What Do You See When You Get?

Presentatie-aspecten in het gegevensdomein

Frido van Orden

Het behandelen van het onderwerp 'presentatie' in een artikelserie over functionaliteit van het dbms doet bij een groot aantal lezers ongetwijfeld de wenkbrauwen fronsen. Toch roeren we dit onderwerp aan, omdat ook dit heeft te maken met de vertaling van regels in gedrag. Voor wie hogere eisen stelt dan aan de discipline van de gebruiker...

Ervaren rotten zullen wellicht grappen dat het enige wat zij in hun carrière aan presentatiezaken in de nabijheid van het dbms hebben aangetroffen het Cobol-'PICTURE'-element is, dat echter evenveel met presentatie heeft te maken als IDMS met streaming video.

De architectuurfreaks zullen een vies gezicht trekken en roepen dat het combineren alleen al van de termen 'dbms' en 'presentatie' een flagrante schending is van het principe van *model-view-controller* (MVC), waarin datastructuur, presentatie en logica juist expliciet worden gescheiden. Zij zullen daarin worden gesteund door dbms-leveranciers, die daarbij voor het gemak vergeten dat juist zij de afgelopen jaren hun producten hebben opgetuigd met zo ongeveer alles wat op grond van datzelfde MVC verboden is. Waarom dan toch het onderwerp presentatie aangeroerd?

VAN REGELS NAAR GEDRAG

Centrale gedachte in de afgelopen artikelen in deze serie is het vertalen van regels in gedrag. Een systeem dwingt regels af; als die veranderen, moet het gedrag van het systeem navenant worden aangepast, vanuit het perspectief van de gebruiker. Enkele voorbeelden van dergelijke vertalingen van regels in gedrag zijn weergegeven in tabel 1.

Het bewerkstelligen van het correcte gedrag overeenkomend met een regel kunnen we globaal op twee manieren bewerkstelligen:

- voor elk presentatie-element (veelal een scherm) wordt handmatig bepaald welke regels relevant zijn. Vervolgens wordt het corresponderende gedrag voor deze regels opgenomen als ontwerpspecificatie voor het presentatie-element en geprogrammeerd;
- voor elk presentatie-element wordt de relatie gelegd met een element uit het gegevensmodel. Aan de hand van de regels die gelden op dit element wordt automatisch het gedrag bepaald dat het presentatie-element moet vertonen. Bij het specificeren en ontwikkelen behoeven slechts de autonome functionele behoeften aandacht. Het afdwingen van regels gebeurt 'automatisch'.

CASUS

Het verschil tussen de zojuist geschetste wijzen om regels in gedrag te vertalen wordt al heel snel duidelijk als we een eenvoudige regel eens nader onder de loep nemen: 'Het attribuut Salaris van de tabel Medewerker mag alleen worden gemuteerd door medewerkers van de afdeling P&O'. We richten ons daarbij voor de eenvoud alleen op schermen. In de 'handmatige' aanpak vereist de implementatie van deze regel de volgende handelingen:
identificeer welke schermen medewerkergegevens muteren;

- identificeer door welke (groepen van) gebruikers deze schermen kunnen worden aangeroepen;
- indien er schermen zijn die alleen door niet-P&O'ers worden gebruikt: ontwerp en implementeer dat het schermveld 'salaris' niet te wijzigen is;

Regel	(Mogelijk) Gedrag
Attribuut A in tabel T mag niet worden gemuteerd	Veld V in scherm S is read-only
Attribuut A in tabel T mag de waarde 'W1','W2' of 'W3' bevatten	Veld V in scherm S manifesteert zich als een keuzelijst
	Bij het verlaten van veld V wordt gecontroleerd of de ingevoerde waarde 'W1', 'W2' of 'W3' is. Bij schending volgt een foutdialoog
Records in tabel T mogen niet worden verwijderd door gebruiker G	Button 'Voeg nieuw record toe' in scherm S is disabled

Tabel 1: Vertalingen van regels in gedrag.

- indien er schermen zijn die zowel door P&O'ers als niet-P&O'ers worden gebruikt: splits dit scherm in twee verschillende schermen, verschillend in de mogelijkheid dat er al dan niet salarisgegevens in kunnen worden gemuteerd.

Een complex en foutgevoelig proces, dat ook nog eens door diverse omstandigheden kan worden getriggerd, namelijk wijziging:

- van functionele eisen (bijvoorbeeld: het toevoegen van de betreffende regel);
- van schermspecificaties (bijvoorbeeld: in een scherm dat vroeger alleen medewerkergegevens toonde, mag nu ook worden gemuteerd);
- in rechten van gebruikersgroepen (bijvoorbeeld: ook direct leidinggevenden van een medewerker mogen zijn of haar salaris muteren).

Voorwaar, alle kenmerken van een nachtmerriescenario. Weinig organisaties zullen dit soort complexiteiten in hun traject van systeemontwikkeling en -onderhoud accepteren. Als ontwijkscenario wordt, in dit voorbeeld, veelal gekozen voor het simpelweg afdichten van alle autorisatie-issues met recht toe, recht aan functionele autorisatie: scherm S is alleen toegankelijk voor gebruik G1, G2 en G3. Probeert iemand anders het scherm te starten, dan valt hem een foutmelding ten deel. Het bepalen van de impact van eenvoudige wijzigingen in de systeemregels blijft dan natuurlijk grotendeels bestaan - al is ook hier weer een uitweg mogelijk: beschrijf het systeem in termen van 'in scherm S kan dit en dat' en 'alleen die en die gebruikers hebben toegang tot scherm S'. Uiteindelijk leidt zelfs dat meestal nog wel tot een werkbaar systeem. Maar geluiden van de werkvloer geven meestal een goed beeld van wat er eigenlijk aan de hand is: 'Nee, als je dat gegeven wilt wijzigen, dan moet je niet dit scherm maar dat scherm gebruiken'.

Hoe anders ziet de wereld eruit als we de 'automatische' aanpak volgen: de regel wordt automatisch geïmplementeerd zonder wijzigingen in specificaties en broncode van programmatuur. De prijs die men daarvoor betaalt, is uiteraard dat het mechanisme om regels automatisch in gedrag te vertalen wel eerst moet worden ontwikkeld of aangeschaft. Daarnaast geldt dat elementen in de gebruikersinterface waarvan het gedrag door regels kan worden beïnvloed (denk aan tekstvelden, buttons en menu-items) zich ervan bewust moeten zijn dat dit gedrag deels extern bepaald is. Voorbeeld: een menu-item dat een schermfunctie activeert kan uitgeschakeld ('disabled') worden, omdat de gebruiker van de applicatie niet is geautoriseerd om het betreffende scherm op te starten. Ideaal is het natuurlijk als intelligentie in het menu-item zelf het 'slimme' programmeurs onmogelijk maakt om het menu-item alsnog op 'enabled' te zetten.

De casus toont het belang aan van het onderkennen van de relatie tussen regels in het systeem en een adequate vertaling in gedrag. Bij administratieve systemen zijn regels in het systeem veelal regels over gegevens in het systeem. Een aanzienlijk deel van de functionaliteit (en de kosten) van het systeem bestaat uit het afdwingen van deze regels, naast de 'echte' functionaliteit als het berekenen van salarissen of het versturen van facturen. Een

Het dbms voorbij (7)

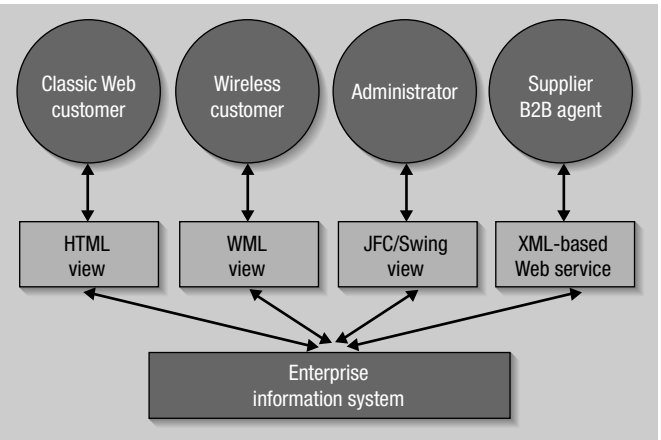
In de artikelen die tot dusverre in deze serie verschenen, is ingegaan op twee belangrijke aspecten van gegevensverwerkende systemen: validatie en autorisatie. De *communis opinio* is dat de huidige generatie dbms'en op deze gebieden voldoende ondersteuning biedt, maar we zagen dat daarop nog heel wat valt af te dingen. Deze keer gaan we in op een aspect dat van alle onderwerpen in deze serie ongetwijfeld het verst voorbij het dbms ligt: de presentatie van gegevens.

systeem dat de regels niet afdwingt is het equivalent van een kaartenbak, een Word-document of een Excel-sheet: de bruikbaarheid valt of staat met de discipline van de gebruiker. Wie dat niet erg vindt, moet alleen al uit kostenoogpunt ernstig worden afgeraden een administratieve applicatie te kopen of bouwen. Wie hogere eisen stelt, vindt in de casus de gezochte justificatie voor het behandelen van iets ogenschijnlijk onbelangrijks als 'presentatie' in een artikelserie over databasefunctionaliteit.

PRINCIPE VAN MODEL-VIEW-CONTROLLER

Hierboven is de term model-view-controller (MVC) gevallen, een architectuurprincipe ('design pattern') dat zijn oorsprong vindt in de Smalltalk-wereld. Kern van MVC is een scheiding tussen gegevens (het model), presentatie (de view) en logica (de controller).

Een eenvoudig voorbeeld van MVC uit de dagelijkse praktijk is de tekstverwerker. De gegevens in een tekstverwerker bestaan uit de teksten en opmaakstructuren van een document. Op deze gegevens zijn verschillende views mogelijk. In Microsoft Word bijvoorbeeld zijn dat 'Normaal', 'Weblay-out' en 'Afdrukweergave', alsmede het 'Afdrukvoorbeeld'. Elke view beschikt over een controller, die ervoor zorgt dat de weergave op het scherm en de inhoud van het document synchroon blijven lopen. De drie eerstgenoemde Word-views maken het mogelijk te muteren. In de view 'Afdrukvoorbeeld' kan alleen naar de gegevens worden gekeken, muteren is niet mogelijk.



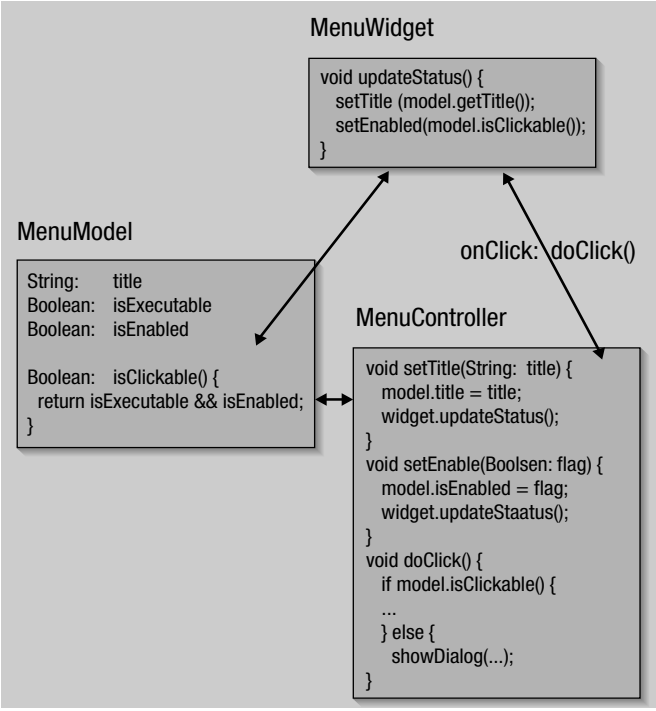
FIGUUR 1: MODEL-VIEW-CONTROLLER OP HOOG NIVEAU.

In de complexe administratieve omgevingen van vandaag vindt MVC gretig aftrek. Terwijl legacy applicaties nog jaren onder ons zullen blijven, vraagt de business om fancy internettoegang. Middleware-leveranciers positioneren zich graag als vendor van de controller-technologie die nieuwe Internet-views kan koppelen aan bestaande legacy gegevensverzamelingen (de term 'model' voor deze veelal monolithische applicaties is misplaatst, maar vooruit). Moderner klinkt het al als we voor 'legacy applicatie' de term 'enterprise information system' (EIS) gebruiken, we 'Internet' uitbreiden met 'mobiel Internet' en 'XML-berichtenverkeer', en natuurlijk voor intern gebruik ook nog een mooie GUI willen. Dat levert mooie plaatjes op zoals die in figuur 1.

Op grond van architecturen zoals die in figuur 1 wordt veelal deze conclusie getrokken: vanuit een perspectief van architectuur is het juist om presentatie volledig te scheiden van logica en gegevens. Het pleidooi voor een sterke integratie dat uit onze casus naar voren komt, valt dan niet in vruchtbare aarde. Toch biedt MVC hier een uitkomst.

MVC IN HET JUISTE PERSPECTIEF

Architectuurprincipes en design patterns als MVC beschrijven algemene oplossingen voor bepaalde typen van problemen. Dat algemene karakter is zowel een kracht als een zwakte. Mensen zijn geneigd algemene zaken te verduidelijken met concrete voorbeelden, waarna vervolgens het onderscheid tussen beide ver-



FIGUUR 2: MODEL-VIEW-CONTROLLER OP LAAG NIVEAU.

vaagt. Leest u er het recente regeerakkoord ook maar eens op na: het onderscheid tussen algemene 'mobiliteit' en het voorbeeld 'automobiliteit' wordt niet of nauwelijks gemaakt.

Als we de associatie tussen de term MVC en architecturen als die in figuur 1 loslaten, blijkt MVC ook een uitkomst te bieden om de zo gewenste koppeling tussen regels en gedrag te kunnen bewerkstelligen. Als voorbeeld nemen we het menu-item uit de casus.

Een menu-item presenteert zich naar gebruikers toe als een element in een (grafische) userinterface (aangeduid met de term 'MenuWidget' in figuur 2). Een visueel aspect van het menu is het al dan niet enabled staan ('grijs menu'). Dit visuele aspect dekt echter maar een gedeelte van de lading.

Proberen we het concept menu-item te vangen in een model (MenuModel in figuur 2), dan komen we attributen tegen als de titel van het menu en het al dan niet toegankelijk zijn ('enabled'). Als we de in de casus genoemde wens willen honoreren dat het menu-item bij voorkeur zelf moet kunnen bepalen of het wel in de 'enabled' status *mag* staan, is het ook van belang dat het model informatie heeft over het door de gebruiker uitvoerbaar zijn van de actie die aan het menu item is gekoppeld. Ter onderscheiding van de term 'enabled' gebruiken we hiervoor de term 'executable'.

Ten slotte dienen we het controller-gedeelte van MVC nog in te vullen. De logica van een menu item is te vatten in enkele regels:

- indien de enabled en/of executable status van het menu-item verandert, dient het schermobject navenant al dan niet toegankelijk te worden gemaakt;
- indien de titel verandert, dient het schermobject dezelfde titel te tonen;
- indien het schermobject wordt geactiveerd (meestal door een muisklik), moet de aan het menu-item gekoppelde actie worden uitgevoerd;

- de aan het menu-item gekoppelde actie kan alleen worden uitgevoerd indien het menu-item zowel enabled als executable is. In alle andere gevallen wordt een foutboodschap getoond.

Het volledige overzicht van het menu-item, uitgewerkt conform de MVC-architectuur, is weergegeven in figuur 2.

CORRECTE INITIALISATIE

Wat hebben we hiermee nu bereikt? Het antwoord ligt besloten in het MenuModel en in alle andere modellen die ontstaan als we de userinterface-elementen uiteenrafelen volgens het MVC-principe. We wilden bewerkstelligen dat regels (in het voorbeeld 'gebruiker G is niet geautoriseerd om scherm S te starten') werden vertaald in gedrag ('het menu-item dat scherm S activeert is uitgeschakeld'). De plaats in het model voor deze informatie is het attribuut isExecutable. Dan moeten we ervoor zorgen dat dit attribuut op correcte wijze wordt geïnitialiseerd. Dit kan op twee manieren:

- we kiezen ervoor (in dit voorbeeld) het menu niet hard te programmeren, maar te laten genereren op basis van de regels die we over het systeem hebben vastgelegd - liefst dynamisch, zodat niet bij elke wijziging in het autorisatieschema nieuwe programmatuur moet worden gegenereerd;
- we gebruiken een userinterface-library waarbij het koppelen van acties aan menu's niet meer gebeurt door dit te programmeren (het specificeren van 'on click' event handlers en dergelijke) maar door het opgeven van een speciaal object dat deze actie beschrijft, eventueel weer te genereren uit een systeemspecificatie. In het voorbeeld bevat het object dan informatie over het scherm (bijvoorbeeld een schermnummer) en informatie over de toegangsrechten tot dat scherm.

Met deze laatste mogelijkheid zijn we echt aangeland bij de hogeschool-applicatie-architecturen.

LUSTEN EN LASTEN

De vertaling van regels in gedrag doortrekken tot het gaatje is zowel lastig als lonend, zoals de casus en de voorbeelden in dit artikel aangeven. De lusten komen echter niet zonder lasten, en dat geldt hier wellicht nog meer dan voor de eerder in deze serie behandelde onderwerpen. De volgende keer keren we weer terug naar onze data roots. ●

Ir. F.G.W. van Orden (fridoo@faapartners.com) is managing partner van FAA Partners BV.

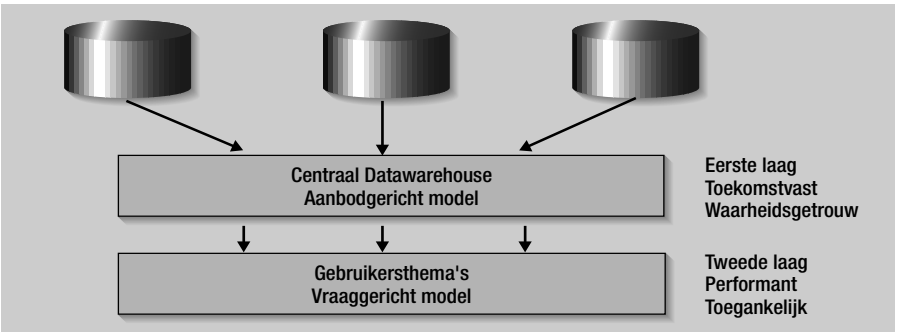
Vervolg van pagina 29

shotdatabase. Dit vraaggerichte model moet alle problemen oplossen die de performance en de toegankelijkheid betreffen.

VLOEKEN IN DE KERK

Als centraal datawarehouse werd in dit geval gekozen voor een kopie van het transactionele systeem, waarin de gehele mutatiehistorie tastbaar werd gemaakt door middel van *volgnummers*. Het is vloeken in de kerk, maar deze gekozen oplossing werkt niet met system keys. Tot overmaat van ramp is het volgnummer ook nog eens redundant. Het is immers direct afleidbaar uit de invoerdatum en in die zin verboden in een relationeel model. Voor een afschildering van de werkelijkheid praat het echter wel een stuk gemakkelijker. Het maakt bovendien de primary key een stukje korter.

De hoogte van het volgnummer bezit in zekere zin nog informatieve waarde. Het zegt iets over de mutatiegraad en dat kan nuttige informatie zijn. Bij een verzekeringsmaatschappij kan het bijvoorbeeld een aanwijzing zijn voor fraude of een indicatie van de kosten van een schaderespondent: was één telefoontje genoeg of is de hoogte van het schadebedrag twaalfmaal aangepast? Als we zoveel doen voor het gemak van de gebruiker, mogen wij zelf, bouwers, dan niet een onschuldig redundant volgnummertje aanmaken als dat zo uitkomt?



FIGUUR 2: HET TWEE-LAGENMODEL.

STOF TOT NADENKEN

Een succesvolle implementatie van een business intelligence-omgeving, en misschien wel de charme van het vak, is vrucht van het voortdurende geschipper tussen systeemkosten en performance-eisen, tussen business en ICT, tussen lokaal en centraal. Dit is een communicatief zwaar en creatief vak, waarbij denken buiten de eigen grenzen en het voortdurend praten met collega's -vooral over gemaakte fouten!- nodig is om de beste oplossing te vinden voor elk BI-probleem.

We hebben het gewaagd de algemeen aanvaarde theorie over system keys ter discussie te stellen en nota bene versienummers te introduceren in een datawarehouse. Dat geeft misschien ook geen universeel toepasbare oplossing, maar wel stof tot nadenken. ●

Drs. C. Verhagen (karien.verhagen@vba.nl) is programmamanager datawarehousing bij de Verenigde Bloemenveilingen te Aalsmeer.