

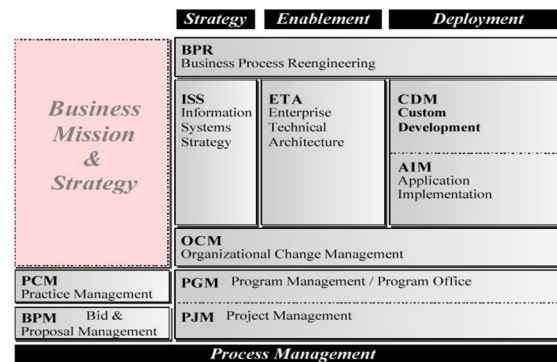
CDM Advantage 2.0

De nieuwe generatie is compleet

De laatste jaren hebben we amper de tijd gehad om op adem te komen tussen de in rap tempo verschijnende releases van zowel Oracle Designer als Oracle Developer. De methode om deze gereedschappen efficiënt in te zetten voor het ontwikkelen van maatwerkapplicaties, Custom Development Method (CDM), bleef echter ongewijzigd. Eindelijk, na zo'n vier jaar, hebben we een nieuwe release van CDM: CDM Advantage 2.0. Volledig in lijn gebracht met de nieuwe generatie Designer en Developer, en zelfs met verwijzingen naar de zeer actuele Oracle Designer 6i. Het meest revolutionaire is echter de volledig nieuwe benadering van het implementeren van business rules m.b.v. het zogenaamde 'CDM RuleFrame'. Met behulp van de bijbehorende Headstart versie wordt business rule modeling vanaf requirements definition tot production volledig afgedekt met transformers en generatoren. En het allerbeste van het RuleFrame: eindelijk zijn we van het mutating table probleem verlost.

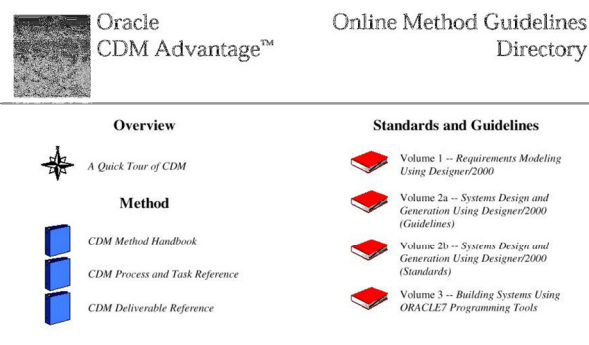
CDM1

Om een duidelijk beeld te krijgen van CDM2 is het verstandig het geheugen nog even op te frissen over de huidige CDM1. Onder de verzamelnaam Oracle Method heeft Oracle in de loop der jaren een raamwerk ontwikkeld van methoden die de lifecycle afdekken van Enterprise Information Management (zie afbeelding 1). Zo is er bijvoorbeeld een methode voor het bepalen van de juiste technische architectuur (Enterprise Architecture Method), voor het implementeren van Oracle Applications (Application Implementation Method) en voor projectmanagement (Project management Method). Ook CDM is een telg van de Oracle Method familie.



Afbeelding 1 Oracle Method. Bron: Oracle

Als we CDM in meer detail bekijken (afbeelding 2) zien we dat deze in grote lijnen is opgebouwd uit twee delen: aan de ene kant vinden we beschrijvingen van verschillende approaches voor het doen van projecten (classic, fast track, lite; zeg maar de watervalmethode, RAD en prototyping) en van de bijbehorende processes, tasks en deliverables. Aan de andere kant vinden we de zogenaamde standards en guidelines libraries (S&GL). Deze libraries zijn hoofdzakelijk gericht op het gebruik van Oracle Designer en verwante producten, de andere documenten zijn ook toepasbaar in maatwerk software ontwikkelingsprojecten waar niet-Oracle producten worden toegepast.



Afbeelding 2 CDM 1.x. Bron: Oracle

Als we kijken naar het verband tussen de twee delen van CDM (approach versus S&GL), dan zien we dat de libraries in feite een erg gedetailleerde beschrijving vormen van het uitvoeren van een project volgens de fast track approach. Bij met behulp van CDM uitgevoerde projecten was de fast track approach dan ook de meest gehanteerde projectaanpak.

Om de drempel zo laag mogelijk te maken om de adviezen die in CDM1 werden gegeven zoveel mogelijk in de praktijk te brengen, heeft de Consultancy afdeling van Oracle Nederland het Headstart Template Package ontwikkeld. Door dit Template Package te gebruiken bij de softwareontwikkeling wordt het naleven van CDM erg gemakkelijk gemaakt. Met behulp van de templates kunnen de GUI richtlijnen eenvoudig worden gevolgd, met behulp van de utilities kunnen de standaarden snel en doeltreffend worden gecontroleerd ('Quality Checks') en kan programmacode worden gegenereerd in de repository waar de standaard functionaliteit van Designer iets te wensen over laat ('Performance Boosters'). In feite is Headstart een out-of-the-box implementatie van al het goede dat CDM te bieden heeft.

Een belangrijk onderdeel van CDM is het Business Rule Modeling. Dit is een theoretische aanpak om tijdens het functioneel ontwerp Business Rules te onderkennen en in Designer te documenteren. Het Designerobject dat hiervoor ter documentatie gebruikt (of misbruikt, zo je wilt) wordt, is de Business Function. Hierin wordt tekstueel vastgelegd welke regel de business rule beschrijft.

Vervolgens wordt de CRUD van de business function gevuld overeenkomstig het entiteitgebruik van de business rule. In de optimale situatie wordt ook nog het Event-object van Designer gebruikt om vast te leggen bij welke gebeurtenis de business rule gevalideerd moet worden. Op deze wijze worden de business rules separaat van het ERD en de 'normale' business functions gemodelleerd. Het grote voordeel hiervan is dat tijdens System Design en Generation er even gemakkelijk voor kan worden gekozen de business rules met behulp van clientlogica als met serverlogica te implementeren. Waar de logica waarmee de rules vervolgens worden geïmplementeerd zich precies bevindt is een heel ander verhaal. Ervaren ontwikkelaars zien het probleem waarschijnlijk niet eens meer, maar het onderhouden van volgens CDM ontwikkelde informatiesystemen kan voor minder ervaren personen behoorlijk complex zijn. Business rules kunnen op de server geïmplementeerd worden in check constraints, database triggers, stored procedures, stored functions en packages, in de client kunnen ze worden geïmplementeerd in (COMPLEX) check constraints, en module libraries. En dan hebben we het alleen nog maar over business rules waarvoor geen specifieke property in Oracle Designer is te vinden, dus de non-native vast te leggen rules.

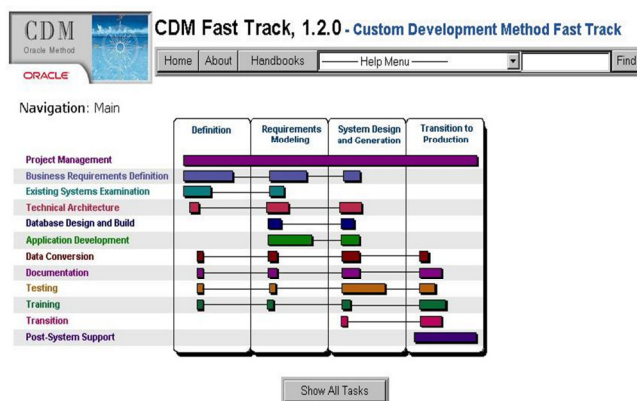
Behalve de onderhoudbaarheid heeft de business rule implementatie volgens CDM1 nog meer nadelen: allereerst kan niet elke rule in de server worden geïmplementeerd. Het is bijvoorbeeld onmogelijk een verplichte master-detail te implementeren in de server. Immers, zodra de mastergegevens door de client naar de server zijn verzonden wordt door de logica in de databasetriggers van de master geconcludeerd dat er (nog) geen details aanwezig zijn. Het belangrijkste nadeel is echter wel het kunnen optreden van het mutating-table probleem in triggers. In een rowlevel databasetrigger mag geen select worden gedaan op de onderliggende tabel omdat als deze trigger afgaat de tabel 'muterende' is, en er dus geen eenduidige definitie bestaat van de kolomwaarden van de tabel. Om dit probleem te omzeilen moet de gewenste select worden uitgesteld tot een statementlevel trigger. Hier is de select wel toegestaan omdat bij het voltooien van het statement de tabel weer stabiel is. De

programmacode die komt kijken bij het uitstellen van de gewenste acties van de ene trigger naar de andere beslaat al gauw enkele A4-tjes. Hoewel Headstart een performance booster bevat voor het genereren van deze code, kunnen we niet spreken van een erg optimale situatie.

CDM2

Nu we CDM1 weer scherp voor de geest hebben, is het tijd om naar CDM2 te kijken. Allereerst is voor wat betreft de naamgeving een opmerking op z'n plaats. De marketingmensen van Oracle hebben met de introductie van release 6 voor de verschillende clientproducten gepoogd een wat uniformer geheel te scheppen. Omdat Oracle Forms toe was aan release 6 zijn alle gerelateerde producten meegelift: Designer ging van 2.1 naar 6.0, Reports ging van 3.0 naar 6.0, en Forms en Reports gingen gebundeld van 1.6 naar 6.0. Het ligt dan ook voor de hand om de nieuwe CDM en de nieuwe Headstart eveneens te liften naar 6.0. Maar als we de jongste releases van deze producten bekijken dan treffen we weer een oerwoud van verschillende versienummers aan. Voor het gemak zullen we in dit artikel zoveel mogelijk slechts over CDM2 en Headstart6 spreken.

Het eerste verschil tussen CDM1 en CDM2 blijkt al direct als we CDM2 opstarten. Mensen die ervaring hebben in de wereld van de component based development krijgen dan direct een déjà vu. De HTML look-and-feel van CDM vertoont toch wel erg veel overeenkomst met het in de CBD wereld erg populaire Rational Unified Process. In afbeelding 3 is een weergave te zien van de nieuwe CDM interface. Het getoonde schema geeft niet alleen informatie over de projectfasering, maar door op een willekeurige plaats te klikken met de muis wordt naar een beschrijving genavigeerd van het geselecteerde item. Dit inzoomen kan vervolgens worden herhaald tot het diepste niveau, alwaar Word wordt opgestart met een macro waarmee een prototype van de betreffende deliverable wordt gegenereerd. Dit mechanisme is de evolutie van CDM Advantage.



Afbeelding 3 CDM 2.0, de nieuwe HTML look-and-feel

De term 'opstarten van CDM2' is iets bezijden de waarheid. Met ingang van release 6 bestaat CDM uit twee verschillende 'richtingen'. Er bestaan zelfs twee setups voor de installatie. CDM2 bestaat uit de onderdelen CDM Classic 2.6 en CDM Fast Track 1.2 en de gemeenschappelijke Standards en Guidelines Library 6.0. Stonden in CDM1 de verschillende approaches nog als aparte hoofdstukken genoemd in de respectievelijke documenten, met ingang van CDM2 is er dusdanig veel meer beschreven over elk van de approaches, dat specifieke documenten op hun plaats waren. De voorheen bestaande Lite approach was kennelijk dusdanig straightforward, dat uitgebreide toelichting hiervan niet meer belangrijk werd gevonden. In overzicht 4a en 4b is aangegeven waaruit de respectievelijke onderdelen van CDM bestaan.

CDM Quick Tour, 2.0.0
 CDM Classic Method Handbook, 2.6.0
 CDM Classic Process and Task Reference, 2.6.0
 CDM Classic Deliverable Templates, 2.6.0
 including the following customized PJM
 deliverable templates:
 WM.020 Workplan Templates
 CDM Standards and Guidelines Library
 Volume 1 - Requirements Modeling using
 Oracle Designer, 6.0.0
 Volume 2 - Design and Generation of Multi-Tier
 Web Applications, 6.0.0
 Volume 3 - Building Systems using Oracle8
 Programming Tools, 6.0.0
 Volume 4 - CDM Standards, 6.0.0
 PJM Add-in, 2.6.1:
 PJM Method Handbook, 2.6.0
 PJM Process and Task Reference, 2.6.0
 PJM Deliverable Templates, 2.6.0

Overzicht 4a CDM 2.0: Classic 2.6. Bron: Oracle

CDM Quick Tour, 2.0.0
 CDM Fast Track Method Handbook, 1.2.0
 CDM Fast Track Process and Task Reference, 1.2.0
 CDM Fast Track Techniques Handbook, 1.2.0
 CDM Fast Track Addendum, 1.2.0
 CDM Fast Track Deliverable Templates, 1.2.0
 including the following customized PJM
 deliverable templates:
 CR.010 Project Management Plan
 CR.070 Status Monitoring and Reporting
 QM.010 Quality Management Procedures
 WM.020 Workplan Templates
 CDM Standards and Guidelines Library
 Volume 1 - Requirements Modeling using
 Oracle Designer, 6.0.0
 Volume 2 - Design and Generation of Multi-Tier
 Web Applications, 6.0.0
 Volume 3 - Building Systems using Oracle8
 Programming Tools, 6.0.0
 Volume 4 - CDM Standards, 6.0.0
 PJM Add-in, 2.6.1:
 PJM Method Handbook, 2.6.0
 PJM Process and Task Reference, 2.6.0
 PJM Deliverable Templates, 2.6.0

Overzicht 4b CDM 2.0: FastTrack 1.2. Bron: Oracle

CDM Fast Track

Vergeleken met de fast track approach van CDM1 is de nieuwe fast track zeker niet 'meer van hetzelfde'. Als we eerlijk zijn was de vroegere fast track (hoewel zoals gezegd de meest gebruikte aanpak) niets meer dan een uitgekleden classic approach. De RAD methode die zo'n vijf jaar geleden erg populair begon te worden, werd door Oracle met de fast track approach vooral geïnterpreteerd als 'minder belangrijke taken en deliverables niet uitvoeren c.q. opleveren'. Inmiddels is veel meer kennis over de RAD methode opgebouwd door verschillende partijen in de markt. De bekendste partij die hier uitgebreid werk van heeft gemaakt is het DSDM consortium, www.dsdm.org. Deze onafhankelijke groep heeft de Dynamic Systems Development Method ontwikkeld, de de-facto onafhankelijke wereldstandaard voor Rapid Application Development. Om de fast track approach daadwerkelijk een RAD methode te laten zijn, is voor CDM2 het deel over de fast track approach volledig herschreven volgens de principes van DSDM. Dit betekent bijvoorbeeld dat er reeds tijdens de requirements definitie een functioneel prototype wordt ontwikkeld. Verder wordt de aanpak gekenmerkt door incrementeel en iteratief ontwikkelen, timeboxing, en het prioriseren met behulp van de MoSCoW-list, een lijst met Must have's, Should have's, Could have's en Would have's. Hierdoor is de fast track aanpak niet langer een kwestie van selectiever taken uitvoeren, maar de meeste taken een aantal keren uitvoeren, steeds gedetailleerder, met steeds concretere resultaten. De skills die de teamleden maar zeker ook de projectleider nodig hebben voor deze methode zijn niet per definitie gelijk aan die voor de 'oude' fast track. De kennis die nodig is om de benodigde vaardigheden onder de knie te krijgen staat beschreven in de nieuwe manual 'CDM Fast Track Techniques Manual'.

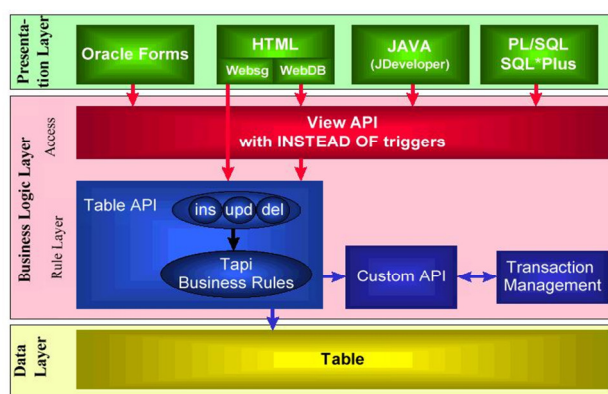
CDM RuleFrame

Van alle wijzigingen die CDM heeft ondergaan is het meest spraakmakende toch wel de toevoeging van het CDM RuleFrame. Het RuleFrame is vergelijkbaar met frameworks die in de CBD wereld gebruikelijk zijn. Zo is er voor de Java ontwikkelomgeving het componentenmodel Enterprise Java Beans bedacht. Bij dit framework wordt vaak als voordeel genoemd dat het 'low-level plumbing' (het basale loodgieterswerk) niet meer door de ontwikkelaar hoeft te worden gedaan, maar dat

dit door het framework wordt afgehandeld. Iets dergelijks kan gezegd worden voor het CDM RuleFrame. Bij gebruikmaking van het RuleFrame hoeft eigenlijk geen programmacode meer handmatig gecodeerd te worden die regelt dat een business rule op het juiste moment gevalideerd wordt. Deze code wordt automatisch gegenereerd door Headstart Utilities op basis van het functioneel en technisch ontwerp van de Business Rules. Wat resteert is de code voor validatie van de Business Rule zelf. Dus niet wannéer de rule afgaat, maar wat hij dan moet validéren. En zelfs hierbij ondersteunt het RuleFrame. Het grootste deel van de code voor validatie bestaat normaalgesproken uit een cursordefinitie en een bijbehorend fetchstatement. Met Headstart kan voor elke tabel een zogenaamde CAPI (zie verder) met een aantal generieke routines worden gegenereerd die, als ze aangeropen worden, een locale cursordefinitie overbodig maken.

Maar om het RuleFrame goed op z'n waarde te schatten moeten we kijken naar het totaalplaatje dat in CDM2 wordt geschetst van het implementeren van business rules. Allereerst propageert CDM voor het eerst sinds haar bestaan een pure implementatie van het drie-lagen model. Nu claimt zo ongeveer elk softwarebedrijf het drielagenmodel te ondersteunen, maar Oracle maakt het met CDM2 daadwerkelijk waar: de ontwikkelmethodiek beoogt te leiden tot een separate presentatielaag, een business logic layer (de geïmplementeerde business rules) en een data layer. Met andere woorden: we kunnen de presentatielaag even gemakkelijk implementeren met Visual Basic als met WebDB, of met Oracle Forms, de business logic laag implementeren we met een Oracle database met behulp van het RuleFrame, en de data laag zou net zo gemakkelijk met Microsoft Access als met Oracle geïmplementeerd moeten kunnen worden. Uiteraard zullen de lagen in de praktijk hoofdzakelijk met Oracle producten worden geïmplementeerd, maar omdat elke laag zijn eigen verantwoordelijkheid heeft en daartussen in principe geen overlap is, kan voor elke laag theoretisch het meest geschikte product worden ingezet. Op zich lijkt dit niet zo bijzonder, maar om het drielagen model in z'n pure vorm te ondersteunen, moet er een behoorlijke concessie worden gedaan aan het gebruik van de standaard functionaliteit van het Oracle RDBMS. Immers, de meest gebruikte vehicels voor event-driven business logica in het Oracle RDBMS zijn de check constraint en de database trigger. Beide

kunnen echter uitsluitend voorkomen als 'aanhangel' van een tabel. En dat zou betekenen dat er tóch business logica in de datalaag zou komen. CDM2 adviseert dan ook geen gebruik meer te maken van check constraints en database triggers. Na vijf jaren niets anders te hebben aanbevolen dan deze databaseobjecten juist zo veel mogelijk te gebruiken, moet er wel een heel goed verhaal zitten achter deze 180° koerswijziging. Gelukkig is dat ook zo. Om de achtergrond hiervan te begrijpen moeten we ons verdiepen in de totale opzet van het CDM RuleFrame, zoals geïllustreerd op afbeelding 5. Voor de volledigheid zij opgemerkt dat de navolgende toelichting op de werking van het RuleFrame bewust zeer globaal is gehouden. In CDM (S&GL vol. 2 hfdst. 6 t/m 8 en de RuleFrame User Guide) zijn hier ruim 250 pagina's aan gewijd.



Afbeelding 5 CDM RuleFrame. Bron: Oracle

Als we de opbouw (afbeelding 5) bekijken van het RuleFrame, dan zien we allereerst de indeling in de reeds genoemde drie lagen. De presentatielaag en de datalaag spreken voor zich. Zoomen we verder in op de business logic laag, dan treffen we daar een viertal hoofdcomponenten aan. De meeste namen die voor deze onderdelen zijn bedacht zijn modieus afgeleid van het buzzword API. Achtereenvolgens zien we de VAPI (View API), de TAPI (Table API), de CAPI (Custom API) en het TM (Transaction Management).

TAPI

De TAPI is een oude bekende die we reeds vanaf Designer 2.1 kennen. Dit is een package met business logica die met standaard Designer-functionaliteit kan worden gegenereerd. Voorbeelden hiervan zijn denormalisatie en het vullen van auditingkolommen. Deze business logica zit verweven in TAPI-procedures (ins, upd, del) die aangeroepen kunnen worden om DML uit te voeren op tabellen in de data laag. Voorwaarde voor de business logica om geëffectueerd te worden is dus wel dat deze TAPI-procedures aangesproken worden en er niet direct met DML statements, buiten de TAPI om, op de tabellen wordt gewerkt. Gegeneerde programma's voor de PL/SQL cartridge van de Oracle Application Server, de zogenaamde WebServer Generator modules (Websg), doen dit keurig. Voor alle andere frontends moet er een voorziening worden getroffen om het omzeilen van de TAPI te voorkomen. Daarom bestaat de mogelijkheid met Designer TAPI-triggers te genereren. Hierdoor wordt de TAPI altijd aangesproken bij DML op de onderliggende tabellen. De keerzijde hiervan is echter dat er wel weer logica wordt ondergebracht in de data laag. Om te voldoen aan een pure implementatie van het drielagenmodel, dient van een andere component van het CDM RuleFrame gebruik te worden gemaakt, de VAPI.

VAPI

De VAPI bestaat uit een view die in z'n eenvoudigste vorm qua structuur één op één is met de onderliggende tabel. Op deze view worden INSTEAD OF-triggers gedefinieerd. Met behulp van dit feature (op dit moment alleen terug te vinden in de Enterprise Edition van Oracle8) leidt DML op een view tot het afgaan van de overeenkomstige trigger. In deze trigger kan zoals in elke database trigger eigen logica gecodeerd worden. In het geval van de VAPI zorgen de INSTEAD OF-triggers er voor dat de DML die op de view wordt uitgevoerd, wordt omgezet in een aanroep van de juiste TAPI procedure. Door de naam van de VAPI gelijk te laten zijn aan die van de onderliggende tabel, is het aanspreken van de business logica voor de frontend dus volledig transparant geworden.

CAPI

De CAPI is een package (één per tabel) waarin alle business logica zit die niet standaard door

Designer kan worden gegenereerd. Alles wat dus volgens CDM1 met check constraints en database triggers werd opgelost, wordt nu gecodeerd in de CAPI. De CAPI's worden met behulp van Headstart Utilities gegenereerd. De utility die de CAPI aanmaakt zorgt er ook voor dat er in Designer 'Table API/Trigger Logic' wordt toegevoegd. Hierdoor wordt automatisch de bijbehorende CAPI aangeroepen, zodra een TAPI-procedure wordt uitgevoerd.

TM

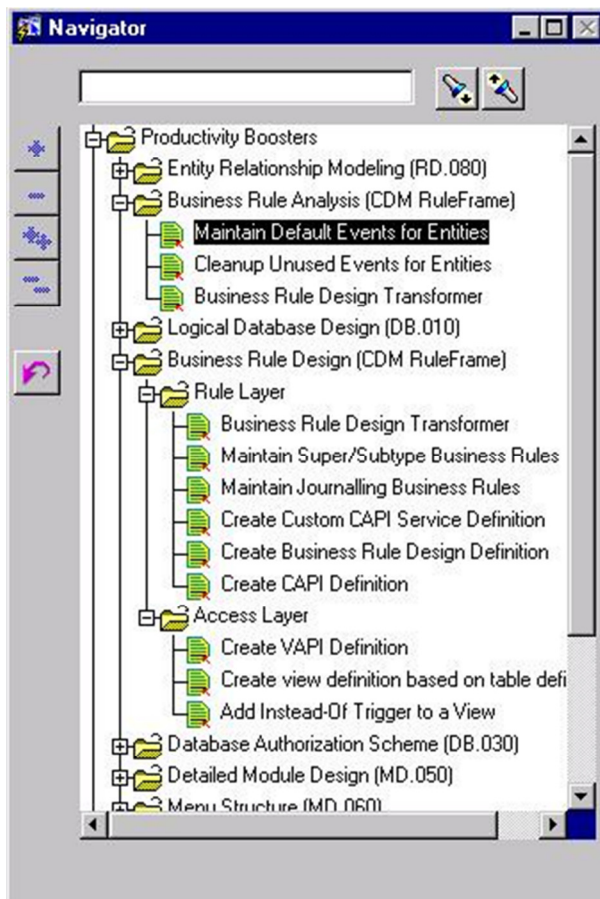
Het laatste onderdeel van de business logic laag is het Transaction Management. Voor de wat grijzere ontwikkelaars is dit een bekend fenomeen. Zo'n twintig jaar geleden was er in elke ontwikkelomgeving een TP (Transaction Processing) -monitor te vinden, met wellicht als meest bekende CICS in de Cobol omgeving. De TM module uit het RuleFrame heeft een vergelijkbare taak. Voortaan moet een frontend zodra er aan een transactie wordt begonnen, dat doorgeven aan de TM-module. Vanaf dat moment wordt elke businessrule die gevalideerd moet worden op een stack gezet, en niet direct uitgevoerd. Pas zodra de frontend heeft doorgegeven dat de transactie kan worden afgesloten, worden alle rules uitgevoerd. De voordelen zijn legio. Voor de eindgebruiker betekent dit bijvoorbeeld dat deze in één keer alle foutmeldingen te zien krijgt waaraan niet wordt voldaan. Dit in tegenstelling tot de denkbare situatie waarbij een invoerfout wordt gecorrigeerd en het opnieuw committen de volgende foutmelding oplevert, eventueel meerdere malen achtereen. De grote winst is echter gelegen in het feit dat de rules pas gevalideerd worden nadat de DML op de tabellen heeft plaatsgevonden. Eindelijk, na vijf jaar behelpen, zijn we verlost van het spook van de mutating table.

Headstart

De concepten van het CDM RuleFrame zouden lang zo waardevol niet zijn, als ze niet in hoge mate ondersteund zouden worden door Headstart (zie afbeelding 6). Als tijdens requirements modeling de business rules nauwkeurig volgens de standaarden worden vastgelegd (vrijwel ongewijzigd volgens de aanpak van Business Rule Modeling zoals beschreven in CDM1), dan kunnen deze door Headstart Utilities worden getransformeerd naar zogenaamde Business Rule Definitions in de System Design and Generation fase. In deze fase is vrijwel de enige handmatige actie

op het gebied van business rule implementatie, het in Designer toevoegen van de regels code die de constraint daadwerkelijk valideren. Zoals reeds eerder is opgemerkt kan hierbij gebruik worden gemaakt van standaard routines in de CAPI. De definitie van deze CAPI kan met Headstart worden aangemaakt. Bij het genereren bundelt deze utility alle business rule definitions die van toepassing zijn op een bepaalde tabel, vertaalt die naar procedures en functions, voegt daar standaard functions en procedures aan toe en schrijft dat alles weg als één package.

Naast de utilities die we gebruiken om het Business Rule Model uiteindelijk om te zetten in een CAPI, bevat Headstart ook utilities voor het maken van VAPI definities. Zo kunnen we in één keer een volledig functionerende VAPI genereren, of een bestaande view uitbreiden met INSTEAD OF-triggers.



Afbeelding 6 Ondersteuning CDM RuleFrame in Headstart Utilities 6.0

Rozengeur en maneschijn?

Bij al het goede dat het CDM RuleFrame en de ondersteuning door Headstart te bieden heeft, zijn er toch ook wat kanttekeningen te plaatsen. Allereerst is het zo dat de trend om zo veel mogelijk van de functionaliteit van de Headstart Utilities op te nemen in Oracle Designer, steeds verder doorzet. Zo is er een groot aantal utilities overbodig geworden en daarmee in release 6.0 ook verdwenen. Hierbij is men echter te ver doorgeschooten. Zo dienen bijvoorbeeld de controles om toegestane waarden van statische domeinen te valideren weer handmatig gecodeerd te worden. Vroeger was er een utility die hiervoor checkconstraints genereerde, maar deze is verwijderd. Waarschijnlijk omdat code voor het valideren van dynamische domeinen wel standaard gegenereerd wordt, in de TAPI. Wellicht is er voor het publiceren van dit artikel al weer een patch op Headstart uitgebracht die deze omissie corrigeert.

In CDM wordt geadviseerd om in applicaties waar Oracle Forms als frontend wordt gebruikt geen VAPI's toe te passen. In plaats daarvan zouden bij voorkeur TAPI triggers gegenereerd moeten worden als mechanisme om de TAPI aan te spreken. De woorden 'bij voorkeur' zijn in dit geval echter een behoorlijk eufemisme. De VAPI in combinatie met Forms werkt überhaupt niet¹.

Naast allerlei kleine en soms wat grotere ergernissen is er één aspect waaraan bij gebruik van het CDM RuleFrame zeker aandacht moet worden besteed. Als in de applicatie die wordt ontwikkeld enkele tabellen een prominente rol spelen, dan leidt dat er al gauw toe dat er een aanzienlijke hoeveelheid business rules voor deze tabellen zal zijn gespecificeerd. Zoals we inmiddels hebben kunnen leren heeft dat tot gevolg dat de uiteindelijk gegenereerde CAPI voor deze tabellen erg groot wordt. En juist dat gegeven kan een probleem veroorzaken. Niet zozeer in Designer, maar bij deployment. De wat oudere databases (tot en met de eerste versies van Oracle8i) zijn nog niet goed toegerust om PL/SQL packages van grote omvang te kunnen compileren. In dat geval is het een alternatief om de Headstart Utility die de CAPI genereert aan te passen zodat deze de CAPI bij generatie opsplijst in verschillende packages. Echter, omdat onderdelen van de sub-CAPI's elkaar overal aanroepen, moeten de namen van de sub-CAPI's eveneens in de gegenereerde code worden opgenomen. Om deze complexiteit, en het überhaupt aan

moeten passen van standaard Utilities te vermijden, kan het een beter alternatief zijn een recente versie van Oracle8i te kiezen als platform waarop de CAPI (en daarmee de hele applicatie) in gebruik wordt genomen².

Resumerend

Met het beschikbaar komen van de nieuwe generatie Oracle ontwikkelmethodiek en tools hebben we een bijzonder goed doordacht, krachtig middel in handen gekregen om met relatief weinig handmatig geprogrammeerde code, goed onderhoudbare applicaties te ontwikkelen. De manier van het implementeren van Business Rules is echter behoorlijk gewijzigd. Zelfs voor ervaren ontwikkelaars is het een aanzienlijk leerproces om het CDM RuleFrame goed toe te passen.

Een project definiëren om de kennis, nodig voor het effectief gebruiken van CDM en het CDM RuleFrame, in te bedden in de ontwikkelorganisatie, is dan ook geen overbodige luxe.

¹ Er wordt code gegenereerd om bij inserts de autogenerated fields (auditing e.d.) die in de server zijn gevuld of gewijzigd, in de POST-INSERT trigger van het form uit de database te selecteren. Dit gaat op basis van rowid. De rowid van de VAPI is op dat moment in het form echter nog niet bekend.

² Bij het fysiek aanmaken van de CAPI in de database kan een te grote package leiden tot de melding 'PLS-00123 Program too large'. De juiste databaseversie biedt hier uitkomst. Tot versie 7.3 is de beperkende buffer van het RDBMS 16 kilobytes groot, van versie 7.3 tot 8.1.3 is deze 32kb en recentere versies hebben een ogenschijnlijk oneindige buffer van 64 Mb. Wees hierbij alert dat versie 8.1.5 op NT (serverbug 966970) niet wordt ingezet.

Kijk voor meer publicaties op <http://www.anewlink.nl/ict/nl/publicaties/>.

(c) Copyright 2000 A New Link bv