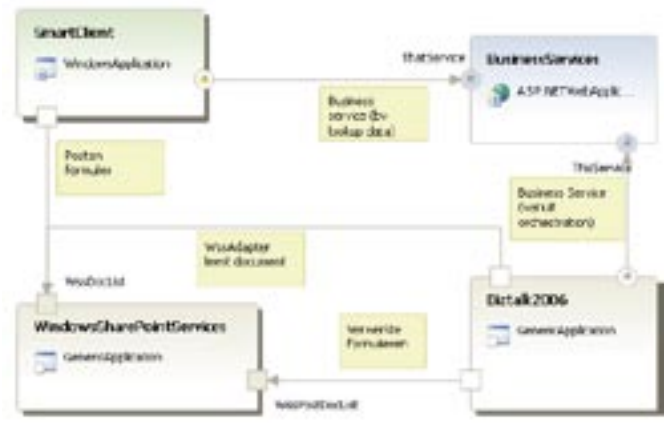
Ilske Verburg en René Schrieken zijn softwarearchitecten bij LogicaCMG

Schema-first en smart clients

DE ONTWIKKELFASE VAN EEN SMART CLIENT-OPLOSSING

De groei van de diversiteit aan devices met substantiële processor- en geheugencapaciteit in combinatie met los gekoppelde servicegeoriënteerde architecturen roept om specifieke architectuurkeuzes. Eén van de mogelijke architecturen is een smart client-architectuur. Maar brengt deze keuze ook nog bepaalde ontwerpprincipes met zich mee naast de principes die gelden op designniveau? Heeft een smart client-architectuur misschien een schema-first aanpak nodig? In dit artikel gaan de auteurs in op wat een schema-first aanpak is, hoe een mogelijke smart client-architectuur eruit ziet en waar schema's gebruikt worden in deze architectuur. De focus ligt vooral op de ontwikkelfase van een smart client-oplossing

Een smart client-architectuur benut volledig de mogelijkheden die de client biedt. Hiervoor is het noodzakelijk dat de client-applicatie op een device draait met enige processor- en geheugencapaciteit. De client is altijd onderdeel van een groter gedistribueerd systeem, zoals een SOA. Dit betekent dat de applicatie communiceert met services die toegang geven tot data of een bedrijfstoepassing. In de client-applicatie wordt logica gerealiseerd om te reageren op het wel of niet kunnen beschikken over een netwerkverbinding. De client-applicatie zal bijvoorbeeld een implementatie voor een serviceagent bevatten. De serviceagent bevat de specifieke logica voor caching (in- en uitgaand) in het geval de smart client geen connectie kan maken met de (web)server. Bovendien bevat het de transformatie van de berichten naar het domeinmodel van de client-applicatie. De serviceagent is ook verantwoordelijk voor het authenticeren van encryptie en signing van de berichten met de juiste credentials en eisen. Uitgaande van deze kenmerken is er een heel scala mogelijk aan verschillende implementaties met verschillende toolsets. De architectuur waar dit artikel aan refereert, is voor een groot deel opgebouwd uit programmeerbare standaardproducten, waarbij InfoPath wordt ingezet als eenvoudige smart client. InfoPath biedt een rijke interface aan de gebruiker en validaties kunnen eenvoudig worden uitgevoerd op de client. Aan de serverkant wordt gebruikgemaakt van webservices met daarnaast BizTalk Server die de orchestratie uitvoert voor de verwerking van de InfoPath-documenten en Windows SharePoint Services als store voor de InfoPath-documenten (zie afbeelding 1).



Afbeelding 1. De verschillende implementaties met verschillende toolsets

In deze smart client-architectuur maakt de gebruiker een nieuw formulier aan vanuit SharePoint. Op dat moment wordt InfoPath geopend met een nieuw formulier. InfoPath kan voor het weergeven van informatie zelf gebruikmaken van diverse datasources waaronder webservices en SharePoint-libraries. Zodra het formulier is ingevuld, wordt het gepost naar een SharePoint form library waar het document wordt 'opgepakt' door een orchestration in BizTalk. In de orchestration wordt het document verwerkt door het aanroepen van webservices om data te verwerken. Daarna wordt het document in een volgende form library geplaatst voor een volgende actie. Een soortgelijke architectuur wordt ook (in meer detail) beschreven in het artikel 'Bouwen van workflow-toepassingen' in het .NET Magazine # 7.

Eerst het schema

Wat is een schema eigenlijk? En wat is een schema-first aanpak? Een schema beschrijft de structuur, inhoud en semantiek van XML-documenten: de metadata. Het schema kan als een zelfstandig onderdeel worden gebruikt om een XML-document te valideren, maar maakt ook onderdeel uit van een webservicecontract. Bij een schema-first aanpak wordt dit schema gedefinieerd en vastgesteld, voordat er een letter code wordt geschreven. Dit in tegenstelling tot de code-first aanpak waarbij het schema impliciet wordt gedefinieerd door in een .NET-taal een class met fields/properties te coderen.

Waarom zou je de ene of de andere aanpak willen gebruiken? Hierover wordt heftig gediscussieerd tussen de verschillende 'kampen'. Het belangrijkste voordeel van een schema-first aanpak, iets wat verder in dit artikel wordt toegelicht, is het vroeg in de iteratie opstellen van het schema als resultaat van een informatieanalyse. De verantwoordelijkheid voor de schema's kan in dat geval bij de businessanalist worden belegd. Hij of zij heeft een overzicht van de data in het systeem, de businessservices die de applicatie moet bieden en de gegevensbehoefte van het bedrijfsproces. Het schema kan gevalideerd worden door materiedeskundigen, en indien nodig, al afgestemd worden met andere partijen in of buiten de organisatie. Dit alles voordat er code wordt

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="MainSchema"
           targetNamespace="http://schemas.company.org/main/2006/07"
           elementFormDefault="qualified"
           xmlns="http://schemas.company.org/main/2006/07"
           xmlns:mstns="http://schemas.company.org/main/2006/07"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:parts="http://schemas.company.org/parts/2006/07">
  <xs:import id="parts"
             schemaLocation="Parts.xsd"
             namespace="http://schemas.company.org/parts/2006/07" />
  <xs:element name="Customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Surname" type="xs:string" />
        <xs:element name="Firstname" type="xs:string" />
        <xs:element name="Birthdate" type="xs:date" />
        <xs:element ref="parts:Address" />
        <xs:element name="Gender" type="GenderType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="GenderType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Male" />
      <xs:enumeration value="Female" />
      <xs:enumeration value="Other" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Codevoorbeeld 1.

geschreven. Door deze validatie en afstemming vooraf ontstaat er een schema dat stabieler is dan een schema dat ontstaat tijdens de realisatie.

Een schema maken

Schema's worden gedefinieerd in de standaard Visual Studio 2005 XML-schema-editor of in de BizTalk-schema-editor. De grafische omgeving biedt met een properties-window en intellisense voldoende functionaliteit om een businessanalist de schema's te laten definieren als resultaat van de informatieanalyse. In het schema wordt per element gedefinieerd aan welke voorwaarde(n) de inhoud van het element moet voldoen. Voorbeelden van dit soort regels zijn: type, cardinaliteit, lengte en bereik. Door dit type regels in het schema te definiëren, kunnen er al basale validaties plaatsvinden zonder dat er code voor hoeft te worden geschreven. Bij het vooraf definiëren van het schema kun je ook gemakkelijker gebruikmaken van include- en importmechanismen in de berichtenstructuur. Een krachtig principe waardoor je generieke schema-elementen kunt creëren die je later kunt hergebruiken in andere schema's. In codevoorbeeld 1 wordt een schema getoond waar het gebruik van een import en een enumeratie op het element Gender wordt gedemonstreerd. In het voorbeeld is uitgegaan van een bestaand schema: parts.xsd waarin zich de definitie voor het element Address bevindt. Door het GenderType-schema-element wordt alleen een waarde van Male, Female of Other toegestaan in het element Gender.

Schema's en componenten in de architectuur

In de eerder beschreven architectuur komen schema's niet alleen terug in de definitie van de webservices, maar komen ook terug in InfoPath als basis voor het formulier en in BizTalk voor de berichtenuitwisseling met de webservices en SharePoint. In InfoPath is er een één-op-één koppeling tussen een schema en het formulier. Het formulier is feitelijk niets anders dan een middel om een XML-bestand op te bouwen volgens de definitie van het schema. Om dit mogelijk te maken, wordt het schema geïmporteerd in InfoPath en kunnen de gegevenselementen vanuit het schema, al dan niet automatisch, op het formulier worden geplaatst. De structuur van het formulier ligt vast in het schema en de mogelijkheden om hier invloed op uit te oefenen zijn beperkt. Wijzigt de schemadefinitie, dan moet het schema opnieuw geïmporteerd worden en de gegevenselementen opnieuw gekoppeld en/of geplaatst worden. Een InfoPath-formulier kan dus eenvoudig gegenereerd worden op basis van het schema; wat sneller werkt dan andersom, het schema opbouwen door het formulier te definiëren. Aan de andere kant leidt deze sterke koppeling ook direct tot rework als een nieuwere versie van het schema wordt geïmporteerd.

In de geschetste architectuur wordt een BizTalk-orchestration toegepast om de informatie in een InfoPath-document te verwerken. In de orchestration wordt de informatie verwerkt door een webservice aan te roepen en wordt er een nieuw document samengesteld en gepost in een SharePoint-form library. In deze situatie komen schema's op een aantal plaatsen terug in BizTalk. De receive-port van de orchestration ontvangt alleen berichten volgens het schema dat ook is gebruikt voor het InfoPath-document. Een operatie op de webservice kan alleen worden aangeroepen met een XML-document zoals gespecificeerd in de webreference (WSDL) van de webservice. De informatie uit ontvangen berichten wordt via mappings vertaald naar te verzenden berichten. En het nieuwe document dat gepost wordt in de SharePoint-form library via een send-port moet voldoen aan het schema voor de desbetreffende form library. In BizTalk moeten de verschillende schema's bekend zijn voordat je de orchestration kunt realiseren. De wijzigingen in een schema hebben vooral invloed op de mappings waarin het schema wordt gebruikt.

Voor de webservice komt het schema terug bij de definitie van het contract (de WSDL). Een contract is opgebouwd uit datatypes, messages en operations en die kunnen al in de vorm van een schema worden weergegeven. Als je bij het maken van de webservices wilt uitgaan van een bestaand schema, dan kun je gebruikmaken van tools die op basis van dit schema de WSDL kunnen genereren. Hiervoor biedt Visual Studio geen ondersteuning, maar er zijn genoeg (freeware) add-ins die dit wel kunnen. De WebServices-ContractFirst (WSCF) tool is hiervan de bekendste. Door op deze manier datatypes en messages in het schema vast te leggen, heb je maximale controle over namespaces, elementnamen en operaties. De schema's en WSDL vormen samen het contract van de webservice. De gebruikers van de webservice genereren met behulp van een tool (wsdl.exe/svcutil.exe) de webservice client-proxy-classes ('de webreference'). De schema's en WSDL zijn onafhankelijk van de gebruikte technologie. Voor het ontwikkelteam van de service is een wijziging in het schema redelijk eenvoudig te ondervangen. Het bestaat voornamelijk uit het toevoegen en/of verwijderen van operaties en het aanpassen van de mapping van datatypes naar business-entities. De meeste impact hebben wijzigingen op de gebruikers van de webservice, omdat zij een nieuwe proxy moeten genereren. In deze smart client-architectuur bevinden zich ook InfoPath- en BizTalk-gebruikers van de webservice. Wijzigingen hebben vooral impact op de reeds ontwikkelde formulieren en orchestrations.

Schema-first en smart client?

In de voorgaande alinea's is toegelicht wat een schema-first aanpak is, hoe een mogelijke smart client-architectuur eruit ziet en waar schema's gebruikt worden in deze architectuur. De focus heeft hierbij vooral gelegen op de ontwikkelfase van een smart client-oplossing, omdat de beheer- en onderhoudsproblematiek (waaronder versioning van schema's) buiten het bereik van dit artikel valt.

Maar er is nog geen antwoord gegeven op de initiële vraag of deze smart client-architectuur een schema-first aanpak nodig heeft. In een smart client-architectuur heb je altijd te maken met expliciete koppelvlakken tussen het client- en het serverdeel binnen de architectuur. In de geschetste architectuur zijn expliciete koppelvlakken te onderkennen tussen de verschillende componenten in de architectuur: InfoPath, BizTalk en de webservices. Het schema wordt op deze koppelvlakken gedeeld door twee of meer componenten. Een wijziging in het schema op een dergelijk koppelvlak raakt dus

altijd verscheidene componenten in de architectuur. De impact van de wijziging zal beïnvloed worden door de gekozen toolset, maar zeker in het geval van InfoPath en BizTalk leveren schemawijzigingen nogal wat werk op. Door de schema's vroegtijdig op te laten stellen door de businessanalist en ze als uitgangspunt te nemen voor de ontwikkeling worden schemawijzigingen tijdens de realisatie beperkt. Dit heeft een gunstig effect heeft op de ontwikkeltijd. Een schema-first aanpak is niet noodzakelijk bij een smart client-architectuur, maar een best practice vinden wij het zeker wel.

Ilske Verburg is softwarearchitect bij LogicaCMG (www.logicacmg.nl) voor vragen en opmerkingen is zij te bereiken via ilske.verburg@logicacmg.com

René Schrieken is softwarearchitect bij LogicaCMG (www.logicacmg.nl) voor vragen en opmerkingen is, hij te bereiken via rene.schrieken@logicacmg.com

Referenties

Smart Clients: http://msdn.microsoft.com/architecture/community/newsletter/022004newsletter.aspx

Contract First Web Services Interoperability between Microsoft .NET and IBM WebSphere: http://msdn.microsoft.com/vstudio/java/interop/websphereinterop/default.aspx

Place XML Message Design Ahead of Schema Planning to Improve Web Service Interoperability: http://msdn.microsoft.com/msdnmag/issues/02/12/webservicesdesign/default.aspx

WSCF - Schema-Based Contract-First Web Services: http://www.thinktecture.com/Resources/Software/WSContractFirst/default.html