Wie zegt dat relationeel en objectgeoriënteerd niet samengaan?

Problemen die er niet zijn

Patrick Savalle

Het is niet gemakkelijk een relationele database te koppelen aan een objectgeoriënteerd ontworpen softwaresysteem. Er zijn boeken volgeschreven over deze materie. Relationeel en objectgeoriënteerd zijn volgens velen als water en vuur. Het probleem heeft inmiddels verschillende namen gekregen: de impedance-mismatch, het OR-mapping-probleem. Er wordt zelfs door verschillende bedrijven gewerkt aan een nieuwe soort 'DBMS-sen': de objectgeoriënteerde 'database management systems' of ODBMS-sen, in plaats van een relationeel systeem of RDBMS. Is er wel sprake van een slecht samenspel tussen relationeel en objectgeoriënteerd?

De veronderstelling dat een objectgeoriënteerde structuur niet goed samengaat met een relationele structuur is echter gebaseerd op een denkfout. Beide zijn modellen van hetzelfde systeem en zijn daarom per definitie juist zeer goed te combineren. Een relationeel datamodel is 'gewoon' een bijzondere vorm van een class-model. Het enige dat daar voor nodig is, zijn slim gekozen definities. De huidige generatie class-modellen (UML, Booch, OMT) zijn vanwege hun inconsistente definitie ongeschikt voor een goede integratie van verschillende soorten modellen. De 'slim gekozen' definities komen uit de Synalyse-methode; het besproken class-model is dus geen UML-model, maar een Synalyse-model dat speciaal voor de gewenste integratie van class-, data- en andere modellen is opgesteld. Verder zal worden aangetoond dat objectgeoriënteerde databases onnodig zijn en dat huidige implementaties verkeerd in elkaar zitten.

HET SOFTWARESYSTEEM

Om een systeem te ontwerpen moeten modellen worden gemaakt. Om een bepaald soort model te kunnen maken moet bekend zijn hoe een systeem er in het algemeen uit ziet. Een gebruikelijke manier in geval van software is het systeem te zien als een structuur van objecten: dit heet objectoriëntatie. Volgens objectmodellering is het softwaresysteem een veranderlijke verzameling van objecten. Objecten die samenwerken door elkaar berichten te sturen (berichtrelaties), die bij elkaar horen door verwijzingen naar elkaar op te slaan (aggregatie- en associatierelaties) en die elk een eigen toestand en daardoor gedrag kunnen vertonen (attribuutwaarden en methoden). 'Het systeem' bestaat dus uit objecten. Objecten zijn eenheden van programmacode en data. Ze voeren methoden uit bij het ontvangen van berichten en slaan hun momentane toestand op in attribuutwaarden. Het is deze momentane toestand die ook af en toe moet worden opgeslagen in een database. Hiervoor is het nodig een relationeel datamodel en class-model te combineren, omdat de database wordt beschreven door een datamodel.

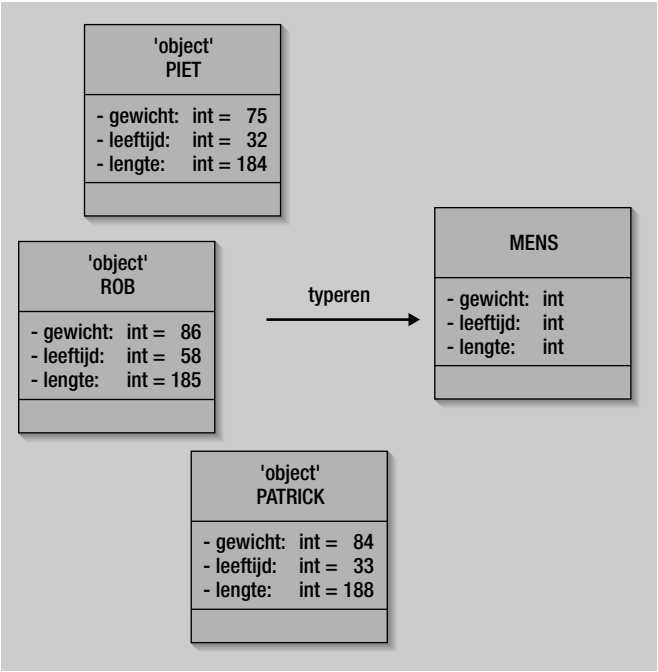
Behalve dat objecten elkaar berichten sturen, kunnen ze ook bij elkaar horen zonder dat ze informatie uitwisselen, zoals de piloot bij zijn vliegtuig hoort: dit zijn associatierelaties. Ze kunnen ook bij elkaar horen zoals een wiel bij een fiets hoort: dit zijn aggregatierelaties. Associatierelaties kunnen eenvoudig worden onderscheiden van aggregatierelaties door te kijken of twee objecten op hetzelfde niveau van aggregatie bestaan of op een verschillend niveau, zoals een geheel en zijn delen (de fiets en zijn wielen). Opsommend bestaat een softwaresysteem uit de volgende elementen.

- Objectsysteem (instanties)
  - Objecten
    - Attribuutwaarden
    - Methoden
  - Berichtrelaties
  - Associatierelaties
  - Aggregatierelaties

HET CLASS-MODEL

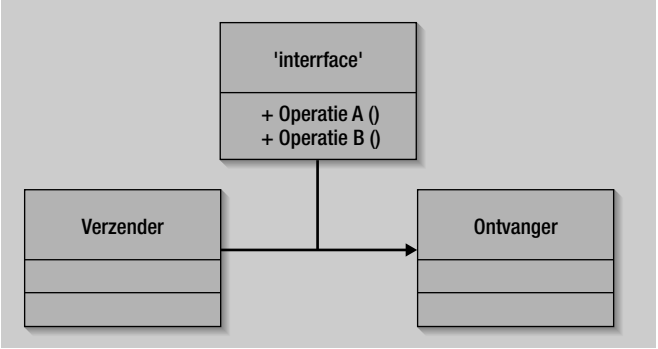
Het class-model is de bouwtekening van een softwaresysteem dat volgens objectgeoriënteerde werkwijze wordt ontworpen. Het toont de karakteristieke structuur van een systeem dat bestaat uit objecten. Omdat het in het leven van een softwaresysteem een komen en gaan is van objecten en objectrelaties, is het lastig de objectstructuur zelf vast te leggen. In plaats daarvan noteren we de karakteristieke structuur van het softwaresysteem: het class-model.

Een class-model is opgebouwd uit classes en class-relaties. De verzameling elementen en regels waarmee een class-model kan worden opgebouwd, vormt samen een class-metamodel. Zo'n metamodel is onderdeel van de methode of semantiek waarmee de software wordt ontworpen. De opbouw van het class-model kan uit de opbouw van objectmodel worden afgeleid (via typering). Een objectstructuur is een veranderlijke structuur waarvan alleen 'snapshots' kunnen worden gemaakt. Een class-model is een onveranderlijke structuur waarin alle mogelijke relaties en toestanden van de objecten staan aangegeven. Het opbouwen van een class-model gebeurt door alle mogelijkheden die objecten hebben samen te nemen en toe te kennen aan hun classes. Objecten worden getypeerd door classes, attribuutwaarden door attributen en methoden blijven na typering gewoon methoden. Een class bestaat dus uit attributen en methoden.

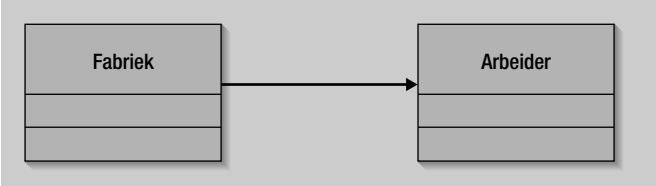


Figuur 1: Opbouw class-model via typering van het objectmodel

Relaties die objecten met elkaar hebben, bestaan ook tussen classes. Berichtrelaties worden in een class-model dus ook aangegeven.



Figuur 2: Interface als verzameling berichttypen van een object



Figuur 3: Associatierelatie

Objecten sturen elkaar vaak verschillende berichten. Om dit aan te geven worden interfaces gebruikt. Een interface is de verzameling berichttypen die een object kan verzenden of ontvangen.

In figuur 2 staat aangegeven dat objecten van Verzender-class twee soorten berichten kunnen sturen aan objecten van Ontvanger-class. Ook elke associatie- en aggregatierelatie waaraan een object kan deelnemen, is zichtbaar in het class-model; in figuur 3 een associatierelatie.

In korte stappen is nu een class-model opgesteld door eerst te bepalen hoe we het systeem graag willen zien (als een objectstructuur) en vervolgens die opbouw te typeren tot een model (het class-model). Opsommend bestaat een class-model uit de volgende elementen:

- Class-model (types)
  - Classes
    - Attributen
    - Methoden
  - Interfaces / berichttypen
  - Berichtrelaties
  - Associatierelaties
  - Aggregatierelaties

Verderop zullen nog enkele elementen worden toegevoegd, maar dit zijn de basiselementen. Merk op dat de UML geen berichtrelatie kent in het class-model, interfaces niet goed definieert en het onderscheid tussen associatie- en aggregatierelaties anders (bijna niet) definieert. Verder bestaan er in de UML slechts links tussen objecten en is er geen verband tussen het objectmodel en het class-model, ondanks het feit dat de één het onderwerp van de ander is (zou moeten zijn).

ACTIEVE EN PASSIEVE ELEMENTEN

Om gemakkelijk de samenhang tussen een class-model en een (relationeel) datamodel te kunnen zien, is het handig om binnen het class-model onderscheid te maken tussen een zogenaamd passief en een actief aspect. Elementen zijn actief als er data-uitwisseling of dataverandering plaatsvindt, terwijl elementen passief zijn als dat niet het geval is. Een berichtrelatie is actief, een methode ook. Een interface bestaat alleen als er een berichtrelatie bestaat, wat actieve elementen zijn. Zonder berichtrelaties vervalt ook de noodzaak tot interfaces. Een attribuut en een associatierelatie zijn daarentegen passief: ze wisselen geen data uit of veranderen die, net als een aggregatierelatie.

De volgende indeling is nu ontstaan:

- Class-model
  - Actief aspect
    - Methoden
    - Interfaces
    - Berichtrelaties
  - Passief aspect
    - Attributen
    - Associatierelaties
    - Aggregatierelaties

ABSTRACTE EN CONCRETE ELEMENTEN

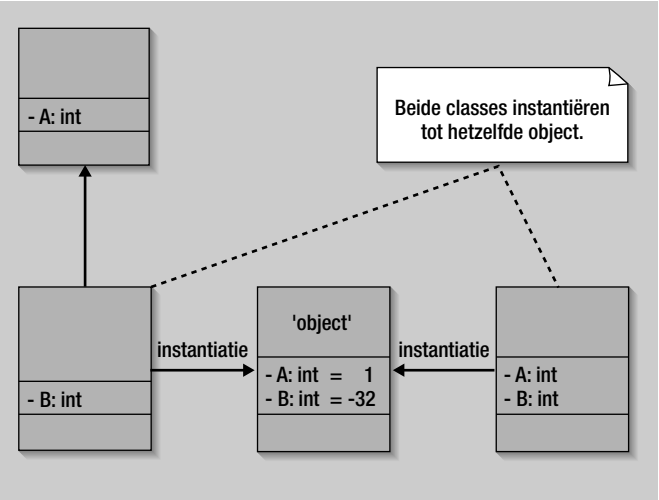
Een tweede opdeling die moet worden gemaakt, is de opdeling in zogenaamde concrete en abstracte elementen. Abstracte elementen zijn elementen die niet zichtbaar zijn in het onderwerp van het model (het systeem) maar 'slechts' dienen ter optimalisatie van het model zelf. Een voorbeeld is de spiegellijn in een bouwtekening: het onderwerp van de bouwtekening heeft zelf geen spiegellijnen. Een abstract element is als het ware een notatietruc. In een class-model bestaan ook zulke elementen; generalisatierelaties bijvoorbeeld. Dit zijn abstracte elementen omdat ze in het systeem (de veranderlijke structuur van objecten) onzichtbaar zijn: aan een object is niet te zien of zijn definitie bestaat uit een enkele class of een hele hiërarchie van super- en subclasses. Figuur 4 laat dat zien.

Elementen die wel zichtbaar zijn in het uiteindelijke systeem zijn concrete elementen - het zijn de concrete elementen die het onderwerp van het model vormgeven. Een class is een concreet element, het modelleert objecten van hetzelfde soort. De uiteindelijke indeling van class-modelelementen is daarmee als volgt:

- Class-model
  - Abstracte elementen
    - Generalisatierelaties
    - Abstracte classes
  - Concrete elementen
    - Actief aspect
      - · Methoden
      - · Interfaces
      - · Berichtrelaties
    - Passief aspect
      - · Attributen
      - · Associatierelaties
      - · Aggregatierelaties

HET RELATIONELE MODEL

De opbouw van het class-model is nu bepaald. De volgende stap is het bepalen van de opbouw van een relationeel datamodel. Het datamodel is de bouwtekening van een database: het toont de



FIGUUR 4: GENERALISATIERELATIES

karakteristieke structuur van een database die bestaat uit records, precies zoals het class-model de karakteristieke structuur toont van een softwaresysteem dat bestaat uit objecten. Vanwege deze analogie en dankzij de opbouw van het Synalyse class-model, is het eenvoudig om aan te geven hoe een datamodel is opgebouwd. Een datamodel is namelijk 'gewoon' het passieve aspect van een class-model. Punt. Een database is daarmee ook het passieve aspect van een objectmodel.

Een class zonder methoden is een recordtype ofwel een tabel. En een class-model zonder methoden, interfaces of berichtrelaties (het actieve aspect) is een datamodel ofwel een schema. Door uit

Het OR-mapping-probleem is ontstaan door slechte metamodellen met onhandig gekozen definities.

een class-model de passieve elementen te selecteren, ontstaat een datamodel. Of, andersom, door een datamodel te voorzien van actieve elementen, ontstaat een class-model. Voor een class-model dat ook abstracte elementen heeft (generalisatierelaties, abstracte classes), geldt dat deze eerst moeten worden verwijderd. Hoe dat moet en welke consequenties dat heeft, bespreken we zo dadelijk.

Een relationeel datamodel kent daarnaast ook zogenaamde normalisatieregels. Deze regels worden gebruikt om het datamodel zo te herstructureren dat de resulterende database zo min mogelijk herhaling van attribuutwaarden bevat. Het is gebruikelijk om drie opeenvolgende normalisatieregels te gebruiken, elk voor een grotere mate van herhalingvrijheid. Het datamodel, dat ontstaat door het passieve aspect te selecteren uit een class-model, is niet altijd even herkenbaar als volwaardig datamodel, omdat datamodellen wel en class-modellen niet worden genormaliseerd. Toch is het een geldig datamodel. Bovendien, omdat het datamodel onderdeel uitmaakt van het class-model, kunnen deze normalisatieregels ook worden toegepast op een class-model. De slimste manier om dat te

doen is door eerst een datamodel op te stellen, dat te normaliseren en op basis daarvan het class-model op te stellen.

HET 'OR-MAPPING PROBLEEM

Als een class-model en een relationeel datamodel zo gemakkelijk te combineren zijn, waar komt dan het idee vandaan dat relationeel en objectgeoriënteerd niet samengaan? Waarom zijn sommige softwareleveranciers dan zelfs bezig met het ontwikkelen van objectgeoriënteerde databases? Een deel van het probleem is dat het passieve aspect, het datamodel, niet in alle soorten class-modellen herkenbaar is. De constatering dat een datamodel een onderdeel van een class-model is, lijkt een open deur, maar ontwerpers die gebruik maken van bijvoorbeeld UML, Booch of OMT kennen helemaal geen actieve of passieve aspecten in een class-model. Ze kennen ook geen berichtrelaties in hun class-modellen. De constatering is dus alleen triviaal voor ontwerpers die gebruik maken van een methode of semantiek die het genoemde onderscheid wel maakt. Het Synalyse-model doet dat omdat het speciaal voor dit soort unificatie en integratie is opgesteld.

Het andere probleem is de manier waarop wordt omgegaan met overerving. Overerving wordt in een class-model gebruikt om een class-structuur anders (slimmer) te formuleren zonder dat de class-structuur een andere objectstructuur gaat weergeven - bijvoorbeeld om verschillende soorten objecten die onderling wel herhaling vertonen te modelleren met classes die onderling geen herhaling vertonen. Het is opnieuw een notatietruc die onzichtbaar zou moeten zijn in het systeem. Ook als twee classes onderling een generalisatierelatie hebben, hebben de objecten (of de records die daaruit kunnen worden geselecteerd) onderling nog steeds geen relatie. Figuur 5 toont dat.

Negen van de tien boeken die uitleggen wat overerving in een class-model is of hoe een class-model moet worden afgebeeld op een datamodel doen dat fout. Negen van de tien objectdatabases maken dezelfde fout. De fout die wordt gemaakt is dat generalisatierelaties worden overgenomen in de database, ofwel ook tussen records aanwezig zijn. Dat is alsof de timmerman de spiegellijnen en centerkruizen natuurgetrouw overneemt in de nieuwe tafel die hij aan het maken is.

EEN VOORBEELD

Het 'reduceren' van een (Synalyse) class-model tot een relationeel datamodel is gebaseerd op een paar eenvoudige constateringen. Een class bestaat uit attributen en methoden. Een recordtype

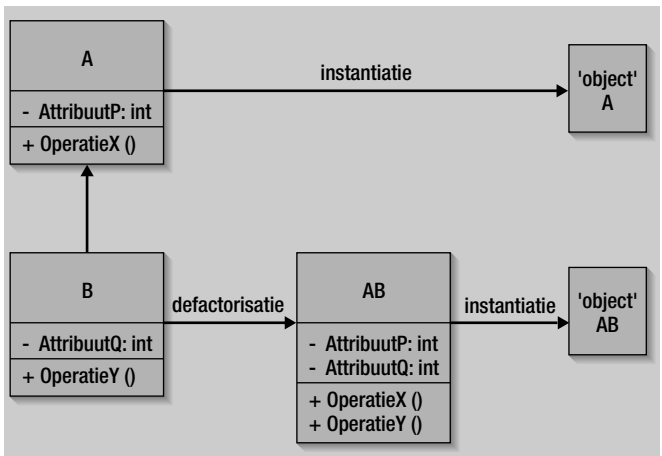
Objectgeoriënteerde databases zijn overbodige dingen.

slechts uit attributen. Daarom wordt gesteld dat recordtypes classes zonder methoden en interfaces zijn. Simplistisch, maar correct. Het extraheren van het relationele schema uit een class-model in een stappenplan:

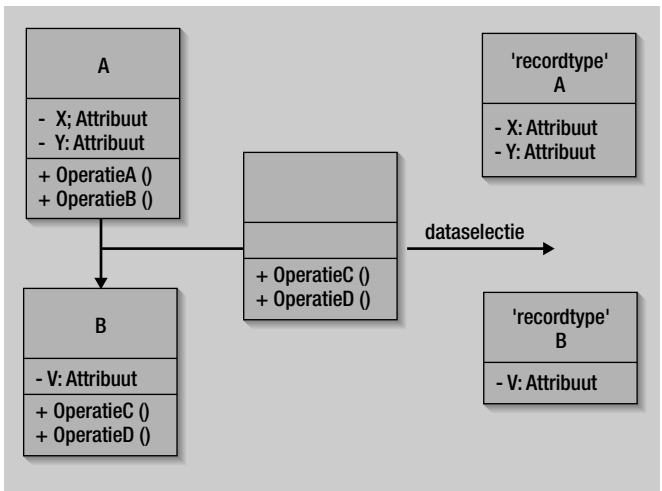
- De class-methoden worden uit het class-model verwijderd (gewoon 'weggegumd'), classes worden hierdoor recordtypes.
- Berichtrelaties worden uit het class-model verwijderd (gewoon 'weggegumd').
- Interfaces worden uit het class-model verwijderd (gewoon 'weggegumd'), eventuele generalisatierelaties met zo'n interface (polymorfie) vervallen.

Dit model bevat nog wel eventueel abstracte elementen zoals generalisatierelaties. Abstracte elementen hebben per definitie alleen maar betekenis binnen het gegeven model en worden daarom niet overgenomen van class-model naar datamodel. Nadat ook deze abstracte elementen zijn verwijderd (door zorgvuldige defactorisatie), ontstaat een 'traditioneel' datamodel. De laatste regel voor het extraheren van een datamodel is daarom:

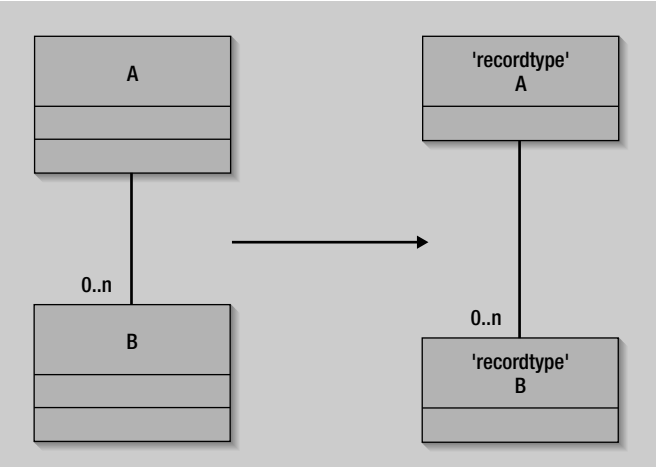
- Abstracte elementen worden zorgvuldig verwijderd middels defactorisatie



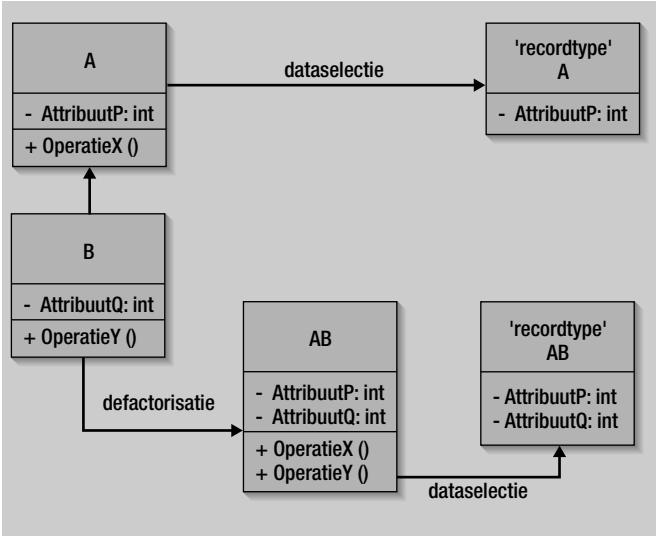
Figuur 5: Objecten hebben geen relatie, ook als twee classes die wel hebben



Figuur 6: Berichtrelaties en interfaces



Figuur 7: Voorbeeld van een aggregatierelatie



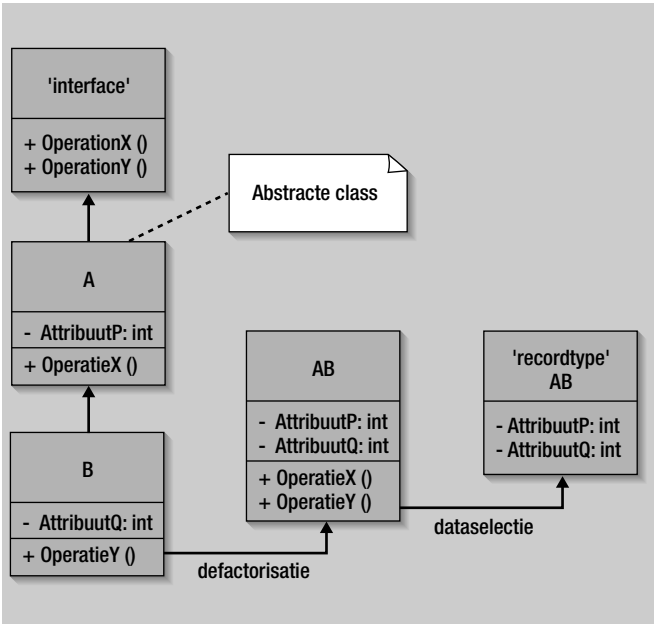
Figuur 8: De vertaling van generalisatierelaties

Het model dat zo ontstaat, kan direct worden ingevoerd in de definitietaal - de DDL - van het relationele model.
De volgende figuren tonen enkele class-modellen en de datamodellen die daaruit kunnen worden afgeleid. Allereerst een constructie met berichtrelaties en interfaces. Berichtrelaties en interfaces worden gewoon genegeerd bij de vertaling.

Vervolgens een voorbeeld van een aggregatierelatie. Deze wordt zonder eisen te stellen aan de navigeerbaarheid overgenomen. Hetzelfde geldt voor associatierelaties.

De volgende figuren tonen hoe generalisatierelaties moeten worden vertaald. Allereerst een situatie waarin beide classes concreet (instantieerbaar) zijn. Elke instantieerbare class krijgt een eigen recordtype toegewezen. Tussen de resulterende recordtypes bestaan geen relaties. In het datamodel is de generalisatierelatie dus op geen enkele wijze zichtbaar. Dataselectie betreft het selecteren van het aspect 'data'.

Ten slotte een situatie waarin één van beide classes abstract (niet instantieerbaar) is. Abstracte classes worden (net als andere abstracte elementen) niet overgenomen in het datamodel.



Figuur 9: Niet instantieerbare abstracte klasse

OO-DATABASES OVERBODIG

Relationele modellen en class-modellen zijn prima te combineren: het ene model is 'gewoon' een aspect van het andere model. Zelfs de normalisatieregels van het relationele model zijn, met wat inzicht, bruikbaar voor een class-model. Het ontwerpen van een systeem zou kunnen gebeuren door eerst een datamodel op te stellen en dat dan later uit te breiden tot een class-model. Naar mijn mening is het hele OR-mapping probleem ontstaan door slechte metamodellen (UML, Booch, OMT) met onhandig gekozen definities, met de juiste definities bestaat het probleem helemaal niet. Objectgeoriënteerde databases zijn overbodige dingen. Een relationele database is prima te gebruiken bij objectgeoriënteerde software. Sterker nog, een RDBMS sluit beter aan op OO-software dan een ODBMS. Raar maar waar. Hooguit sluit de huidige generatie RDBMS-en wat betreft jargon wat slecht aan op class-modellen. Vreemd is de manier waarop de meeste auteurs en producten omgaan met generalisatie of overerving. Het is het slimst om overerving te zien als een notatietruc die onzichtbaar zou moeten zijn buiten het model waarin het wordt toegepast. De meeste problemen kunnen trouwens worden vermeden door class-modellen gewoon op te stellen zonder generalisatie. Zulke modellen zijn absoluut niet slechter dan modellen die er wel gebruik van maken, maar dat is een artikel op zich waard. ●

Patrick Savalle is een van de ontwerpers van de Synalyse-methode en momenteel werkzaam als consultant bij Turnkiek Technical Systems (een onderdeel van Imtech ICT). Zie www.synalyse.nl voor meer informatie.