De kern van een Service Oriented Architecture is ontkoppelen

De Oracle database en Oracle SOA Suite

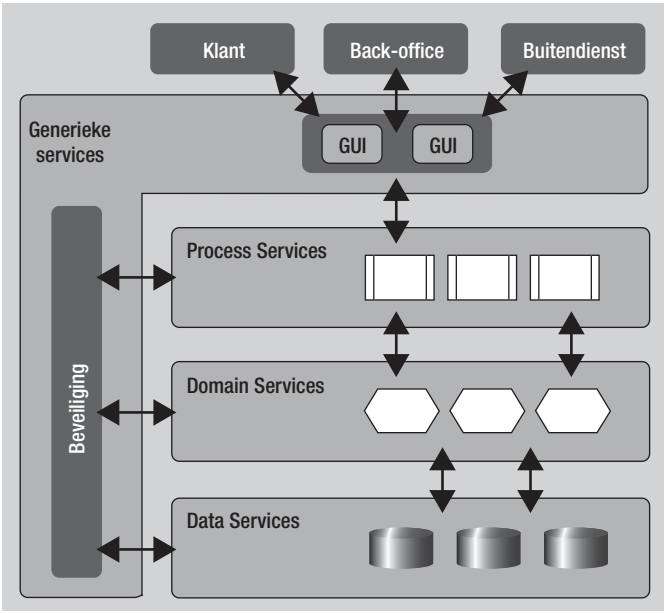
Jules de Ruijter

De Oracle SOA Suite biedt een brede ondersteuning voor software-ontwikkeling en applicatie-integratie op basis van services en een service-georiënteerde architectuur. De SOA Suite bevat tools gericht op ontwikkeling van front-ends, bedrijfsprocessen, bedrijfsregels, security, beheer etcetera.

De toepassing van een servicegeoriënteerde architectuur wordt langzaam gemeengoed bij het ontwikkelen van applicatie-functionaliteit. In dit artikel gaan we in op de effecten op de database van het werken onder deze architectuur en met de Oracle SOA Suite. Na een historisch perspectief gaan we dieper in op de effecten op bedrijfsregels, front-ends en de ontwikkelstraat.

Historisch perspectief

Sinds het ontstaan van de database heeft deze verschillende rollen vervuld in de architecturen van het moment. De eerste hiërarchische en netwerk-databases waren niet meer dan opslag-mechanismen. Referentiële integriteit en bedrijfsregels werden bewaakt in de programmatuur die de database benadert.



Afbeelding 1: Referentiearchitectuur.

Hetzelfde geldt ook voor de eerste generaties relationele databases (na 1970).
In de jaren daarna komen zowel programmeertalen in de database (PLSQL, Java), als een reeks functionaliteiten zoals queues, OLAP, Spatial, XML. We zien dan een verschuiving van applicatie-functionaliteiten naar de database: referentiële integriteit, bedrijfsregels tot en met complete applicaties (APEX). De database is dan verworden tot een container van data met de bijbehorende logica. Eén van de problemen die we zagen ontstaan in de client/server-architectuur is de verdubbeling van functionaliteit op het gebied van referentiële integriteit en het toepassen van de bedrijfsregels.

Het ontstaan van de n-tier architecturen (n >= 3) leidt tot ontkoppeling van de presentatielaag van de business-logica laag, maar het probleem van 'verdubbeling' blijft. Zo ontstaan bedrijfsbrede datamodellen, volgebouwd met referentiële integriteits-triggers. De aanleiding voor deze oplossingen is wel duidelijk: het zorgt voor één punt waar alle eisen aan de consistentie worden bewaakt. Het leidt echter wel tot een omgeving die zich niet of nauwelijks laat onderhouden.

Ontkoppelen

SOA staat voor een stijl van ontwikkelen waarbij het ontwerp-criterium 'ontkoppelen' (loosely coupled) een centrale rol speelt. De ervaring leert dat het noodzakelijk is om behalve een scheiding in lagen (n-tier architecturen), ook een scheiding in de data (domeinen) aan te brengen. Dit is een absolute vereiste om het gewenste niveau van ontkoppeling in een SOA te kunnen realiseren. Het uitgangspunt daarbij is namelijk dat een service zelf verantwoordelijk is voor de kwaliteit en integriteit van de gegevens die hij beheert. Dat vereist dat deze data ook autonoom zijn geregistreerd, *zonder* harde relaties met andere gegevens. Dit heeft geleid tot een aanpak waarbij de gegevens op zijn te

delen in 'basisregistraties' (of data-agglomeraties). Dit zijn functioneel samenhangende groepen van gegevens, die via één set van services beschikbaar worden gesteld. De integriteit binnen dit domein is prima in de database te realiseren met gebruik van referentiële integriteit, triggers enzovoort. De referentiële integriteit met gegevens buiten het domein – die beheerd worden door andere services – zal door de service moeten worden bewaakt. Hiermee wordt de ontkoppeling van de domeinen bewerkstelligd, waarmee het bouwen van een echte service-gerichte architectuur mogelijk wordt.
In de gehanteerde referentiearchitectuur (zie afbeelding 1) kunnen we verschillende typen services onderkennen:
- data services, diensten die de database benaderen;
- domain services, diensten die een domein beheren;
- process services, diensten die een proces besturen;
- generic services, algemene diensten (beveiliging, printen, etcetera).

De architectuur is opgebouwd uit vele lagen, gebaseerd op het architectuurprincipe van ontkoppelen. Zo zijn de volgende zaken ontkoppeld:
- de data van de wijze waarop zij gepresenteerd zijn;
- de logica van de data;
- de logica van de procesvoering;
- de procesvoering van de data.

Bedrijfsregels

De kern van een Service Oriented Architecture is dus ontkoppelen. Hierbij wordt zowel een onderverdeling in horizontale lagen gemaakt als in een aantal verticale kolommen (basisregistraties) in de database. Deze basisregistraties kennen geen onderlinge afhankelijkheid. Een voorbeeld hiervan is een relatieservice en een orderservice. In de wereld van de data driven applicaties is de vraag over referentiële integriteit de eerste vraag die ontwikkelaars dan opwerpen. Immers hoe voorkom je dat een relatie weggegooid wordt terwijl er nog lopende orders zijn, hoe ga je om met bedrijfsregels in een SOA-wereld?
In de Oracle-wereld was Oracle Designer dé tool om applicaties in te ontwikkelen. In de loop van de tijd is rond Designer een ontwikkel-framework ontstaan met CDM, CDM RuleFrame en Headstart. CDM schrijft voor dat bedrijfsregels geclassificeerd worden:
- static data constaint rules (postcode is verplicht);
- dynamic data constraint rules (een order bestaat uit een of meerdere orderregels);
- change event rules (een order kan pas verzonden worden als deze betaald is);
- authorization rules (een order van meer dan 10.000 euro moet worden gefiatteerd).

CDM RuleFrame implementeert deze regels in een package-laag in de database. Iedere DML-actie op een tabel moet door deze laag. Het inzetten van andere GUI's (bijvoorbeeld mobiele toepassingen) is hiermee eenvoudiger. Een hele generatie ontwikkelaars heeft hiermee projecten ontwikkeld en is terecht erg tevreden met het resultaat. De switch naar een Service Oriented Architecture lijkt een streep door deze methodiek te halen. Maar is dat ook zo?

Laten we eerst eens met een andere benadering naar de classificatie van de bedrijfsregels kijken:

	Data-oriëntatie	Proces-oriëntatie	Low level	High level
Static data constraint	X		X	
Dynamic data constraint	X			X
Change event rule		X	X	
Authorization rule		X		X

De bedrijfsregels zijn nu ingedeeld naar data/procesoriëntatie en naar het niveau van aggregatie. Zoals eerder is gesteld, is SOA de architectuur van het ontkoppelen. Eén van de zaken die we graag ontkoppeld zien betreft gegevens en processen, maar verder ook programmatuur en bedrijfsregels. Concreet betekent dit dat we services maken om data te beheren en services orkestreren om processen uit te voeren. Dat betekent derhalve dat datagerichte regels in een data-service thuishoren en procesgerichte regels in een orkestratie (lees: BPEL). Ergo, 'Change event' en 'Authorization' horen in een process service thuis.
Een tweede aspect dat uit de tabel naar voren komt is het aggregatieniveau van de regel. Dat heeft rechtstreeks te maken met het begrip 'granulariteit', de grofkorreligheid van een service.

Granulariteit

Services worden ontwikkeld met verschillende granulariteit. Als we de eerder genoemde type services (zie afbeelding 1) in aflopende granulariteit weergeven, is dat als volgt: process services; domain services; basic services/data services; generic services. Binnen ieder onderkend type service bestaat een gelaagdheid in granulariteit. Ook in de data services kunnen we onderscheid maken naar granulariteit. Grofkorrelige services worden veelal samengesteld uit één of meerdere fijnkorrelige services. Om nu terug te keren naar de bedrijfsregels: in de fijnkorrelige data-services zullen de static data constraint rules op exact dezelfde wijze worden geïmplementeerd zoals we dat altijd gedaan hebben. Om de tabellen wordt een package-laag (Table API) gecreëerd.

De Oracle SOA Suite

Deze suite is een complete set hulpmiddelen voor het ontwikkelen en beheren van software op basis van SOA. Voor elke component is er een plug-in voor JDeveloper, voor het ontwikkelen van de services en toebehoren. Daarnaast zijn er binnen de SOA Suite diverse consoles beschikbaar (in een browser) die fungeren als een gebruiksvriendelijke schil om de complexe configuratie-files en monitor-faciliteiten.

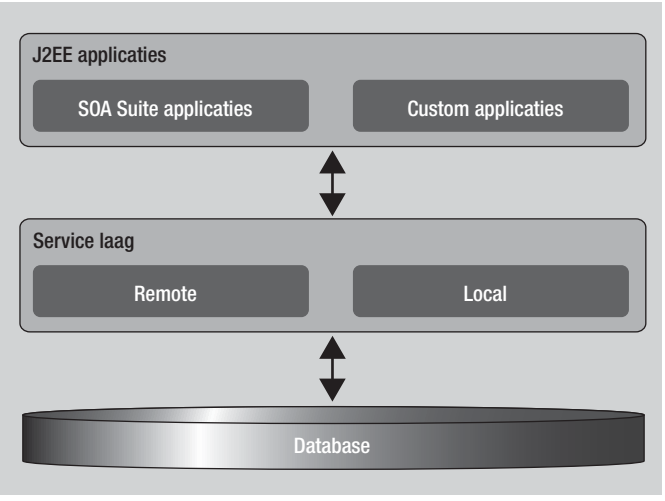
Iedere DML-actie op een tabel wordt hiermee netjes aan de rules onderworpen.
Daar waar constraints de fijnkorrelige dataservices overstijgen moet er wel iets gebeuren. Er wordt dan veelal een grofkorrelige samengestelde (composite) service ontwikkeld: de domain service. In deze domain service wordt dan de dynamic data constraint afgehandeld.

Procesgericht

De implementatie van de change event rules en authorization rules vindt in de process services plaats. Processen zijn feitelijk niet meer dan een orkestratie van services. Het toepassen van de regel zien we dan ook als het uitvoeren van een service. Of we deze regels dan zelf coderen (in BPEL of in een service) of onderbrengen in een decision service (Oracle Business Rule Engine) is afhankelijk van de complexiteit en veranderlijkheid van de betreffende regel. Vooral sterk aan verandering onderhevig zijnde regels lenen zich er sterk voor om in een rule engine te worden ondergebracht.
Met deze oplossingswijze lijkt het beeld te ontstaan dat de regels op verschillende plekken worden geïmplementeerd en daarmee niet consistent worden afgedwongen. Dit is niet het geval. Ja, regels kunnen op verschillende plaatsen worden geïmplementeerd, maar dit gebeurt maar éénmalig! De afweging om te bepalen op welke plaats een regel wordt geïmplementeerd blijkt in de praktijk eenvoudig genoeg te maken.
Hiermee is en blijft het mogelijk om de kracht van de database optimaal te benutten. Er zit zoveel functionaliteit in de database die vaak onbenut blijft, en het kan niet de bedoeling zijn van een SOA om de database 'uit te kleden' tot een veredeld opslagmechanisme. Het moet alleen duidelijk zijn dat ontkoppeling het *centrale ontwerpcriterium* is van een SOA en dat daaraan niet getornd mag worden!

Front-ends

Een SOA heeft als een van de belangrijkste uitgangspunten dat de componenten binnen de architectuur ontkoppeld zijn om de flexibiliteit te verhogen. Dit vereist dat ook front-end applicaties zoveel mogelijk losgekoppeld dienen te zijn van de database.
In een traditionele Oracle-omgeving is de front-end applicatie gebaseerd op het 2-tier client-server principe, waarbij de businesslogica en functionaliteit in de database geïmplementeerd zijn (Forms) of er heel dicht tegenaan zitten (JHeadstart). Daarnaast is de applicatie vaak een administratieve applicatie, waarbij de formulieren op de schermen vaak een een-op-een relatie met de database-tabellen hebben. Deze twee aspecten zorgen ervoor dat de front-end applicatie veel database-specifieke kennis bevat, waardoor de applicatie sterk gekoppeld is met de database.
Door het uitgangspunt van ontkoppeling tussen componenten is er een ander type front-end applicatie nodig. In plaats van direct in de database-tabellen acties uit te voeren, wordt gebruik maakt van de services[1] (remote of local). Specifieker, de front-end applicatie zal meer in een 3-tier architectuur ontwikkeld worden, waarbij de database-laag met een service-laag ontkoppeld is van de UI-laag. In afbeelding 2 is dit schematisch uiteen gezet, waarbij de front-end applicaties in J2EE[2] zijn geïmplementeerd. Door de ontkoppeling is het eenvoudig een andere database te gebruiken zonder dat deze impact heeft op de draaiende front-end applicatie.



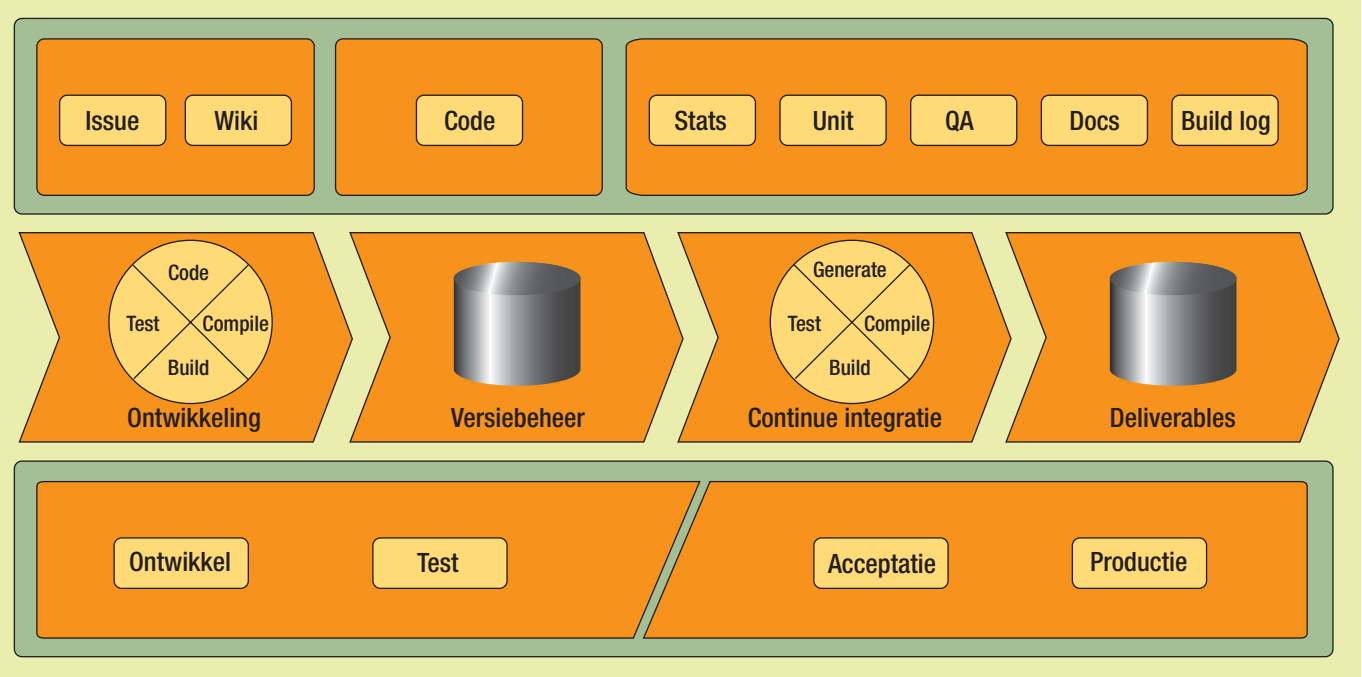
Afbeelding 2: Een service-laag verbindt de database-laag met de UI-laag.

Technologiekeuze

Oracle biedt zowel met de combinatie Application Server en JDeveloper een uitstekende ondersteuning om 3-tier applicaties mee te ontwikkelen. Om de front-end laag (ook wel view-laag genoemd) te implementeren, biedt Oracle ondersteuning voor de Sun-standaarden JSP[3] en de opvolger JSF[4], waarmee in combinatie met AJAX-technologie[5] de rijke UI eigenschappen van Forms en JHeadstart grotendeels geëvenaard kunnen worden. Daarnaast biedt Oracle met ADF Faces[6] een krachtig tool om applicaties snel mee te kunnen genereren.
Voor de service-laag heeft Oracle zijn eigen, op Toplink gebaseerde, EJB3-implementatie[7]. EJB3 biedt de mogelijkheid om services eenmaal te implementeren en vervolgens als webservice, remote bean of local bean aan te bieden aan de verschillende componenten binnen de SOA. Oracle ziet de bovenstaande technologie-indeling voor de view- en service-laag als een de facto standaard voor het ontwikkelen van front-end applicaties binnen een SOA.

Ontwikkelstraat

Het grote verschil tussen SOA- en traditionele software-ontwikkeling is dat we bij SOA niet meer spreken over applicaties. Een SOA kenmerkt zich door hergebruik en het combineren van al bestaande functionaliteit. Functionaliteit is niet meer onderdeel van één applicatie, maar van een domein of een bedrijfsproces. Als er bij applicaties al hergebruik was, dan vond dit vooral plaats in de database, en iedere nieuwe applicatie was een losstaand iets dat veelal alleen de database deelde met andere applicaties.
Het hergebruik vindt plaats op veel niveaus en dat heeft impact op de software-ontwikkeling. Een SOA-toepassing bestaat uit meer componenten (nieuw en bestaand, binnenshuis en bij



Afbeelding 3: Virtual machine.

derden) waardoor coördinatie en integratie een stuk belangrijker worden. Het los van elkaar ontwikkelen van alle onderdelen, en pas op het laatste moment integreren van deze onderdelen, is vragen om moeilijkheden.
Het naar voren halen van de integratie tijdens de ontwikkelfase is niet iets nieuws maar het wordt dus wel steeds belangrijker. Tijdens het ontwikkeltraject is het verstandig om minimaal een keer per nacht alle onderdelen te integreren: een nachtelijke build. Met behulp van een continuous integration tool kan dit zelfs nog vaker. Dergelijke tools zorgen ervoor dat automatisch een build wordt gedaan van alle software zodra een ontwikkelaar wijzigingen aanbrengt in het versiebeheer-systeem. Een automatische build bestaat hierbij niet alleen uit het compileren van alle code, maar vooral ook uit unit-testen, deployen, en integratietesten. Alleen op deze wijze is het mogelijk goed te testen of alle onderdelen correct samenwerken.

Scripts

Het maken van build- en tests-scripts voor BPEL-processen wordt standaard ondersteund door de Oracle SOA Suite. Voor alle BPEL-processen wordt automatisch een Ant build file gemaakt, waarmee processen gecompileerd en gedeployed kunnen worden. De gegenereerde test-scripts kunnen handmatig of als onderdeel van de build uitgevoerd worden. Hierbij kunnen externe services gesimuleerd worden, zodat je verschillende scenario's eenvoudig kunt testen, zonder afhankelijkheid van de externe services. Omdat er gebruik gemaakt wordt van Ant scripts is het eenvoudig deze stappen te integreren in de automatische nachtelijke build. Dit systeem is niet alleen toe te passen op de standaard SOA-onderdelen zoals BPEL-processen en web services, maar bijvoorbeeld ook op de onderdelen die in de database draaien. Dit betekent dat ontwikkelaars SQL scripts opleveren die iedere nacht gedraaid kunnen worden om de database-objecten naar de laatste stand te zetten. Ook schrijven de ontwikkelaars unit-tests voor database-objecten. Als je op deze manier werkt, helpt het enorm als je de database eenvoudig terug kunt zetten naar een laatst bekende stand, meestal de laatste productieversie. Dit kan met behulp van backups, maar een virtual machine met ondersteuning voor snapshots maakt dit veel eenvoudiger, zie afbeelding 3.
Virtual machines zijn ook voor een ander doel te gebruiken: om aan het begin van een nieuw project niet teveel tijd kwijt te zijn met het inrichten van een ontwikkelstraat, is een ontwikkelstraat virtual appliance te ontwikkelen. Dit betekent dat je binnen vijf minuten een draaiende ontwikkelstraat kunt hebben, met onder andere versiebeheer, continuous integration, test-tools, documentatie-tools, en issue management. Dit bespaart een hoop tijd aan het begin van een project.

Noten

1. *Onder services verstaan we niet per definitie webservices. Een EJB3 module met session facades kan evengoed als service dienen, zowel in de lokale als remote variant.*
2. *Java 2 Enterprise Edition.*
3. *Java Server Pages.*
4. *Java Server Faces.*
5. *Asychronous Javasript And XML.*
6. *ADF Faces is Oracle's JSF implementatie met o.a. ADF en AJAX mogelijkheden.*
7. *Enterprise Java Beans 3.0.*

Jules de Ruijter (Jules.de.Ruijter@it-eye.nl) is Managing Consultant bij IT-eye.