

W e l o v e i t

Magazine voor en door **Oracle & Java** gebruikers en ontwikkelaars

Getronics
PinkRocade

DBA.nl

BACKBASE

Belastingdienst
Centrum voor ICT

CAESAR
GROEP

centraal boekhuis

Inter Access

Jom IT

KADENZA
HAVING A CLUE
KeyOn

MediaCompetence
Tel: (+31) 035 - 548 6008

ORDINA
CONSULTING | ICT | OUTSOURCING

PARTNER

Provide
consultancy

QUALOGY

Quobell
IT-dienstverlening

RULEGEN
Delivering the Relational Promise

Ten+ SAM

UNISYS

VX Company
ENTERPRISE IT

POWERED BY SHART



BACKBASE

Van Inside-out naar Outside-in

Ria's verrijken presentatielaag van
IT-toepassingen

Door de snelheid waarmee technologische ontwikkelingen
zich voltrekken, kunnen we ons moeilijk voorstellen hoe
veel jaren geleden zonder internet en zonder
communicatie werkpacties konden leveren.

29-33 34-39



Over virtualisatie en licenties



Adressenbeheer met PL/SQL en JHeadstart

De werkprocessen van ondernemingen zijn lange tijd verborgen gebleven voor de buitenwereld. Die afsluiting vindt zijn oorsprong in de industriële productie. Daar werken mensen gezamenlijk in een fabriek, letterlijk achter gesloten deuren. In de laatste omgeving werd dezelfde gesloten werkwijze overgenomen. Zelfs de medewerkers die de commerciële contacten onderhouden, deden weinig moeite om hun werk ietwat open te maken. Niet voor niets spreken we lange tijd over een verkoopbinnenland en een verkoopbuitensland. Met de opdeling van grote afdelingen in kleine business units zijn de processen transparanter geworden en daarmee ook de organisaties. De buitenwereld mag zien hoe we werken. In veel gevallen maar dat ook, omdat in de kanten bedrijven veel meer met elkaar te maken hebben. De afzonderlijke bedrijfsprocessen grijpen steeds vaker op elkaar in. Daarmee raken ook de IT-voorzieningen met elkaar verweven. En dankzij de mogelijkheden van het internet is integratie niet langer een onmogelijke opgave.

Het inside-out denken, maakt plaats voor outside-in. Laat de klanten of partners maar binnen en richttoets op onze systemen transacties activeren. Los van het beveiligingsvraagstuk stelt een dergelijk openheid ook eisen aan het bedieningsgemak. Bij gebruikers van bijvoorbeeld een bedrijfsgebonden ERP-systeem mag je na jaren van training en praktische ervaring voldoende expertise veronderstellen. Die weten welke schermen ze moeten opstarten en welke velden moeten worden ingevuld om het gewenste proces uit te voeren. Indien we ook 'niet ingewijden' toegang bieden tot het primaire bedrijfsinformatiesysteem, dan moet het bedieningsgemak aanzienlijk verbeteren. Met andere woorden: aan de presentatielaag moet intelligentie worden toegevoegd.

Kent u nog de klassieke theorie over systeemontwikkeling volgens het J-ter model? Eerst komt de data laag, daarna de laag met de procesbeschrijvingen en vervolgens de presentatielaag. Voor de eerste 2 hebben we de zaken uitstekend geregeld. Van object gestructureerd tot aan re...

... informatie nu in de kleinsten haast...
... op te slaan en te herleiden....
... modellen zoals stellen ons in staat...
... alle gegevens en business processen...
... in een organisatie perfect te beschrijven...
... en te versieren naar maximaal...
... code. Een afzonderend service...
... etienneel architectuur zorg ervoor dat...
... de processen op verschillende platformen...
... voor uiteenlopende doeleinden perfect...
... via het web met elkaar samenwerken.

Maar aan de presentatielaag heeft de IT industrie gedurende de afgelopen 50 jaar niet zo veel gedaan. Tot dat terwijl het de laag is, die direct de gebruikers aanpreekt, gebruikers waaraan iedereen in de IT denkt dat we er zo lang hebben zitten. Want daar door we het resultaat toch afkomstig voel. Niet dat het niet de introductie van Web 2.0 kwam lang in het veld. Rich Internet Applications (RIA) gaan alle jaren ten aanzienheid goed maken.

Volgens onderzoeksbureau Gartner maakt in 2010 minstens 60% van de nieuw te ontwikkelen applicaties gebruik van RIAs, terwijl 25% er primair op leunt. RIAs zijn webapplicaties die de methode Ajax en rijkdom van de desktop vermogen met de unieke eigenschappen van het internet. De combinatie zorgt voor dynamische en gebruiksvriendelijke interacties.

De IT-industrie van vandaag hermaakt zich door een toereiding. Aan de ene kant staat Microsoft, die met Windows al jaren boerst over de GUI (Graphical User Interface) op de desktop, de presentatielaag dus. Die dominantie wordt de gigant uit Seattle er niet van om toen laterzet zich als industrie revolutie aandende met... Niet een vinger in de IT-ontwikkeling te houden. Het andere kamp vestigde zich rond Java, de universele ontwikkeltaal voor het web. In die hoop kreeg de presentatielaag eindelijk erkenning met de samenwerking van twee ontwikkelstandaarden: Asynchronous Javascript and XML (AJAX) (XML-based Markup Language) met Ajax, een browser onafhankelijk, breed gebruikte technologie voor het maken van bedieningsgeschiktheid webpagina's.

'n gesprek met jou is 500 euro waard
(zie bijgevoegde kaart)



"Ik ben geen voorstander van database onafhankelijk programmeren, om het simpele feit dat de database zoveel rijke functionaliteit bezit die je laat liggen als je alle logica en business rules in de applicatie stopt"

J O M - I T

Adressenbeheer met PL/SQL en JHeadstart

Door Toine van Beckhoven

Als enthousiast Oracle ontwikkelaar mag je altijd hopen op een uitdagende opdracht. Eentje waar je al je creativiteit in kunt leggen, waarin je nieuwe paden bewandelt en waarvan het eindresultaat zowel succesvol is als het onderhoud blijvend boeiend. Eentje waar je met recht trots op bent.

Toen ik bij een klant tien jaar geleden gevraagd werd om mee te schrijven aan een Centraal Adresboek dat moest dienen als het hart van het adressenbeheer en als de bron van adresgegevens voor alle andere applicaties binnen de organisatie, had ik geenszins het idee dat dát zo'n applicatie voor mij zou worden. Deze applicatie kreeg echter in de loop der jaren steeds weer nieuwe dimensies, waardoor ie zelfs tot nu toe interessant is gebleven om aan te werken. Uitbreiding naar andere delen van de organisatie, vervanging van interfaces en een nieuwe web based userinterface met behulp van Oracle's JHeadstart zijn een paar van die interessante uitdagingen geweest de afgelopen jaren. Eén ding bleef zeer stabiel, niet in het minst door de initiële opzet: het database-centrische karakter van de applicatie in Oracle. In dit artikel wil ik de verschillende facetten van het programma belichten.

In 1997, als werknemer van IT dienstverlener Dedicate en een jaar daarna van Motiv werd ik bij een nieuwe klant gedetacheerd: een verkooporganisatie in het Zuiden van Nederland. Bij deze organisatie zijn steeds zo'n 80 vertegenwoordigers onderweg om de producten bij diverse doelgroepen onder de aandacht te brengen. Accurate "adresgegevens" van die doelgroepen is voor hun werk van groot belang: het bepaalt welke personen en organisaties op hun laptop geladen worden vanwege de indeling in rayons en uiteraard is het juiste adres belangrijk om bij een bezoek niet aan de verkeerde deur te staan. Mijn eerste opdracht was destijds het repliceren van mutaties in adresgegevens uit een AS/400 applicatie naar het toenmalige CRM systeem. Die adresgegevens werden door een aparte afdeling cartotheek nauwlettend bewaakt: wekelijks werden mutaties in adresgegevens aangeleverd door een dataleverancier (een externe partij gespecialiseerd in het bijhouden van adresgegevens voor de sector). Die mutaties werden allemaal via het programma bekeken en vervolgens geaccepteerd, geweigerd of geaccepteerd na wijziging. Edel maar arbeidsintensief werk!

De AS/400 echter had zijn langste tijd gehad en daarmee was de tijd

gekomen om de daarop draaiende applicatie te migreren naar een nieuw platform. Windows NT als Operating system en Oracle (versie 7.3.2 destijds) waren de standaard in de organisatie en de keuze voor met name Oracle was dus een gegeven. Als Oracle programmeur verzorgde ik het datamodel van de nieuwe applicatie die 'Centraal Adresboek', kortweg CAB, ging heten. Twee Visual Basic programmeurs programmeerden de frontend. De nieuwe applicatie had de volgende belangrijke specificaties (naast het kunnen opslaan van de gewenste gegevens):

- **Het moet een importmodule bevatten waarbij gegevens komende van een ander systeem gecontroleerd geïmporteerd konden worden in de database.** Er werkten destijds twee mensen in de afdeling 'cartotheek' die in de AS/400 applicatie een reeks van 'fiaterring'-schermen hadden om mutaties komende van de adresleverancier te verwerken..
- **Het moet mogelijk zijn om te specificeren welke mutaties van de adresleverancier automatisch (ongezien) doormogen en welke door de cartotheek gefiatteerd moeten worden.**
- **Het moet mogelijk zijn om mutaties op gegevens, gedaan in de applicatie, te exporteren naar diverse andere applicaties.** Dit onderstreepte de wens van de applicatie als 'hart' van alle adresgegevens. De toenmalige geïdentificeerde 'export'systemen waren het nieuwe CRM-systeem Siebel (versie 5 destijds), het financiële systeem SAP en een ander van strategisch belang zijnde applicatie. Daarnaast zouden de gegevens uit CAB gebruikt worden in een reeks van zelf geschreven kleinere (Oracle) applicaties.

De architectuur

Uiteraard werd als eerste goed nagedacht over het datamodel. In dat opzicht namen we het beste uit de bestaande AS/400 applicatie en het door Siebel te vervangen CRM systeem Sales+. Het datamodel bestond uit om en nabij vijftien basistabellen:

- personen, organisaties, afdelingen, adressen en postcodes
- adreskoppelingen voor personen, organisaties en afdelingen

- persoon/organisatie en organisatie/organisatiekoppelingen
- een codetabel (domains)
- kenmerken en de koppeling van kenmerken met relaties (personen/organisaties en afdelingen)
- een primaire sleuteltabel.

Dat er aan dit originele datamodel een paar nadelen kleven voor wat betreft schaalbaarheid en performance is bekend (eventuele serialization in verband met een primaire sleuteltabel i.p.v. het gebruik van sequences en een codetabel met een type kolom: dit is niet ideaal voor de Cost based optimizer tot en met release 10). Voor de applicatie is dit tot nog toe geen enkel probleem geweest door de aard van het gebruik.

Gegeven de eis van een importmodule waarin externe systemen hun data aan konden bieden en de bouw van een Visual Basic GUI via welke wijzigingen gedaan konden worden, besloot ik direct dat de validatie en business rules zo dicht mogelijk bij de data moesten liggen en het database-centrische karakter van de applicatie was daarmee geïnitieerd.

Op die wijze werden deze altijd afgedwongen, ongeacht de bron van de mutaties. Het is een werkwijze waar ik mede door deze applicatie volledig achter sta. Het gaf ons bijvoorbeeld twee jaar geleden de kans om enorm snel een nieuwe GUI te bouwen met behulp van voor ons op dat moment nog onbekende webtechnologie (gebruik makend van JHeadstart). De business rules, gebouwd in PL/SQL, zouden bestaan in de vorm van database triggers en database packages.

PL/Generator en TTG

In beginsel begon ik met het definiëren van de business rules in een Word document, maar na een paar uur kreeg ik de ingeving dat het makkelijk zou zijn als ik een kleine 'regel' database aan zou leggen met een simpele interface (geschreven in Oracle Forms) en daarin de business rules op wat meer gestructureerde wijze zou gaan vastleggen. Van het een kwam vervolgens het ander. Ik bedacht dat met een kleine rule generator trigger templates gemaakt konden worden. In eerste instantie dacht ik dat

Adressenbeheer met PL/SQL en Jheadstart

Vervolg

ik dan alvast CREATE TRIGGER statements zou genereren die niets deden, behalve in commentaar de business rules weergeven die ik op die plek moest coderen. De simpele documentatie tool groeide echter uit tot een lichtgewicht case tool waarmee ik nog altijd de backend triggers en packages genereer en compileer. Dit werd niet in het minst mogelijk gemaakt door een integratie met een PL/SQL generator tool die de bekende PL/SQL expert Steven Feuerstein destijds gebouwd had (en helaas veel te weinig gebruikt is): PL/Generator. Deze PL/SQL generator was in beginsel vooral een Table Encapsulator generator: je geeft een tabelnaam op en de tool genereert een PL/SQL package met standaard functies, cursoren en procedures om inserts, updates, deletes, “upserts” en allerlei queries te kunnen doen via PL/SQL. De tool is echter volledig open: je kunt eigen PL/SQL mee laten genereren in de packages en het heeft een API om settings te manipuleren. Vanuit mijn ‘case tool’, die ik TTT doopte, genereerde ik de trigger files en customization files om vervolgens PL/Generator de uiteindelijke packages te laten genereren. TTT stond voor niets originelers dan “Toine’s Trigger Generator”. De positieve aspecten van deze aanpak waren een hoge mate van standaardisatie in het uiterlijk van de (database) code, het kunnen uitdraaien van een deel van de documentatie en vanwege de template language die ik zelf in de tool bouwde en die in PL/Generator aanwezig is (CGML), een hoge mate van generiek programmeren. Het bekende mutating table probleem dat vaak optreedt bij database triggers was ook eenvoudiger op te lossen. We hebben de tool in de afgelopen jaren dan ook voor andere maatwerkapplicaties gebruikt. Oracle Designer was destijds niet overwogen als optie. Achteraf was dat wellicht een goede mogelijkheid geweest met het CDM RuleFrame. Echter vooraf had ik geen idee waar TTT uiteindelijk zou belanden. Over het database centrische

karakter (alle business logica dicht bij de data, dus in de database) kan ik tot op de dag van vandaag nog laaiend enthousiast worden. Ik ben geen voorstander van database onafhankelijk programmeren, om het simpele feit dat de database zoveel rijke functionaliteit bezit die je laat liggen als je alle logica en business rules in de applicatie stopt, naast al het extra werk dat nodig is als de frontend van omgeving wijzigt. En dat gebeurt veel vaker dan dat gekozen wordt voor de database van een andere leverancier.

Import en Export

Om mutaties te kunnen identificeren ten behoeve van export naar externe systemen besloot ik om elke tabel te voorzien van een zogeheten ‘interface’ tabel, met de naamconventie <tabelnaam>_IF. Deze tabellen bevatten naast de basistabel kolommen tevens metadata kolommen, zoals een uniek mutatie-nummer voor elke wijziging, de operatiecode I(insert), U(pdate) of D(elete), exportstatus kolommen en een kolom die aangeeft of het mutatierecord de Before (voor update/delete of After image (na update/insert) van een mutatie is.

Aangezien we niet alleen een export-mechanisme, maar tevens een import-mechanisme moesten hebben, gingen we de zogenaamde “IF-tabellen” ook gebruiken voor importrecords. Daartoe werden nog als extra metadata kolommen opgenomen:

- **if_impexp**: om aan te geven of het een I(mport) of E(xport) record betreft
- **id_system**: identifier van het importsysteem; ieder extern systeem rondom CAB werd opgenomen in een systeemtabel en van een uniek nummer voorzien
- **error_code en error_message**: tijdens importeren van records kunnen allerlei fouten optreden waarvoor deze kolommen dienen
- **<primary_key>_Foreign kolommen**: voor een juiste import dienden we ook te weten wat de sleutels waren in het importsysteem. Deze sleutels gingen we tevens, gekoppeld aan de CAB sleutels, vastleggen in een centrale sleutel koppeltabel.

Eén set interfacetabellen was niet voldoende en een tweede set werd aangemaakt, de zogeheten “IF2 tabellen” met weinig verrassende naamconventie <tabelnaam>_IF2.

Waarom? Dat had zowel voor het import als exportmechanisme een reden:

Te *importeren* data in de IF-tabellen waren gevalideerde mutaties en konden met een generieke importprocedure allemaal in CAB verwerkt worden, terwijl mutaties in de IF2 tabellen nog on gevalideerde mutaties waren, welke na validatie (automatisch goedgekeurd volgens de gespecificeerde regels of omdat de cartotheek ze had geaccepteerd) in de IF-tabellen terecht kwamen om tezamen met andere gevalideerde gegevens in CAB verwerkt te worden.

Aan de *export* kant wordt voor elke mutatie, ongeacht welk systeem daar geïnteresseerd in is, een mutatierecord aangemaakt in de “IF-tabellen” (ik noem ze de exportsysteem *onafhankelijke* interfacetabellen). Vervolgens worden ze voor elk exportsysteem dat zich ‘geabonneerd’ heeft op een (sub)set van de gegevens door de voor dat systeem gedefinieerde filters gehaald en naar de IF2 tabellen geschreven (daarmee zijn het systeem *afhankelijke* interfacetabellen). Elk exportsysteem kan daarmee op het voor dat systeem gewenste moment zijn mutaties ophalen zonder andere systemen in de weg te zitten. IF-records die voor geen enkel exportsysteem meer benodigd zijn worden vervolgens verwijderd. Helemaal weg zijn daarmee de mutaties niet: ze zijn daarvoor eerst ook nog naar LOG-tabellen geschreven. Wie het bijhoudt weet inmiddels dat voor die grofweg 15 basistabellen er evenveel IF, IF2 en LOG-tabellen zijn. En dan heb ik de FLAG tabellen nog niet genoemd: de import gebruikt “vlaggetjes” tabellen om te identificeren welke kolommen er daadwerkelijk als gemuteerd worden aangeboden. Hiermee kan ik onder meer bewerkstelligen dat kolommen in CAB, welke door het importsysteem helemaal niet geleverd worden, met rust gelaten worden in CAB: deze worden in de corresponderende FLAG tabel op “Nee” gezet ten teken dat voor die mutatie die kolom als niet gewijzigd

beschouwd mag worden. De hele export en importfunctionaliteit leunt zwaar op het gebruik van DBMS_SQL en metadata (zowel zelf gedefinieerde als data-dictionary metadata).

Het import- en exportsysteem is uitgegroeid tot een vernuftig generiek mechanisme met veel functionaliteit, waarvan slechts een kleine greep is:

Import

- Inserts worden updates als het record al bestaat.
- De sleutels van het importsysteem worden automatisch bewaard na succesvolle invoer in CAB, zodat een volgende keer een insert vanzelf als een update wordt aangeboden.
- Mutaties komende van een bronsysteem gaan niet weer als mutaties naar het exportsysteem als dat systeem hetzelfde is als het bronsysteem.
- Per 'operatiecode' (I/U/D) kunnen kolommen wel of niet in CAB overschreven worden (zo kan bijvoorbeeld een veld tijdens aanmaak van een nieuwe persoon wel vanuit het importsysteem gevuld worden, maar bij updates niet).

Export

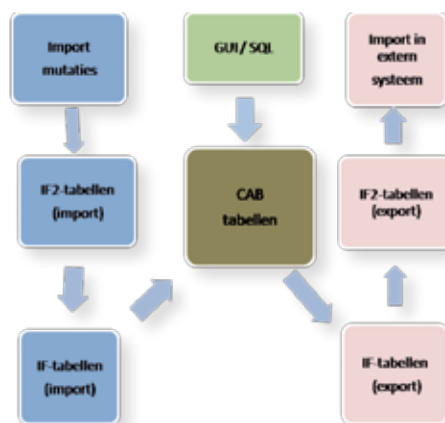
- Het systeem weet per exportsysteem de laatst gestuurde mutatie en stuurt alleen delta mutaties.
- Het is mogelijk om buiten de automatische delta's om manueel gegevens opnieuw aan te bieden aan een exportsysteem.
- Het exportsysteem gaat slim om met meervoudige updates op hetzelfde record tijdens een delta periode (deze worden in de IF2 tabellen teruggebracht tot slechts één Before Image en één After Image record).
- Het systeem maakt van Updates Inserts of Deletes al naar gelang een Before of After Image record niet of wel door de filters komt van het exportsysteem.

Afbeelding 1 (rechts) geeft een zeer eenvoudige weergave van de flow van mutaties van en naar CAB.

Een succes voor CAB

In de organisatie, die vestigingen in bijna 20 landen van Europa heeft,

werden meerdere landen geïnteresseerd in het succesvolle concept dat de vestiging in Nederland had ten aanzien van adressenbeheer. Elk land worstelde met zijn eigen oplossingen en problemen ten aanzien van data leveranciers, import in CRM systemen en slechts enkelen hadden een ver geautomiseerd systeem hiervoor. Vanuit België en de Scandinavische landen (genaamd "De Nordics") ontstond belangstelling voor de Nederlandse oplossing met CAB als centrale 'profieldata' applicatie en onze geautomiseerde export en importmechanismes eromheen. Helemaal omdat alle landen in Europa te maken kregen met de van boven opgelegde centralisatie van het CRM systeem: Siebel 5 en 6 werden de afgelopen jaren bij de landen zelf weggehaald om vervangen te worden door een centrale OLTP en OLAP applicatie (Siebel 7.7). Daarmee werden ook de rechten die elk land op die databases hadden geheel beperkt: rechtstreekse database toegang was niet meer mogelijk en interfaces met deze centrale database moesten gaan lopen via WebMethods. Hiertoe werd een flat file definitie opgesteld waarmee de landen hun wijzigingen in profieldata konden aanbieden aan WebMethods dat de wijzigingen doorvoerde in de centrale OLTP database. Voor ons betekende dat 'slechts' een nieuw "exportsysteem": de flat file van Webmethods. Bijna alle al gereed zijnde functionaliteit van het exportmechanisme kon blijven bestaan, alleen het doelformaat moest gewijzigd



Afbeelding 1. CAB import/export

worden. Omdat België en de Nordics ook hun interfaces zouden moeten aanpassen aan WebMethods en zelf nog niet zover als wij met hun beheer van profiel-

gegevens, werd besloten om CAB en de interface naar WebMethods ook uit te rollen naar deze landen. Een succes voor onze aanpak en CAB! Besloten werd tot twee centrale databases waarin CAB zou draaien: eentje voor Nederland en België (met de database in België) en eentje voor De Nordics (met de database in Zweden). Om de gegevens behorende bij een land in de applicatie te onderscheiden, heb ik CAB uitgerust met Virtual Private Database (VPD) policies: afhankelijk van de ingelogde gebruiker en het land waar deze toe behoort, geeft een 'SELECT * FROM ab_person' alleen Belgische of Nederlandse personen als resultaat (in het geval van de database voor België en Nederland). Een heel krachtig concept.

JHeadstart

De Visual Basic GUI, die door extra functionaliteit en een aantal goede verbeteringen in CAB niet meer zou voldoen, moest plaats gaan maken voor een web based interface. Op dat moment sprongen mijn Motiv collega en ik in een gat dat we al tijden vurig wensten: het maken van een JHeadstart applicatie als opstap naar een kennismaking met J2EE technologie. We hadden enkele jaren daarvoor een hele goede presentatie van Steven Davelaar en Sandra Muller gezien (twee drijvende krachten achter JHeadstart, werkzaam bij Oracle Consulting) en die had zoveel indruk gemaakt dat het JHeadstart moest worden en wel om de volgende redenen:

- Java en J2EE verdrongen steeds duidelijker Oracle Forms (gezien vanuit het perspectief van ons Oracle ontwikkelaars) dus we wilden die overstap ook gaan maken
- JHeadstart maakt het heel gemakkelijk om te werken met ADF Business Components en JDeveloper maakt het maken van je Model op basis van Business Components zeer gemakkelijk. Omdat al onze business logica al in de database wordt afgedwongen, zou het presentatiedeel (de View) samen met het Controllerdeel (dat onder meer de navigatie regelt tussen de verschillende schermen) relatief licht zijn. Dus alle componenten waren met behulp van JHeadstart en JDeveloper relatief gemakkelijk te realiseren

Adressenbeheer met PL/SQL en Jheadstart

Vervolg

- JHeadstart verbergt heel veel complexiteit van J2EE en aangezien we relatief weinig Java en J2EE kennis hadden, was het een redelijk veilige keuze om in de korte tijd die we hadden de GUI te maken

In de vorige We Love IT heeft Steven Davelaar al een uitgebreid artikel geschreven over de functionaliteit van JHeadstart, dus dat zal ik hier niet herhalen. Voor ons was het een zeer leerzaam project. Terwijl we nog nooit met JHeadstart gewerkt hadden, geen J2EE en nauwelijks Java programmeerachtergrond hadden, hebben we binnen de korte gestelde tijd een werkende applicatie afgeleverd met alle gewenste basisfunctionaliteit. En de meeste problemen die we kenden kwamen voort uit het in teamverband programmeren en het gebruik van subversion (een version control system). Niet zozeer subversion zelf was lastig (in tegendeel: subversion heeft zijn werk goed gedaan), maar het feit dat we vanaf elke werkplek aan de applicatie wilden kunnen werken, ook vanuit huis. Een goeie infrastructuur daarvoor ontbrak echter voor ons. Dit heeft ons toch enkele keren negatief verrast. Het programmeren/definiëren in JHeadstart daarentegen verliep relatief vlot. Een minpunt in onze situatie was dat we in het beginstadium moesten werken met een Beta versie van de voor het eerst op ADF Faces gebaseerde JHeadstart versie 10.1.3. De productieve versie van JHeadstart op dat moment was 10.1.2 en was gebaseerd op Struts en UIX. ADF Faces was echter het pad dat we in wilden, maar versie 10.1.3 was weliswaar bijna, maar nog niet geheel productiegereed. Ook dat heeft ons tijdens die eerste maand enkele malen voor voldongen feiten geplaatst. Echter het uitstekende forum van JHeadstart en de snelheid waarmee met name Steven en Sandra reageren op problemen en vragen hielp ons steeds snel vooruit. En door het zeer open karakter van JHeadstart met zijn template driven

generator en de mogelijkheid om delen van de JHeadstart classes te subclassen konden we zelfs bug fixes voor zijn door een workaround in een eigen subclass te programmeren. Op deze wijze kon ik een zelfs een enorm performance-probleem verhelpen.

Om de logica binnen een JHeadstart/ADF applicatie zo simpel mogelijk te houden is af en toe slim gebruik van Oracle functionaliteit in de database wenselijk. We hebben met behulp van Views met Instead of Triggers een aantal lastige problemen weten te omzeilen. We hebben overigens heel erg veel gehad aan de gevolgd ADF/JHeadstart workshop van Oracle: hierdoor hadden we ook daadwerkelijk een "(J)headstart".

In *afbeelding 2* ziet u een voorbeeld van een detailscherm voor de invoer/mutaties van personen in CAB. JHeadstart geeft je een zeer rijke user interface toolbox, waarvan wij maar een beperkt deel gebruikt hebben omdat dat voor de applicatie afdoende was. Out of the box navigatie, scrollen op detailniveau, simpele en geavanceerde zoekschermen, tabs, regions, dynamische sorteringen en hele eenvoudige List of Values definitie maken het een omgeving die ik vrijwel even productief vind als Oracle Forms en wellicht zelfs productiever. Zeker voor de doorgewinterde JHeadstart gebruiker. We hadden binnen twee dagen een heel aardig prototype met behoorlijk wat werkende functionaliteit.

Tot slot

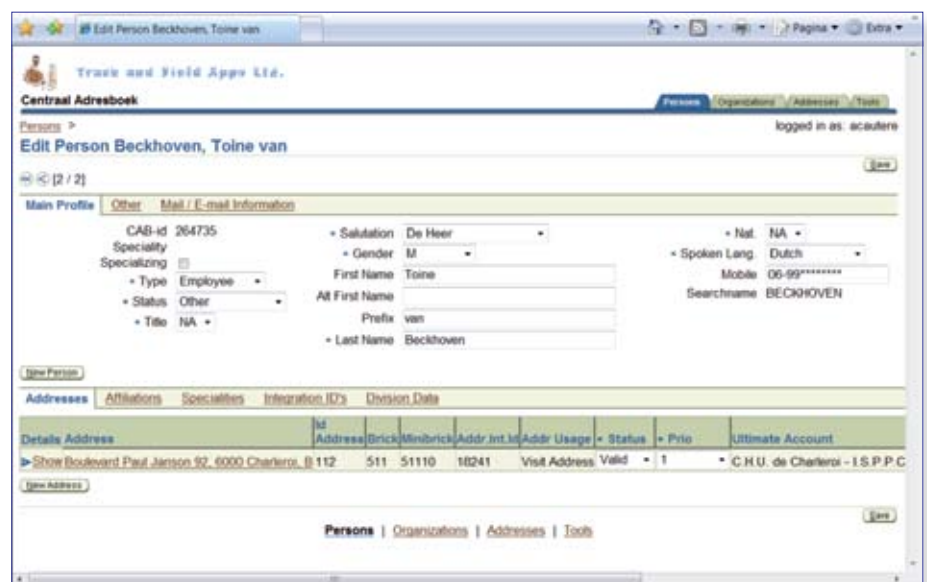
CAB is in beginsel simpel, maar in zijn totaliteit een zeer uitgebreid systeem geworden dat het beheer van profielgegevens van doelgroepen heeft verrijkt voor de klant waar ik vanuit Motiv zat en nog altijd deels zit. We hebben er veel creativiteit in kunnen stoppen en het heeft me heel veel van Oracle geleerd, zowel van de database als zijn ontwikkelomgevingen. Nu nog eens zien of er een tijd komt om hem wat te moderniseren, want de applicatie bevat een aantal eigenschappen waar heden ten dage wellicht betere (standaard)oplossingen voor zijn, bijvoorbeeld:

- CDC/Streams voor het opvangen van mutaties om naar andere systemen te zenden
- Bulk Collect/ForAll in PL/SQL. Maar vooral ook Oracle versie 10 en 11: CAB draait momenteel in alle landen nog altijd in Oracle 8.1.7 databases!
- JHeadstart is inmiddels ook aan nieuwere versies toe met prachtige nieuwe features. Op moment van schrijven migreren we naar versie 10.1.3.2

Toine van Beckhoven

(<http://www.jom-it.nl/>)

is zelfstandig en allround Oracle specialist. Oracle Performance Tuning en PL/SQL hebben zijn grootste interesse.



Afbeelding 2. Een detailscherm van CAB, gegenereerd met JHeadstart