Volgens veel mensen zijn componenten niet goed te combineren met databases. Op diverse forums wordt momenteel deze discussie gevoerd, er verschijnen regelmatig artikelen over dit onderwerp en ook in de praktijk lijkt de combinatie componenten en database een lastige. Is dat terecht? Laten we proberen deze vraag te beantwoorden door op fundamenteel niveau naar componenten en databases te kijken.

Componenten en relationele databases

Is de combinatie mogelijk?

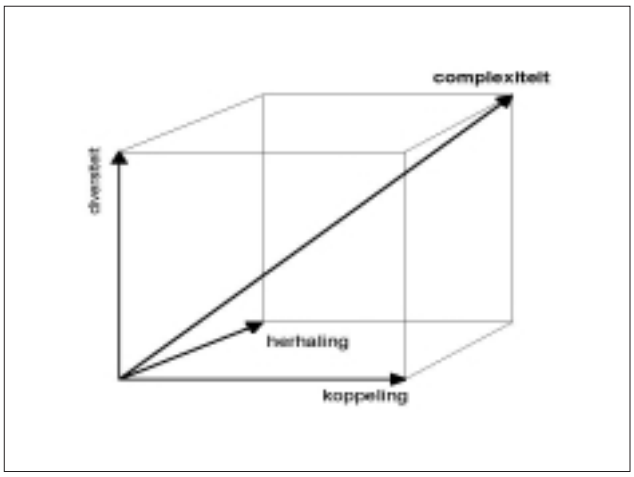
COMPONENTEN Een component is een zelfstandig deel van een systeem, een deel dat in relatieve afzondering kan worden behandeld. Een component heeft duidelijke interfaces, expliciete koppelingen met zijn omgeving en een duidelijke functie. Volgens de Synalyse-methode is een softwarecomponent 'de functionele clustering van classes, procedures of andersoortige softwarebouwstenen met expliciete afnemende en aanbiedende interfaces'. Een component is dus een deelsysteem dat aan bepaalde voorwaarden voldoet. Componenten zijn de eenheden van ontwerp, hergebruik, test en oplevering.

Componenten komen voort uit het functiemodel van een systeem. Een functiemodel verdeelt een systeem in delen op basis van functionaliteit en een component is de drager van een (deel-) functie. In een functiemodel van een audiosysteem komen bijvoorbeeld deelfuncties voor als opname, weergave, ontvangst, voorversterking, eindversterking en toonregeling. In het systeem zelf worden deze functies 'gedragen' door componenten als de cassettespeler, de voorversterker, eindversterker en eventueel een tuner.

Elk component heeft duidelijke in- en uitgangen. In geval van een audiosysteem zijn dat tulpstekers, in geval van software zijn het 'berichtinterfaces'. Afnemende interfaces die aangeven welke berichten de component zelf verstuurd (of welke procedures de component zelf zal aanroepen) en aanbiedende interfaces die aangeven welke berichten de component kan afhandelen (of welke procedures de component zelf kan uitvoeren). De logische opbouw is hetzelfde maar de fysieke 'formaten' verschillen omdat audiocomponenten nu eenmaal van andere materialen worden gemaakt dan softwarecomponenten.

ZELFSTANDIGHEID Componenten hebben als belangrijke eigenschap zelfstandigheid. Een component kan eenvoudig worden vervangen door een gelijksoortig component, bijvoorbeeld als het kapot is. Of als een betere maar gelijkwaardige component beschikbaar komt. Een component kan ook los van andere componenten worden getest en ontwikkeld. Om dit te kunnen bereiken worden de interfaces en de functie van de componenten voor een bepaald systeem zorgvuldig vastgelegd en mag bij voorkeur alleen de inhoud van de component, de vorm, worden veranderd. Zelfstandigheid kan het beste worden bereikt met een structuur met weinig koppelingen waarin dan automatisch behoorlijk wat herhaling voor komt. Alleen door delen te herhalen kan er een lagere koppeling worden verkregen. Dat is een algemeen geldig mechanisme dat we zodadelijk zullen bespreken.

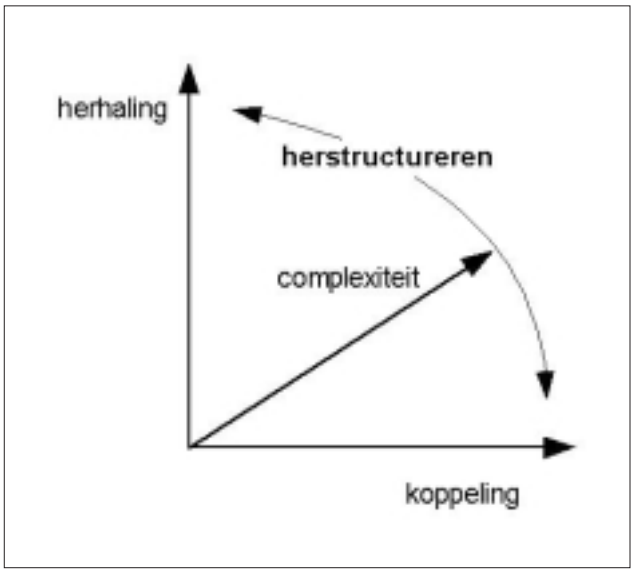
DATABASES Een database is een permanente opslag voor gegevens. De meeste programma's moeten hun gegevens voor een volgende sessie kunnen bewaren en gebruiken daarvoor een database. Databases worden ontworpen door een zogenaamd datamodel op te stellen. Het datamodel geeft de karakteristieke structuur weer van de gegevens in de database. De inhoud van de database zelf is georganiseerd in records en omdat het aantal records en de inhoud van de records telkens anders zijn, is het moeilijk om daar een bouwtekening van de maken. Daarom gebruikt een ontwerper de karakteristieke structuur. van de records om de database te ontwerpen, deze structuur is namelijk onveranderlijk. Dat model heet een datamodel. In dit artikel zullen wij ons concentreren op een speciaal soort database, namelijk de relationele database en het bijbehorende relationele datamodel. De reden daar-



FIGUUR 1. Herhaling, koppeling en diversiteit bepalen de complexiteit van een structuur

voor is dat dit soort verreweg het meest gebruikte is. Ieder groot systeem is wel op één of andere manier gekoppeld aan een rdms zoals Oracle of DB/2.

CONSISTENTIE Een database heeft als belangrijkste eigenschap (naast natuurlijk persistentie) consistentie. Een database moet, ondanks de voortdurende wijzigingen van zijn inhoud, een consistente verzameling van gegevens bevatten. De RDBMS moet de consistentie van gegevens in uiteenlopende situaties kunnen garanderen. Bijvoorbeeld als twee gebruikers tegelijkertijd dezelfde gegevens wijzigen, of als een hele reeks van gerelateerde gegevens in één keer moet worden gewijzigd. Consistentie kan het beste worden bewaard in een structuur waarin weinig tot geen herhaling bestaat, ook dat principe is universeel. Een wijziging hoeft dan maar op één plek worden doorgevoerd. De datamodelleerder noemt herhaling vanwege de ongewenstheid trouwens 'redundantie'.



FIGUUR 2. Datanormalisatie poogt de herhaling uit de structuur te halen

DATANORMALISATIE Om gegevens een structuur met zo min mogelijk herhaling te geven, biedt datamodellering een techniek die normaliseren heet. Normaliseren maakt gebruik van wiskundige regels om gegevens te 'clusteren' naar hun functionele samenhang wat resulteert in een database met zo min mogelijk herhaling aan gegevens. Er bestaan verschillende gradaties van 'herhalingvrijheid', elk gerepresenteerd door een andere zogenaamde normaalvorm. Een datamodel dat voldoet aan de derde normaalvorm beschrijft een herhalingvrijere database dan een datamodel dat 'slechts' voldoet aan de eerste normaalvorm. Normaliseren voorbij de derde normaalvorm, tot en met de vierde of zelfs vijfde normaalvorm, is ongebruikelijk. De derde normaalvorm is voldoende om alle gangbare vormen van herhaling te verwijderen uit de database.

HET PROBLEEM Wat is precies het probleem? Waarom zouden componenten en databases slecht te combineren zijn? Is het probleem de manier waarop een ontwerper zijn componentmodel samenstelt? Is het de manier waarop een relationeel datamodel wordt samengesteld? Zijn de twee soorten modellen gewoon per definitie onverenigbaar?

Deze vragen zijn verrassend eenvoudig te beantwoorden. Er is in principe namelijk geen probleem! Er is alleen een verschil tussen het gezichtspunt van waaruit een componentmodelleerder en een datamodelleerder naar een systeem kijken en dat maakt dat zij andere beslissingen nemen om hun in principe goed verenigbare modellen samen te stellen. Een datamodelleerder heeft een relatief beperkte blik op een systeem. Voor hem is consistentie verreweg de belangrijkste eigenschap. Aan een datamodelleerder is gevraagd: 'maak een structuur waarin gegevens veilig kunnen worden opgeslagen en waarin consistentie zo goed mogelijk is te garanderen!'. En dat heeft hij met behulp van datanormalisatie gedaan. Een componentmodelleerder houdt niet alleen rekening met consistentie maar ook met de handelbaarheid van het systeem. Aan de componentmodelleerder is gevraagd: 'maak een opdeling van het systeem die er voor zorgt dat het een begrijpelijk en handelbaar geheel van delen wordt!' En dat heeft hij met behulp van een functiemodel gedaan. Hij heeft gezorgd voor een overzichtelijke (functionele) clustering van bij elkaar horende bouwstenen, heeft ze netjes verpakt en heeft de doos waarin de onderdelen worden samengebracht (de component) voorzien van ingangen en uitgangen (berichtinterfaces). Zijn systeem heeft een soort plug-and-play opbouw gekregen die het mogelijk maakt delen in afzondering te behandelen. Elk deel is een relatief autonome eenheid.

GEZAMENLIJK GEZICHTSPUNT De functionele opdeling van het componentmodel heeft echter geen rekening gehouden met de wensen van de datamodelleerder. De datamodelleerder heeft een applicatiebreed datamo-

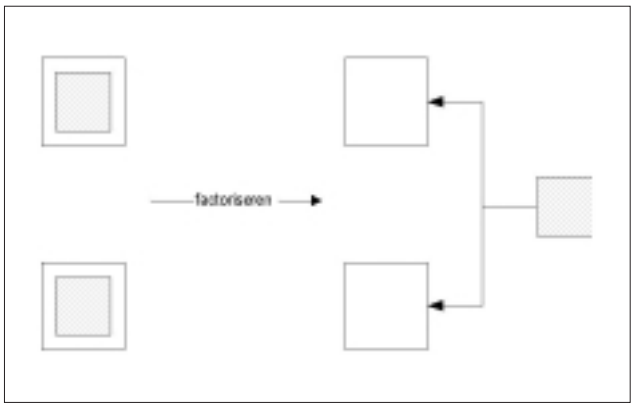
del gemaakt, zonder rekening te houden met eisen die nóg belangrijkere zijn dan consistentie, zoals eenvoud en handelbaarheid. De ideale structuur vanuit het gezichtspunt van de datamodelleerder bevat bovendien een bijna maximale hoeveelheid koppeling van delen, dat is de prijs die moet worden betaald voor herhalingsvrijheid. Die herhaling is lastig voor de componentmodelleerder want componenten zijn juist los gekoppeld en bevatten daarvoor onderling wél herhaling. Herhaling kan worden geruild voor koppeling en andersom maar herhaling én koppeling tegelijkertijd verminderen kan alleen met functioneel onsamenhangende delen en die vormen per definitie samen geen systeem.

Als het zojuist beschreven datamodel moet worden gecombineerd met een componentenstructuur van dezelfde applicatie, dan geeft dat problemen. Beide structureren sluiten slecht op elkaar aan, het datamodel zal dwars door alle componentgrenzen heen lopen. Hier is op fundamenteel niveau echter geen reden voor. Het enige probleem is dat beide modellen vanuit andere gezichtspunten zijn gemaakt. De oplossing is dus beide ontwerpers een gezamenlijk gezichtspunt te bieden.

HERHALING OF KOPPELING? Herhaling en koppeling zijn factoren die elkaar bijten. Het verminderen van de éne factor betekent vrijwel altijd het vermeerderen van de andere factor en andersom. Om te kunnen begrijpen hoe dit werkt, kan de volgende 'rekentruc' worden gebruikt. Herhaling, koppeling en ook diversiteit kunnen worden gezien als factoren die samen de complexiteit van een structuur bepalen. Dit geldt voor elke soort structuur, dus ook die van componenten, records of tabellen. Figuur 1 toont dit.

Deze definitie is oorspronkelijk bedacht voor toepassing in de Synalyse-methode maar is in principe algemeen toepasbaar. Het is niet per se 'de waarheid' maar een bedenksel om verstandige beslissingen te kunnen nemen bij het herstructureren van systemen. Datanormalisatie is een vorm van herstructurering en volgt dezelfde wetmatigheden.

Elke structuur heeft een bepaalde 'intrinsieke' complexiteit. Deze complexiteit is niet te verminderen: het is de complexiteit die nodig is om de gewenste functionaliteit te kunnen leveren. De verdeling van de complexiteit over de drie factoren is echter wel te veranderen. Dit heet herstructureren. Het is bijvoorbeeld mogelijk om te kiezen voor een structuur met veel herhaling. Zo'n structuur heeft dan wel minder koppeling en diversiteit. Complexiteit kan namelijk worden gezien als een vector waarvan wel de richting maar niet de grootte kan worden veranderd. (Data-) normalisatie is een vorm van herstructurering die er op is gericht de factor herhaling uit een structuur te halen. Zoals figuur 2 laat zien, kan dit



FIGUUR 3. Herhaling wordt afgewogen tegen koppeling

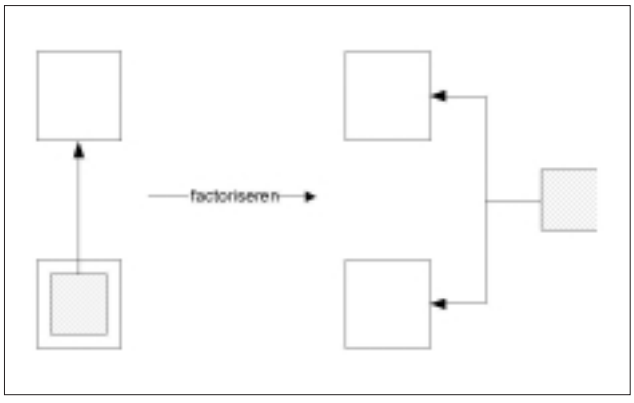
echter alleen maar door koppeling aan te brengen.

Er blijken maar een verassend klein aantal fundamentele herstructureringen mogelijk te zijn. De onderstaande figuren tonen de belangrijkste. In figuur 3 wordt herhaling afgewogen tegen koppeling. Een structuur kan een bepaalde functie leveren mét herhaling van delen (links) of zónder herhaling, maar met koppeling (rechts).

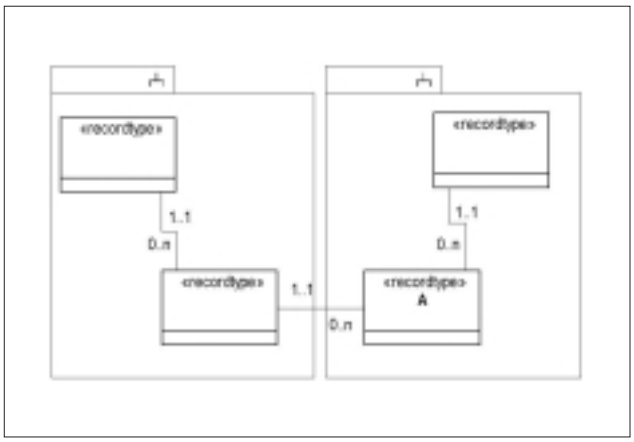
Een andere vorm is het ontkoppelen van modules door delen buiten haakjes te brengen. Hierdoor vergroot de diversiteit in de structuur. Dit wordt getoond in figuur 4.

Elke herstructurering heeft voor- én nadelen. Redenen om te kiezen voor koppeling in plaats van herhaling zijn compactheid en aanpasbaarheid (gemakkelijker te bewaren consistentie bij veranderingen). Redenen om juist voor herhaling te kiezen zijn zelfstandigheid, robuustheid en snelheid van verwerking. Voor niets gaat de zon op. De ontwerper moet zorgvuldig afwegen welke eigenschappen het zwaarst wegen en de structuur die hij bedenkt daarop afstemmen.

GENORMALISEERDE DATABASE GEWENST? Een deel van de oplossing is dat de componentmodelleerder en de datamodelleerder elkaar eerst beter moeten begrijpen. Maar wie moet wie nou beter begrijpen? Ik neig er naar om te



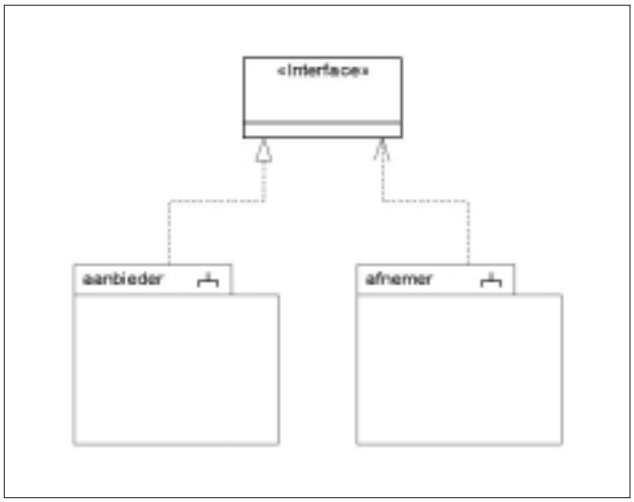
FIGUUR 4. Ontkoppeling van modules vergroot de diversiteit van de structuur



FIGUUR 5. De getoonde componenten gebruiken een gezamenlijk datamodel

stellen dat de datamodelleerder de componentmodelleerder beter moet leren begrijpen. De datamodelleerder zou zijn blik moeten verwijden en rekening moeten houden met andere aspecten van het softwaresysteem dan alleen consistentie van gegevens. Het probleem is namelijk het bewaren van consistentie door te kiezen voor een genormaliseerde / herhalingsvrije database. Consistentie zou misschien anders moeten worden bewaakt.

Een genormaliseerde database is misschien erg handig als een softwaresysteem als enige eis consistentie van gegevens heeft, als het softwaresysteem echter ook in begrijpelijke, losgekoppelde delen moet worden verdeeld, is de maximale koppeling die ontstaat door normalisatie erg onhandig. Als een datamodelleerder zijn zin zou krijgen, zou er uiteindelijk één grote, wereldwijde database ontstaan. En dat is natuurlijk onzin. Dat zal ook de datamodelleerder niet ontkennen. Iedereen kent het verschijnsel van de grote, logge onhandelbare 'corporate database' die voor grote problemen zorgt omdat zijn performance te laag is en de gebruikers elkaar in de weg zitten. Want dat is een andere prijs die moet worden betaald:



FIGUUR 6. Hoogwaardige componenten zijn slechts gekoppeld door middel van hun externe interfaces

uiteindelijk zorgen al die koppelingen die ontstaan door normalisatie voor een slechte performance. Om nog maar te zwijgen over de robuustheidsproblemen van een enkele database waar alles van af hangt.

Is het trouwens niet raar dat een ontwerper van objectgeoriënteerde software nooit zal overwegen om al zijn objecten centraal te laten beheren in een soort 'objectbase' maar dat zoiets voor een datamodelleerder heel normaal is? Een record is gewoon het data-aspect van een object. Waar komt dan dit verschil in inzicht dan vandaan?

OPLOSSING Ik noemde zojuist enkele van de problemen die ontstaan door te kiezen voor een database met een te grote reikwijdte. Deze problemen zijn de ontwerper van de database natuurlijk ook niet ontgaan en de meeste datamodellen worden na initiële in bedrijfstelling dan ook in de loop der tijd weer enigszins gedenormaliseerd. Denormaliseren is het bewust aanbrengen van herhaling. Denormaliseren is echter een gevoelsmatig proces, zo wiskundig correct als het normaliseren is (het kiezen van een datamodel dat zo min mogelijk herhaling van gegevens in de database oplevert), zo onhandig is het denormaliseren. Toch ligt in dit denormaliseren juist de oplossing besloten. Als een datamodelleerder nu eens niet meer kiest voor een bedrijfsbrede of applicatiebrede database maar voor veel kleinere databases, laten we zeggen voor elke component een aparte database? De datamodelleerder zou dan de overwegingen van de componentmodelleerder respecteren en de structuur van zijn datamodellen laten overeenstemmen met de functionele structuur van het softwaresysteem, waarop ook de componentenstructuur is gebaseerd. En als de datamodelleerder nu eens zorgt voor een herhalingvrije of genormaliseerde structuur binnen de grenzen van een component maar eventueel wel herhaling aanbrengt of denormalisatie toepast over de grenzen van de component? Om de twee verschillende componenten te kunnen ontkoppelen. In dat geval zouden verschillende componenten dezelfde tabellen krijgen en zouden hun datamodellen overeenkomsten vertonen (herhaling). Wat voor problemen zouden er dan ontstaan? Want het probleem van het combineren van componenten en databases is dan namelijk opgelost...

CONSISTENTIE De problemen die zouden ontstaan door zo'n aanpak zijn voornamelijk te herleiden tot een moeilijker te bewaren consistentie, niet per se tot een uitbundige herhaling van gegevens. Het feit dat meerdere componenten in het systeem dezelfde soort tabellen zouden hebben, is namelijk nog geen herhaling- misschien nog wel van type, maar niet van instanties. Van één bepaald record hoeft nog steeds maar één instantie te bestaan in het hele systeem, ondanks dat het type van dat record misschien wel bij alle componenten van het systemen bekend is. En zelfs al is die herhaling van instan-

ties er wel, dan nog is dat geen probleem, zolang alle gegevens maar consistent zijn en kunnen worden gehouden.

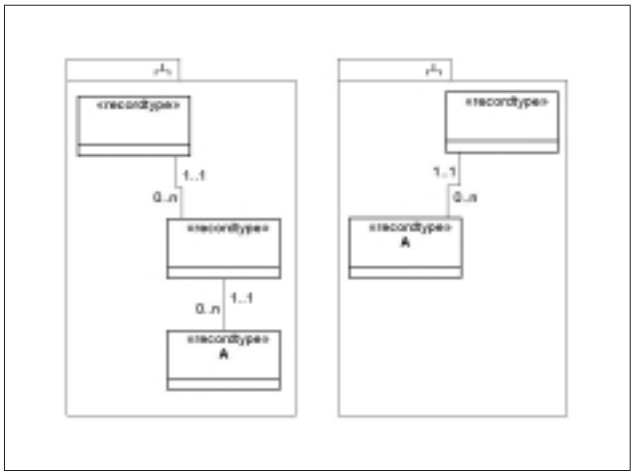
Consistentie is ook geen probleem, het moet alleen anders worden benaderd. Consistentie kan ook worden bewaard door componenten met elkaar te laten communiceren. Het echte probleem is dat consistentie bewaren met regels en procedures een enge gedachte is voor een datamodelleerder, en dat de huidige generatie rdms'en daar niet of nauwelijks voor gemaakt zijn. Maar het kan wel, het kost wat meer moeite en code, maar de beloning is nog veel groter. Als nergens in het systeem kopieën van een bepaald record worden gemaakt en de componenten dat in onderling overleg regelen, dan kan de consistentie net zo goed worden bewaard als met een genormaliseerde, applicatiebrede database. Echter met de voordelen van een componentenstructuur: begrijpelijkheid, robuustheid, snelheid en onderhoudbaarheid!

EEN DATAMODEL PER COMPONENT Als een datamodel wordt opgesteld zonder rekening te houden met de componentgrenzen, dan ontstaat een enkel, herhalingvrij datamodel met koppelingen die de componentgrenzen overspannen. Figuur 5 toont zo'n situatie: twee componenten en een datamodel dat (ongewenste) relaties bevat tussen die twee componenten.

De figuur toont precies díe situatie die het combineren van een database met componenten zo moeilijk maakt. De getoonde componenten kunnen niet meer los van elkaar worden gebruikt omdat ze een gezamenlijk datamodel gebruiken. Het zijn daarom geen hoogwaardige (lees: zelfstandige) componenten. Hoogwaardige componenten zijn slechts gekoppeld door middel van hun externe interfaces, zoals het figuur 6 laat zien.

De enige relaties die componenten onderling mogen hebben, zijn relaties tussen hun interfaces. Relaties in het datamodel die daar buiten om lopen, moeten worden verwijderd. Dat kan door de gekoppelde tabellen, of recordtypes, te herhalen. Figuur 7 toont dat.

De zojuist geïntroduceerde herhaling zorgt echter voor een mogelijk consistentieprobleem: als beide A-tabellen hetzelfde record bevatten, dan moeten bij wijzigingen van dat record ook beide tabellen worden aangepast. De truc is ervoor te zorgen dat beide tabellen daarom nooit hetzelfde record bevatten. Een herhaling van types (tabel) is niet per se een herhaling van instanties (record) en het is best mogelijk dat een bepaalde tabel in meerdere componenten wordt herhaald zonder dat er een herhaling van records optreedt. Om dit te kunnen doen, moeten componenten samenwerken: ze mogen records bijvoorbeeld alleen maar doorgeven, niet kopiëren. Dit zorgt voor een extra uitwisseling van berichten, dat is de prijs die moet worden betaald. Consistentie die oorspronkelijk inherent was aan de structuur van het datamodel, moet nu actief worden bewaakt. Mocht het éne component toch een kopie van een record van het andere component nodig hebben, dan moet die kopie dus ook als zodanig worden behandeld. De kopie mag dus niet meer worden opgeslagen in de eigen tabellen, kan verouderd zijn omdat het origineel misschien al is aangepast en moet worden weggegooid na gebruik. Voor een goede programmeur is dat echter geen probleem, zolang hij het maar van tevoren weet en de ontwerpen die hij moet programmeren er maar rekening mee houden!



FIGUUR 7. Deze herhaling zorgt voor een mogelijk consistentieprobleem

KIEZEN OF DELEN Voor ontwerpers die dit een teleurstellende oplossing vinden, heb ik een even teleurstellende mededeling: het is kiezen of delen en slechts door meerdere gezichtspunten in te nemen, kan de elegantie van de oplossing worden gezien. De hele problematiek is trouwens een gevolg van het feit dat het componentmodel en het datamodel, en ook het eventuele class-model, los van elkaar worden ontworpen. En dat de ontwerper van elk soort model zich blind staart op de gewenste eigenschappen en mogelijkheden van alleen zijn eigen model. Wanneer men een geïntegreerde aanpak volgt, zoals de Synalyse-methode die beschrijft, en het datamodel wordt ontworpen als onderdeel (het passieve aspect) van het class-model, is er helemaal geen probleem, dan ontstaat automatisch een goed 'component-combineerbaar' datamodel. Eerst een functiemodel, dan een componentmodel, dan een class-model per component. En uit dat classmodel kan het datamodel worden geselecteerd. Simpel toch? Wonderen bestaan niet, bovendien, gezond verstand is in de informatica al wonderlijk genoeg.

Patrick Savalle
Savalle is ontwerper methoden en technieken bij Synalyse, www.synalyse.com