Ervaringen met object-georiënteerd programmeren bij de programmering van het Technisch Model Varkensvoeding

Ing. H.J.P.M. Vos

Proefstation voor de Varkenshouderij
Postbus 83, 5240 AB Rosmalen
tel. 04192-86555
fax 04192-18214
e-mail VOS@PVV.AGRO.NL

Referaat

Object-Oriented Programming (OOP) is een moderne techniek van programmeren, die ondersteund wordt door speciale programmeertalen. Steeds meer bestaande programmeertalen worden uitgebreid met mogelijkheden tot OOP. Ook Turbo-PASCAL van BORLAND heeft sinds enige tijd een OOP-uitbreiding. We hebben deze mogelijkheden van Turbo-PASCAL gebruikt bij het programmeren van het Technisch Model Varkensvoeding. Wij hebben er positieve ervaringen mee.

Trefwoorden: TMV, Voeding, Modellen, Object Oriented Programming

Inleiding

Hoewel Object-Oriented Programming (OOP) vooral de laatste tijd een enorme vlucht neemt, is de techniek zélf niet zo nieuw. Eind jaren zestig ontstond SIMULA, een simulatietaal en de eerste programmeertaal waarin OOP-ideeën waren verwerkt. Toch heeft het nog zo'n twintig jaar geduurd, voor die ideeën echt doorbraken. OOP ondersteunt het hergebruik van bestaande programmatuur beter en maakt het eenvoudiger om het overzicht te houden over grote programmeerprojecten. Dat zou de reden zijn waarom OOP nu zo doorbreekt. Veel bedrijven zitten met verouderde programmatuur en staan momenteel voor de keuze om toch maar door te gaan met steeds kostbaarder wordend onderhoud of alles te herschrijven met gebruik van nieuwe technieken, waarbij men hoopt daar op den duur voordeel bij te hebben. Zoals elke nieuwe ontwikkeling, kent ook OOP een aantal fanatieke aanhangers maar ook vele sceptici. In dit artikel wordt uitgelegd wat de belangrijkste kenmerken van OOP zijn en hoe deze bij één toepassing - het Technisch Model Varkensvoeding (TMV) - zijn toegepast.

TMV

In 1989 is een zogenaamde 'Werkgroep TMV' in het leven geroepen. De werkgroep had tot taak het opleveren van een 'Informatiemodel TMV'. Dit gebeurt in drie fasen. In april 1991 is de eerste fase afgesloten met de presentatie van het genoemde informatiemodel (Werkgroep TMV, 1991), dat de processen en rekenregels beschrijft die

```
type

voergift = record
    hoeveelheid : real; { kg }
    gehalte_ruw_eiwit : real; { kg/kg }
    gehalte_ME : real; { MJ/kg }
    gehalte_lys, gehalte_met, gehalte_cpm,
    gehalte_thr, gehalte_try, gehalte_iso : real;
    { Essentiele aminozuren (kg/kg) }
end;

vleesvarken = record
    eiwit, vet, as, water : real; { kg }
    opgenomen_voer : voergift;
end;

var
    v : vleesvarken;
begin
.
.
writeln('Opgenomen cystine + methionine is ',
    1000.0 * v.opgenomen_voer.hoeveelheid *
    v.opgenomen_voer.gehalte_cpm:8:3, ' gram');
.
.
v.eiwit := v.eiwit + 0.130; { Zet 130 gram eiwit aan. }
writeln('Eiwit is nu ', v.eiwit:8:3);
.
```

Figuur 1
Voorbeeld van de definitie en het gebruik van record-types in PASCAL

```
function gewicht(var v : vleesvarken) : real;
(*
** Componenten eiwit, vet, water en as, vormen samen het
** gewicht van het dier, exclusief de darminhoud.
** Deze darminhoud, zo wordt verondersteld, maakt 5% uit
** van het totale levend gewicht van het dier:
*)
begin
    gewicht := (v.eiwit + v.vet + v.water + v.as) / 0.95
end;

var
    v : vleesvarken;
begin
    .
    .
    writeln('Gewicht is nu ', gewicht(v):6:2);
    .
    .
```

Figuur 2
Procedures die gekoppeld zijn aan een zeker record-type

te maken hebben met de groei van gezonde vleesvarkens. In de tweede en derde fase wordt onder andere de invloed van klimaat, huisvesting en individuele variatie uitgewerkt.

Rekenmodel

Tijdens de ontwikkeling van het informatiemodel is ook een rekenmodel ontwikkeld. Het rekenmodel diende voornamelijk om de rekenregels en processen van het informatiemodel te kunnen valideren. Daarnaast kon het worden gebruikt voor de ontwikkeling van programmatuur ten behoeve van demonstratie en onderzoek. Het rekenmodel is ontwikkeld op het Proefstation voor de Varkenshouderij. Daarbij is gebruik gemaakt van Turbo-PASCAL v5.5. Deze versie van Turbo-PASCAL biedt OO-programmeermogelijkheden. De 'Demonstratie-programma's TMV', die sinds mei 1991 worden verspreid door het IKC-centraal in Ede, zijn gebaseerd op het rekenmodel.

Eind 1991 is TMV de tweede fase ingegaan. Om klaar te zijn voor de komende uitbreidingen op het informatiemodel, is in de zomer van 1991, de programmatuur grondig herzien. Wat hieronder wordt beschreven is het systeem zoals dat er voor de tweede fase uit ziet.

OOP

OOP is een reactie op de problemen die het ontwikkelen van grote informatie-systemen met zich meebrengt. Deze problemen zijn onder andere:
- slechte herbruikbaarheid van bestaande programmacode;
- slechte onderhoudbaarheid, waardoor grote kans op bugs (fouten).

Dergelijke problemen blijken vooral op te treden bij de ontwikkeling en het onderhoud van grote programma's (>10000 programmaregels) (Stroustrup, 1986).
De eerste echte OO-talen waren 'zuivere' OO-talen; dat wil zeggen: talen die speciaal waren ontwikkeld ten behoeve van OOP. Later zijn zogenaamde 'hybride' OO-talen ontstaan uit traditionele talen: C++ en Objective C uit programmeertaal C en Object-Oriented PASCAL uit PASCAL. Het voordeel van de 'hybride' OO-talen is dat bestaande code er mee kan worden gecompileerd en een organisatie niet persé geheel opnieuw hoeft te beginnen met de opbouw van een verzameling standaardprocedures.
In het vervolg van dit artikel worden de specifieke kenmerken van OOP behandeld. OOP onderscheidt zich van traditionele programmeermethoden door:
- inkapseling (encapsulation);
- overerving (inheritance);
- meervormigheid (polymorfism)

(Anonymous, 1988). Deze kenmerken worden hieronder behandeld.

Inkapseling

Hogere programmeertalen bieden de mogelijkheid om 'structures' te creëren. Een structure is een groep variabelen die als geheel een entiteit

```
type

vleesvarken = object
    eiwit, vet, as, water : real;
    opgenomen_voer : voergift;
    procedure leg_op(opleg_gewicht : real);
    procedure verstrek_voer(var vg : voergift);
    procedure verteer_voer;
    function gewicht : real;
end;

procedure vleesvarken.leg_op;
var
    leeg_gewicht : real;
begin
    leeg_gewicht := 0.95 * opleg_gewicht;
    eiwit := 0.173 * leeg_gewicht;
    vet := 0.130 * leeg_gewicht;
    as := 0.035 * leeg_gewicht;
    water := leeg_gewicht - (eiwit + vet + as)
end;

{ Methoden 'verstrek_voer' en 'verteer_voer', worden hier niet
verder uitgewerkt }

function vleesvarken.gewicht;
begin
    gewicht := (eiwit + vet + as + water) / 0.95
end;

var
    v : vleesvarken;
    d : byte;
begin
    .
    .
    d := 0;
    v.leg_op(23.0);
    while v.gewicht 105.0 do begin
    d := d + 1;
    .
    .
    v.verstrek_voer(voergift);
    v.verteer_voer
end;
writeln('Het eindgewicht op dag ', d, ' is ', v.gewicht);
.
.
```

Figuur 3
Voorbeeld van de definitie van klasse 'vleesvarken' en het gebruik ervan in programmatuur

voorstelt. Binnen PASCAL heet een dergelijk type een 'record', bij programmeertaal C een 'struct'. Figuur 1 toont een voorbeeld van de definitie en het gebruik van record-types in PASCAL. Om een procedure of functie te laten omgaan met een dergelijk record, moet het adres ervan, steeds worden doorgegeven aan die routine. Figuur 2 geeft daarvan een voorbeeld. Daar zo'n routine enkel dient om iets met een record van het betreffende type te doen, zijn het record-type en de routine sterk met elkaar verbonden. Het is daarom logisch om dat verbond tot uiting te laten komen in de definitie van een record-type, namelijk door de routine op te nemen als onderdeel van het record-type. Bij Turbo-PASCAL wordt keyword 'record' dan vervangen door 'object'. Een object-type is een record-type, waarin ook procedures en functies kunnen worden opgenomen. Procedures en functies heten dan de 'methods' van het object-type en de variabelen worden gewoonlijk de 'fields' genoemd.
Voor een goed begrip van wat gaat volgen, is het van belang, nog wat termen uit te leggen. Een 'object' is een vóórkomen van een 'object-type'. Object-type 'vleesvarken' geeft aan wat onder een vleesvarken wordt verstaan. Van dit type kunnen meerdere objects worden gedeclareerd. Een gebruikelijker term voor object-type is 'class'. Een object wordt wel een 'instance' van een class genoemd. Hierna zal van 'klassen', 'objecten', 'velden' en 'methoden' worden gesproken.
Figuur 3 geeft een eenvoudig voorbeeld van een mogelijke definitie van klasse 'vleesvarken'. De uitwerking is slechts een voorbeeld en volgt niet het 'Informatiemodel TMV'. De methoden (functies en procedures) van een object, hebben toegang tot de velden alsof het lokale variabelen zijn.
Een groot voordeel is dat methoden met eenduidige namen kunnen worden opgenomen in de klasse-definities (Bulman, 1991). Het is mogelijk om in elk van de klassen een methode 'print' op te nemen, welke een beschrijving van het object naar het scherm geeft. Zelfs namen van systeem-routines kunnen worden gebruikt (zoals 'writeln'), waardoor de leesbaarheid verder kan worden vergroot. Het samenvoegen van code (methoden) en data (velden) wordt 'inkapseling' genoemd.
Bij OOP geldt als regel dat men slechts met de objecten communiceert middels de methoden. De methoden vormen bij een goed ontworpen klasse, de interface

a

```
type

vleesvarken = object
    .
    .
    function onderhoudsbehoefte_lys : real;
    { Onderhoudsbehoefte lysine in kg/dag }
    .
    .
    procedure verteer_voer;
end;

function vleesvarken.onderhoudsbehoefte_lys;
begin
    onderhoudsbehoefte_lys := 36 * 1E-6 * pow
    (gewicht, 0.75) / 0.7
end;
```

b

```
type

onze_borg = object(vleesvarken)
    function onderhoudsbehoefte_lys : real;
end;

function onze_borg.onderhoudsbehoefte_lys;
    onderhoudsbehoefte_lys := 30 * 1E-6 * pow
    (gewicht, 0.75) / 0.7
end
```

Figuur 4
Voorbeeld van de manier waarop de herdefinitie van een methode mis kan gaan

a

```
type

tmv_zeug = object(vleesvarken)
    function pdmax : real; virtual;
    { Maximale eiwitaanzetcapaciteit in kg/dag }
    function R : real; virtual;
    { Minimale verhouding vetaanzet/eiwitaanzet }
end;

function tmv_zeug.pdmax;
begin
    pdmax := 0.130
end;

function tmv_zeug.R;
begin
    R := 1.0
end;
```

b

```
type

nieuwe_tmv_zeug = object(tmv_zeug)
    function R : real; virtual;
end;

function nieuwe_tmv_zeug.R;
var
    nieuwe_R : real;
begin
    nieuwe_R := tmv_zeug.R; { Oude definitie }
    if gewicht 60 then
    nieuwe_R := nieuwe_R + (gewicht - 60) / 100;
    R := nieuwe_R
end;
```

Figuur 5
Juist gebruik van virtuele methoden

met de buitenwereld. Het is de bedoeling dat de interne opslagstructuur nooit gebruikt wordt bij de communicatie met het object. Eén en ander is in veel OOP-talen af te dwingen.
Het verbergen van de opslagstructuur heeft als voordeel dat aanpassen ervan geen invloed heeft op de programma's die gebruik maken van een klasse. Mocht op zeker moment blijken dat het verstandig is om een linked-list opslagstructuur te vervangen door een opslagstructuur die gebruik maakt van een tabel, dan hoeft enkel de sourcefile te worden aangepast, waarin de klasse is uitgewerkt en niét alle andere sourcefiles waarin de klasse wordt gebruikt.

Bij de definitie van een klasse, wordt aangegeven welke methoden die klasse 'kent'. Die methoden kunnen functies zijn (routines die waarden teruggeven) of procedures. Er bestaan twee bijzondere vormen van procedures, namelijk constructors en destructors. Deze kunnen gebruikt worden om een object te initialiseren respectievelijk bij het opruimen van een object.
Bij de aanroep van een constructor wordt een structuur opgezet die de 'virtual method table' (VMT) genoemd wordt. Wat de VMT inhoudt wordt onder *meervormigheid* verder uitgelegd.
Constructors worden gebruikt om een object te initialiseren. Bij Turbo-PASCAL moet je de constructor (of één van de constructors, want er mogen er ook meerdere zijn) aanroepen vóór één van de andere methoden wordt gebruikt.
Een 'destructor' is een methode die gebruikt kan worden om een object netjes op te ruimen. In het algemeen heeft een klasse maar één destructor en worden bij de aanroep ervan geen parameters gebruikt.

Overerving

Het komt vaak voor dat een klasse aardig aan zekere wensen voldoet, maar dat er voor een bepaald project nog wat extra faciliteiten nodig zijn. Bij traditioneel programmeren heeft men dan de keuze uit twee mogelijkheden:

- extra faciliteiten toevoegen aan bestaande programmatuur;
- kopie maken van bestaande programmatuur en deze kopie uitbreiden.

De eerste mogelijkheid heeft als nadeel dat de ballast van de extra faciliteiten, ook wordt meegenomen in situaties waar deze niet nodig zijn en de tweede mogelijkheid heeft als nadeel dat vaak dubbel onderhoud nodig is. Het overervings-mechanisme van OOP, biedt de mogelijkheid om zonder extra ballast en zonder de noodzaak van dubbel onderhoud, een nieuwe klasse te definieren op basis van een bestaande klasse (Constantine 1990). Daarbij kunnen methoden worden toegevoegd of worden vervangen door andere methoden en kunnen velden worden toegevoegd. De nieuwe klasse erft automatisch alle velden en methoden van de 'ouder'.
Door zo veel mogelijk van dit mechanisme gebruik te maken, kan maximaal gebruik worden gemaakt van bestaande code. Dat vermindert de kans op bugs (fouten) omdat er veel minder programmatuur hoeft te worden bijgeschreven voor een nieuw project. Verder ontstaan daardoor boomstructuren van klassen. Een duidelijke tekortkoming van Turbo-PASCAL, is dat in Turbo-PASCAL alleen kan worden afgeleid van één oudertype, terwijl veel andere OOP-talen zogenaamde 'multiple inheritance' (MI) bieden en het dus mogelijk maken om klassen te definieren op basis van twee of meer 'ouder'-klassen. Borland is niet van plan om MI-mogelijkheden in te bouwen in Turbo-PASCAL. "Unlike C++, Turbo still does not support multiple inheritance. Borland claims that this capability adds too much complexity for the programmer. Perhaps, but that's a decision better left to the programmer." (Veale, 1991).

Meervormingheid

Methoden van de ene klasse, kunnen, indien gewenst, worden herschreven in een afgeleide klasse. Onder bepaalde omstandigheden geeft dit echter problemen, zie figuur 4a. Stel dat methode 'verteer_voer' in dat voorbeeld, gebruik maakt van 'onderhoudsbehoefte_lys'. Indien nu een nieuwe klasse wordt afgeleid van 'vleesvarken', zoals in figuur 4b wordt getoond, dan is het zo dat als nu een object van het type 'onze_borg' wordt gecreëerd, het direkt opvragen van 'onderhoudsbehoefte_lys' aan 'onze_borg', het verwachte resultaat zal geven. Echter de aanroep ervan door 'verteer_voer', loopt verkeerd; deze roept de oude methode aan voor de berekening van de onderhoudsbehoefte lysine. Natuurlijk kan dat ondervangen worden door 'verteer_voer' ook opnieuw op te nemen in de nieuwe klasse, maar dat druist in tegen de bedoelingen van OOP. Door nu van methode 'onderhoudsbehoefte_lys' op te geven dat deze virtueel is (voorwaardelijk), gebeurt datgene wat men wil, namelijk de aanroep van de laatst bekende variant, ook door methoden die in één van de ouder-types zijn uitgewerkt. In figuur 5 wordt een voorbeeld gegeven van het juiste gebruik van virtuele methoden. Klasse 'tmv_zeug' wordt in figuur 5a afgeleid van klasse 'vleesvarken', waarbij een tweetal virtuele methoden van klasse 'vleesvarken' wordt hergedefinieerd. In figuur 5b wordt weer een nieuwe klasse gedefinieerd, die op één punt afwijkt van klasse 'tmv_zeug'.
Bij de initialisatie van een object door middel van een constructor, wordt een structuur opgezet die de 'virtual method table' (VMT) heet. De virtual method table zorgt er voor dat bij aanroep van een virtual method, de laatst gedefinieerde versie ervan wordt uitgevoerd. Uitgaande van het voorbeeld in figuur 5, van de afleiding van 'nieuwe_tmv_zeug' via 'tmv_zeug' uit 'vleesvarken', een voorbeeld van hoe een variabele van het type 'nieuwe_tmv_zeug' in een programma kan worden toegepast:

```
z : nieuwe_tmv_zeug;
.
.
z.init;
.
.
{ Programma dat voor gegeven type iets
berekent. }
.
.
z.done;
```

Methoden 'init' en 'done', zijn hierbij de constructor respectievelijk de destructor van de klasse. Het nadeel van een dergelijke constructie is dat een afgeleid type alleen kan worden doorgerekend door een kopie van het programma te maken en het type van één van de variabelen in de source te veranderen. Het kan echter mooier als er gebruik wordt gemaakt van pointers. Pointers zijn variabelen die wijzen naar een plaats in het geheugen. Op die plaats kan bijvoorbeeld een zeker object staan. Bij de meeste OOP-talen is een constructie mogelijk, vergelijkbaar met die in het volgende stukje pseudo-code:

```
p : pointer naar object van type vleesvarken
q : pointer naar object van type vleesvarken
.
.
p := new vleesvarken
q := p
```

Op de voorlaatste regel wordt opdracht gegeven om een stukje geheugen te reserveren voor een object van type 'vleesvarken' en wordt 'p' verwezen naar dat stukje geheugen. De velden en methoden van dat object kunnen - op een iets andere manier - gewoon worden benaderd.
Na de assignment op de laatste regel, verwijzen zowel 'p' als 'q' naar het zelfde object. Tot zover is dit alles nog vergelijkbaar met programmeren in traditionele programmeertalen. Anders wordt het als het programmafragment er als volgt uit ziet:

```
p : pointer naar object van type tmv_zeug
q : pointer naar object van type vleesvarken
.
.
p := new tmv_zeug
q := p
q->leg_op(23.0)
do while q->gewicht 105.0
.
.
q->verstrek_voer(voergift)
q->verteer_voer
enddo
```

In deze pseudo-taal moet u 'q->' lezen als het object waar 'q' naar wijst. Bij juiste initialisatie van object 'p->', wordt voor dat object een Virtual Method Table (VMT) aangemaakt. Bij de 'verteer_voer'-opdracht aan 'q->', wordt inderdaad methode 'verteer_voer' van klasse 'vleesvarken' aangeroepen. Deze roept echter een aantal virtuele methoden aan, die in klasse 'tmv_zeug' anders zijn uitgewerkt dan in klasse 'vleesvarken'. Wat nu gebeurt, is dat de methoden van 'tmv_zeug' worden gebruikt en niet die van 'vleesvarken'. Dit gedrag biedt veel mogelijkheden. Een voorbeeld dat in TMV is uitgewerkt is die van een groep vleesvarkens, die bestaat uit pointers naar objecten van type 'vleesvarken'. Zo'n groep kan worden opgelegd, gevoerd en dergelijke. Door het pointer-mechanisme, kunnen er allerlei soorten vleesvarkens in worden geplaatst; een paar borgen en een paar zeugen of vleesvarkens waarvan de

eigenschappen stochastisch zijn vastgesteld. De programmatuur voor het behandelen van een groep vleesvarkens, hoeft maar één keer te worden geschreven en kan voor allerlei soorten vleesvarkens worden gebruikt. OOP biedt hier een mogelijkheid om zeer efficient te programmeren.

Afsluiting

De bouw van de programmatuur voor de eerste fase, heeft veel tijd gekost. Een belangrijke oorzaak was de onervarenheid met OOP. Al beschik je als programmeur over alle technische kennis over OOP, dan vraagt het nog een hele tijd om de mogelijkheden ervan in te zien (Lee, 1991). Zeer regelmatig is de programmatuur volledig omgegooid om nieuwe ideeën er in toe te passen. De ervaring die bij de bouw van de TMV-programmatuur is opgedaan, is toegepast bij een ander simulatieprogramma, waarvan de opzet dusdanig complex is dat deze met traditionele programmeertechnieken zeer moeilijk te programmeren is. Gebruik van OOP maakt dus de bouw van meer complexe programma's mogelijk. Object Oriented Programming maakt programmeren eenvoudiger, waardoor de kans op fouten afneemt. De programmatuur is ook beter te testen.

Literatuur

ANONYMOUS
Turbo Pascal version 5.5 Object-oriented programming guide, Borland International, 1988

BULMAN D.
Refining Candidate Objects, Computer Language, Volume 8(1), 30-39, 1991

CONSTANTINE L.
Objects, Functions, and Program Extensibility, Computer Language, Volume 7(1), 34-54, 1990

LEE E.
The Journey of a Thousand Miles, Computer Language, Volume 8(10), 45-54, 1991

STROUSTRUP B.
The C++ Programming Language, ISBN 0-201-12078-X, Addison-Wesley Publishing Company, 1986

VEALE D.
Turbo Pascal for Windows, Computer Language, Volume 8(8), 82-88, 1991

WERKGROEP TMV
Informatiemodel Technisch Model Varkensvoeding, Proefverslag P 1.66, Proefstation voor de