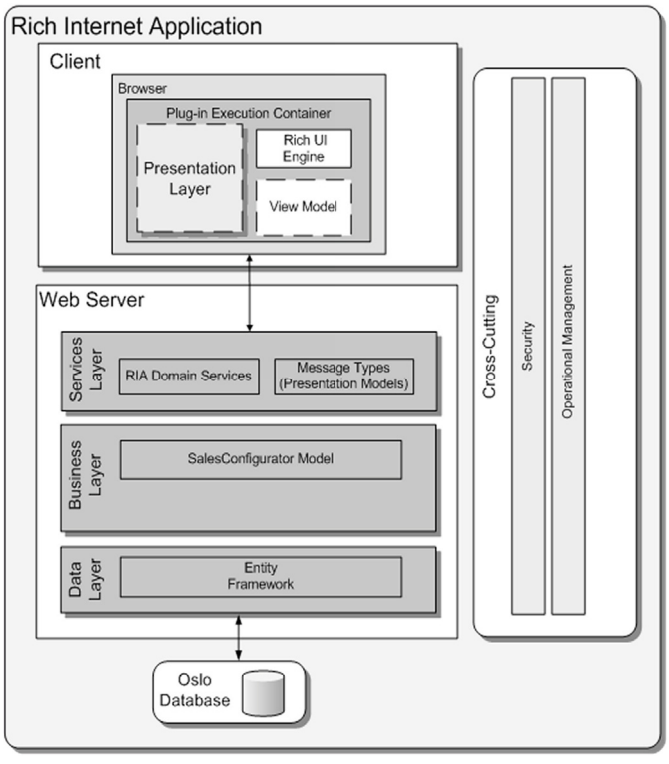
FIGUUR 1

Figuur 1 toont de details van deze architectuur. In deze guide staat beschreven dat Rich Internet Applications met de server communiceren door middel van web services, bijvoorbeeld WCF web services. Wat in deze guide nog niet is opgenomen is het gebruik van RIA services (voor een beschrijving van RIA Services zie ook het artikel van Sander Schutten en Wouter Overmeer in .Net Magazine 2 van dit jaar). Dat is dan ook meteen de eerste keuze die je dient te maken. Maak je gebruik van RIA Services of niet.
Het gebruik van RIA verhoogt de productiviteit in hoge mate maar zorgt van de andere kant voor een sterke koppeling tussen de verschillende lagen waardoor het toepassen van de RIA Architectuur Guideline bemoeilijkt wordt. Omdat RIA Services het ontwikkelen zoveel vergemakkelijkt hebben we uiteindelijk toch gekozen voor RIA Services. De uiteindelijke architectuur staat beschreven in figuur 2 (als vereenvoudiging is de integratie met de back office systemen weggelaten). De architectuur beschrijft dat we gebruik maken van een Silverlight cliënt die gebruik maakt van ViewModel (zie ook MVVM verder op in dit artikel). Als service laag gebruiken we RIA Services en we hebben een aparte business laag (wat een aparte assembly is) die het Entity model bevat. Het Entity model is onderdeel van het Entity Framework, het bevat de specifieke entiteiten van een applicatie. Door je Entity model te beschrijven weet het Entity Framework hoe entiteiten opgehaald moeten worden uit de database en hoe entiteiten gepersisteerd dienen te worden zonder SQL te schrijven.

RIA Services

RIA Services helpt je om snel een service georiënteerde architectuur op te zetten. Binnen RIA Services heb je nog verschillende smaken. Standaard krijg je ondersteuning voor Entity Framework en Linq to SQL. Je kunt er ook voor kiezen om je eigen implementatie te maken om data op te halen en te persisteren. Aangezien wij gebruik maken van Entity Framework hebben we gekozen voor RIA Services voor Entity Framework. Door deze keuze krijg je heel veel cadeau. Je hele entiteiten model wordt ontsloten door middel van web services, je krijg een Read, Insert, Update en Delete services per entiteit uit je Entity model.
Onder architecturen wordt de discussie gevoerd of je RIA services wel moet gebruiken voor Enterprise applicaties. Volgens de ene groep zijn RIA services vooral bedoelt voor het maken van prototypes en kleine applicaties. Volgens andere is RIA services het platform om Rich Internet Applicaties te maken. Wij hebben uiteindelijk voor RIA services gekozen vanwege de hoge productiviteit. Een groot voordeel van RIA services is namelijk dat die ook wijzigingsbeheer doet. Wanneer je verschillende services aanroept die verschillende entiteiten terug geven en je gaat die wijzigen (change\insert\delete) dan wordt die status door de RIA Services cliënt bijgehouden. Wanneer je dan op een gegeven moment een SubmitChanges aanroept dan stuurt de cliënt alleen de wijzigingen door naar de server. Wanneer je met traditionele web services zou werken dan zou je dat beheer aan de cliënt zelf moeten doen.

Het gebruik van RIA Services heeft wel meerdere keren voor opontholud gezorgd. In eerste instantie ben je met RIA services snel op weg. Je kunt door een paar keer klikken je Entity model ontsluiten met web services. Echter daarna kom je voor een aantal uitdagingen te staan. Eén van die uitdagingen is hoe je je domain services moet groeperen. Een domain service is een verzameling van entiteiten. Je kunt er voor kiezen om één grote domain service voor je hele entiteiten model te maken. Dit heeft als nadeel dat je de controle bij validaties verliest. Je domain service bevat dan namelijk een Get, Insert, Update en Delete voor al je entiteiten. Stel dat je een applicatie hebt voor online winkelen met een module hebt voor het plaatsen van orders (door klanten) en een module voor het beheer van prijzen (voor beheerders) en beide zouden gebruik maken van dezelfde domain service, dan zou een klant ook een prijs update naar de service kunnen versturen. Uiteraard



FIGUUR 2

kun je dit tegen gaan door security op services te zetten maar dan heb je nog steeds het probleem met validaties. Wanneer je op een server een Submit binnen krijgt dan kan hier in potentie elke entiteit in zitten (order updates, prijzen updates, etc.). Wanneer je dan een validatie wilt doen voor bijvoorbeeld het plaatsen van een order dan wordt het moeilijk, je weet immers niet welke actie er aan de cliënt is gedaan. Is er een order geplaatst door een klant? Is er een prijs gewijzigd door een beheerder? Je zou de volledige change set kunnen analyseren en de conclusie kunnen trekken datt het om het plaatsen van een order gaat omdat er een nieuwe order in de change set zit maar dan wordt het één grote spaghetti in de Submit methode van de domain service. Van de andere kant zou je er ook voor kunnen kiezen om heel kleine domain services te maken: bv. één per entiteit (bijvoorbeeld één voor order en één voor orderregel en één voor prijs). Dit heeft weer als nadeel dat het wijzigingsbeheer van RIA Services niet lekker loopt. Wanneer er door de cliënt een nieuwe order wordt aangemaakt met één of meerdere orderregels dan moet de cliënt een SubmitChanges aanroepen voor zowel order als order regel. Op de server worden deze twee acties dan ook niet als één transactie uitgevoerd maar als twee afzonderlijke. Wij hebben uiteindelijk een aantal regels opgesteld voor de groepering van domain services. Zo moet een domain service één business transactie ondersteunen. Dus we zouden een aparte domain service maken voor het plaatsen van orders (OrderDomainService die het aanmaken van orders en order regels ondersteund en die prijzen alleen maar ophaalt) en een domain service voor het beheer van prijzen (PriceDomainService die het muteren van prijzen ondersteund).

Documentatie

Een andere uitdaging is de documentatie. Deze is nog niet al te best. Er zijn genoeg artikelen, voorbeelden en videos te vinden voor relatief simpele scenario's maar zodra het complexer wordt en je vrij basale architectuur principes wilt gebruiken ben je toch veel aan het zoeken. De documentatie die er is kan soms verwarrend zijn omdat er al een aantal pre-releases van RIA services zijn geweest en de API's van de verschillende versies onderling best verschillen. De documentatie die je dan vind is daarom niet altijd van toepassing op de laatste versie van RIA Services. Gelukkig is er een Silverlight forum waar je al je vragen kunt posten voor zowel Silverlight vragen als RIA services vragen: (referentie toevoegen). Vaak vind je daar al het antwoord op je vraag. Wanneer dat niet het geval is kun je je vraag posten en krijg je eigenlijk altijd snel antwoord.

Toen we begonnen was Silverlight 4 en Visual Studio 2010 nog in Beta. RIA Services was toen ook nog in Beta, er was zelfs nog geen officiële release van RIA Services beschikbaar. Het was wel duidelijk dat Silverlight 4 en Visual Studio 2010 ons veel voordeel op zou leveren. Er was echter twijfel of het switchen van versies tijdens het project geen project risico was. We zijn begonnen met Silverlight 3, Visual Studio 2008 en een release candidate voor RIA Services. Zodra Silverlight 4, Visual Studio 2010 en RIA Services officieel uitkwamen zij we over gegaan. Dit ging opvallend soepel. In RIA services zaten de meeste breaking changes maar omdat die goed gedocumenteerd waren kostte het ons maar een dag of twee om over te gaan op de nieuwste platforms.

De rol van Blend

Blend is het tool dat door designers gebruikt wordt om de applicatie grafisch te ontwerpen (zie ook het artikel van G. van der Pol in .Net Magazine 2 van dit jaar). Ons project had geen specifieke grafisch design eisen. De enige eis was dat het moest passen in de Océ huisstijl voor intranet applicaties (web applicaties dus). De vraag waar we als technisch team voor stonden was of Blend ons zou helpen om de Océ huisstijl te implementeren in Silverlight. Al vrij snel werd duidelijk dat ontwikkelaars erg snel geneigd zijn rechtstreeks met de XAML te werken in plaats van een tool als Blend te gebruiken. Visual Studio 2010 heeft daarbij ook de mogelijk om schermen grafisch te bewerken waardoor de behoefte aan Blend verder af nam. Als uitgangspunt voor de layout hebben we één van de standaard themes voor Silverlight genomen (zie (http://timheuer.com/blog/archive/2010/05/03/new-silverlight-4-themes-available-for-download.aspx). Deze themes bevat een aantal mogelijke layouts. We kunnen jullie aanbevelen om één van deze layouts als basis te kiezen en die verder te bewerken. Opgemerkt dient verder te worden dat de grafisch eisen aan business applicaties in de regel minder hoog zijn dan wanneer je een 'Flash'-achtige toepassing wilt maken. Wanneer het grafisch design wel belangrijk is dan is het effectiever om het design door een designer uit te laten voeren met Blend.

MET DE NIEUWE SILVERLIGHT APPLICATIE WORDT VOORKOMEN DAT EEN VERTEGENWOORDIGER EEN PRODUCT VERKOOPT DAT TECHNISCH NIET KAN WORDEN GELEVERD OMDAT DE VERSCHILLENDE ONDERDELEN NIET MET ELKAAR SAMENWERKEN.

MVVM

Een andere belangrijke keuze die we moesten maken was of we MVVM zouden gebruiken en hoe. De details van MVVM zijn al aan de orde gekomen in het artikel van K. Dockx in .Net Magazine 2 van dit jaar. MVVM staat voor Model - View - ViewModel en lijkt op Model - View - Controller (MVC). In plaats van een controller heb je dus een ViewModel. Het ViewModel is de klasse waar je een pagina aan bind. In onze situatie is het model de RIA Services cliënt, de View een Silverlight pagina en het ViewModel een custom class met properties waar de page aan kan binden. Het lastige van MVVM is dat iedereen adviseert om het te gebruiken maar dat het niet standaard in .Net of Silverlight zit. Je moet het dus zelf maken, de

opmerking die dan gemaakt wordt is dat het pattern zo eenvoudig is dat je het zelf makkelijk kunt maken (en dat is ook zo) maar heeft wel als nadeel dat wanneer je voorbeelden van andere bekijkt je ze nooit rechtstreeks kunt gebruiken. Wanneer je voorbeelden van Microsoft bekijkt in MSDN dan wordt vaak de DomainDataSource als object gebruikt om aan te binden en dat is geen MVVM. Er zijn wel open source MVVM frameworks beschikbaar zoals MVVM light maar die bieden dan weer geen directe ondersteuning voor RIA Services: dergelijke framework gaan ervan uit dat ze tegen een abstract model aanpraten en dat kan van alles zijn, niet noodzakelijk RIA Services. Omdat er RIA services toch wat specifieke eigenschappen heeft zoals het asynchrone gedrag kost het toch wat tijd om een goed MVVM framework te implementeren.

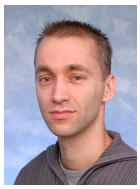
Conclusie

We zijn overtuigd de juiste keuze gemaakt te hebben om de Sales Configurator te ontwikkelen met Silverlight en RIA Services. Beide technologieën hebben een vrij steile leercurve dus het is zeer aan te raden dat je expertise van deze technologieën in je team hebt. Wat ons betreft is de combinatie van Silverlight en RIA services de manier om applicaties te ontwikkelen: zowel bekeken vanuit de eind gebruiker vanwege de rijke gebruikers ervaring als vanuit de ontwikkelaar bekeken vanwege het programmeer gemak en de hoge productiviteit. Toen de eind gebruikers het eerste resultaat zagen waren ze erg onder de indruk. Een Silverlight applicatie geeft een veel betere gebruikers ervaring dan ze gewend waren met applicaties gebaseerd op HTML.

René Titulaer, is software architect en is nu 7 jaar werkzaam voor Océ. Als architect is René is verantwoordelijk voor het opstellen van standaarden en richtlijnen voor Microsoft applicaties en helpt teams met het maken van de software architectuur voor projecten. René is te bereiken via rene.titulaer@oce.com

Danny op het Veld, is technisch teamleider en is 12 jaar werkzaam voor Océ. In die rol is hij verantwoordelijk voor de aansturing van het bouwteam en de afstemming met de rest van de project organisatie. Je kunt contact opnemen met Danny via Danny.ophetveld@oce.com

Marcel Poppe, is na ruim 20 jaar voor diverse softwarehuizen werkzaam te zijn geweest, sinds 2008 in dienst bij Océ. Zijn interesses gaan uit naar software architectuur en design, in combinatie met database management systemen en Microsoft .Net technologie. Marcel is te bereiken via marcel.poppe@oce.com



Wilbert Janssen, is 14 jaar werkzaam voor Océ, waarvan de laatste 3 jaar als Software Ontwikkelaar. Hij ontwikkelt met focus op kwaliteit en is zeer geïnteresseerd in het toepassen van nieuwe technologieën en methodieken. Je kunt contact opnemen met Wilbert via wilbert.janssen@oce.com