KMG Klynveld Kraayenhof & Co.
Automatisering & Controle-groep

COMPACT 852

Computer en Accountant

"Capabilities, het wenkend perspectief",
theorie, praktijk en toekomst van op capabilities gebaseerde computersystemen
door H. Roos

Knowledge based systems: een stap vooruit in de beheersbaarheid van administratieve processen
door A. van der Drift

Vierde generatietalen
door drs. J.E. Huizenga

Informatie over (elektronische) informatie
"Werken met een database"
door D. Boom

Nummer 38 12e jaargang lente 1985

COMPACT

COMPUTER EN ACCOUNTANT

INHOUDSOPGAVE

- Van de redactie 1
- Actualiteit 5
- "Capabilities, het wenkend perspectief", theorie, praktijk en toekomst van op capabilities gebaseerde computersystemen door H. Roos 6
- Knowledge based systems: een stap vooruit in de beheersbaarheid van administratieve processen door A. van der Drift 45
- Vierde generatietalen door drs. J.E. Huizenga 67
- Informatie over (elektronische) informatie "werken met een database" door D. Boom 77
- Tijdschriften "Computers that are "never" down" besproken door W.C. Bakker 83

nummer 38 12e jaargang lente 1985

Lente 1985

VAN DE REDACTIE

Voor ons ligt een nieuw nummer van Compact het lentenummer 1985, tevens eerste aflevering van een nieuwe jaargang, de twaalfde! Het nummer is toekomst-gericht, een schuchtere poging om min of meer onbekende gebieden in kaart te brengen.

Mogen wij, voordat wij in dit voorwoord ingaan op de afzonderlijke hoofdartikelen, eerst een rectificatie doorgeven. Wij zijn door belanghebbenden hierop attent gemaakt.

Het betreft: Rectificatie Compactnummer 37 (pag. 73).
Door een onjuiste interpretatie van de redactie is een tweetal onjuistheden opgenomen in het artikel "Toename in uitbesteding/overname automatiseringsactiviteiten".

1. Goudse Verzekeringen b.v. beschikt zelf over een eigen computercentrum met de naam Informaticum. Van verzelfstandiging daarvan respectievelijk overname door een software-house is geen sprake.

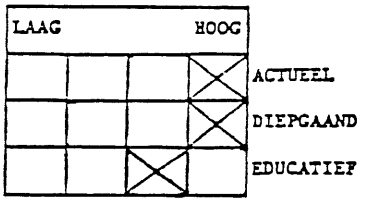
2. Niet IMDATA B.V. is van Internatio Müller N.V. naar Raet C.V. overgegaan, maar het Rotterdamse Rekencentrum van de IM-groep. IMDATA maakt daarentegen nog onveranderd deel uit van de Internatio-Müller Groep.

Onze excuses voor de onjuiste berichtgeving.

Onze keus is dit keer gevallen op het publiceren van voornamelijk eigen werk in de vorm van een viertal hoofdartikelen. Gevolgd door de bespreking van het artikel "Computers that are "never" down".

De rubrieken Boeken, Micro, Tijdschriften, ABC-Nieuws en Onderwijs kunt u weer in het komende nummer aantreffen. Door het grote tijdsbeslag van onze medewerkers uit andere hoofde is hiertoe besloten.

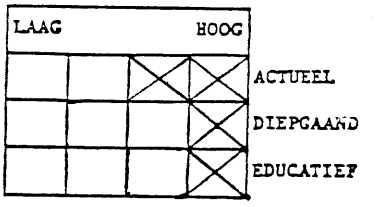
"Capabilities, het wenkend perspectief"
theorie en toekomst van
op capabilities gebaseerde computersystemen
door H. Roos



Een poging (in de zin van "essay") wordt ondernomen om enig licht te werpen op de eigenschappen en het gebruik van capabilities. Als uitgangspunt is gekozen een drietal publicaties, de oudste van 1966 en de nieuwste van 1984.
Verschillende invalshoeken die aan de orde komen zijn bescherming (protection), het gemeenschappelijk gebruik (sharing) en het verbergen van informatie (information hiding) in verband met modularisatie en de rol die op capabilities gebaseerde adressering daarbij kan spelen. Het is uitdrukkelijk niet de bedoeling om iets nieuws te poneren.
Voorop staat het etaleren van een aantal in de vakliteratuur gepubliceerde modellen van op capabilities gebaseerde systemen. Getracht is daartoe zoveel als mogelijk van Nederlandse termen en uitdrukkingen gebruik te maken. Het begrip capability is onvertaald gelaten. Als praktisch voorbeeld wordt de realisatie van een database file in het IBM System/38 behandeld.

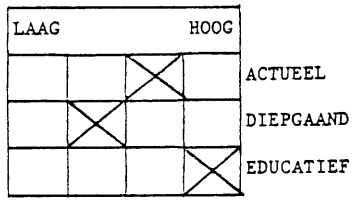
Ter inleiding en situering van de problematiek volgen eerst enige overwegingen over het onderwerp van het symposium "Certificering van software", 8 november 1984, NGI sectie EDP-Auditing". Aan het einde van deze overwegingen zal verband worden gelegd tussen het vraagstuk van de certificering van software en het gebruik van capability systemen.

"Knowledge based systems"
door A. van der Drift



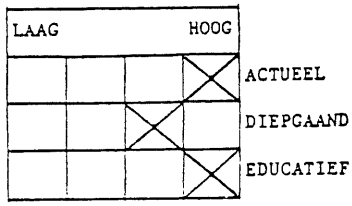
Door onder meer ontwikkelingen vanuit Artificial Intelligence, hogere verwerkingssnelheden en grotere geheugens krijgt een relatief nieuw verschijnsel binnen de automatisering gestalte onder de naam Knowledge Based Systems (KBS).
Dit artikel gaat in op deze KBS in relatie tot beheersbaarheid van geautomatiseerde administratieve processen, waarbij de stellige verwachting bestaat, dat het gebruik van KBS ten behoeve van deze processen zonder twijfel zal plaatsvinden (zo dit al niet plaatsvindt).

"4e Generatietalen"
door J.E. Huizenga



In het artikel over Vierde generatietalen wordt de invloed van deze hulpmiddelen op de betrouwbaarheid belicht, maar ook de accountant als gebruiker van deze hulpmiddelen blijft niet buiten beschouwing.

"Informatie over (elektronische) informatie"
Werken met een data base
door D. Boom



De laatste jaren ondervindt het gedrukte tijdschrift sterke concurrentie van de zogenaamde bibliografische databases, die de tijdschriftartikelen in elektronische vorm aanbieden. Hoe dat gebeurt, welke informatie, en tegen welke kosten dit geschiedt wordt in dit artikel uiteengezet.

Redactiewisseling

Het vertrek van J.E. Huizenga betekent tevens een afscheid van de Redactie van Compact.

Jur, namens de lezerskring van Compact danken wij je voor een aantal jaren succesvol redacteurschap.
Dick Steeman
Dries Neisingh
Henk van der Wielen

De plaats van Jur zal worden ingenomen door Aad H.C. Koedijk.

Lente 1985

COMPACT (R) is een uitgave van de
Automatisering & Controle-groep van
KMG Klynveld Kraayenhof & Co.

Deze informatie is in de eerste plaats bestemd voor diegenen, die in de algemene controlepraktijk werkzaam zijn. De in dit tijdschrift weergegeven meningen mogen niet altijd gezien worden als officiële zienswijzen van KMG Klynveld Kraayenhof & Co. De in de rubrieken besproken artikelen worden soms geheel opgenomen of verkort aangehaald, tevens als regel voorzien van commentaar.

Redactie:
Drs. J.E. Huizenga,
A.W. Neisingh,
Prof. D. Steeman en
H.J.M. van der Wielen (secr.).

Kopij kunt u inleveren bij de
secretaris van de redactie.
Adres:
Prinses Irenestraat 59,
1077 WV Amsterdam.
Postadres:
Postbus 7137
1007 JC Amsterdam.

Indien u belangstelling heeft voor meerdere exemplaren kunt u deze aanvragen bij de secretaris van de redactie, evenwel zolang de voorraad strekt (telefoon 020 - 5461394).

ACTUALITEITEN

KMG Computer Newsletter

De inhoud is kort weergegeven onder het hoofd **Editorial** .

Spotlight on MCS

Much positive reaction has been received on the last issue of K-Memory – an encouraging sign that the need for a greater exchange of information in the field of computer technology is a real one. A sign also that KMG partners everywhere are genuinely interested in technological developments taking place in KMG worldwide.

Since the publication of K-Memory, two computer newsletters have been sent out to Computer Audit Liaison Partners keeping them up-to-date with happenings in and around KMG. The computer newsletter is a vehicle of communication open to everybody and we would encourage KMG staff to use it as such. Information should be sent to the editor, Han Urbanus.

This edition of K-Memory takes a look at the use of EDP in management consultancy. Of particular interest is the article on the new methodology developed by KMG KKC for analysing and developing organisations' information needs and systems. Experience gained by MCS staff in the Netherlands has proved that the system, PRISMA, is an effective one, which is being further refined in the light of new requirements and discoveries.

The brief introduction to the IBM System 38 Review will be followed in the next issue of K-Memory by a report on a seminar on System 38 held in Ottawa, Canada at the end of May. If this seminar is a success, there are plans to hold more in other parts of the world. We'll keep you posted! (zie Compact, nr.37 rubriek Onderwijs. Red.)
The update on Concept Consolidation aims to inform readers of the power and scope of the new package, introduced in the February edition of K-Memory, which is now available to all KMG member firms. Whilst of particular importance to EEC firms facing the implementation of the EEC Seventh Directive, the package should be recognised as a powerful tool for improving reporting of all financial information. Parts of this article have been based on a KMG book on consolidation – written by John Baillie, KMG Glasgow – to be released in autumn 1985.

Finally, to keep you in touch with what KMG firms are providing in the field of computer services – an introduction to KMG Hungerfords' exciting new venture 'MicroLab', and KMG U.K.'s computer service bureau which specialises in direct mailing. If your firm offers specialised computer services, please let us know about them so that we may pass on the information to others.

De juni-editie (nr. 2) is kortelings beschikbaar gekomen.
K-memory wordt uitgegeven door het Bureau Praktijkontwikkeling van KMG Executive Office onder supervisie van J.H. Urbanus KMG CASC.
Het blad is in eerste aanleg voor intern gebruik binnen KMG. Voor belangstellenden - ook voor onze cliënten - is een exemplaar beschikbaar.
U kunt een exemplaar bij de redactie van Compact aanvragen.

Symposium "Certificering van software", Jaarbeurs, Utrecht, 8 november 1984, NGI sectie EDP-Auditing.

"Capabilities, het wenkend perspectief", *

Theorie, praktijk en toekomst van op capabilities gebaseerde computersystemen.

door

Herman Roos RA

SAMENVATTING

Een poging (in de zin van "essay") wordt ondernomen om enig licht te werpen op de eigenschappen en het gebruik van capabilities. Als uitgangspunt is gekozen een drietal publicaties, de oudste van 1966 en de nieuwste van 1984. Verschillende invalshoeken die aan de orde komen zijn bescherming (protection), het gemeenschappelijk gebruik (sharing) en het verbergen van informatie (information hiding) in verband met modularisatie en de rol die op capabilities gebaseerde adressering daarbij kan spelen. Het is uitdrukkelijk niet de bedoeling om iets nieuws te poneren. Voorop staat het etaleren van een aantal in de vakliteratuur gepubliceerde modellen van op capabilities gebaseerde systemen. Getracht is daartoe zoveel als mogelijk van Nederlandse termen en uitdrukkingen gebruik te maken. Het begrip capability is onvertaald gelaten. Als praktisch voorbeeld wordt de realisatie van een database file in het IBM System/38 behandeld.

1. Inleiding

Ter inleiding en situering van de problematiek volgen eerst enige overwegingen over het onderwerp van dit symposium. Aan het einde van deze overwegingen zal verband worden gelegd tussen het vraagstuk van de certificering van software en het gebruik van capability systemen.

Mogelijkheden en beperkingen

Welke verwachting kan iemand die kennis neemt van een software certificaat koesteren? Waarom zou hij überhaupt behoefte aan zo'n certificaat hebben?
Een mogelijke reden is dat hij een zo groot mogelijke zekerheid wil hebben dat een bepaald software produkt zijn prijs waard is.

---

* Dit hoofdstuk is opgenomen in de Proceedings van het Symposium.

Hij heeft bepaalde verwachtingen omtrent de bruikbaarheid van het produkt. Die verwachtingen zullen zijn gewekt door de produktbeschrijving; de commerciële belofte wat het zo al voor mogelijkheden biedt, verpakt in een verkoopbrochure. Dit zal hij hebben getoetst aan een probleem waar hij een oplossing voor zoekt.
De vraag is nu of de gekoesterde verwachting redelijkerwijs wordt gehonoreerd.

De totstandkoming van elk software produkt wordt gekenmerkt door één of meer vertalingen.
De maker of zijn opdrachtgever (hier kunnen zo al ten minste twee zeer verschillende situaties voorkomen; standaard software voor de open markt en maatwerk dat voor een specifieke situatie wordt gemaakt) heeft een idee dat wordt uitgewerkt in een produktspecificatie, op zodanige wijze dat er over gepraat kan worden. De vorm van die specificatie zal noodzakelijkerwijs informeel zijn. Dit te meer daar het steeds de bedoeling zal zijn om met het te maken software produkt een probleem op te lossen.
Het zal immer gaan om het behandelen van informatie, of het nu om informatie voor de besturing van een bedrijf in ruime zin gaat of b.v. om de besturingsinformatie voor een industriële robot.
Dit betekent inpassing in een organisatorisch proces dat uiteindelijk wordt gedragen door mensen van vlees en bloed en een brein waarin opgeslagen een cultuurpatroon dat sterk mede bepalend is voor de interpretatie van de produktspecificatie en voor de beoordeling van de mate waarin het produkt daaraan voldoet.
Het gaat er nu onder meer om of de software maker in staat is om de specificaties zodanig om te zetten in een werkend produkt dat het in overeenstemming is met het verwachtingspatroon van de koper/opdrachtgever.

Ten minste twee vertaalslagen zijn hiervoor nodig.
De produktspecificatie moet worden omgezet in een formele specificatie, die (sterk vereenvoudigd) vervolgens door een vertaalprogramma wordt omgezet in een werkend produkt. Aan het in gebruik nemen gaat voorts nog vooraf de afbeelding van het "programma" op het werkgeheugen (reëel of virtueel, dat maakt in wezen geen verschil) van de computer. De consequentie van dat laatste is dat de vertaling in de vorm van uitvoerbaar programma moet geschieden m.b.v. een vertaalprogramma (compiler) die behoort bij (is afgestemd op) het specifieke computersysteem waarop het programma zal gaan worden uitgevoerd (computersysteem = hardware + besturingssoftware).

Tijdens elke "overgang" wordt informatie geïnterpreteerd. Dit betekent dat aan de formele data, die de invoer van de "overgang" belichamen, betekenis wordt gehecht. In de eerste slag wordt een "informele" produktspecificatie omgezet in een formele. Hiermee wordt bedoeld dat de eerste specificatie is gesteld in een "taal" waarvan vorm noch betekenis eenduidig zijn.

De bedoeling van de eerste vertaalslag is het elimineren van dit gebrek aan eenduidigheid door omzetting in een programmeertaal met een eenduidig gedefinieerde vorm en betekenis. Tijdens deze omzetting bepaalt de software maker een vaste formele betekenis van de oorspronkelijke specificatie. Hij interpreteert de specificatie zoals hij de bedoelingen van de specificeerder begrijpt.

Een belangwekkende vraag is in hoeverre de specificatie impliciet uitgaat van de context van de specificeerder (zijn "universe of discourse") en of het mogelijk is dat verschillende specificeerders denken en dus specificeren vanuit niet signifikant verschillende contexten. Dat laatste zou dan doorgetrokken kunnen worden naar de software maker. Kan zijn universe of discourse zo lijken op dat van de specificeerder dat daaruit een hoge mate van gelijke interpretatie volgt?
Het antwoord op deze vraag is in sterke mate bepalend voor de mogelijkheid om te komen tot specificaties die in meer dan een concrete context een zinvolle betekenis hebben. Dat dit in enige mate het geval moet zijn wordt geïllustreerd door het omvangrijke aanbod van standaardpakketten, zowel voor eindgebruikertoepassingen als voor besturingssoftware. Dit is tevens de ratio voor een eventueel software certificaat.

De functie van zo'n certificaat is het wegnemen van onzekerheid over de kwaliteit van een pakket bij de potentiële koper/gebruiker. Als hij het pakket eenmaal heeft gekocht, kan hij zelf wel vaststellen of het gekochte het hem mogelijk maakt op de verwachte wijze zijn probleem op te lossen.

Tot zover echter alleen het passen van de mogelijkheden van het pakket bij de specifieke situatie van de gebruiker.
Er schuilen echter nog andere problemen in het gebruik van programma's. Deze problemen worden veroorzaakt door de moeilijkheden waarmee de omzetting van informele naar formele taal gepaard gaat.
Het is in de praktijk moeilijk gebleken om een dergelijke omzetting foutloos te verrichten. De compiler kan hierbij behulpzaam zijn. Voor zover er is gezondigd tegen de vormregels zal een goede compiler dat aangeven. Dat geldt echter slechts voor de beoordeling of een bepaalde "bewering" in de taal volgens de regels om uit de taalelementen beweringen te construeren inderdaad een geldige bewering is. Dat zegt niets over de logische samenhang tussen de achtereenvolgende beweringen waaruit het gehele programma is opgebouwd. Daarvoor is het nodig dat de taal tevens beschikt over een stel regels die de betekenis van combinaties van beweringen definieert.
De in de praktijk gebruikte formele talen voldoen in verschillende mate aan deze wenselijkheid. Zo kunnen talen met sterke en met zwakke typering van constructies worden onderscheiden.

Waar moet een software certificaat zich over uitspreken? Kan het ooit meer zijn dan een testrapport dat nauwkeurig de testomstandigheden specificeert en de testuitkomsten weergeeft? Daar heeft een gebruiker, die per definitie onvoldoende deskundig is om de resultaten op hun voor zijn situatie juiste waarde te schatten, niet veel aan. Richt een certificaat zich op een gebruiker met een bepaald niveau van deskundigheid? Hoe dat aan te geven? Wat wordt in de praktijk zoal aangeboden op de open software markt? Toepassingen, meestal geparameteriseerd en geschikt voor een beperkt aantal computersystemen. Besturingssystemen, eigenlijk niet los van specifieke hardware. De op het gebied van de besturingssoftware aangeboden produkten betreffen vrijwel steeds uitbreidingen van bestaande besturingssystemen of de vervanging van bepaalde functies daarvan.

Steeds is een concreet produkt bedoeld voor een eenduidig gedefinieerde combinatie hardware/software. Dit geldt bv. ook voor een produkt als het besturingssysteem QNIX en het daarvoor beschikbare postbussysteem QMAIL. Naar onze ervaring werkt die combinatie niet vlekkeloos. Het is echter in elk concreet geval niet steeds duidelijk of dat komt door het produkt of door de wijze waarop het is geïnstalleerd en wordt gebruikt. Dat is blijkbaar ook een facet van het probleem. Wat moet de certificeerder hieraan doen?

Verwoording van het certificaat

Zo bezien lijkt het alsof het bij de huidige stand van de wetenschap uitsluitend mogelijk is om te komen tot meer of minder sterk geclausuleerde certificaten.

Het kan nuttig zijn na te gaan wat de systematiek is van "normale" accountantsverklaringen. Zonder op volledigheid te willen bogen volgt daartoe een beknopte samenvatting.
Voor een verklaring die betrekking heeft op een financiële verantwoording zijn als strekking slechts toelaatbaar: 'goed', 'fout' met de mededeling wat er fout is en wel op zodanige wijze geformuleerd dat daaruit blijkt hoe de verantwoording goed zou zijn geweest, een 'non-oordeel' dat slechts toelaatbaar is bij z.g. objectieve verhindering, hetgeen wil zeggen dat het op grond van een of meer in de non-mededeling nauwkeurig te vermelden specifieke omstandigheden niet mogelijk is om tot een goed of fout te concluderen. Bij uitzondering zijn 'voorbehouden' mogelijk mits de invloed op de verantwoording nauwkeurig wordt aangegeven hetgeen evenals bij de foutmededeling tot gevolg heeft dat uit de formulering blijkt hoe de verantwoording er uit zou moeten zien om het predicaat 'goed' te verdienen. In wezen dus twee uitersten: 'goed' of 'non-oordeel'.

De term non-oordeel zij niet negatief opgevat als zou de accountant niets hebben gedaan. Het tegendeel is in dergelijke -schaarse- gevallen waar. Het impliceert dat de accountant het mogelijke heeft gedaan en dat tevens geen tekortkomingen zijn geconstateerd. Een non-oordeel kan dus voor de lezer wel degelijk een belangrijke positieve betekenis hebben.

De systematiek van accountantsverklaringen strekt zich ook uit over andere door accountants gegeven oordelen dan die over financiële verantwoordingen. Daarvoor is reglementair de term "mededeling" gereserveerd. Over de toepassing op het gebied van de interne controle-aspecten van automatiseringsorganisaties en geautomatiseerde systemen is een NIVRA publicatie beschikbaar (23). De opzet daarvan is te komen tot een zekere mate van gelijkheid in de formuleringen van door accountants afgegeven mededelingen over de betrouwbaarheid en de continuïteit van geautomatiseerde gegevensverwerking.

Uit de daarmee inmiddels opgedane ervaringen blijkt dat er duidelijk sprake is van een kloof tussen de verwachtingen van de lezer van min of meer sterk geclausuleerde mededelingen (of certificaten!) en hetgeen daaruit af te leiden valt. Een gebruiker wil eigenlijk net als van een financiële verantwoording eenvoudig weten of het goed is of niet. Kampert (24) stelt dan ook de vraag of er in wezen geen sprake is van "een kloof tussen rationele behoeften en het vermogen van de accountant". Met evenveel recht zou daar echter aan toegevoegd kunnen worden de vraag of die op zich rationele behoeften wel reëel zijn in het licht van de stand van de techniek. En hier is niet alleen bedoeld de controletechniek van de accountant doch vooral ook de techniek van het maken en beheersen van geautomatiseerde informatiesystemen.

Die clausuleringen vormen even zo vele open deuren.
Geen enkele weldenkende (in de zin van rationeel) lezer van zo'n mededeling zal uit een positief oordeel over een automatiseringsorganisatie inclusief de daarin operationele informatiesystemen afleiden (de verwachting putten) dat de daaruit afkomstige informatie steeds juist zal zijn. Dit kan op logische gronden nimmer met een voor een dergelijke mededeling uitgevoerde audit worden bereikt. Een open deur dus. Maar wel een belangrijke waarover geen misverstand mag bestaan.
Bij verklaringen over jaarrekeningen doet zich eenzelfde verschijnsel voor. Zo'n verklaring biedt geen garantie dat de desbetreffende onderneming ook over het volgende boekjaar weer op zijn minst eenzelfde resultaat zal behalen.
Niettemin worden verklaringen over financiële verantwoordingen en meer in het bijzonder over jaarrekeningen niet van dergelijke "open deur" clausules voorzien. De voorbehouden die zijn verwoord in de clausules worden geacht axiomatisch te zijn en als zodanig aan elke weldenkende lezer bekend. Het doet dan ook in de optiek van de lezer van een hiervoor bedoelde mededeling wat krampachtig aan dat daar wel uitdrukkelijk wordt geclausuleerd. Zou dit achterwege blijven dan is daarmee

althans een deel van Kampert's bezwaren ondervangen. De vraag is hoe weldenkend de lezer geacht kan worden te zijn. Er zou best sprake kunnen zijn van een kloof tussen de mogelijkheden van geautomatiseerde informatiesystemen "an sich" en de verwachtingen die een deel der "weldenkende" lezers daarover koestert.
Essentieel is dat deze verklaringen en mededelingen steeds betrekking hebben op concrete unieke situaties die aan een bepaalde context zijn gebonden. Bij software certificatie gaat het echter om produktcertificering los van de context van een eventueel gebruik.

Het is zeer de vraag of zich in de huidige praktijk voldoende concrete gevallen voordoen waarin een in simpele bewoordingen gesteld oordeel mogelijk is om pogingen om tot een (software) certificaat als produkt te komen te rechtvaardigen. Gevreesd moet worden dat het toepassen van de stelling dat om iets uit te leggen het "zo eenvoudig mogelijk moet worden voorgesteld, doch niet eenvoudiger", indien toegepast op de formulering van een oordeel over software, noodzakelijkerwijs resulteert in een inherent complexe formulering ter overbrugging van de kloof tussen droom en werkelijkheid. Gecompliceerde problemen laten zich niet vangen in eenvoudige taal zonder oversimplificatie die de waarheid geweld aandoet.
Je mag evenwel een gebruiker van een certificaat niet opzadelen met een reeks clausules waarvan hij de draagwijdte nauwelijks kan vaststellen of beoordelen. Het komt erop neer dat de formulering van een mededeling slechts voor de lezer waar die voor bestemd is, begrijpelijke, duidelijke en daarmee bruikbare informatie bevat (Kampert 24). Dit betekent dat daarin opgenomen voorbehouden de strekking van de verklaring of mededeling niet mogen aantasten. De strekking moet ondubbelzinnig en duidelijk zijn. Kan dat? Is de kloof overbrugbaar?

Hoe kan het systeem van accountantsverklaringen en -mededelingen nu worden toegepast op software?
In NIVRA 26 is de kunstgreep toegepast om een geautomatiseerd systeem analoog te behandelen aan een (financiële) verantwoording. Waarom kunstgreep? Wel omdat een verantwoording over een uitgevoerde opdracht nu eenmaal niet hetzelfde is als het resultaat van de opdracht zelf.

Een verantwoording over een gevoerd financieel beheer moet zodanige informatie bevatten dat de opdrachtgever kan beoordelen of het beheer doeltreffend en doelmatig is geweest.
Doelmatig t.a.v. een software produkt heeft twee kanten. De verbruikte middelen om het produkt te maken en de doelmatigheid van de werking van het produkt zelf. Dat laatste is een aspect van de doeltreffendheid.
Nu is aan een produkt zelf niet te zien of het doelmatig is, daarvoor zal het in zijn werking moeten worden waargenomen.
Een verschil met een financiële verantwoording is dat die zich beperkt tot de weergave van de in geld gemeten economische gevolgen van gevoerd beheer. Dit aspect staat duidelijk naast het produkt als gevolg

van gevoerd beheer. Het gaat in wezen om een kwaliteitsoordeel over het produkt, naast de financiële verantwoording over de aangewende middelen.
Hiertussen liggen nog de exploitatiekosten. In het geval van software, de kosten van gebruik gedurende de totale gebruikstermijn (life-cycle).

Om te kunnen beoordelen of een produkt doeltreffend is moet dus niet alleen het functionele doel doch tevens het nagestreefde niveau van exploitatiekosten van tevoren gegeven zijn als onderdeel van de opdrachtformulering.
Let wel, het onderzoek naar de doelmatigheid kan zich nog verder uitstrekken en wel naar de mogelijkheden tot het bereiken van een grotere doelmatigheid dan à priori in de opdrachtformulering (voor de bouw van de software) is gegeven. Hier is bedoeld het voldoen aan de produktspecificaties zowel in functioneel als in operationeel opzicht.
Indien het een specifieke opdracht betreft bevat de produktspecificatie mogelijk beide elementen. In het geval van beoordeling en vergelijking van bij voorbeeld een aantal verschillende tekstverwerkingssystemen worden de operationele aspecten geheugenbeslag en snelheid wel onderzocht maar alleen vergelijkenderwijs. Het oordeel is impliciet relatief; er is geen norm in de vorm van een à priori produktspecificatie. Die is althans niet aan de onderzoeker bekend.

Gegeven de te vervullen functies en het niveau van exploitatiekosten moet de software certificeerder trachten vast te stellen of en in hoeverre het te certificeren produkt aan de gegeven normen voldoet.
Naast deze specifieke normen moet tevens worden vastgesteld in hoeverre het produkt betrouwbaar is.

Het op juiste wijze in een organisatie inpasbaar zijn wordt geacht onderdeel te zijn van de functionaliteit. We hebben kennelijk te maken met het produkt-als-black-box waar het de functionaliteit betreft en met het produkt-in-zijn-samenstelling waar het betreft de onderhoudbaarheid in de zin van binnen redelijke grenzen aanpasbaar aan de zich wijzigende eisen van de functionaliteit. Een uitermate star produkt zal immers in de tijd gezien een te geringe aanwendingsmogelijkheid hebben. Het produkt-in-zijn-samenstelling is tevens bepalend voor de betrouwbaarheid en de inspanning, benodigd voor het aanbrengen van wijzigingen in de functionaliteit als belangrijk deel van de exploitatiekosten. De samenstelling is tevens in hoge mate bepalend voor de mate waarin aanpassingen mogelijk zijn zonder de doelmatigheid van het gebruik in de zin van snelheid, het andere deel van de exploitatiekosten, aan te tasten.

Functionaliteit en doelmatigheid kunnen worden getoetst door het simuleren van het gebruik. De aanpasbaarheid en de doelmatigheid waarmee dat kan vereisen echter analyse van de produktsamenstelling. Diezelfde produktsamenstelling is tevens in hoge mate bepalend voor de betrouwbaarheid van de werking.

Verband met capabilities

Wat is nu de relatie met het begrip capability? De certificeerder zal moeten vaststellen of het produkt goed in elkaar zit. Hij zal de samenstelling moeten toetsen aan algemeen geldende principes voor de bouw van software. Dergelijke algemene principes zijn gericht op het beheersen van de bouw en het onderhoud van programma's. Uitgangspunt daarbij is dat het voor de mens onmogelijk is om een programma boven een bepaalde omvang en complexiteit als een eenheid te begrijpen, d.w.z. in zijn werking te doorgronden uitsluitend op basis van de programmatekst.

Een van de principes die worden toegepast om dit op te lossen is de verdeling van programma's in modules die elk klein genoeg zijn om wel begrepen te kunnen worden. De wijze waarop dat dient te geschieden om het beoogde verbeterde inzicht te verkrijgen is het object van de tak van computerwetenschap "software engineering".

Een basisprincipe bij de verdeling van een programma in afzonderlijke modules is dat de afzonderlijke modules zo los als mogelijk van elkaar moeten zijn. Elk module moet vervangen kunnen worden door een andere versie met een totaal andere interne structuur zonder dat dit enig effect heeft op de overige modules van hetzelfde programma. Hetzelfde principe kan worden toegepast op programma's die te zamen één systeem vormen. De informatie die specifiek is voor de interne werking van een module dient volledig te zijn afgeschermd van alle overige modules. Dit principe van "information hiding" (Parnas 22) vereist dat het voor een bepaalde module niet mogelijk is om buiten zijn eigen geheugengebied te adresseren. Om echter met andere modules te kunnen samenwerken is de overdracht van informatie tussen modules nodig. Hiertoe moeten twee samenwerkende modules over een gemeenschappelijk adresseerbaar geheugengebied beschikken. Dit levert voor modules die een zodanige functie binnen het geheel van het programma vervullen dat ze met een groot aantal modules moeten samenwerken uiteraard een probleem op. Moet er voor elk paar een communicatiegebied zijn of kunnen ze alle van hetzelfde gebied gebruik maken? Hoe kan worden voorkomen dat ze elkaars data beïnvloeden?

Deze en dergelijke vragen kunnen met de traditionele wijze van geheugenbescherming zoals b.v. in het IBM 370 complex niet tot een afdoende oplossing worden gebracht. Een daarvan fundamenteel afwijkende computerarchitectuur die uitgaat van capability based adressering biedt daarvoor wel perspectieven. Een groot praktisch probleem voor de huidige praktijk is echter dat het overgrote deel van de toegepaste commerciële software niet voor capability machines is gemaakt, waardoor de opmars van capability based adressering slechts geleidelijk zal kunnen geschieden.

Opmerkelijk is bv. dat de GUIDE organisatie geen System/38 gebruikers als lid accepteert. De motivering hiervoor heeft een interessante ontwikkeling doorgemaakt. Aanvankelijk werd het System/38 beschouwd als een mini in dezelfde lijn als de 34 en later de 36. In het acroniem GUIDE staan de letters IDE voor "integrated data equipment". Dit type valt daar volgens dit criterium niet onder. Het is overigens de vraag of menige 43xx installatie wel aan dit criterium voldoet. Nu echter de grootste 38 niet meer onder doet voor verschillende machines uit de 43xx lijn wordt merkwaardigerwijs als argument gebruikt dat de "filosofie" van de 38 zozeer verschilt van hetgeen men gewend is dat samengaan maar tot spraakverwarring zou leiden!! Er lijkt dus duidelijk sprake van op conservatisme gebaseerde weerstand. Alle reden derhalve om het fenomeen capability based adressering meer bekendheid te geven.

Opgemerkt moet echter worden dat een gebruiker van een op capabilities gebaseerd systeem als bv. System/38 zich geenszins behoeft te verdiepen in de achterliggende geheimen. Die zijn afdoende verstopt. Hij heeft slechts te maken met de merendeels positieve effecten, niet met de wijze waarop die dank zij de architectuur van het systeem zijn gerealiseerd. Een fraai voorbeeld van "information hiding".

De voordelen van capability based adressering zijn door Linden (3) als volgt samengevat:

Capability based adressering biedt de mogelijkheid tot:
- kleine protectie domeinen
- extended type objecten
- flexibele sharing

Kleine protectie domeinen bevorderen:
- het gebruik van extended type objecten
- systeem veiligheid
- betrouwbare programma's

Extended type objecten bevorderen:
- systeem veiligheid
- betrouwbare programma's

Flexibele sharing bevordert:
- betrouwbare programma's.

Het principe van capability based adressering zal worden behandeld tegen de achtergrond van het adresseringssysteem van het IBM MVS computersysteem. Dat is redelijk representatief te achten voor non-capability based adressering. Een beschrijving van dat adresseringssysteem is niet in de doorlopende tekst opgenomen maar in een afzonderlijke appendix (zie blz. 41).

2. Verschillende definities van capability

Het woord capability is niet goed in het Nederlands te vertalen. Mogelijke vertalingen zouden kunnen zijn "mogelijkheid" of "potentie". Het begrip heeft evenwel een zo specifieke betekenis gekregen dat het beter is het woord capability ongewijzigd in een in het Nederlands gestelde tekst over te nemen.
Begonnen zal worden met het vermelden van enige door verschillende auteurs gegeven definities. (De vertalingen zijn van mijzelf en eventuele onjuistheden of onduidelijkheden komen dan ook voor mijn rekening.)

Linden (3), 1976 geeft als definitie van een capability:
"een teken dat wordt gebruikt voor het identificeren van een object en waarvan het bezit bepaalde gebruiksrechten op dat object impliceert. Het kan worden beschouwd als een toegangsbewijs. Wijziging van een capability is niet toegestaan behalve het beperken van de impliciete gebruiksrechten. Het reproduceren van een capability (in de zin van kopiëren) is echter in tegenstelling tot een (b.v. theater-) toegangsbewijs wel toegestaan".

Fabry (2), 1974 definieert een capability als:
"een absoluut adres voor een virtueel object. Het systeem is vrij om het virtuele object te herplaatsen ("relocate") op voorwaarde dat de overeenstemming tussen het object en zijn capability blijft gehandhaafd".

Corsini e.a.(18), 1984 definiëren een capability als:
"een paar (AR,ID), waarin AR (Access Right) staat voor een verzameling van mogelijke toegangsrechten en ID (IDentifier) een segment identificeert, wordt een capability genoemd. Een verzameling van capabilities specificeert de wijze waarop een deelverzameling van de segmenten van een gemeenschappelijk gesegmenteerd virtueel geheugen (commom segmented virtual memory) kan worden benaderd, d.w.z. specificeert een toegangsdomein d(h)".

Deze definities zijn niet zonder meer duidelijk. De essentie is echter de begrenzing van een object als een afzonderlijk protectie domein en een uniek adres voor dat object dat, eenmaal vastgesteld, nimmer meer kan worden gewijzigd.

3. Performance en beveiliging

Een van de meest omstreden aspecten van capability based adressering is of het nu wel of niet mogelijk is om op basis van een dergelijke architectuur een aanvaardbare performance te bereiken.

Jones (13) die capability based adressering kortweg synoniem acht met "protected pointer"-adressering gaat er blijkbaar impliciet van uit dat het toepassen van protected pointers het bij uitstek geschikte mechanisme is voor de realisatie van een capability based adresseringssysteem. Dat hoeft echter niet per sé.
Er is ook een andere oplossing mogelijk, gebaseerd op de verdeling van het geheugen in segmenten die capabilities kunnen bevatten en segmenten die data (inclusief programma-instructies) bevatten.
Jones behandelt oplossingen voor het vermijden van performance verlies dat wordt veroorzaakt door het scheiden van capabilities en data in verschillende objecten.

In HYDRA (9) wordt dit opgelost door een scheiding aan te brengen tussen de interne representatie van een object en de wijze waarop de programmeur het ziet. Voor de programmeur is het een object. Om de integriteit van de capabilities te handhaven bestaat de interne representatie echter uit een afzonderlijke capability vector naast een data vector. In HYDRA zijn de operaties op de data vector alleen via de kernel') mogelijk om te voorkomen dat data operaties onrechtmatig op capabilities worden uitgevoerd. Jones signaleert echter dat een data-operatie daardoor ongeveer 300 maal zoveel tijd kost in vergelijking met dezelfde data operatie direct als machine-instructie.

') Kernel = hart van het besturingssysteem ook wel nucleus genaamd. (Red.)

Het spreekt vanzelf dat er naar oplossingen is gezocht voor dit probleem. Jones geeft er twee aan. Een voorstel van Wilkes (14), waarbij gebruik wordt gemaakt van "tags" en de methode, toegepast in het STAROS (8) operating systeem, die berust op het bijhouden van de grens die de capabilities scheidt van de object data binnen de vector waarin een object is gerepresenteerd. Bij adressering wordt een extra "bounds check" uitgevoerd om te voorkomen dat capabilities onrechtmatig worden geadresseerd en gemanipuleerd.

Die laatste oplossing is mogelijk in computers die geen speciale hardware ondersteuning bieden voor capability adressering. Waarschijnlijk zou die in het HYDRA systeem ook op die wijze mogelijk zijn. Dit betekent echter dat de veiligheid van een dergelijke uitvoering van capability adressering minder moet zijn dan bij specifieke hardware ondersteuning mogelijk zou zijn (zie over dit probleem ook Lopriore 4). Bounds checking wordt immers uitgevoerd door een operating systeem-functie. Die moet wegens de security gevoeligheid uiteraard in de kernel worden ondergebracht. Nu is een kernel in beginsel wel beschermd, doch de vraag blijft of dat evenveel beveiliging biedt als directe hardware-ondersteuning. Hardware-ondersteuning zal altijd uit moeten gaan van een of andere vorm van tagging. Dit wordt bevestigd door Myers (11) en Wilkes (14) en is gerealiseerd in het IBM System/38 (19) en in de iAPX 432 (6).

4. Domeinbescherming; het model van Dennis en Van Horn

Hierna volgt een samenvatting van het model van Dennis en Van Horn. De door hen gegeven definities en gebruikte notatie van meta-instructies is niet strak gevolgd. Getracht is de essenties zo duidelijk mogelijk in het Nederlands weer te geven. Geheel buiten beschouwing blijft b.v. de behandeling van verschillende soorten van bijzondere condities als inbreuken op bevoegdheden, overloop etc.

De term meta-instructie wordt gebruikt om aan te geven dat het krachtiger instructies betreft dan de normale machine interface instructies. In dit verband is een opmerking van Lopriore (4) van belang nl. dat er met de vertaling van hoog niveau broncode in laag niveau object code in de regel veel van de op het toepassen van software technieken als inkapseling (zie hierover verder) in de broncode getroffen maatregelen weer verloren gaan. Effectieve toepassing van dergelijke technieken vereist machine-ondersteuning. Dit impliceert de noodzaak van meta-instructies op machine interface-niveau.

Het model van Dennis en Van Horn geeft een goed inzicht in het programmagebruik van capabilities. Het gaat uitdrukkelijk over de betekenis (semantics) van capabilities en niet over de realisatiemogelijkheden. Aan het eind van de samenvatting volgt een beschrijving van de gedachte werking van het model.

Uitgangspunt van het model van Dennis en Van Horn vormen een aantal definities die de karakteristieke elementen beschrijven van "multigeprogrammeerde berekeningen".
Dit begrip kan synoniem worden geacht met "parallelle processen".
Ze maken onderscheid tussen "berekeningen" (computations) en processen. Een proces is een abstract ding dat door de instructies van een procedure (programma) beweegt terwijl de procedure wordt uitgevoerd door een processor.
Elke "berekening" vindt plaats binnen een beschermde ruimte (sphere of protection), die wordt bepaald door een lijst met capabilities of kortweg een C-list.
Een "berekening" is een verzameling processen met een gemeenschappelijke C-list zodanig dat alle processen die dezelfde C-list gebruiken leden zijn van dezelfde "berekening".
Elk object kan worden geadresseerd via een capability. Er zijn vier typen capabilities.

1. Segment capabilities bevatten procedures of data.
2. I/O functies verzorgen het verkeer van data tussen intern geheugen en de periferie.
3. Protected entry points (kortweg entry points) maken verkeer mogelijk tussen verschillende beschermde ruimten die elkaar niet kunnen vertrouwen.
4. Directories bevatten de externe namen van permanente objecten en de daarmee geassocieerde capabilities die wijzen naar een segment, een i/o functie, een entry of een andere directory.

Elke systeemgebruiker (in de terminologie van Dennis en Van Horn een "principal") heeft een eigen oorsprong-directory (root directory), die het beginpunt is van de hiërarchie van de permanente objecten waarvan hij eigenaar is.
Een permanent object is adresseerbaar via een deelnaam die bestaat uit de ketting van namen via welke het object bereikbaar is vanuit een bepaalde directory. Deze directory structuur lijkt veel op het UNIX file system (21).

Voor een systeemgebruiker die van het systeem gebruik wil maken wordt door de supervisor (dat is de combinatie van hardware en software die "berekeningen" creëert en meta-instructies uitvoert) een C-list gemaakt met een capability voor de oorsprong-directory van die gebruiker. Met de volgende meta-instructies kan hij capabilities:

Toevoegen:
acquire<directory naam, capability naam>

Verwijderen:
remove<directory naam, capability naam>

Een object creëren:
create<segment, i/o functie, entry, directory>

Het object heeft daarna noch slechts een type aanduiding maar nog geen externe naam. Die naam wordt eraan toegevoegd door het object permanent te maken met de meta-instructie:

place<directory naam, capability naam, index nummer>

(In praktische systemen kunnen deze beide instructies worden samengevoegd; zie b.v. System/38.)

Als resultaat van een meta-instructie die een capability toevoegt aan de C-list wordt het indexnummer van die capability die immers in de C-list staat bekend aan het proces en kan vervolgens worden gebruikt voor het adresseren van het object of voor het permanent maken ervan (conform System/38 en Corsini e.a.).

Het verwijderen van permanente objecten geschiedt met de meta-instructie:

delete<directory naam, capability naam>

Het gebruik van een directory van een andere systeemgebruiker is mogelijk door het toevoegen van de betreffende directory capability met behulp van de meta-instructie:

link<systeemgebruikers naam>

Uiteraard kunnen capabilities uit die vreemde directory slechts worden gebruikt indien dat door hun eigenaar uitdrukkelijk is toegestaan door ze voor vrij gebruik te bestemmen. De capability heeft dan de toegangsindicator F en de bijbehorende gebruiksindicator geeft de aard van het toegestane gebruik aan (uitvoeren, lezen, schrijven en combinaties daarvan). (In principe is een "link" met elke directory mogelijk. Echter alleen effectief voor "free" capabilities; vgl. "public" objects in System/38.)

Voor het gebruik van een capability van een andere systeemgebruiker op ad hoc basis (dus niet via een "free" of "public" capability) wordt gebruik gemaakt van de entry-capability en de meta-instructie:

enter<indexnummer1, indexnummer2>

Het volgende voorbeeld illustreert het gebruik hiervan. Stel dat proces A de eigenaar is van capability X en dat proces B van die capability gebruik wil maken. De verschillende stappen zijn schematisch weergegeven in figuur 1. Proces A en proces B worden kortweg aangeduid als A en B.

1. B begint de operatie met het creëren van een normale segment capability die we Z zullen noemen met de meta-instructie:

   i:=create, segment(Z)

   Deze capability is van tijdelijke aard en alleen bekend onder het indexnummer. Om verwijzing mogelijk te maken zullen we als naam Z gebruiken. B zet in het segment zijn verzoek aan A om de capability X te mogen gebruiken en wat de aard van het verlangde gebruik is.

2. A creëert een "free-entry" capability in een van zijn directories die we eveneens A zullen noemen. Dit stoort niet omdat de namen van de processen niet in de meta-instructies van het voorbeeld behoeven te worden gebruikt. A voert hiertoe uit de meta-instructie:

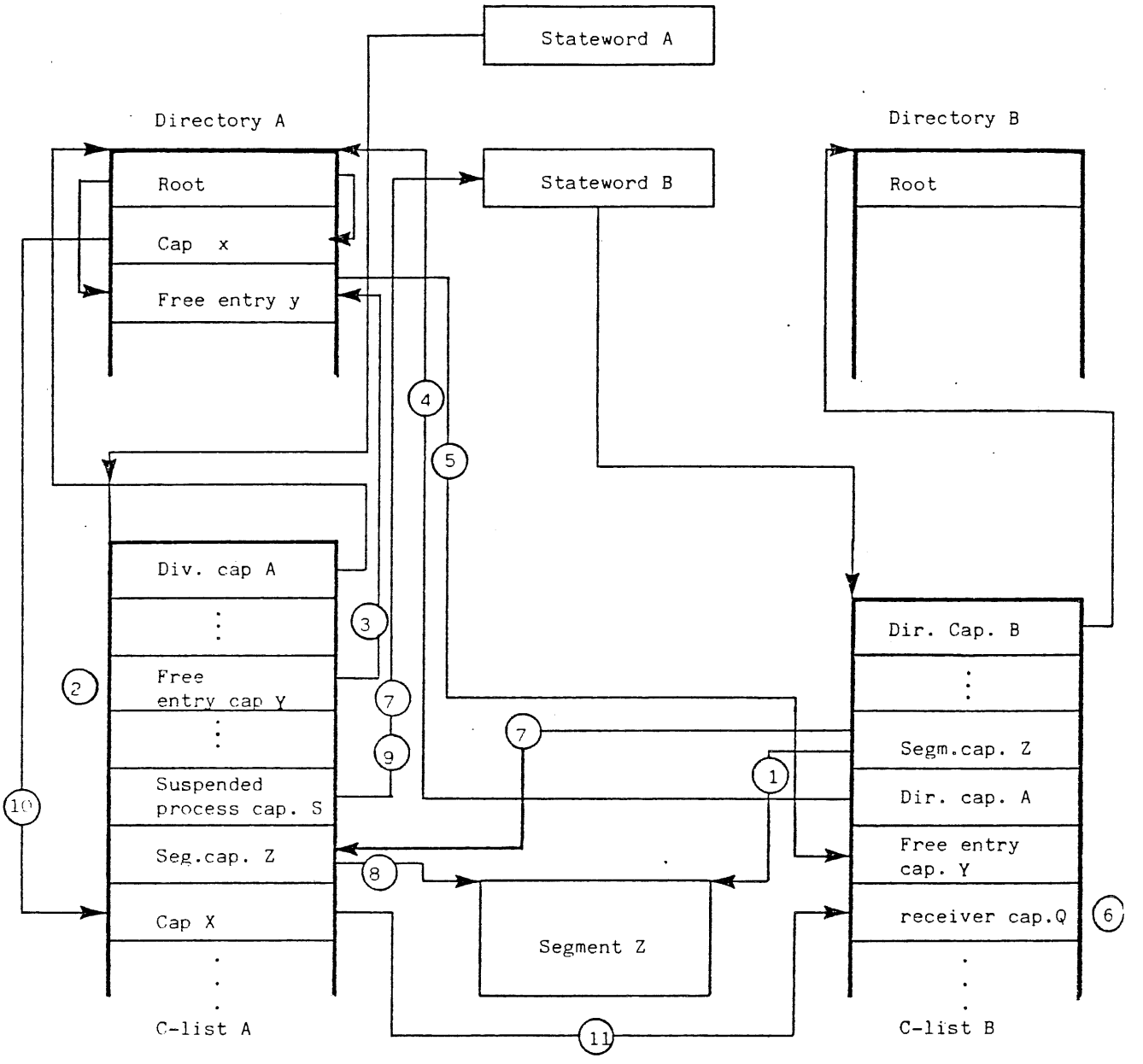
   i:=create, entry (F)

gevolgd door

3. Het permanent maken ervan met de meta-instructie:

   place, A, Y, i

   Hierin is Y de externe naam van de zojuist gecreëerde entry capability die bij het creëren als toegangsindicator een F (free) heeft meegekregen ten teken dat hij vrij gebruikt mag worden. A is de directory waarin de nieuwe "free-entry" wordt opgenomen en i is het indexnummer van de gecreëerde capability in de C-list van het proces A. Het indexnummer wordt ter vereenvoudiging verder niet gebruikt in dit voorbeeld.



1 B: Create Z
2 C: Create Y
3 A: place y
4 B: link A
5 B: acquire A.Y
6 B: receive Q
7 B: enter Y, Z
8 A: read Z
9 A: owner S
10 A: acquire X
11 A: transmit B,Q,X

Voorbeeld domein-overschakeling volgens DVH

figuur 1

4. B voegt een directory-capability toe aan zijn C-list door het uitvoeren van de meta-instructie:

   link, A

   Hierna heeft proces B toegang tot de directory van A en kan capabilities van A die als vrij zijn gekenmerkt aan zijn eigen C-list toevoegen.

5. B voegt nu de zojuist door A gecreëerde free-entry capability aan zijn C-list toe met de meta-instructie:

   acquire, A, Y

   Het zal duidelijk zijn dat proces B de namen A en Y moet kennen om deze acties te kunnen uitvoeren. Daar moeten afspraken over bestaan tussen A en B.

6. B treft vervolgens voorbereidingen om de verlangde capability in zijn C-list te kunnen ontvangen door het creëren van een z.g. receiver-capability met de meta-instructie:

   i:=receive

   Deze capability is evenals het segment Z tijdelijk van aard en krijgt daarom geen externe naam. Voor het voorbeeld zal in plaats van het indexnummer de fictieve naam Q worden gebruikt.
   Alles is nu gereed voor A om de capability X te vragen. Dit geschiedt door

7. Het uitvoeren door B van de meta-instructie:

   enter, indexnummer Y, indexnummer Z

   Het effect van deze instructie is dat het proces B wordt opgehouden en een nieuw proces wordt gestart. De C-list van het nieuwe proces is de C-list van proces A, die als onderdeel van de enter-instructie wordt aangevuld met een capability S die verwijst naar het statuswoord van het opgehouden proces B en met een kopie van de segment capability Z uit de C-list van proces B. Proces B wacht nu en proces A gaat verder.

8. A leest de informatie uit Z, stelt vast wat B wil en

9. Controleert de identiteit van B met de meta-instructie:

    s:=owner

    Deze instructie zoekt de naam van de systeemgebruiker die eigenaar is van het proces B, via de aan de C-list van A toegevoegde capability S die naar het statuswoord van het opgehouden proces B verwijst. A kan nu nagaan of B volgens eerder gemaakte afspraken bevoegd is om op de verlangde wijze van het gevraagde segment X gebruik te maken. Hiertoe zal A b.v. over een tabel beschikken waarin die informatie staat. Blijkt B over voldoende bevoegdheden te beschikken dan

10. Voegt A eerst de gevraagde capability aan zijn eigen C-list toe met de meta-instructie:

    acquire, directory A, capability X

en

11. Stuurt die vervolgens door naar B met de meta-instructie

    transmit, process B, receiver Q, indexnummer X (N)

    Deze operatie vervangt de receiver capability Q door de capability X met de eigendomsaanduiding N die staat voor "niet-eigenaar".

12. Daarna geeft A met de meta-instructie:

    continue

    aan dat de operatie is voltooid. A wordt opgehouden en B wordt hervat.

    Het model werkt tijdens processing met woordadressen die bestaan uit:

    index, offset

De index is de waarde die het resultaat is van de uitvoering van een meta-instructie waarmee een capability aan de C-list wordt toegevoegd. De functie van de naam van de capability als absoluut adres vereist dat het reële adres voor alle actieve C-lists identiek is
Indien de totale adresruimte voldoende is (stel > 2**40) kan dat worden bereikt door het adres eenmaal, tijdens object-creatie vast te stellen en in de directory op te nemen. Vertaling in een reëel adres dient vervolgens te geschieden door een afzonderlijk mechanisme.

Wat nu indien het object een andere plaats in het reële geheugen krijgt toegewezen, terwijl er C-lists van meer dan een actief proces naar verwijzen?
Dit is geen probleem; er is immers altijd een functie nodig die het woordpaar <index, offset> vertaalt in het reële adres. Voor een proces bestaat slechts het woordadres dat echter door elk afzonderlijk proces uit dezelfde directory regel in zijn eigen C-list is geplaatst.

5. Gemeenschappelijk programma- en data-gebruik; de overwegingen van Fabry

Fabry (2) geeft als definitie:
"een capability is een speciaal soort adres voor een object, dat alleen door het systeem kan worden gecreëerd en dat, om het object te gebruiken, het uitsluitend via zo'n adres kan worden geadresseerd" (2-p.403).

Fabry behandelt capability-based adressering vanuit de wenselijkheid om in een multiprogrammeringsomgeving de mogelijkheid te hebben om verschillende processen vrijelijk van elkaars programma's en data-gebruik te kunnen laten maken.
Het probleem is in het kort dat het proces - via het programma dat de procesafwikkeling bepaalt - gebruik wil maken van data of van een ander programma, die tot dan geen deel uitmaakten van de status van het eigen proces.

Stel dat beide, data en programma, zich wel reeds in het geheugen bevinden en dus reeds deel uitmaken van de processtatus van een of meer andere actieve processen. Een proces kan alleen van die data en dat programma gebruik maken indien het in staat is om ze te adresseren. Zowel data als programma's plegen echter telkens wanneer ze voor een actief proces in het geheugen worden gebracht op een andere plaats terecht te komen. Die plaats hangt af van de op dat moment beschikbare vrije ruimte. Tijdens het plaatsingsproces worden de beginadressen van de verschillende delen van een programma in absolute vorm geplaatst in de operand velden van de programma-instructies die deze startadressen in het juiste register plaatsen en worden de juiste waarden van de adressen in sprongopdrachten in de betreffende programma-instructies geplaatst.
Voor het proces waarvoor het programma is geladen blijven die adressen voor de duur dat het programma van het proces deel blijft uitmaken ongewijzigd.
Indien nu in een dergelijk programma een sprong naar een niet geladen programma voorkomt kan de adreswaarde niet worden bepaald en is, gesteld dat het plaatsingsproces niettemin zou lukken, de operand van de sprongopdracht onbepaald.
Fabry gaat een aantal praktische mogelijkheden na om dit probleem op te lossen.

Het probleem wordt aangeduid als "verwijzing naar gedeelde segmenten". Voor een oplossing moet het in beginsel mogelijk zijn elk object van berekening in een afzonderlijk virtueel geheugensegment op te nemen.
Uitgangspunt is dus adressering gebaseerd op segmenten van in principe variabele lengte.
Achtereenvolgens gaat hij in op verschillende mogelijkheden tot het interpreteren van het segmentadres van een gedeeld segment door alle processen die het gebruiken en wel

- op een functioneel uniforme wijze;
- op het gebruik van speciale koppelingssegmenten die het mogelijk maken een adresverwijzing naar een ander object indirect om te zetten in een voor het actieve proces correcte waarde;
- een uitbreiding van die indirecte verwijzingsmethode door het toepassen van meerdere segmenttabellen per proces en
- tenslotte het gebruiken van capabilities als adresseringsmethode.

De eerste drie methoden hebben alle het nadeel dat ze een centrale besturing vereisen. Dynamische wijziging van de context van een actief proces is daarbij niet of in zeer beperkte mate mogelijk. Voor de details wordt verwezen naar het artikel zelf (2).

Een capability is zo'n absoluut adres voor een virtueel object.
Het door Fabry gesignaleerde probleem werd veroorzaakt door relatieve adressering ten behoeve van het op doelmatige wijze in het interne geheugen kunnen herplaatsen van programma's. Het werken met absolute adressen neemt de oorzaak in feite weg.
De doelmatigheid behoeft niet in gevaar te komen indien het gebruik van absolute adressen geschiedt in de virtuele adresruimte. Het is zeer goed mogelijk om daarnaast een volkomen transparant mechanisme te gebruiken om virtuele adressen in reële te vertalen. Het herplaatsen van een object in het virtuele geheugen is ook nog mogelijk mits het verband tussen het object en zijn virtuele adres blijft gehandhaafd.

Fabry behandelt capabilities als een oplossing voor efficiënt gemeenschappelijk gebruik van objecten van computerprocessen.
Capabilities hebben als voordeel boven andere manieren van adresseren dat de interpretatie van een capability als adres volledig onafhankelijk is van de context van het proces dat de capability gebruikt om een object te adresseren. Een capability is in feite een absoluut adres voor een object.

Door het toepassen van adresrelocatie in andere adresseringsmethoden worden relatieve adressen absoluut gemaakt voor de duur van die relocatie. Een volgende relocatie zal de relatieve adressen weer opnieuw absoluut maken. De waarde van de adressen is afhankelijk van de beschikbare vrije adresruimte en de daaruit door de supervisor die de relocatie verzorgt gekozen segmenten of pagina's.
Dit geldt zowel voor een adresseringsmethode die de te reloceren adressen direct afbeeldt op het interne geheugen als voor een virtuele adresseringsmethode die de te reloceren adressen afbeeldt op een virtueel geheugen.

6. Kritiek van Fabry op Denis en Van Horn

Fabry stelt dat het oorspronkelijke adresseringsschema van Dennis en Van Horn (1) niet voldoet aan het door hem gestelde criterium, dat, om te bereiken dat voor het adres van elk object een capability als basisonderdeel wordt gebruikt, gebruikersprogramma's in staat moeten zijn om capabilities naar behoefte te kunnen opnemen in hun permanente gegevensstructuren (uiteraard onder handhaving van de integriteit van de capabilities) (2-p.403).
Deze kritiek berust op de interpretatie van het model van Dennis en Van Horn dat capabilities uitsluitend worden opgenomen in C-lists, die de beschermde ruimte van een berekening bepalen (1-p.145).

Elke capability in een C-list bestaat uit een (logische) pointer naar een object. De index van een capability in een C-list correspondeert met de naam van het object. Door DVH wordt echter in een noot ter realisering van het door Fabry gestelde criterium gesuggereerd om met elk object een unieke code te associëren op het moment dat het object wordt gecreëerd (1-p.145). Bij de behandeling van de organisatie van de directories en de naamgeving van objecten wordt voorts uitdrukkelijk gesteld dat "de naam van een object nimmer kan worden gewijzigd voor de duur van het bestaan van het object" (1-p.151).
Elk lid van een directory bestaat uit een associatie van een naam en een capability. Een capability bevat een "pointer" naar een object, een eigendomsindicator, een toegangsindicator en een indicator voor het soort gebruik dat is toegestaan (1-p.152).
Het ligt mijns inziens voor de hand ervan uit te gaan dat het de bedoeling is dat de capability in de directory dezelfde is als die in de C-list. Dat blijkt ook uit de gegeven voorbeelden. Impliciet gaat het model derhalve uit van een zeer royale (virtuele) adresruimte, die unieke adressering mogelijk maakt.

Het afbeelden en het adresseerbaar maken van de objecten op reëel geheugen wordt door Dennis en Van Horn niet uitdrukkelijk aan de orde gesteld. De enige verwijzing hiernaar is dat "verondersteld wordt dat de toewijzing van indexnummers (die refereren aan een segment) geschiedt door het systeem gedurende de uitvoering van een berekening" (1-p.152).
Dit wekt inderdaad de suggestie dat toewijzing van unieke adressen niet tijdens de creatie van een object doch op het moment van gebruik door een proces geschiedt.
Dat wordt veroorzaakt door het gebruik van het woord "allocation" dat door mij is vertaald met toewijzing. De Webster geeft als betekenis: to apportion for a specific purpose or to particular persons or things. Dit is te vertalen als: toedelen met een speciaal doel of aan bepaalde personen of dingen. Het is niet duidelijk of er is bedoeld toedelen van nummers aan objecten of dat bedoeld is toedelen van nummers aan geheugenplaatsen. Deze onduidelijkheid wordt veroorzaakt door het ontbreken van een derde naamval in de zin. Hieruit moge blijken hoe belangrijk het begrip naamval kan zijn. Belangwekkende beschouwingen hierover zijn te vinden in Charniak en Wilks (5).

Het idee van de C-list is afkomstig van de "program reference table" in de Burroughs B5000 computer en is enigszins te vergelijken met het "access segment" in de Intel iAPX 432 (6-p.2-9/13) en met de "User profiles" in de IBM System/38 (22-p.2-35/47). Mapping respectievelijk via de "segment table" en via een "context" (= directory).

7. "Information hiding" en "encapsulation"; het model van Corsini e.a.

Een recente publicatie van de hand van drie Italianen Corsini, Frosini en Lopriore (4) beschouwen capability based adressering in het licht van de realisatie van abstracte objecten.

Onder een abstract data type is in dit verband te verstaan een programma dat bepaalde functies kan vervullen ten behoeve van een ander programma en waarbij de interne werking van dat programma volledig is afgeschermd voor de programma's die er gebruik van maken.
Een dergelijk programma of object bevat een interne data structuur die verborgen is gehouden voor de buitenwereld. Zo een data structuur wordt ingekapseld genoemd naar de door Parnas (22) gebruikte term bij de introductie van het begrip "verbergen van informatie" (information hiding) als fundamentele en noodzakelijke techniek bij het verdelen van programma's in modules.

Corsini e.a. gaan ervan uit dat voor het toepassen van abstracte objecten in een programmeertaal (en dus door middel van een compiler) essentieel is dat de inkapseling van de data van het abstracte object wordt gehandhaafd gedurende de gehele levensduur van het object in een beschermde omgeving.
Het is duidelijk dat de handhaving als zodanig vereist dat er sprake is van een beschermde omgeving. Deze kan worden geboden door een adresseringssysteem dat is gebaseerd op capabilities.

Corsini e.a. definiëren een capability als:
"een segmentnaam en een verzameling toegestane operaties op dat segment" (naam is gebruikt als vertaling voor identifier).
Een verzameling van capabilities bepaalt de grenzen van een (beschermd) domein.

Dit betekent dat een proces dat loopt in een bepaald domein uitsluitend segmenten kan gebruiken die deel uitmaken van dat domein.
Een combinatie van een proces en het domein waarin het loopt wordt "subject" genoemd. Het is niet goed mogelijk om hier een ander woord voor te gebruiken als onderwerp of persoon omdat het een kunstmatig begrip is dat nauwkeurig is gedefinieerd en benoemd. Een proces moet van domein kunnen wisselen of in de termen van Corsini e.a. een subject moet een ander subject kunnen binnengaan ("enter").

Subjecten worden verhinderd om capabilities te vervalsen. Nieuwe segmenten worden op aanvraag door het systeem beschikbaar gesteld. Subjecten kan de bevoegdheid worden verstrekt om capabilities over te brengen van een domein naar een ander domein. Deze twee aspecten worden verder niet behandeld.

Uitgegaan wordt van een segmentnaam van 48 bits, hetgeen voldoende is om elke 10 microseconden gedurende 75 jaren een nieuw segment en dus een nieuwe capability aan te maken. Dit biedt de garantie dat elke segmentnaam voor zijn gehele bestaan binnen het systeem uniek is.
Het begrip subject verschilt van het begrip eigenaar (principal) van Dennis en Van Horn. Een subject is een proces in een domein. Een "principal" is de eigenaar van een directory. Eigenaar kan zowel een programma als een persoon zijn.
Alle objecten bevinden zich in geheugensegmenten, ook capabilities. De verzameling van alle capabilities die een domein bepalen bestaan uit een basis-capability segment als wortel(root) en daarvan afhankelijke hulp-capability segmenten.

8. Het principe van "information hiding" met gebruikmaking van capabilities

Alvorens in te gaan op een voorbeeld dat is gebaseerd op het model van Corsini e.a. zal eerst in algemene zin worden ingegaan op het begrip "information hiding".
Wat is de essentie?
Dat functie x gegevens nodig heeft.
Of die nu moeten worden berekend, waarbij x niet zelf "weet" hoe, of worden geselecteerd uit reeds beschikbare gegevens, maar x weet niet waar die zich bevinden, of een combinatie van beide, is voor x niet van belang. Het enige van belang is dat functie x "weet" dat functie y de verlangde gegevens kan verstrekken. (In theorie zouden de gevraagde gegevens een string met uitvoerbare code kunnen zijn; als zodanig wordt dit principe reeds lang toegepast bij b.v. het toevoegen van routines aan een programma door alleen de naam van die routine op te geven, de compiler of de linker doet dan de rest).
Om van functie y gebruik te kunnen maken moet y de vraag van x ontvangen en "begrijpen". Er moet dus tenminste een gemeenschappelijke "taal" worden gebruikt.

In Cobol is dat bijvoorbeeld de Call-conventie, waarbij een parameterlijst wordt "doorgegeven".
Daarmee kunnen zelfs specifieke procedures van de aangeroepen functie worden geselecteerd. Dat is niet de bedoeling van "information hiding". Met andere woorden de Call-conventie is zeer vrij toe te passen. Bovendien wordt een aangeroepen functie (in Cobol-termen een subprogramma) gedurende het "vertaal"proces (in ruime zin, dat wil zeggen met als resultaat een uitvoerbaar programma) deel van het aanroepende programma en daarmee van hetzelfde protectie domein!

Dit is van veel principiëler betekenis. Het impliceert dat het in beginsel mogelijk is om vanuit elk programmadeel iedere programma-variabele en -constante te adresseren.

Met capabilities kan in de eerste plaats de effectuering van de call worden uitgesteld tot het moment van de call tijdens een reëel proces. Dit komt doordat elk "object" een vast adres heeft en er geen relocatie behoeft te geschieden. Relocatie is bij het gebruik van vaste adressen zinloos.
Vervolgens kan de bevoegdheid van het aanroepende programma naar behoeven worden ingeperkt. Een capability is immers een naam met daaraan gekoppeld een of meer bevoegdheden.

Stel nu dat x bevoegd is om y uit te voeren. Dat is niet voldoende omdat de enige communicatie tussen x en y dan zou bestaan uit het starten van het (virtuele) proces dat door y wordt gedetermineerd zonder enige input en zonder expliciete bestemming van het resultaat van het proces y. Het is met andere woorden nodig om instructies mee te geven op basis waarvan dat proces zal gaan werken. y heeft daarvoor variabelen nodig als data om mee te werken (bijvoorbeeld te toetsen aan interne data van y, of een berekening uit te voeren en het resultaat terug te geven).

In het Cobol-geval komt het bijvoorbeeld voor dat het aanroepende programma als een van de parameters zijn eigen volledige data-structuur meegeeft. Daarmee zijn voor y in beginsel alle data van x manipuleerbaar. Dit is evenmin de bedoeling van "information hiding".

Dit kan worden opgelost door een afzonderlijk object te creëren dat fungeert als "postbus".
B.v. x creëert (dynamisch!) een tijdelijk object en vult dat met de boodschap voor y. Vervolgens geeft x aan y de bevoegdheid om de postbus te legen en te vullen (te lezen en te schrijven).
Hoe kan dat?

Dit vereist een speciale instructie waarmee x een capability kan toevoegen aan de verzameling capabilities van y. Hiertoe moet x kunnen schrijven in het object van y waarin de capabilities van y staan. Dus ofwel x moet vooraf een specifiek recht daartoe hebben, of dat recht moet algemeen zijn d.w.z. voor alle systeemgebruikers gelden ofte wel voor het "publiek". Als aan die voorwaarde is voldaan kan x de functie y aanroepen. Dat kan indien x een capability heeft voor y met de bevoegdheid "uitvoeren".

Hoe "weet" y echter waar de boodschap staat? Het "programma" y heeft nu een bevoegdheid die binnen het programma niet bekend is. Er ontbreekt blijkbaar nog iets aan de procedure.

Het is evident dat een van de eerste acties van y zal moeten zijn het creëren van een referentie naar de postbus. Het is m.a.w. noodzakelijk dat de communicatie tussen x en y vooraf exact is afgesproken. y moet in zijn programma de naam van de postbus weten, anders is het niet mogelijk er aan te refereren. De naam van de postbus zal in y dus een constante zijn. De capability die y in concreto gebruikt is echter een unieke naam die bij creatie van de postbus door x door het systeem is bepaald en die voor de gehele levensduur van de postbus geldt. Op deze wijze adresseert y via eenzelfde constante naam steeds een andere postbus.

Hieruit blijkt ook nog eens waarom de adresruimte van een capability based adresseringssysteem zo omvangrijk moet zijn. Ook alle tijdelijke objecten moeten een uniek adres toegewezen krijgen.

Het zal duidelijk zijn dat op deze wijze ook een universele oplossing wordt geboden voor het gemeenschappelijk gebruik van dezelfde programma's voor tegelijkertijd verschillende eindgebruikers. Voor elke eindgebruiker wordt een eigen proces gecreëerd dat een eigen kopie van de datastructuur heeft. Voor het programma hebben al die verschillende datastructuren dezelfde naam. De systeemnamen of capabilities zijn echter verschillend.

De enige "kennis" die de programma's x en y gemeenschappelijk hebben is de naam van de postbus en de grammatica van de boodschappen die ze via die postbus uitwisselen. (Grammatica omvat zowel de vorm als de betekenis.)

Uiteraard dienen er wel maatregelen te worden getroffen om te voorkomen dat y niet de verkeerde postbus adresseert. y kan immers door verschillende programma's worden gebruikt of wat op hetzelfde neerkomt door verschillende processen die elk van programma x gebruik maken.

Het moet daarom mogelijk zijn om af te dwingen dat de reeks operaties die door x wordt uitgevoerd vanaf de creatie van de postbus en het geven van de capability daarvoor aan y tot het lezen van het antwoord van y en de vernietiging van de postbus, als een ononderbroken reeks wordt afgehandeld.
Zou de reeks halverwege worden onderbroken en b.v. ten behoeve van een ander proces het programma x opnieuw worden gestart, dan zouden er twee capabilities met weliswaar verschillende systeemnamen doch met dezelfde bij de programma's bekende naam bestaan en daardoor geen eenduidige adressering meer mogelijk zijn.

Een eenvoudige oplossing in dit geval zou kunnen zijn dat het proces dat het eerst het programma x start voorkomt dat een tweede proces het gebruikt door er een exclusief slot op te leggen. Dat zou dan moeten gebeuren door het programma dat x aanroept.

Een andere mogelijkheid is om de beschreven reeks te behandelen als een "critical region". Een tweede proces zou b.v. wel alvast een postbus kunnen creëren doch nog niet de betreffende capability aan y mogen geven. In dit geval zou de actie door het programma x moeten worden uitgevoerd.

9. Het model van Corsini e.a.

Corsini e.a.(18), 1984 definiëren een capability als: een paar (AR,ID), waarin AR staat voor een verzameling van mogelijke toegangsrechten (Access Rights) en ID een segment identificeert, wordt een capability genoemd. Een verzameling van capabilities specificeert de wijze waarop een deelverzameling van de segmenten van een gemeenschappelijk gesegmenteerd virtueel geheugen (commom segmented virtual memory-CVM) kan worden benaderd, d.w.z. specificeert een toegangsdomein d(h). (Hierin staat (h) voor een index bij d, waarbij voor h elke kleine letter kan staan. De subscript werkt niet op de voor dit stuk gebruikte printer.)

Een proces, aangeduid als p(i) moet, en kan slechts, binnen een domein d(h) blijven. Dit houdt in dat het alleen segmenten kan adresseren die in d(h) zijn gespecificeerd.
Een paar (p(i), d(h)) wordt een subject s(i,h) genoemd.
Een proces kan in een ander domein werken indien het overschakelt van domein d(h) naar domein d(k). Dit overschakelen wordt genoemd het betreden van het subject s(i,k) door het subject s(i,h).
Om een bepaald segment te kunnen adresseren moet een subject behalve over de segment ID in de domeindefinitie tevens beschikken over het recht om de betreffende operatie op dat segment uit te voeren.

Een praktische voorwaarde voor de realisatie is dat uitsluitend het systeem segment IDs kan vaststellen en dat die segment IDs voor de totale levensduur van het segment gelden. Daartoe wordt het CVM geacht voldoende groot te zijn. Een adresgrootte van 48 bits biedt b.v. de mogelijkheid om gedurende 75 jaar elke 10 microseconden een nieuw adres als segment ID te genereren.
Capabilities worden opgeslagen in speciale capability segmenten. De segmenten die een domeindefinitie bevatten bestaan uit een basis capability segment (BCS) en een of meer auxiliary capability segmenten (ACS). De toegangsrechten op capability segmenten kunnen zijn het recht om te lezen en het recht om een capability op te slaan, resp Take en Grant.

Om van domein te wisselen is het nodig om een capability te hebben voor een ander BCS.

Voor de adressering worden speciale capability registers CR0,CR1, CR2,....CRn gebruikt om de capabilities, die daadwerkelijk door een proces worden gebruikt, op te slaan.
Bij de start van een proces wordt de capability voor het segment dat de eerstvolgende uit te voeren instructie bevat in CR0 geladen en de capability voor het BCS van het betreden subject in CR1. Vervolgens wordt het eerste deel van de programmateller (PC) geladen met het adres van CR0 en het tweede deel zal de verplaatsing (offset) binnen het geadresseerde segment bevatten. Bij de start van het programma in het segment waarnaar CR0 wijst zal die waarde 0 zijn.

De eerste activiteiten van het gestarte programma zullen het laden van de benodigde capabilities in de capability registers omvatten. Dit kan doordat CR1 de capability bevat die nodig is om die benodigde capabilities uit het capability segment naar een capability register te kunnen overbrengen.
Hiervoor is de instructie

loadcap<CR(i),W,CR(j)>

beschikbaar. De eerste operand is het capability register met de segment ID van het capability segment, W is de offset binnen het segment en de derde operand is het capability register waar de geadresseerde capability in wordt geplaatst indien de toegangsrechten in CR(i) daarvoor voldoende zijn, d.w.z. de bevoegdheid Take omvatten. Het springen naar een ander programma kan met behulp van de instructie

jump<CR(i),W)>.

Het effect daarvan is het laden van het adres van het capability register CR(i) en de offset W in de programmateller.
Andere instructies komen aan de orde bij het hierna te geven voorbeeld van het toepassen van information hiding d.m.v. een extended object type.

10. Voorbeeld van een extended data type gebaseerd op het model van Corsini e.a.

Het opvragen van debiteurengegevens met behulp van een beeldscherm toepassing

De eindgebruiker werkt met een programma waarmee hij een vraag kan samenstellen. Dit programma omvat onder meer een validatieprocedure die aangeeft of er een fout is gemaakt in de vraagstelling en wat in concreto fout is. Een vraag wordt alleen dan uitgevoerd indien hij correct is geformuleerd.
Dit proces speelt zich af in het domein van de eindgebruiker E(i), waarbij i staat voor de individuele gebruiker i. Dat domein is gedefinieerd in een basis capability segment E(i) (BCS/Ei).

Van de capabilities die zich in het BCS E(i) bevinden zijn voor het voorbeeld van belang de capability "vraagstelling", L, S, (L=lezen, S=schrijven), de capability "enter", die wijst naar het BCS/Deb.vr. en de capability "postbus", T, G, waarbij T staat voor "Take" en G voor "Grant" die resp. de bevoegdheid tot lezen en opslaan van een capability inhouden.

Het BCS/Deb.vr. definieert het domein waarbinnen de vraag wordt geanalyseerd en het antwoord wordt samengesteld. BCS/Deb. bevat een aantal capabilities voor resp. een initialisatieroutine, een routine die de vraag analyseert en bepaalt welke routines het antwoord moeten samenstellen, een capability voor de file waarin de debiteurengegevens zich bevinden en een capability voor een "auxiliary capability segment" (ACS/Deb) dat dient voor de ontvangst van de vraag en het doorgeven van het antwoord. Naar dat laatste segment wijst de capability "postbus" in het BCS/E(i).

Zolang E(i) bezig is met het formuleren van zijn vraag bevat het capability register CR0 de capability voor het vraagprogramma en wijst CR1 naar het BCS/E(i). De vraag wordt door het vraagprogramma opgeslagen in het segment "vraagstelling". Zodra nu de vraag compleet en correct is moet de beantwoording in het domein van BCS/Deb geschieden. Daartoe moet van domein worden gewisseld.

BCS/E(i) bevat hiertoe een routine die de capability "postbus" voor ACS/Deb laadt in CR2

```
loadcap <CR1,W,CR2>
```

CR1 wijst naar BCS/E(i), W is de index in BCS/E(i) die de over te dragen capability bevat, CR2 bevat de bestemming.
In het model is het niet mogelijk om capabilities direct van CRi naar CRj over te dragen. Vandaar dat dit via de capability segmenten moet verlopen.

Vervolgens wordt de capability voor het segment "vraagstelling" overgedragen aan ACS/Deb:

```
transfer <MSK Read,Write CR1,W,CR2,0>
```

Hierin bevat CR1 de capability voor BCS/E(i) en is W de index die de plaats aangeeft waar in BCS/E(i) de capability staat voor het segment "vraagstelling". In CR2 staat de capability voor het lege ACS/Deb. 0 geeft de index in het ACS/deb aan waar de capability voor het segment "vraagstelling" wordt opgeslagen. De bevoegdheden van de overgedragen capability zijn beperkt tot lezen en schrijven. Het is dus b.v. niet mogelijk dat deze capability via het BCS/Deb wordt overgedragen aan b.v. domein E(j) van een andere eindgebruiker.

Vervolgens wordt de capability voor het BCS/Deb in CR3 geladen met

```
loadcap <CR1,W,CR3>
```

Deze capability heeft alleen het recht "enter". Met de volgende instructie wordt dan ook de domeinomschakeling gerealiseerd.

```
enter <CR3,0>
```

Dit heeft tot gevolg dat alle capability registers van het tot dan actieve domein E(i) worden geconserveerd op de proces stack, dat de capability voor de eerste routine van het complexe object Deb wordt geladen in CR0 en dat de capability voor het BCS/Deb wordt geladen in CR1. De eerste routine moet het nu actieve proces Deb initialiseren. Dit omvat achtereenvolgens:
- laden van de capability voor de debiteurenfile in CR2,
- laden van de capability voor het ACS/Deb in CR3,
- laden van de capability voor "vraagstelling" in CR3,
- laden van de capability voor het analyseprogramma in CR4,
- starten van het analyseprogramma door de inhoud van de programmateller te wijzigen met de instructie

```
jump <CR4,0>
```

Het analyseprogramma analyseert de vraagstelling via CR3, bepaalt welke routines nodig zijn voor de beantwoording en laadt de betreffende capabilities. Het antwoord wordt opgeslagen in "vraagstelling" en via de instructie

```
reenter
```

wordt weer teruggeschakeld naar het domein E(i).

11. Een praktijkvoorbeeld: het IBM System/38

De machine-interface van het System/38 is op een zodanig hoog niveau gebracht dat de machine-interface-programmeur slechts te maken heeft met een enkel geheugenniveau. De instructie-operanden zijn dan ook steeds geheugenadressen. Het is niet mogelijk om registers te adresseren, die zijn voor de programmeur onzichtbaar.

De instructieset kan worden verdeeld in drie groepen:
- "normale" byte georiënteerde instructies voor het manipuleren van data;
- object georiënteerde instructies voor het creëren, vernietigen en adresseerbaar maken van objecten en
- een aantal speciale instructies voor de creatie en de wijziging van objecten van het type "user profile". Deze laatste kunnen alleen worden gebruikt door processen met toereikende bevoegdheden.

De beveiliging van de capabilities berust op het gebruik van storage tags, waarop hierna zal worden ingegaan. Overigens wordt voor het beveiligingssysteem verwezen naar Roos (7).

De System/38 machine interface ofte wel instructieset bevat uitsluitend instructies met pointers als operanden. Het is niet mogelijk om op het niveau van de instructieset andere objecten te adresseren dan objecten die worden gerepresenteerd door data in het virtuele geheugen.

Een programma bestaat uit een rij instructies waarvan de operanden naar een operand-definitietabel (ODT) verwijzen. Via die tabel dienen systeemobjecten buiten het programma-object te worden geadresseerd. System/38 instructies adresseren dus alleen objecten en adressen binnen objecten. Ook I/O loopt via objecten. Elk adresseerbaar object wordt voorgesteld door een datastructuur.

Als een programma is samengesteld (geassembleerd, gecompileerd) bevat het symbolische pointers naar objecten. Deze worden "unresolved" genoemd. Het programma moet instructies bevatten om een pointer voordat die wordt gebruikt in een operand te voorzien van het werkelijke virtuele adres van het object. Alleen een "resolved" pointer wijst naar een object. Het is op verschillende manieren mogelijk om een resolved pointer te verkrijgen.
Indien het een bestaand object bevat kan dat met behulp van de instructie

> Resolve System Pointer <unresolved system pointer, object name and required authorization, context, pointer authority to be set>

Deze instructie zoekt het object via de naam en de context en plaatst een resolved system pointer in plaats van de unresolved pointer. Het proces verkrijgt de voor adressering benodigde in de tweede operand gevraagde bevoegdheid. Die kan ook in de pointer zelf worden geplaatst indien het proces dat de instructie uitvoert "authorized pointer" bevoegdheid heeft voor het object. De in de pointer op te nemen bevoegdheid wordt gedefinieerd in de laatste operand (ref.20-p.3-18/21). De instructie wordt slechts uitgevoerd indien het proces over voldoende bevoegdheden beschikt.

De bevoegdheden van een proces bevinden zich in het "user profile" van het proces. Adresseerbaarheid en bevoegdheden zijn gescheiden opgeslagen in resp. een "context" en een "user profile".
Het adresseren van het user profile is alleen mogelijk voor het wijzigen van de daarin opgeslagen bevoegdheden. Adressering als onderdeel van het verkrijgen van een resolved pointer is een machinefunctie.
User profiles zijn uitsluitend impliciet adresseerbaar via een speciale "machine context" met speciale instructies die als operanden systeembjecten hebben die per definitie alleen via de machine context adresseerbaar zijn.

Opgemerkt moet worden dat een reële machine als de System/38 die gebaseerd is op capability based adressering aanzienlijk gecompliceerder is dan de hiervoor beschreven theoretische modellen.
Om een indruk te krijgen van het gebruik van abstracte objecten in het System/38 zal als voorbeeld een database file worden beschreven. Daartoe zullen eerst de verschillende objecten worden gedefinieerd, die worden gebruikt om een samengesteld object te realiseren.

Database files in het IBM System/38 kunnen direct (keyed) zijn of sequentieel (arrival sequenced). Een file kan :
- fysiek zijn, d.w.z. hij bevat reële records,
- logisch, d.w.z. hij representeert een bepaald uitzicht ("view") op de records in een of meer fysieke files en bevat zelf geen records en is dan een afgeleide file of
- hij bevat uitsluitend geordende sets sleutelwaarden die verwijzen naar bepaalde records in een fysieke file en is dan een directe logische file.

Het file control block (FCB) is een object dat de beschrijving bevat van de file en van elk lid van de file (file member) en de pointers bevat naar de objecten die de file gestalte geven. De beschrijvingen bevatten de file type aanduiding en de definities van de sleutels. De record beschrijvingen staan in het format object.
De objecten waarnaar de pointers in het FCB-object verwijzen zijn het format object, het cursor object en het dataspace object van het eerste member en het cursor object en het dataspace object van het laatste member van de file, een data space index object, een scope list object en verschillende directory objects.

Het format object bevat een record definitie met de beschrijving van de velden waaruit het record bestaat.
Het cursor object bevat het open data path control block. Dit bevat pointers naar de objecten die te zamen een member vormen. Dat zijn tenminste een data space en, indien het een direct file betreft, tevens een data space index. Voorts bevat het een pointer naar eventuele andere members. Een file wordt in werkelijkheid niet via de cursor zelf doch via een kopie ervan geadresseerd.

Het data space object bevat een pointer naar de cursor. Het formaat van de records die in het data space object worden opgeslagen zijn beschreven in het format object.

Het data space index object bevat een pointer naar de cursor. Het bevat voorts een ordening van de records in de data space of data spaces waarvoor de index is gecreëerd. De ordening wordt bepaald door de access path specificatie die deel uitmaakt van de file definitie in het file control block object.

De scope list is geen afzonderlijk object doch bevindt zich in het FCB-object indien dat de definitie van een logische file bevat. Het bevat pointers naar alle betrokken fysieke files en naar de format objecten die het uitzicht van de logische file op die fysieke files beschrijven.

Directory objecten bevatten de gegevens over de afhankelijkheden tussen verschillende files.

Een format directory per format object bevat pointers naar de files die records bevatten die door dat format worden beschreven.

Een "shared-data" directory per fysieke file bevat pointers naar alle logische en afgeleide files die pointers bevatten naar die fysieke file.

Een "shared-data" directory per fysiek member bevat pointers naar alle logische en afgeleide members die pointers bevatten naar dat fysieke member.

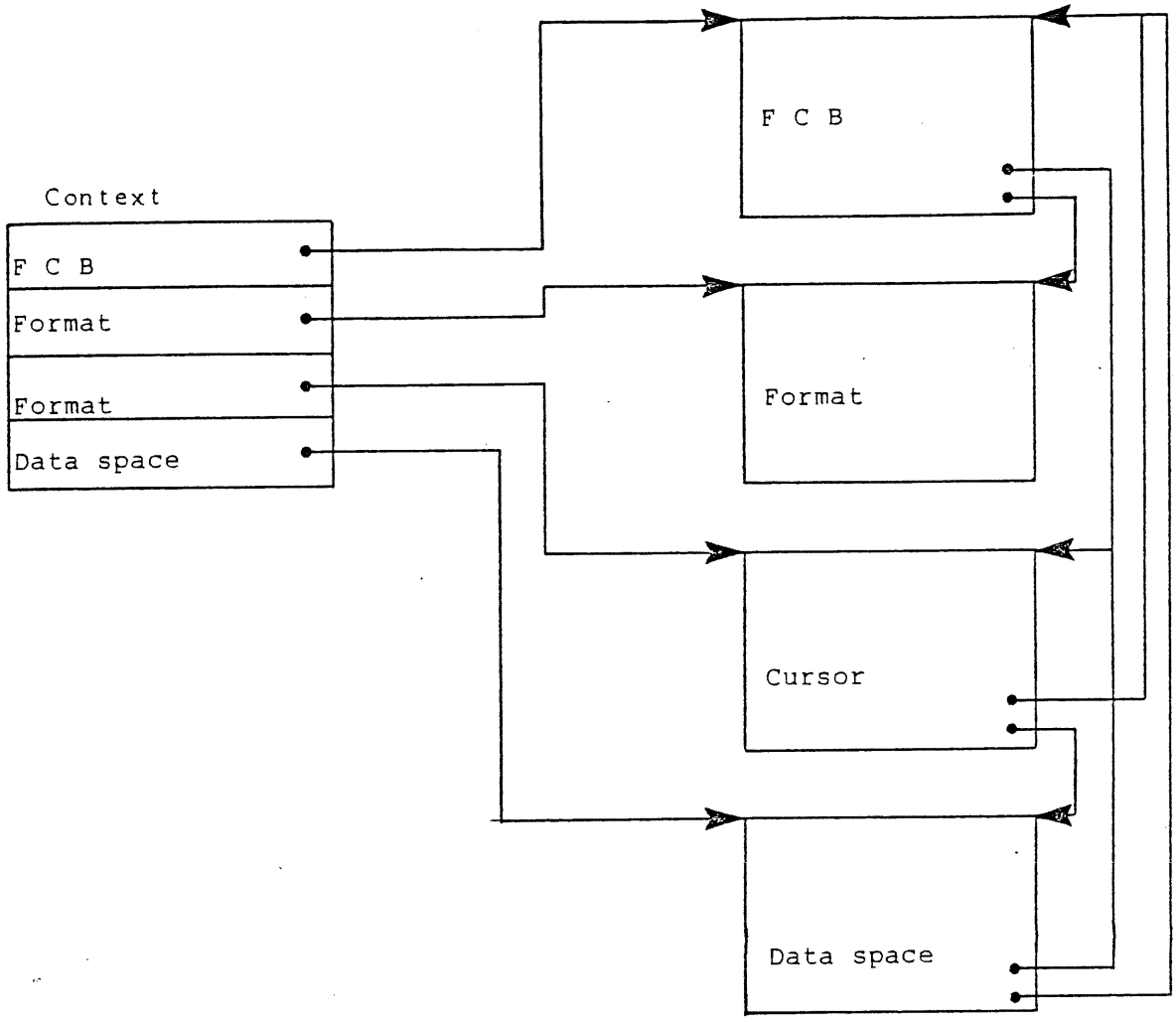
Een "shared-access-path" directory per direct file bevat pointers naar alle afgeleide files die pointers bevatten naar die direct file.

Een "shared-access-path" directory per direct member bevat pointers naar alle afgeleide members die pointers bevatten naar het data space index object van dat member.

De eenvoudigste vorm is een fysieke file met record opslag in aankomstvolgorde. In figuur 2 is de samenhang tussen de objecten schematisch weergegeven.

12. Integriteit van pointers

Het voorbeeld illustreert duidelijk dat pointers vrijelijk kunnen voorkomen in alle objecten.
De integriteit van de pointers wordt gewaarborgd doordat hun waarde waar het de adressen van objecten betreft slechts worden vastgesteld bij de creatie van het betreffende object door het systeem. Het adresdeel van pointers kan daarna niet meer worden gewijzigd. Wel uiteraard het deel dat de bevoegdheden bevat.
Deze bescherming is gerealiseerd door het gebruik van "storage tags". Het System/38 geheugen is opgebouwd uit woorden van 40 bits: 32 vrije bits, 7 "error correction code" bits en 1 "tag" bit.
Een pointer kan alleen in 4 aaneengesloten woorden worden opgeslagen. Per pointer moeten er dus 4 tag bits aanstaan. Indien de pointer wordt gebruikt als adres in een operand wordt door de hardware gecontroleerd of alle 4 de tag bits aanstaan. Er bestaan geen instructies waarmee tag bits geadresseerd kunnen worden (ref.15-p.342).


Context
F C B
Format
Format
Data space
F C B
Format
Cursor
Data space

Figuur 2

Pointers worden dus opgeslagen in geheugenlocaties van 16 bytes (ref.20-p.3-4, beschrijving van de instructie <Copy bytes with pointers>) die elk zijn voorzien van een aantal extra bits.
Indien wordt getracht een byte-georiënteerde operatie uit te voeren, bijvoorbeeld een <copy bytes left ajusted> (ref.20-p.2-36) op een geheugenlocatie die blijkens de tag bits een pointer bevat, dan worden die tag bits zodanig gewijzigd dat de inhoud niet langer door de hardware als een pointer wordt geaccepteerd.
Pointers kunnen in wezen alleen worden gekopieerd met speciale alleen voor pointer manipulatie bestemde instructies.

13. Besluit

In het voorgaande essay is gepoogd enig licht te werpen op capability based adressering. Zowel uit de theoretische modellen als uit de praktische realisaties blijken een aantal speciaal uit een oogpunt van gegevensbeveiliging aantrekkelijke eigenschappen. Mits ondersteund door een op capabilities gebaseerde machine-interface of instructieset is een hoge mate van afscherming mogelijk die in beginsel niet afhankelijk is van de betrouwbaarheid van het control program. Bovendien gaat dit gepaard aan zeer soepele sharing van data en programma's; meer in overeenstemming met de in dit verband gebruikelijke terminologie sharing van objecten die niet allen programma's en data omvatten doch ook (de representaties van) terminals, printers, datacommunicatiepoorten, etc.
Geconstateerd is dat het tempo van acceptatie relatief traag is. Als een der vermoedelijke oorzaken hiervoor is gesuggereerd een zeker conservatisme bij het automatiseringsmanagement. Deels is dit wel te begrijpen uit de ongetwijfeld optredende beheersproblemen die te verwachten zijn bij het inpassen van deze benadering in een bestaande situatie. Laat men de geboden mogelijkheden echter ongebruikt liggen, dan mist men een reële kans om de kwaliteit van de automatisering op een hoger plan te brengen, zowel wat doelmatigheid betreft als ten aanzien van het beveiligingsniveau.
Het veel gehoorde argument dat de conventionele software niet op een capability machine werkt duidt op onbegrip van de mogelijkheden van capability machines. Van een tekstverwerker die voor de kantoorautomatisering wordt ingezet verlangt men dat ook niet. Bovendien liggen de koppelingsproblemen op het niveau van de communicatieprotocollen en ook capability systemen houden zich te dien aanzien aan de daarvoor reeds bestaande en in ontwikkeling zijnde standaarden. In de praktijk blijken dergelijke koppelingen tussen een capability machine en een conventionele registermachine zeer goed mogelijk.

Referenties:

1. Dennis,J.B. and Van Horn,E.C., Programming semantics for multiprogrammed computations, Comm. of the ACM, Vol.9,Nr.3, March 1966.
2. Fabry,R.S., Capability based addressing, Comm. of the ACM, Vol.17, Nr.7, July 1974.
3. Linden,T.A., Operating system structures to support security and reliable software, ACM Computing Surveys, Vol.8,Nr.4, December 1976, p.409-445.
4. Lopriore,L, Capability based tagged architectures, IEEE Transactions on computers, Vol.c-33,No.9, September 1984.
5. Charniak,E and Wilks,Y,ed. Computational semantics, North Holland publishing Cy., ISBN 0-444-11110-7.
6. Introduction to the iAPX 432 architecture, Intel Corporation, manual order number: 171821-001, 1981.
7. Roos,H., Why is the IBM System/38 a secure system? In proceedings Workshop beheer en controle van en in besturingssystemen, Noordwijkerhout 14-16 mei 1984. NGI Sectie EDP-Auditing, p245-260.
8. Jones,A.K. and Wulf,W.A., Towards the design of secure systems, Software-Practice and experience, Vol.5,321-336, 1975.
9. Wulf,W, Cohen,E, Corwin,W, Jones,A,Levin,R, Pierson,C, and Pollack,F, HYDRA: The kernel of a multiprocessor operating system, Comm. of the ACM, June 1974, Vol.17, Nr.6, p.337-345.
10. Presser,L and White,J.R., Linkers and loaders, Computing surveys, Vol.4, No.3, September 1972.
11. Myers,G.J., Advances in computer architecture, snd.ed. 1981, John Wiley & Sons, ISBN 0-471-07878-6.
12. Berstis,V., Truxal,C.D. and Ranweiler,J.G., System/38 addressing and authorization, IBM System/38 technical developments, IBM 1978, ISBN 0-933186-00-2.
13. Jones,A.K., Capability architecture revisited, Operating Systems Review,Vol.14,No.3,p.33-36.
14. Wilkes,M.V., Hardware support for memory protection: capability implementations, Proceedings Symposium on architectural support for programming languages and operating systems, SIGARCH Computer architecture news, Vol.10, Nr.2, March 1982.
15. Houdek,M.E., Soltis,F.G. and Hoffman,R.L., IBM System/38 support for capability-based addressing, Proceedings 8th annual symposium on computer architecture, SIGARCH Newsletter,Vol.9, Nr.3, May 1981.
16. OS/VS2 MVS Overview, IBM GC28-0984-0 June 1978.
17. IBM System/370 Principles of Operation, GA22-7000-5, August 1976, including GN22-0584, June 1979.
18. Corsini,P, Frosini,G and Lopriore,L, The implementation of abstract objects in a capability based adressing architecture, The Computer Journal, Vol.27, No.2, 1984.
19. IBM System/38 Functional concepts manual, GA21-9330-1, June 1982, including GN21-3005, September 1982.

20. IBM System/38 Functional reference manual, GA21-9331-3, September 1982.
21. Banahan,M.F. and Rutter,A, UNIX- The Book, Sigma technical press, 1982, ISBN: 0 905104 21 8.
UNIX is een handelsmerk van Bell Laboratories.
22. Parnas,D.L., On the criteria to be used in decomposing systems into modules, Comm. of the ACM, Dec.1972, p.1053-1058.
23. NIVRA geschrift 26, Automatisering en controle, Deel IV. Mededelingen door de accountant met betrekking tot de betrouwbaarheid en continuïteit van geautomatiseerde gegevensverwerking, 1982, Kluwer. Deventer, ISBN: 90 267 0780 0.
24. Kampert,RA, drs.H.A., Mededelingen met betrekking tot de betrouwbaarheid en continuïteit van geautomatiseerde gegevensverwerking - een kloof tussen rationele behoeften en het vermogen van de accountant?, De Accountant nr.4/december 1982, p.234-240.
25. De gegevens voor dit voorbeeld zijn ontleend aan het IBM System/38 logic manual LY21-0571.
26. Siewiokek,D.P.,Bell,C.G. and Newell,A, Computer structures: principles and examples, chapter 32, The IBM System/38; contains a selection of 4 papers from the IBM publication: IBM System/38 technical developments, 1978 (ref.15).

Appendix.

Als voorbeeld voor een virtueel adresseringssysteem kan het op basis van de IBM 370-architectuur werkende MVS worden genomen.

De 370 gebruikt adressen van 24 bits waarmee een adresruimte van maximaal 16M kan worden geadresseerd. Het besturingssysteem MVS (multiple virtual storage) kan gelijktijdig een groot aantal virtuele adresruimten van elk 16M afbeelden op een reëel geheugen van 16M of kleiner (16).
De 370-architectuur is recent uitgebreid tot 370XA waarmee een aanzienlijk grotere virtuele adresruimte van ca. 2000M kan worden geadresseerd. Deze uitbreiding maakt gebruik van de in het PSW van de 370 tot dan toe ongebruikte bits. De 370XA gebruikt adressen van 31 bits. Deze uitbreiding heeft geen fundamentele wijziging gebracht in het adresseringsmechanisme.

De status van het lopende proces bevindt zich voor een deel in een speciaal register, het PSW (Program Status Word) en voor een deel in een aantal besturingsregisters (control registers). Het adres waar de eerstvolgende uit te voeren instructie staat bevindt zich in het PSW (17-p.22 en 32ev.).

Een instructie heeft in het algemeen het formaat:

Operatiecode, Operand1, Operand2

De operatiecode geeft aan welk concreet formaat de instructie heeft. Er zijn 7 verschillende formaten die alle op soortgelijke wijze worden vertaald. De operand adressen worden schematisch weergegeven door een paar:

Basisadres, Afstand vanaf basisadres

De component basisadres verwijst naar een algemeen register dat het basisadres bevat van 24 bits lengte. De tweede component bevat een 12 bits adres. Het operand adres is de som van beide adressen, waarbij het 12 bits adres voor de optelling wordt aangevuld met 12 nullen ter linkerzijde (17-21). Met het 24 bits registeradres kan maximaal 16M worden geadresserd. Met de 12 bits component kan 4K worden geadresseerd.

Het resultaat van deze adresberekening geeft een z.g. logisch adres. De waarde van een logisch adres hangt uiteraard af van de waarde in het basisregister. Die wordt vastgesteld door het laadprogramma dat het uit te voeren programma in herplaatsbare vorm leest uit een programmabibliotheek op schijf en het op een virtuele geheugenruimte van 16M afbeeldt.

Het logische adres is een adres binnen een virtuele geheugenruimte van 16M en vormt het uitgangspunt voor het proces dat het virtuele geheugen afbeeldt op het reële geheugen (16-2-1ev).
Het virtuele geheugen is verdeeld in 256 segmenten van elk 64K en elk segment is weer verdeeld in 16 pagina's van elk 4K. Het interne geheugen is verdeeld in stukken van 4K, elk weer bestaand uit twee stukken van 2K. Elk 2K-stuk is uitgebreid met een aanhangsel (tag) met een sleutelwaarde van 4 bits die fungeert als toegangsindicator, een bit als indicator voor het soort gebruik dat is toegestaan en twee bits als gebruiksindicator (16-2-4).

Het laadprogramma plaatst het uit te voeren programma vanaf schijf, waar het in herplaatsbare vorm staat, in reële geheugenpagina's en bouwt de segmenttabel en de paginatabellen op, zodanig dat het adresvertalingsmechanisme vanuit elk berekend logisch adres de juiste reële pagina bereikt. Daarna wordt de inhoud van de reële pagina's weggeschreven naar de schijf die wordt gebruikt voor de opslag van programmapagina's die niet actief nodig zijn voor het lopende proces. De paginatabellen worden gemerkt ter aanduiding dat de betreffende pagina's zich niet in reëel geheugen bevinden.

Tijdens het laden zijn de adreswaarden bepaald die in de door het geladen programma gebruikte basisregisters moeten worden geplaatst. Die waarden blijven ongewijzigd zolang het geladen programma actief is dat wil zeggen, zolang het in virtueel geheugen aanwezig is en er dus segment- en paginatabellen van bestaan. Het startadres van de segmenttabel en het beginadres van het programma in de virtuele adresruimte worden opgenomen in het statuswoord van het proces.

Het proces gaat lopen door het laden van het beginadres van de segmenttabel in besturingsregister 1 (segment table origin register) en het laden van het startadres van het programma in het PSW.
Het startadres van het programma is een logisch adres. Het moet via de tabellen worden omgezet in een reëel adres.

Het adresvertalingsmechanisme telt de linker 8 bits op bij de inhoud van het segmenttabel-oorsprongregister. Dit geeft het reële adres van de regel in de segmenttabel die het beginadres bevat van de paginatabel.
De volgende 4 bits van het logische adres worden opgeteld bij het beginadres van de paginatabel. Het resultaat is het reële adres van de regel in de paginatabel die het beginadres bevat van de reële pagina in het geheugen of de indicatie dat de pagina zich niet in het reële geheugen bevindt.
Het reële adres binnen de pagina wordt bepaald door de 12 rechter bits van het logische adres op te tellen bij het paginabeginadres. Dit geeft het adres van de uit te voeren instructie.

De uit te voeren instructie wordt opgehaald en na vaststelling van de uit te voeren operatie die impliciet het formaat (de lengte van de instructie die nodig is voor het vaststellen en in het PSW laden van de volgende uit te voeren instructie en de betekenis van de posities van de bits in de instructie) van de instructie bepaald en worden de operand adressen berekend door de inhoud van het basisregister waar het eerste deel van de operand naar wijst op te tellen bij het tweede deel van het operand adres.
De aldus berekende logische operand adressen worden vertaald in reële adressen via segmenttabel-oorsprongregister, pagina oorsprongadres en de afstand vanaf het begin van de pagina.

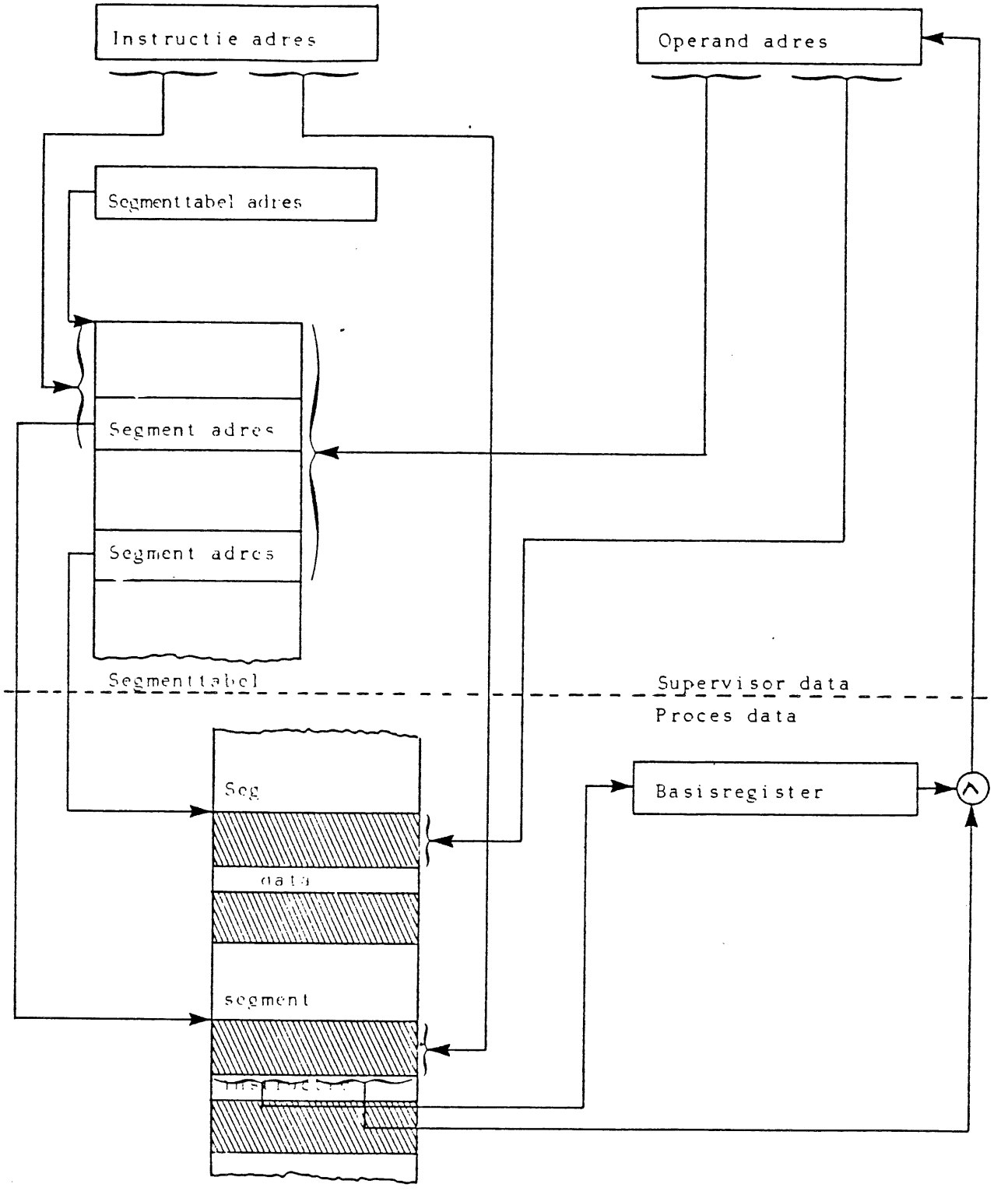
Virtuele of logische adressen zijn door deze vertaalmethode onafhankelijk van elkaar.
Wordt een pagina op een andere plaats in het geheugen gezet, dan wordt het nieuwe beginadres in de paginatabel gezet, met als gevolg dat de vertaling van logisch adres in reëel adres opnieuw het juiste resultaat geeft.

De waarde van de virtuele adressen wordt bepaald door de basisadressen in de basisregisters. Worden die gewijzigd dan zal de berekening van het logische adres uiteraard een andere uitkomst geven en daardoor een andere pagina aanwijzen. Elk virtueel adres is in wezen volledig relatief en kan dus worden weergegeven als een paar <index,offset>. Hetzelfde geldt voor het berekenen (in IBM termen genereren) van de virtuele adressen. De argumenten voor deze berekening zijn paren van dezelfde vorm <index,offset>.

In figuur 3 zijn de wezenlijke onderdelen en hun samenhang van deze adresberekeningsmethode schematisch weergegeven.

Indien een programma eenmaal is geladen liggen de logische adressen vast. De afbeelding van de logische adresruimte op de reële adresruimte kan niet door de programmeur worden bestuurd. Het op dynamische wijze gebruik maken van verschillende processen door dezelfde programma's wordt niet ondersteund. Dat kan alleen indien het vanuit een centraal punt wordt bestuurd. Het is volledig afhankelijk van een speciaal daarop gerichte procesmanager. Een voorbeeld is Cullinet's IDMS/DC. Communicatie tussen gebruikersprogramma en besturingsprogramma vindt daarbij plaats door middel van speciale supervisor calls.

Het gereloceerde programma wordt door middel van STOR, segmenttabel en paginatabel "swappable" en "pageable" gemaakt.
Dit betekent dat de virtuele adressen voor de duur van het bestaan van de adresruimte binnen die adresruimte context onafhankelijk zijn en daarbinnen dus gedeeld kunnen worden tussen verschillende programma's in dezelfde adresruimte.



Schema IBM 370 adressering (vereenvoudigd)

figuur 3

COMPACT

Lente 1985

KNOWLEDGE BASED SYSTEMS:
Een stap vooruit in de beheersbaarheid van administratieve processen

door A. van der Drift

*** Door onder meer ontwikkelingen vanuit Artificial Intelligence, hogere verwerkingssnelheden en grotere geheugens krijgt een relatief nieuw verschijnsel binnen de automatisering gestalte onder de naam Knowledge Based Systems (KBS).
Dit artikel gaat in op deze KBS in relatie tot beheersbaarheid van geautomatiseerde administratieve processen, waarbij de stellige verwachting bestaat, dat het gebruik van KBS ten behoeve van deze processen zonder twijfel zal plaatsvinden (zo dit al niet plaatsvindt).

1. Inleiding

In dit artikel staan Knowledge Based Systems (KBS) als relatief nieuw verschijnsel centraal. Het verschijnsel wordt hierin vanuit de optiek van beheersbaarheid ten aanzien van geautomatiseerde administratieve processen aan de orde gesteld. Ten einde de essenties van KBS te kunnen doorgronden, wordt vooraf een beknopte uiteenzetting gegeven over de achterliggende technieken.

Louter afgaand op de vertaling, gaat het bij KBS kennelijk om systemen, die gebaseerd zijn op kennis. De huidige conventionele systemen bezitten ook al kennis. Namelijk de inbreng van de bij systeemontwikkeling betrokken gebruikers. Deze kennis is in eerste instantie vastgelegd in de vorm van functionele specificaties voor het te ontwikkelen systeem. Later in het ontwikkelingstraject is de verzamelende kennis "vertaald" naar de uiteindelijke instructies in de programmatuur.

*** Over het vergaren en het vertalen van deze kennis is in de loop der tijden een veelheid aan literatuur verschenen (onder meer betreffende ontwikkelingstechnieken waaronder prototyping, programmeertalen en testen). Naast de efficiency in het bewerken van kennis is met name de kwaliteit/betrouwbaarheid van deze (vertaalde) kennis als problematiek en doel centraal gesteld.

Dit hoofdstuk is opgenomen in de Proceedings van het NGI Symposium sectie EDP-auditing "Data Beheer & Controle. 6-8 mei 1985.

Wat is nu het verschil in kennis tussen de KBS en de conventionele systemen? Wat zijn de essentiële verschillen tussen KBS en de conventionele systemen? Wat verandert er nu ten aanzien van de System Development Life Cycle? Vormt het gebruik van KBS een verbetering, en zo ja, in welk opzicht?

Dit artikel gaat in op deze vragen met als doel het belang van het gebruik van KBS voor de beheersbaarheid van geautomatiseerde administratieve processen aan te tonen. Impliciet wordt hierin tevens een model geschetst, waaraan toekomstige KBS uit oogpunt van beheersbaarheid van processen dienen te voldoen.

2. Introductie tot KBS

Dit hoofdstuk geeft een beknopte uiteenzetting over de achterliggende technieken, ten einde een indruk te geven van de essenties van KBS. Een specifiek implementatiemodel (produkt) wordt hierin derhalve niet behandeld.
Het zij vermeld, dat de hierin behandelde technieken in principe zullen worden aangetroffen in toekomstige implementatiemodellen. Om technische en/of efficiency-redenen zullen deze wellicht niet volledig in elk toekomstig model worden geïmplementeerd. Het streven zal er echter wel voor een belangrijk deel op gericht (moeten) zijn.

Kennis

Ten aanzien van kennis wordt in het kader van dit artikel het navolgende onderscheid gemaakt:

- kennis betreffende de bedrijfsprocessen. Deze kennis wordt onder meer aangetroffen in de vorm van gegroepeerde gegevens (informatie), die met behulp van informatiesystemen worden onderhouden in bestanden (bijvoorbeeld in een database). De hier bedoelde gegevens hebben tot doel de bedrijfsprocessen te registreren en/of te sturen;
- kennis betreffende de functie(s) en werking van gegevensverwerkende (informatie-)systemen. Deze kennis wordt onder meer aangetroffen in de vorm van functionele specificaties betreffende het systeem en in de daaruit resulterende applicatieprogrammatuur. De kennis stuurt in wezen de gegevensverwerking ten behoeve van de bedrijfsprocessen en dient derhalve uit die bedrijfsprocessen voort te vloeien.
  Deze kennis is onder te verdelen in kennis betreffende de gegevens (meta-data; onder meer in de vorm van bestandsbeschrijvingen) en in kennis betreffende de gegevensverwerkende processen (onder meer opgeslagen in de vorm van instructies met condities en acties);

- kennis betreffende de functie en werking van algoritmen, die binnen onder meer applicatieprogrammatuur kunnen worden aangetroffen. Deze kennis is impliciet aanwezig in de algoritmen (een algoritme, dat bijvoorbeeld sorteert, bevat immers impliciet de kennis over hoe het moet sorteren). Deze kennis is niet toepassingsgevoelig en vloeit derhalve ook niet direct voort uit de bedrijfsprocessen. Vanuit de bedrijfsprocessen kunnen echter wel eisen zijn gespecificeerd ten aanzien van de beschikbaarheid van bepaalde algoritmen.

Waar navolgend kennis ter sprake wordt gebracht, wordt met name kennis betreffende de gegevensverwerkende processen bedoeld.

Vastlegging van kennis

Kennis kan worden vastgelegd in rules (regels), bestaande uit condities en acties of conclusies. Acties, die ongeconditioneerd moeten worden uitgevoerd (gebruikt), zouden in een rule kunnen worden opgenomen, waarvan de conditie altijd "waar" is.
Veelal is kennis opgebouwd uit en/of te relateren aan andere kennis. Zo kunnen rules acties bevatten, die op zich bestaan uit, dan wel verwijzen naar andere rules. Aan de basis van alle huidige en toekomstige informatiesystemen staat kennis en derhalve de daaruit te definiëren verzameling van rules.
In de huidige (conventionele) systemen, ontwikkeld in bijvoorbeeld de programmeertaal COBOL, kunnen we kennis onder meer aantreffen in de "IF"-instructies (condities), waarin de acties zijn opgenomen achter de "THEN"-clausule en/of zijn opgenomen in programmadelen (paragrafen of secties), waar naar wordt verwezen.
Programma's bevatten aldus - zij het in een gecodeerde vorm - de oorspronkelijk door de gebruiker ingebrachte kennis (de functionele specificaties).

Programmeertalen

Afhankelijk van de generatie programmeertaal, die in een specifiek geval wordt gebruikt, is aanvullende (specifiek computer-georiënteerde) kennis vereist.
Een in de tweede generatietaal, Assembler, geschreven programma vereist kennis betreffende het WAAR (in de geheugens), HOE (in welke volgorde en op welke wijzen) en WAT (conform de functionele specificaties) dient te worden uitgevoerd.
Een in de derde generatietaal, COBOL, geschreven programma vereist kennis betreffende het HOE en WAT. Het WAAR wordt aangevuld door de COBOL-compiler.

Van vierde generatietalen kan als vanzelfsprekend worden verwacht c.q. geeist, dat uitsluitend kennis betreffende het WAT behoeft te worden opgenomen in de programma's. De kennis betreffende het WAT richt zich immers uitsluitend op de probleemstellingen en dus op de kennis (en toepassingen) van de gebruikers. De compilers/interpreters van deze vierde generatietalen zullen derhalve het WAAR en HOE automatisch moeten aanvullen.
(Hierboven is voorbijgegaan aan de betekenis van Linkers en Loaders. Deze bieden veelal ondersteuning ten aanzien van het WAAR.)
Uit het voorafgaande blijkt ondanks de tendens tot afname, dat kennis (in het bijzonder gebruikerskennis) in de programmatuur aanwezig is en blijft. Hieraan zijn nadelen verbonden, die in het volgende hoofdstuk nader zullen worden uitgewerkt.

*** Waar het dus in wezen om gaat is voor het uit te voeren proces "interpreteren" van relevante rules. Hoe krachtiger de "interpreterende" programmatuur c.q. hoe hoger de programmeertaal, hoe meer beperkingen kunnen worden aangebracht in de opgave van (uit het computergebruik voortvloeiende WAAR- en HOE-) rules en des te meer men zich bij de ontwikkeling van informatiesystemen kan beperken tot het vastleggen van kennis, die direct betrekking heeft op uitsluitend gebruikersproblematieken (toepassingen).

KBS functioneren op grond van een verzameling rules, die zijn opgeslagen in de zogenaamde rule base (een specifiek daarvoor ingericht bestand). Door het interpreteren van deze rules uit de rule base op grond van invoergegevens worden door het KBS processen uitgevoerd. Volledigheidshalve zij vermeld, dat kennis ten behoeve van KBS ook op andere wijzen (zoals onder meer in zogeheten frames) kan worden vastgelegd in bestanden. Gelet op de doelstelling van dit artikel wordt hierin ter illustratie uitsluitend uitgegaan van rules in de rule base. Deze implementatieverschillen zijn in dit kader niet van fundamenteel belang.

Kennis vergaren

Uit het voorafgaande valt af te leiden dat in de eerste plaats een rule base met relevante kennis (intelligentie) gevuld zal moeten worden. Dit vullen wordt in de literatuur aangeduid met de term "Knowledge acquisition". Hiervoor kan gebruik worden gemaakt van kennis vergarende algoritmen als onderdeel van KBS-programmatuur. Deze algoritmen zijn niet direct noodzakelijk, echter wel zeer wenselijk (zoals ook later in dit artikel zal worden aangetoond). Het kan - indien daarvoor ontwikkeld - ondersteuning bieden inzake het standaardiseren

van in te brengen kennis, het controleren op bevoegdheden inzake het inbrengen en wijzigen van kennis en het controleren op mogelijke tegenstrijdigheden in de ingebrachte kennis. Zonder deze algoritmen kan gebruik worden gemaakt van willekeurige editors, die in staat zijn de rules in het voor de KBS-programmatuur vereiste formaat in de rule base te plaatsen.

Inferentie mechanisme

In de tweede plaats vindt door de KBS-programmatuur ten behoeve van het uit te voeren proces interpretatie plaats van de rules. Dat deel van de KBS-programmatuur, dat met dit redeneren (het op grond van invoergegevens interpreteren van de beschikbare rules, die onderling in verband met elkaar kunnen worden gebracht) belast is, wordt inferentie mechanisme (gevolgtrekkingsmechanisme/inference engine) genoemd. Dit mechanisme bestaat uit redenerende algoritmen. In de coding van de KBS-programmatuur bevindt zich dus geen kennis, maar algoritmen, die in staat zijn op grond van de combinatie van invoer en rules uit de rule base tot acties (antwoorden) te redeneren.

Redeneren

Er zijn twee verschillende manieren, waarop de redenerende algoritmen gebruik kunnen maken van de rules: voorwaarts redenerend (forward chaining/reasoning of data-driven) en achterwaarts redenerend (backward chaining/reasoning). Beide manieren zijn van belang, zodat ook beide aangetroffen zouden moeten worden.
Aan de hand van figuur 1 volgt een toelichting, waarbij het accent niet ligt op de uitwerking van de geschetste probleemstelling, maar op de gevolgde redeneertrant.
Stel dat in een bepaalde toepassing mutaties worden aangebracht op een stam/standenbestand. In dit administratief proces worden tevens twee overzichten vervaardigd; het ene bevat de aangebrachte mutaties; het andere dubieuze mutaties.

De rule base zou hiervoor de in figuur 1 vereenvoudigd weergegeven rules kunnen bevatten.

```
Rule 1  IF  mutatiegegevens betreffen standengegevens
          THEN  muteer oude stand met mutatie.

Rule 2  IF  mutatiegegevens niet betreffen standengegevens
            AND stamgegevens zijn bijgevoegd
          THEN  voeg toe stamgegevens + mutatie.

Rule 3  IF  muteer
            OR voeg toe
          THEN tel mutaties 1
               print mutaties op lijst 1.

Rule 4  IF  dubieus
          THEN tel mutaties 2
               print mutaties op lijst 2.

Rule 5  IF  oude stand + mutatie  > 1.000.000
          THEN dubieus.

Rule 6  IF  mutatie  > 500.000
          THEN dubieus.
```

Figuur 1.

Bij voorwaarts redeneren:
De redenerende algoritmen zullen op grond van de ingevoerde mutaties rule voor rule doorlopen, ten einde acties aan te geven. De volgorde, waarin de rules worden doorlopen is - zo zal blijken - niet relevant. Een en ander kan als volgt plaatsvinden:

Rule 1: Indien op grond van een nader te specificeren sleutel (bijvoorbeeld het rekeningnummer) een stam/standen-record in het desbetreffende bestand wordt gevonden, wordt aan deze conditie voldaan, hetgeen tot gevolg heeft dat het bewuste record zal worden gemuteerd.

Rule 2: Indien het bovengenoemde record niet in het bestand wordt aangetroffen en de ingevoerde mutatie tevens voorzien is van stamgegevens, zal een geheel nieuw record worden toegevoegd.

Rule 3: Indien er gemuteerd of toegevoegd is, zal de mutatie geteld worden en op lijst 1 worden afgedrukt.

Rule 4: Indien de mutatie als dubieus wordt aangemerkt, zal deze opnieuw worden geteld en op lijst 2 worden afgedrukt. Bij eerste doorgang is nog niets bekend over het al dan niet dubieus zijn van de mutatie, waardoor dan nog niet aan deze conditie wordt voldaan.
Rule 5: Indien de som van de oude stand en de mutatie groter is dan een miljoen, wordt de mutatie als dubieus aangemerkt.
Rule 6: Indien de mutatie groter is dan 500.000, wordt deze eveneens als dubieus aangemerkt.

Vervolgens zullen de algoritmen de rules met condities, waaraan nog niet is voldaan, opnieuw in beschouwing nemen. Dit kan effect hebben bij rule 4, indien de mutatie in de eerste doorgang als dubieus is aangemerkt op grond van de rules 5 en/of 6.

Uit het voorafgaande blijkt duidelijk dat de redenering plaatsvindt op grond van invoergegevens (vandaar de term "data-driven").
Ook blijkt dat de volgorde, waarin de rules zijn opgeslagen respectievelijk worden doorlopen, niet relevant is. (Dit lijkt overigens analoog aan relationele gegevensstructuren, waarin de volgorde van data ook geen informatieve waarde heeft. Relationele opslagstructuren zouden mede daarom wellicht kunnen worden gebruikt voor de opslag van rules.) Dit betekent ook dat rules gemakkelijk kunnen worden toegevoegd, gewijzigd en verwijderd.
Echter, voornoemde zal slechts onder hier niet nader besproken beperkingen kunnen plaatsvinden, waardoor het aannemelijk is te veronderstellen dat in toekomstige implementatiemodellen ten behoeve van het doorlopen de rules "genest" (gekoppeld) zullen zijn. In figuur 1 zou "dubieus" in rule 4 automatisch kunnen worden vervangen door de condities uit de rules 5 en 6.

Bij achterwaarts redeneren:
Indien men wil weten op grond waarvan mutaties op lijst 2 verschijnen, kunnen de algoritmen vanuit de actie in rule 4 terug-redeneren, waarbij de daarin aangegeven conditie "dubieus" als actie in de rules 5 en 6 zal worden aangetroffen. Dit levert als resultaat op dat de algoritmen de condities "oude stand + mutaties > 1.000.000" en "mutatie > 500.000" als antwoord zullen geven.
In hoofdstuk 6 zal kort op een specifiek nut van achterwaarts redeneren worden teruggekomen.

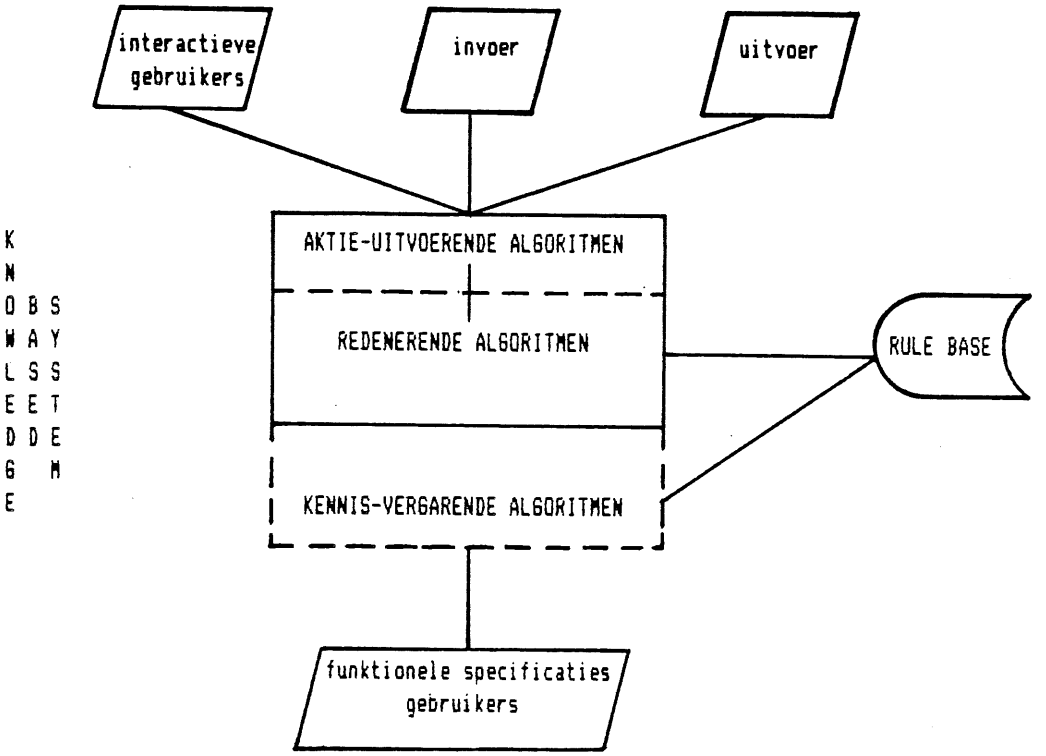
Acties uitvoeren

In het daadwerkelijk uitvoeren van de acties (zoals het opvragen, zoeken, selecteren, tellen, wegschrijven en printen) moet door een aanvullende groep van algoritmen zijn voorzien. Deze voorzieningen worden

al sinds jaar en dag aangetroffen in bestaande meest parametergestuurde talen (bijvoorbeeld report writers en audit packages).
Bij "Expert Systems", die als subset van KBS worden gezien, kunnen specifieke algoritmen worden onderkend, zoals hoogwaardige "User interfaces" en de zogenaamde "Explanation facility". Laatstgenoemde verschaft toelichting aan gebruikers op de gevolgde redenering (lijkt vergelijkbaar met de "TRACE-facility", die door de hedendaagse compilers en interpreters wordt geboden). Hierdoor neemt de acceptatie van de uitspraak van het systeem toe en kan aanleiding worden gevonden tot het aanvullen dan wel corrigeren van de gehanteerde rules.

*** De KBS-programmatuur bevat dus in principe geen kennis, maar uitsluitend algoritmen, die op grond van de extern opgeslagen kennis (in de rule base) processen kan uitvoeren. Dit betekent ook dat deze programmatuur ten behoeve van meerdere administratieve toepassingen moet kunnen functioneren. Immers, de desbetreffende kennis (rules) bepaalt de karakteristieken van de toepassing. Om voor een bepaalde toepassing te functioneren behoeven KBS derhalve nog slechts te weten van welke groep van rules zij gebruik dienen te maken.
Een en ander doet concluderen, dat KBS voor meerdere toepassingen functionerende informatiesystemen zijn met een geheel afwijkende structuur. Deze worden ook wel aangeduid als de vijfde generatie software.

Figuur 2 geeft van genoemde algoritmen een schematische weergave.



Figuur 2.

3. Ontwikkeling van informatiesystemen

Ten behoeve van de ontwikkeling van conventionele informatiesystemen is sinds jaar en dag gepoogd middelen en technieken toe te passen, die onder meer bijdragen tot:
- een verhoging van de efficiency voor wat betreft het ontwikkelen;
- meer betrouwbare en effectieve produkten (in casu programma's);
- grotere flexibiliteit van de produkten, waardoor automatiseringsinspanning kan worden bespaard als het gaat om het uitvoeren van onderhoudswerkzaamheden.

Een aantal middelen en technieken wordt in het kader van kwaliteitsverhoging van systemen navolgend besproken.

Programmeren

Het hanteren van programmeerstijlen heeft voornamelijk tot doel door middel van het inzichtelijk maken van programma's de onderhoudbaarheid hiervan te vergroten. Het spitst zich daarbij in wezen toe op het ordenen van kennis, die in de programmatuur is opgenomen (bijvoorbeeld modulair en gestructureerd programmeren).

*** Onderhoud vloeit maar al te vaak voort uit veranderde gebruikerseisen en -wensen. Juist de ingebrachte gebruikerskennis dient daardoor gewijzigd te worden. Een afsplitsing van die kennis uit de overige programma-coding zou het onderhoud kunnen vereenvoudigen.

Een van de problemen in relatie tot onderhoudswerkzaamheden, betreft de aanwezigheid van constanten (onderdeel van kennis) in de programmatuur. Als voorbeeld moge dienen loonbelastingtabellen. Constanten blijken toch nog te vaak aan wijzigingen onderhevig te zijn, waardoor ontwikkelaars zullen trachten deze constanten zo veel mogelijk buiten de programma-coding te houden. Dit zal resulteren in het gebruik van constantenbestanden, die met behulp van aparte mutatieprogramma's door gebruikers zelf kunnen worden onderhouden. Hierdoor treedt een door zowel de gebruikers- als de ontwikkelingsorganisatie gewenste taakverschuiving op; namelijk het plegen van onderhoud op constanten door gebruikers in plaats van door programmeurs. De inhoudelijke verantwoordelijkheid ligt immers bij de gebruikers.

*** Het verwijderen van constanten uit programma's resulteert in het introduceren van algoritmen in de programmatuur, die indien gewenst de constanten uit de bestanden benaderen. Derhalve zou de verwijdering van constanten als eerste stap in de richting van algoritme-rijke en kennis-arme programmatuur (KBS) kunnen worden gezien.

Een volgende stap is de verwijdering van de (eveneens onderhoudsgevoelige) regels, waarin condities en acties zijn opgenomen, naar een regelbestand (rule base), dat door gebruikers kan worden onderhouden liefst met behulp van daarop zeer specifiek gerichte programmatuur (de eerder genoemde kennis vergarende algoritmen).

Hogere programmeertalen

De kracht van de taal in het uitvoeren van acties alsmede programmeursvriendelijkheid en efficiency van verwerking spelen bij de selectie van een programmeertaal een beduidende rol. Afhankelijk van de mate van kracht (mogelijkheden) wordt het werk van de programmeur beperkt. Dit zal de efficiency van ontwikkelen en de betrouwbaarheid van het eindprodukt ten goede komen. De compiler- dan wel interpreterprogrammatuur zou - zoals eerder gesteld - voor het HOE en WAAR immers zorg moeten dragen. Het "coderen" van louter de probleemstellingen (kennis) van de gebruikers blijft derhalve nog over voor de programmeur. Dit wordt reeds als zodanig ervaren bij gebruik van bijvoorbeeld report-writers en dergelijke.

Welke zijn nu de essentiële verschillen tussen KBS en de hogere programmeertalen, indien als uitgangspunt geldt dat zowel de hogere programmeertalen als de actie-uitvoerende algoritmen van de KBS gelijke kracht (mogelijkheden) bieden? Een tweetal meest essentiële verschillen wordt navolgend genoemd.

In de eerste plaats het gebruik van een "redundant-vrije" opslag van kennis, die ten behoeve van meerdere toepassingen kan worden geraadpleegd, waardoor een "sharing" van kennis op een veel verfijnder niveau, dan bij de reeds bekende "program-sharing" het geval is, kan plaatsvinden. Bij gebruik van talen is mogelijk dezelfde kennis in meerdere programma's aanwezig. Dit verzwaart het onderhoud en heeft in negatieve zin invloed op de betrouwbaarheid van programma's. Evenwel is in de loop der jaren herhaaldelijk getracht bij gebruik van talenkennis, die in meerdere programma's benodigd is, onder te brengen in afzonderlijke deelprogramma's (subroutines). Dit voldoet echter gelet op de daarvoor in aanmerking komende kennis slechts in beperkte mate. Ten aanzien van redundant-vrije opslag van kennis geldt de volgende aanvulling. De essentie hierbij ligt niet zo zeer in de besparing van de voor kennis benodigde opslagruimten, maar primair in de onderhoudsproblematiek. Binnen een specifiek implementatie-model van KBS kan het voorkomen dat toch bepaalde kennis fysiek redundant wordt opgeslagen, maar dat de kennis vergarende algoritmen dit compenseren door bij verandering van kennis de wijzigingen op alle plaatsen automatisch door te voeren.

In de tweede plaats het gebruik van kennis vergarende algoritmen, die - zoals reeds vermeld - specifieke ondersteuning kunnen bieden bij het ontwikkelen en onderhouden van toepassingen.

*** Identiek aan de taakverschuiving ten aanzien van de constanten, zullen gebruikers (vooralsnog afhankelijk van onder meer de gebruikersvriendelijkheid van de kennis-vergarende algoritmen) zelf in staat zijn de rules te specificeren (te ontwikkelen en onderhouden). Hierdoor zullen ontwikkelaars zoals programmeurs ten aanzien van applicatieprogrammatuur overbodig worden en zich derhalve kunnen richten op de ontwikkeling van KBS en overige systeemprogrammatuur.
Eveneens afhankelijk van de door de kennis vergarende algoritmen geboden mogelijkheden, zullen op specificatie - tijd logicafouten (tegenstrijdigheden in de specificatie van kennis) kunnen worden aangegeven, hetgeen de betrouwbaarheid en ontwikkelingssnelheid wel zeer ten goede zal komen. Een ideaalsituatie, die langzamerhand benaderd zal gaan worden.

Zoals uit de voorafgaande alinea kan worden afgeleid, zullen KBS zich bij uitstek lenen voor prototyping. De betrokkenheid van de gebruikers en de snelheid van ontwikkelen kunnen immers optimaal zijn.

Testen

Het testen van programma's heeft vooral tot doel aanwezige fouten op te sporen, ten einde de applicatieprogrammatuur daarop te kunnen verbeteren, zodat het functioneert conform de daaraan gestelde eisen en wensen.
Zeker bij conventionele systemen blijkt 100 procent testen (liever gezegd het verkrijgen van 100 procent zekerheid, dat alle oneffenheden, die aanwezig zijn, met behulp van het middel testen ontdekt worden) praktisch onhaalbaar. Dit ondanks geboden hulpmiddelen, zoals faciliteiten en pakketten als COMBI, dat beperkte inzage verstrekt in de volledigheid van de ingevoerde testgevallen in relatie tot de uitgevoerde programma-instructies op grond van de test.
Ook en wellicht als alternatief wordt getracht wetenschappelijk te bewijzen, dat algoritmen (onder alle omstandigheden) juist zijn (correctness-proving). Echter door onder meer de aanwezigheid van constanten wordt dit in de praktijk ernstig bemoeilijkt. Zo ook heeft het gebruik van constanten in de programmatuur tot gevolg, dat zonder codereview testen een onvolledige indruk over de kwaliteit van de applicatieprogrammatuur zal geven. Dit vereist dus handmatige arbeid, hetgeen op zich qua betrouwbaarheid veelal te wensen overlaat.

Voor wat betreft de coding, bevatten KBS in principe uitsluitend algoritmen, die eenmaal getest zijnde ten behoeve van meerdere toepassingen kunnen functioneren. Dit kan worden vergeleken met bestaande systeemprogrammatuur. Wellicht zal in de toekomst voor dit soort coding een correctheidsbewijs kunnen worden geleverd.
Resteert nog een betrouwbaarheidsuitspraak over de inhoud van de rule base. Gelet op het feit, dat de rule base in wezen gevuld is met door gebruikers samengestelde functionele specificaties, is een review hiervan door hen met ondersteuning van KBS goed mogelijk. Het voorbeeld uit het voorafgaande hoofdstuk betreffende achterwaarts redeneren illustreerde dit al. Dit neemt niet weg, dat testen naar aller waarschijnlijkheid noodzakelijk zal blijven. Dit zal zich dan richten op de toepasbaarheid van het totale systeem in de werkelijkheid. Een en ander zal echter mede door de inzichtelijkheid en abstractie van techniek (het WAAR en HOE) eenvoudiger worden.

*** Bij gebruik van KBS zal het (nog uitsluitend door gebruikers uit te voeren) testen beperkt kunnen blijven tot de functionele specificaties in de rule base. Bedacht dient te worden dat de inhoud van de rule base tot stand is gekomen ten gevolge van een vertaalslag op grond van het beeld dat de gebruikers over de werkelijkheid hebben naar deze functionele specificaties. Het verkrijgen van een juist beeld alsmede deze vertaalslag kan foutief verlopen, waardoor het testen van de resultaten hiervan van betekenis blijft. Door geboden prototypingfaciliteiten zal dit mede kunnen worden ondersteund.

4. Beheersbaarheidsaspecten

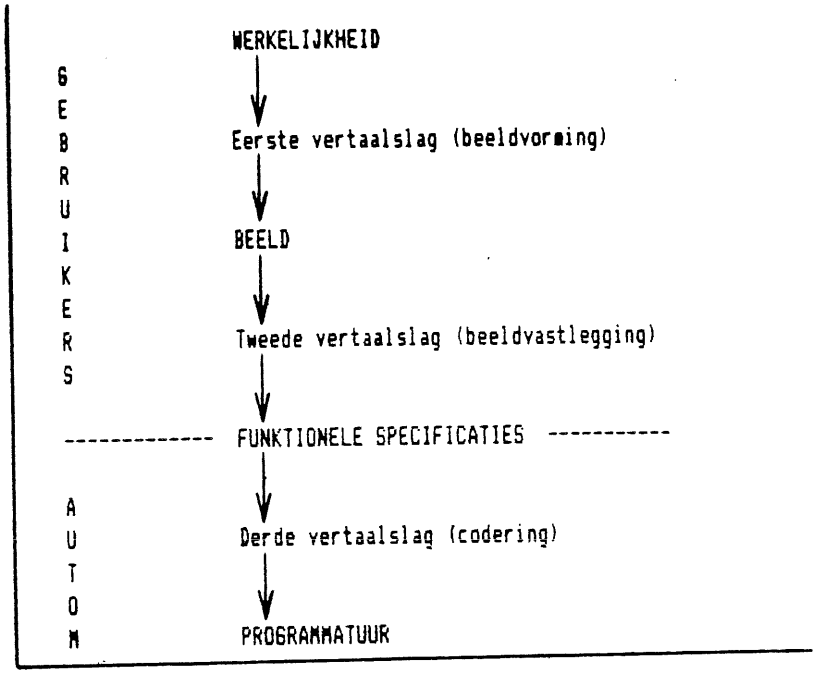
Ten behoeve van het contrast tussen KBS en conventionele systemen geldt in dit hoofdstuk het navolgende als uitgangspunt:
- conventionele systemen worden door de automatiseringsorganisatie in bijvoorbeeld tweede of derde generatie programmeertalen ontwikkeld en onderhouden;
- KBS-programmatuur bevat uitsluitend algoritmen en derhalve geen gebruikerskennis. Deze programmatuur draagt zorg voor de verwerking op grond van de gespecificeerde rules (als weergave van functionele specificaties).

Conventionele systemen

In principe specificeert en wijzigt (al dan niet met ondersteuning van informatici) alsmede accepteert de gebruikersorganisatie de functionele specificaties op grond waarvan de ontwikkeling dan wel het onder-

houd zal plaatsvinden. De ontwikkelaars (de automatiseringsmedewerkers) zullen vaststellen dat de beschikbare specificaties door de juiste (verantwoordelijke/bevoegde) gebruikers zijn goedgekeurd. Afhankelijk van de binnen automatisering van kracht zijnde prioriteiten zullen de genoemde werkzaamheden aanvangen.

Figuur 3 geeft de in dit verband te onderkennen vertaalslagen weer, die bij conventionele systemen van toepassing zijn.



Figuur 3.

Na gereedkoming zal de gebruikersorganisatie het systeem testen om te trachten vast te stellen dat:

- het systeem voldoet in de werkelijkheid c.q. de eerste vertaalslag door hen correct is uitgevoerd;
- het systeem voldoet aan het verwachtingspatroon c.q. de tweede vertaalslag door hen correct is uitgevoerd;
- het systeem conform de gespecificeerde eisen en wensen functioneert c.q. de derde vertaalslag door de ontwikkelaars correct is uitgevoerd;
- bij onderhoud: de overige coding van het systeem, die niet voor wijziging in aanmerking dient te komen, ongewijzigd is gebleven c.q. de derde vertaalslag door de ontwikkelaars correct is uitgevoerd.

Tevens bestaat veelal de wens om ook bij voortduring vast te stellen dat de programmatuur (in machine-uitvoerbare vorm) nog steeds voldoet aan het beeld over de werkelijkheid en de functionele specificaties. In sommige organisaties past men mede hiervoor een specifieke testaanpak toe, genaamd "Integrated Test Facility" (ITF), waarmee in de produktieverwerking met fictieve gegevens kan worden getest.

Van operationele programmatuur dient bij voortduring vastgesteld te worden dat uitsluitend daartoe bevoegde gebruikers de programmatuur uitvoeren. De controle hierop vindt veelal door systeemprogrammatuur als TeleProcessing-monitors (voor on-line-programma's), toegangsbeveiligingsprogrammatuur (als aanvulling op het Operating System van de computer) en de produktie-afdelingen binnen de automatiseringsorganisatie (organisatorische procedures) plaats.

*** Samengevat: Bij conventionele informatiesystemen dient de automatiseringsorganisatie controle uit te oefenen op de bevoegdheid van gebruikers voor wat betreft ontwikkelings- en onderhoudsaanvragen.
Procedureel en/of software-matig vindt controle plaats op het gebruik van de applicatieprogrammatuur.
De gebruikersorganisatie zal trachten met behulp van het middel testen zekerheid te verkrijgen omtrent de betrouwbaarheid van de applicatieprogrammatuur. Dit testen zal zich met name ook richten op de door de automatiseringsorganisatie uitgevoerde (derde) vertaalslag.
Over de waarde van het testen is in het voorafgaande hoofdstuk enige twijfel uitgesproken. Dit betekent dat onzekerheid blijft bestaan omtrent de volledigheid en juistheid van de uitgevoerde vertaalslagen. Indien derhalve vertaalslagen in het ontwikkelingstraject achterwege zouden kunnen blijven, zou dit de mate van onzekerheid doen afnemen.

Systemen op basis van KBS

In principe ontwikkelt en onderhoudt de gebruikersorganisatie zelf door het specificeren van rules, die de functionele specificaties weergeven. Software-matig zal door de kennis-vergarende algoritmen controle op de bevoegdheid daartoe worden uitgeoefend. De vertaalslag van functionele specificaties naar coding door de automatiseringsorganisatie vindt dus niet plaats, waardoor het testen zich daar niet op behoeft te richten. De volledigheid en juistheid van de rules (functionele specificaties als resultaat van de eerste en tweede vertaalslag door de gebruikers) dient te worden vastgesteld. Dit is mede gelet op ondersteuning vanuit KBS (zoals backward reasoning en "Explanation facility") voor de gebruikers eenvoudiger geworden.

Het gebruik van de rules voor het uitvoeren van toepassingen dient gecontroleerd te worden door de redenerende algoritmen van KBS. Indien de data adequaat is beveiligd tegen onbevoegd gebruik aan de hand van een directe relatie tussen data en gebruiker, vereist het gebruik (het lezen) van rules (kennis) - evenals dit het geval kan zijn bij conventionele programmatuur - slechts in uitzonderingsgevallen beveiliging.

Doordat de gegevensverwerking plaatsvindt op grond van de gedefinieerde functionele specificaties, kunnen geen verschillen bestaan tussen de werking en de daarvoor bestaande specificaties. Niet tegenstaande dat, kan bijvoorbeeld ITF nog steeds van belang zijn, teneinde vast te stellen, dat de specificaties de werkelijkheid blijven dekken.

Het zal duidelijk zijn dat een en ander valt of staat met de betrouwbaarheid en beveiliging van de kennis in de rule base.

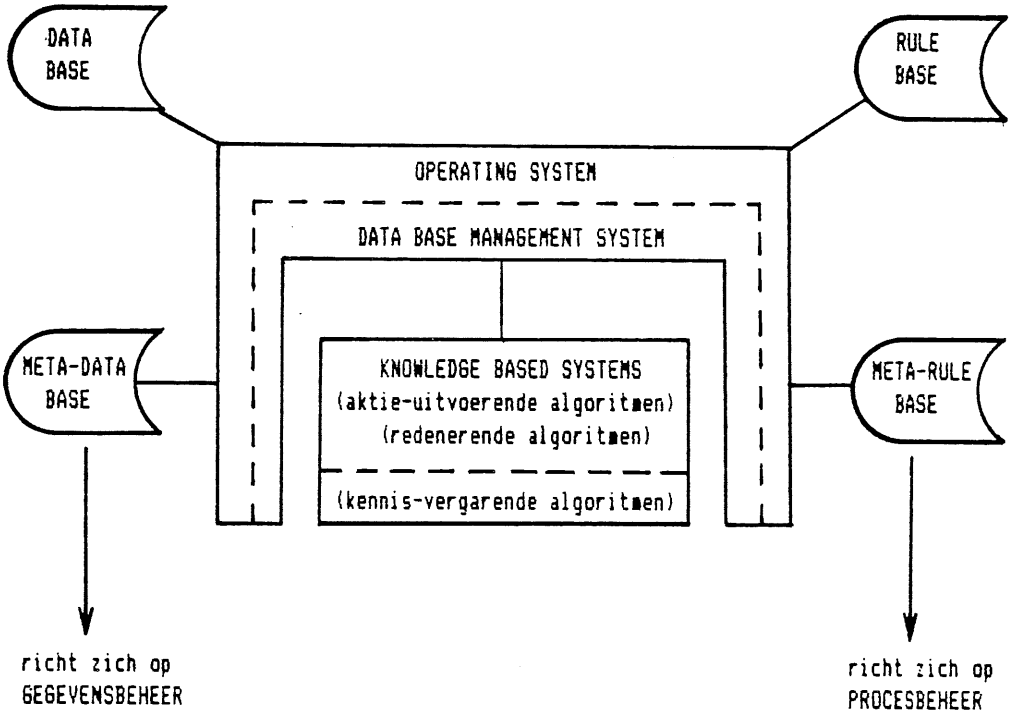
*** Bij gebruik van KBS zal de controle op de bevoegdheid van gebruikers voor wat betreft het ontwikkelen en onderhouden, software-matig kunnen plaatsvinden. Deze controle moet op een verfijnder niveau dan programma's kunnen plaatsvinden, namelijk op kennisniveau. Dit geldt ook voor het gebruik van kennis (de rules in de rule base).
Ten gevolge van het directe gebruik van specificaties voor het uitvoeren van processen, zullen geen verschillen bestaan tussen deze functionele specificaties en de eigenlijke werking.
Het testen richt zich nog slechts op de volledigheid en juistheid van de functionele specificaties in relatie tot de werkelijkheid.

(Voorzichtigheidshalve zij hier benadrukt dat bovenstaande valt af te leiden uit een ideaal beeld over KBS. Implementatiemodellen zouden hier vooralsnog wel eens fors in kunnen verschillen.)

Meta-rules

Analoog aan data en meta-data vereist het gebruik van rules meta-rules. Meta-rules vertellen KBS onder meer hoe de rules zijn gedefinieerd, wie bevoegd is tot het specificeren (ontwikkelen en onderhouden) en wie de rules mogen gebruiken. Aan de hand daarvan zouden immers de kennis-vergarende en redenerende algoritmen bevoegdheidscontrole kunnen uitoefenen. Met de verzameling van meta-rules zal de rule base derhalve worden beheerd op analoge wijze als dit bij meta-data ten behoeve van data het geval is.

Figuur 4 geeft van behandelde componenten een weergave. Hierbij is uitgegaan van een en hetzelfde Data Base Management System (DBMS) voor de fysieke opslag van (meta-) data en (meta-) rules.



Figuur 4.

Het zij vermeld dat in specifieke implementatiemodellen de programmatuur (O.S., DBMS en KBS) alsmede de bestanden (in het bijzonder de meta-data base, rule base en meta-rule base, maar zelfs ook de data base) geïntegreerd kunnen zijn. De individuele functies, die de in figuur 4 getoonde componenten bevatten, zullen als vanzelfsprekend ook bij integratie kunnen en moeten worden onderkend.

5. Organisatorische aspecten

De rule base bevat in wezen stuurinformatie (condities en acties, constanten, feiten, parameters en dergelijke) voor de geautomatiseerde administratieve processen. Rules bepalen c.q. sturen immers de gegevensverwerking, die plaatsvindt ter registratie en ondersteuning van de bedrijfsprocessen. De rules dienen dan ook voort te vloeien uit die bedrijfsprocessen, waardoor deze door de daarbij betrokken gebruikers behoren te worden vastgesteld.
De verwerkte gegevens registreren en sturen vervolgens de desbetreffende bedrijfsprocessen. Derhalve ligt het eigendom en beheer hiervan eveneens in de gebruikersorganisatie. De mate waarin de gegevens betrouwbaar dienen te zijn, is direct afhankelijk van de bedrijfsprocessen, waarvoor de gegevens van betekenis zijn. Dit houdt in, dat desbetreffende betrouwbaarheids- en nauwkeurigheidseisen dienen te worden gesteld vanuit die bedrijfsprocessen. Deze eisen dienen dan ook in het gegevensverwerkend proces geëffectueerd te worden, waardoor deze zullen worden aangetroffen in de rule base (de eisen zijn immers ook kennis) en/of meta-data base.
Een en ander leidt tot de (zij het wellicht triviale) conclusie dat de data en rules eigendom zijn van gebruikers.

Application Administration

Voor onder meer het coördineren van gebruikerseisen en -wensen inzake gegevensverwerkende processen (informatiesystemen) wordt sinds jaar en dag binnen de gebruikersorganisatie een aparte functie van Application Administration aangetroffen. Het vervullen van deze functie door een Application Administrator (AA = applicatie/systeem-beheerder) neemt als vanzelfsprekend in belang toe naarmate het gemeenschappelijk gebruik van deze processen en daarmee de daaraan ten grondslag liggende kennis ("sharing" van rules) toeneemt. Ten einde zijn functie adequaat te kunnen vervullen, dient de AA in staat te zijn te voorkomen, dat onbevoegd toegang wordt verkregen tot de rule base. Dit vereist een beheersinstrument, dat bij KBS in de vorm van meta-rules beschikbaar is. De AA dient daarom de beschikking te hebben over deze meta-rules, waarmee hij de KBS-programmatuur kan instrueren slechts aan bepaalde gebruikers bepaalde handelingen in de rule base toe te staan. Met meta-meta-rules (eveneens op te slaan in de meta-rule base) kan worden aangegeven wie toegang heeft tot de meta-rules. Deze meta-meta-rules dienen onder beheer te staan van de AA.

*** Indien kennis (rules) wordt "geshared" door meerdere informatiesystemen, die onder beheer staan van verschillende AA's, vereist het beheer van de desbetreffende rules coördinatie tussen de betrokken AA's. Deze coördinatie zal worden uitgevoerd door of namens de bedrijfsleiding.
De coördinator dient de meta-meta-rules en meta-rules betreffende de te "sharen" kennis (rules) te beheren. Zonder bedoelde coördinatie dient redundantie in de opslag van rules te worden geïntroduceerd. Dit kan uitmonden in afzonderlijke rule bases. Hierdoor wordt echter het risico van inconsistentie van kennis verhoogd. Vooralsnog is het de vraag of zonder deze coördinatie het "sharen" van kennis (rules) tot conflicten kan leiden, indien in overleg tussen de betrokken gebruikers één gebruiker als eigenaar van de kennis wordt geaccepteerd.

Data administration

Ten behoeve van bedrijfsprocessen kan door meerdere gegevensverwerkende processen, die kunnen ressorteren onder verschillende AA's, dezelfde data worden gebruikt. Hierbij is sprake van gemeenschappelijk gebruik van gegevens ("data-sharing"). Dit vereist ten behoeve van de betrouwbaarheid en nauwkeurigheid van de gemeenschappelijk gebruikte data eveneens coördinatie; zelfs coördinatie over bedrijfsprocessen heen. Hiervoor wordt de functie van Data Administration onderkend. Analoog aan de AA neemt het belang van het functioneren van de Data Administrator (DA = gegevensbeheerder) toe naarmate meer "data-sharing" plaatsvindt. Ook hiervoor geldt de noodzaak tot gebruik van een adequaat beheersinstrument, waarmee onbevoegd toegang tot de gegevens in de data base en het opslaan van onbetrouwbare gegevens kunnen worden voorkomen. Dit beheersinstrument wordt aangetroffen in de vorm van meta-data. Daarom dient de DA hierover de beschikking te hebben. Met meta-meta-data (eveneens op te slaan in de meta-data base) kan worden aangegeven wie toegang heeft tot de meta-data. De meta-meta-data dient derhalve onder beheer te staan van de DA.

Onderscheid data, meta-data en rules

De scheiding enerzijds tussen meta-data en rules en anderzijds tussen data en rules dwingt onwillekeurig duidelijkheid af in wat tot de verantwoordelijkheid van de DA en wat tot die van de AA behoort.
In zowel de meta-data base als de rule base kunnen - zoals uit het voorafgaande blijkt - betrouwbaarheids- en nauwkeurigheidseisen worden opgenomen. Bij een conventioneel informatiesysteem met DBMS worden doorgaans zoveel mogelijk betrouwbaarheids- en nauwkeurigheidseisen

opgenomen in de meta-data base, die onder beheer valt van de DA. Aan de hand hiervan vindt de verwerking plaats. Nader bekeken zijn deze eisen functioneel te scheiden in eisen ten behoeve van het gegevensbeheer en eisen ten behoeve van het procesbeheer. DA zal niet toestaan, dat gegevensverzamelingen onbetrouwbare informatie bevatten. De betrouwbaarheid van deze informatie dient voor alle hiervan gebruik makende gegevensverwerkende processen te gelden. Zo zal een gegevenselement "datum" numerieke informatie dienen te bevatten, waarbij onder meer 30 februari niet zal mogen voorkomen. Een van de gebruikers zou naast deze eis nog aanvullende eisen kunnen onderkennen, die zelfs te relateren zijn aan een beperkt aantal van zijn toepassingen. Zo zal bijvoorbeeld een bepaalde toepassing niet mogen antedateren. Beide eisen dienen van kracht te zijn. Beide functies (DA en gebruiker c.q. AA) zullen de verantwoordelijkheid hierover moeten kunnen dragen. Echter het beheer over de meta-data base ligt uitsluitend en alleen bij de DA. In dit geval zal bij gebruik van KBS de specifieke gebruikerseis, die te relateren is aan een bepaalde toepassing (ten behoeve van een bepaald bedrijfsproces), niet in de meta-data base maar in de rule base worden opgenomen. De DA zal zijn eisen echter in de meta-data base handhaven. Enige overlap in eisen, die zijn opgenomen in zowel de rule als de meta-data base, is naar alle waarschijnlijkheid ten gevolge van de verantwoordelijkheden van de DA en AA's onvermijdelijk. De DA zal in ieder geval een minimale betrouwbaarheid van gegevens (op concernniveau) trachten af te dwingen middels de meta-data. Voor wat betreft de scheiding tussen data en rules ligt het eenvoudiger. Zoals de definitie van data en rules uit hoofdstuk 2 aangeeft, wordt data gebruikt voor het registreren en sturen van de bedrijfsprocessen en rules gebruikt voor het sturen van de daarvoor benodigde gegevensverwerkende processen. Daarvan uitgaand kan gesteld worden dat artikelprijstabellen, zoals deze kunnen worden aangetroffen in de coding van conventionele systemen, dienen te worden beschouwd als data en niet als rules; derhalve dus onder beheer van de DA en niet onder beheer van de AA te staan. Prijzen zijn immers het resultaat van een (in dit verband commercieel) bedrijfsproces en sturen het (verkoop) bedrijfsproces. Voorts kan de prijs gezien worden als een kenmerk (attribuut) van een te registreren object (artikel).

Figuur 5 schetst het behandelde beheersmodel, dat door de daarin opgenomen systeemprogrammatuur (KBS en OS/DBMS) ondersteund dient te worden.

TECHNISCH BEHEER:	BEHEER:	BEHEER:	GERICHT OP:
Automatiseringsorganisatie op grond van gebruikerseisen	Gebruikersorganisatie	Gebruikersorganisatie	
DBMS/OS Dient data en meta-data te ondersteunen	DATA Registreert en stuurt bedrijfsprocessen eigenaar: Eindgebruikers	META-DATA (+ meta-meta-data) Beinvloedt het gebruik van gegevens. eigenaar: Data Administrator	GEGEVENS
KBS Dient rules en meta-rules te ondersteunen	RULES Vloeit voort uit bedrijfs-processen en stuurt de gegevensverwerking. eigenaar: Eindgebruikers	META-RULES (+ meta-meta-rules) Beinvloedt de stuur-informatie voor de gegevensverwerking eigenaar: Application Administrator	PROCESSEN

HEEFT EFFECT OP

Figuur 5.

*** Met de beheersinstrumenten, meta-data en meta-rules, kan de geautomatiseerde gegevensverwerking derhalve worden beheerd. Met name door de opkomst van de meta-rules wordt een stap vooruit in de beheersbaarheid van administratieve processen bereikt.
De DA-functie is verantwoordelijk voor de meta-data base, die gericht is op het beheer van de gegevens; de AA zal verantwoordelijk worden gesteld voor de meta-rule base, die gericht is op het beheer van de gegevensverwerkende processen. Derhalve zal DA de meta-meta-data (bevoegdheden betreffende de meta-data) en de AA de meta-meta-rules (bevoegdheden betreffende meta-rules) vaststellen. Deze bevoegdheidsinformatie kan in de desbetreffende bestanden worden opgenomen.

6. Controle-aspecten

Uit de voorafgaande hoofdstukken blijken gegevensverwerkende processen te worden gestuurd door de inhoud van:
- de rule base voor wat betreft condities en acties, derhalve stuurinformatie betreffende die processen;
- de meta-rule base voor wat betreft stuurinformatie betreffende de rules, waaronder bevoegdheidsinformatie;
- de meta-data base voor wat betreft stuurinformatie gericht op de opgeslagen gegevens, waaronder bevoegdheidsinformatie en integriteitseisen.

Deze bestanden bevatten voor de organisatie documentatie over de gegevens (meta-data base) en de processen (meta-rule base en rule base). Met behulp van specifieke programmatuur kan informatie in de meta-data base worden toegevoegd, gewijzigd en gelezen. Met behulp van de kennis-vergarende algoritmen zal informatie in de meta-rule base en rule base kunnen worden toegevoegd, gewijzigd en gelezen. Met behulp van de redenerende algoritmen zal het eveneens mogelijk zijn (terugwaarts redenerend) informatie uit de rule base te kunnen lezen. Dit is van bijzonder groot belang voor het uitvoeren van een informatiesysteem-onderzoek. Men zou het informatiesysteem zelf immers kunnen vragen op grond waarvan het iets uitvoert.
Hieruit kan worden geconcludeerd dat de meta-data base, rule base en meta-rule base zowel een directory- als dictionary-functie uitoefenen. Dit garandeert de betrouwbaarheid van de documentatie. De hierin opgenomen informatie geeft immers - omdat het (als stuurinformatie) bepalend is voor de gegevens en gegevensverwerking - een juiste weergave van die gegevens en verwerking. Hierdoor zal de inhoud van deze bestanden geraadpleegd kunnen worden door de EDP-auditor en/of accountant voor het baseren van een oordeel over de algehele betrouwbaarheid

van de geautomatiseerde gegevensverwerking. Ten behoeve van dit raadplegen dient de EDP-auditor vast te stellen, dat genoemde bestanden adequaat worden beheerd (waaronder beveiligd).

*** Voor het beoordelen van het gegevensbeheer en informatiesystemen zal de EDP-auditor in vele gevallen gebruik willen maken van de beschikbare documentatie. Een bekend en bijna immer aanwezig probleem is de betrouwbaarheid daarvan. In het voorafgaande is vermeld hoe daarin ten gevolge van het gebruik van KBS kan worden tegemoet gekomen. Een sterk punt daarbij is de mogelijkheid van achterwaarts redeneren, waardoor de te beoordelen systemen zelf kunnen tonen wat zij op grond waarvan verwerken ("auditing information systems by backward reasoning").

7. Slot

In dit artikel is ingegaan op "Knowledge Based Systems" (KBS) in relatie tot de beheersbaarheid van administratieve processen. Hierbij is uitgegaan van een ideaalsituatie voor wat betreft de door KBS geboden faciliteiten, waardoor impliciet een model tot stand is gebracht. In de praktijk kan het echter nog enige tijd duren voordat in enige mate aan dit model zal worden voldaan. Hierdoor zal men nog geruime tijd geconfronteerd worden met afwijkingen op dit model. Dit betekent echter wel dat enerzijds de consequenties van deze afwijkingen duidelijk kunnen worden onderkend; anderzijds het streven naar het bereiken van de ideaalsituatie eveneens duidelijk kan worden geleid. Dit alles in relatie tot beheersbaarheid.

*** Met dit artikel hoopt de auteur, naast de behandeling van KBS inzake de beheersbaarheid van administratieve geautomatiseerde processen, tevens eventuele mysterie rond KBS te hebben opgehelderd, waarbij overigens wel zij opgemerkt dat de benaming wellicht beter zou kunnen luiden "Knowledge Base Systems". Op andere dan binnen conventionele systemen bestaande kennis zijn deze systemen immers niet gebaseerd, maar de kennis (Knowledge) is hierbij wel uit de programma-coding gehaald en in daarvoor speciaal ingerichte bestanden (Base) geplaatst.

COMPACT is een uitgave van de AC-groep van
KMG Klynveld Kraayenhof & Co.

Lente 1985

VIERDE GENERATIETALEN

door drs. J.E. Huizenga

1. Inleiding

In de grote Van Dale wordt "gebruiker" omschreven als "degeen die iets gebruikt of zich van iets bedient, inzake werktuigen of boeken" of als "hij die het zakelijk recht van gebruik van eens anders zaak heeft". In automatiseringsland is het gebruik om een ieder die niet tot de automatiseerders behoort als gebruiker aan te duiden. Zoals uit bovenstaande omschrijvingen valt af te leiden heeft het begrip gebruiker een tamelijk afstandelijke gevoelswaarde.
De gebruiker mag gebruiken, maar heeft geen zeggenschap en dus geen invloed.

Het moge duidelijk zijn dat een dergelijke afstandelijke benadering niet de bedoeling kan en mag zijn!

N.B. Over de voor de hand liggende associatie van "gebruiker" met verslavingsverschijnselen wil ik niet speculeren.

Een van de dingen die gebruikers constateren is dat er vaak een aanzienlijk gat is tussen de door hen gewenste en de gerealiseerde automatiseringstoepassingen. Dit gat heeft zowel een kwantitatief als een kwalitatief karakter.
Andere benamingen voor het gat zijn "application backlog" en voor het verschijnsel "software crisis".
Met name de laatste term duidt op een gat in de markt voor een ieder die belooft de crisis te zullen bezweren zich zal richten.
In dit kader zijn en worden een aantal software-produkten op de markt gebracht die de bedoeling hebben het gat te dichten.
Deze produkten worden doorgaans "Vierde Generatie Talen", danwel "Vierde Generatie Hulpmiddelen" genoemd, verder aan te duiden als 4GL's.

In dit artikel zal getracht worden een aantal kenmerken van 4GL's te presenteren, waarbij niet zal worden ingegaan op de merites van individuele produkten.
Het belang van 4GL's voor de drie niveaus van management (operationeel, tactisch en strategisch) zal worden aangegeven.
Tevens komen aan de orde de invloed van 4GL's op de systeemontwikkelingscyclus en op het onderhoud van toepassingen.

Op de hiervoor genoemde aspecten van automatiseringstechnische aard zal slechts beperkt worden ingegaan met als doel de technische achtergrond te schetsen van de invloed van 4GL's op de organisatie, van zowel de automatisering als de gebruikers en hun beheer over de geautomatiseerde systemen. Hierbij zal bijzondere aandacht worden geschonken aan de invloed op delegatie- en verantwoordingspatronen (lees interne controle).
Langs deze weg kan een verkenning worden uitgevoerd van de invloed op controleerbaarheid van de toepassingen, primair binnen de eigen organisatie, maar ook voor de controlerende accountant.

Hij - de accountant - kan op twee wijzen te maken krijgen met 4GL's.
In de eerste plaats doordat zijn cliënt voor hem relevante toepassingen ontwikkelt en onderhoudt met behulp van een 4GL.
In de tweede plaats kan een 4GL wellicht gebruikt worden door de accountant bij het ontwikkelen van programmatuur ten behoeve van bestandsonderzoeken.

2. Niveaus van automatisering

Voor de behandeling van 4GL's is het nuttig onderscheid te maken tussen de drie bekende beslissingsniveaus:
- operationele beslissingen (frekwent, korte termijn, gedetermineerd, laag aggregatieniveau, proces gericht en met een beperkte reikwijdte);
- tactische beslissingen (regelmatig, middellange termijn, gestructureerd, matig aggregatieniveau, gericht op bedrijfsonderdelen, voornamelijk interne informatie verwerkend);
- strategische beslissingen (incidenteel, lange termijn, slecht gestructureerd, hoog aggregatieniveau, gericht op de gehele onderneming, veel externe informatie verwerkend).

Volgens dezelfde indeling worden informatiesystemen geclassificeerd, waarbij dan vaak de klasse "automaten" wordt toegevoegd.
Traditioneel richtte de administratieve automatisering zich voornamelijk op het ondersteunen en soms nemen van operationele beslissingen.
Voor tactische beslisssingen werd slechts het ruwe basismateriaal aangeleverd, terwijl de automatisering voor de strategische besluitvorming nauwelijks enige betekenis had.
Wanneer bedacht wordt met hoeveel inspanning in de afgelopen jaren systemen van enige omvang en relevantie werden ontwikkeld, is het niet verwonderlijk dat automatisering slechts gerealiseerd werd voor die processen die met een hoge frekwentie worden uitgevoerd.

3. Karakteristiek van 4GL's

Een 4GL zou, om die naam te mogen dragen volgens Martin, moeten bestaan uit de volgende componenten:
- informatie-opvraag taal (query language), waarbij de opdracht direct wordt uitgevoerd;
- overzichten generator (report generator), waarbij de opdracht later, als een batch-job wordt uitgevoerd;
- afbeeldingen generator (graphics generator);
- genereren van toepassingen door en voor de gebruikers (application generator);
- krachtige ondersteuning van programmeurs (high level programming).

Deze componenten dienen in dezelfde taal te worden toegesproken, zodat in ieder geval naar de programmeur/gebruiker toe een eenduidige interface ontstaat.
Hieraan dient een data dictionary/directory te worden toegevoegd. Deze dient zoveel mogelijk accurate beschrijvingen over gegevens en processen te bevatten, welke informatie tevens stuurinformatie bevat voor het verwerkende proces.
Zij het in andere bewoordingen, wordt deze karakteristiek in de literatuur algemeen onderschreven.
De tendens is duidelijk: met het opvolgen van de generaties wordt van degene die de computer opdrachten verstrekt (programmeur of gebruiker) steeds minder machinekennis vereist, waardoor meer inspiratie en concentratie beschikbaar blijft voor het oplossen van het eigenlijke probleem en vooral het goed definiëren daarvan.
Dit gaat gepaard met een aanzienlijke lastenverschuiving naar de apparatuur.

In het algemeen worden de volgende voordelen van 4GL's verwacht:
- gebruikers kunnen zonder tussenkomst van systeemontwikkelaars opvragingen uit de bestanden doen;
- gebruikers kunnen zonder tussenkomst van systeemontwikkelaars overzichten uit de bestanden krijgen;
- betere participatie van de gebruikers tijdens de ontwerpfase, waardoor ontwerpfouten minder zullen voorkomen;
- het wordt eenvoudiger een model van het toekomstige systeem te maken, ten einde de kans op ontwerpfouten te verkleinen;
- gebruikers kunnen meer eenvoudige toepassingen zelf onwikkelen;
- systeemontwikkelaars krijgen geavanceerde hulpmiddelen ter beschikking, waardoor hun produktiviteit sterk kan worden verhoogd;
- onderhoud aan systemen kan eenvoudiger en sneller worden gepleegd.

Ten einde met een geautomatiseerd systeem te kunnen werken, zullen de volgende componenten gedefinieerd moeten zijn alsmede, waar nodig, de relaties tussen de componenten:
- de toepassingsprogramma's;
- de data base;
- het terminalnetwerk;
- de job control-specificaties.

Verder dient in goede afstemming tussen de genoemde componenten de back-up en recovery geregeld te zijn.
Bij voorkeur zijn de parameters waarmede de componenten werken opgeslagen in een data dictionary/directory, zodat ze eenduidig zijn vastgelegd en met hun onderlinge relaties kunnen worden weergegeven.

Naar hun aard kunnen 4GL's verdeeld worden in twee groepen:

Programmageneratoren
De programmageneratoren zijn slechts behulpzaam bij het maken van de toepassingsprogramma's.

De werkwijze is doorgaans als volgt:
De programmeur/gebruiker codeert de toepassing in een specificatietaal of door het afwerken van een vragenlijst. Op grond van de specificaties of de antwoorden op de vragenlijst, wordt een programma gegenereerd in een van de derde generatietalen, zoals COBOL.
Op deze wijze wordt de programmeur/gebruiker gevrijwaard van het coderen van allerlei instructies van huishoudelijke aard, die een derde generatie compiler nu eenmaal nodig heeft, maar die met het oplossen van het probleem, zoals de gebruiker dat ziet, weinig verband houden.
De omgeving, waarin de gegenereerde toepassing moet werken is niet gedefinieerd. Hiervoor zullen de gespecialiseerde automatiseringsmedewerkers op dezelfde wijze als voorheen de nodige parameters opstellen.

Applicatiegeneratoren
Applicatiegeneratoren bevatten wel alle componenten welke nodig zijn om een werkende toepassing te genereren.

Binnen deze groep zijn weer twee klassen te onderscheiden:
- Applicatiegeneratoren, die als pre-processor werken en parameters voor de bestaande systeemprogrammatuur (data base management systeem, teleprocessing monitor en besturingsprogrammatuur) genereren, de modulaire generatoren, enerzijds
- en de applicatiegeneratoren, die in een pakket hun eigen data base management systeem en teleprocessing monitor bevatten, de "self-contained packages", anderzijds.

4. De betekenis van 4GL's voor de systeemontwikkeling

De thans beschikbare 4GL's leveren doorgaans in combinatie met de huidige computers toepassingen op die qua prestaties (verwerkingssnelheid en beslag op de machinecapaciteit) hun meerdere moeten erkennen in de met traditionele middelen gerealiseerde toepassingen. Dit zou echter - als het goed gaat - gecompenseerd dienen te worden door:
- lagere systeemontwikkelingskosten;
- gedeeltelijk wegwerken van de back-log.

Wat kan nu de betekenis zijn van 4GL's voor de onderscheiden beslissingsniveaus?

Operationele beslissingen

Voor de "beslissingen-fabriek" geldt dat de beslissingen per stuk weinig tijd mogen kosten en dat het beslissingsproces slechts incidenteel wordt gewijzigd.
De programmatuur moet dus efficiënt zijn en zal slechts incidenteel onderhoud behoeven.
Hieruit valt af te leiden dat de bijdrage van 4GL's beperkt zal zijn tot de systeemanalyse en -ontwerpfase, waarbij het met name gaat om een zo vroeg mogelijke confrontatie tussen het te ontwikkelen systeem en de daaraan te stellen functionele eisen. De gedachten gaan hierbij in de richting van prototyping in enigerlei vorm.
In het streven naar hoge efficiency in het machinegebruik zal de toepassing slechts met behulp van een 4GL worden gerealiseerd, indien deze voor het te verwachten transactievolume een aanvaardbare combinatie van beslag op de machinecapaciteit, verwerkingssnelheid en per tijdseenheid te verwerken transactievolume kan opleveren.

Tactische en strategische beslissingen

Bij het nemen van tactische/strategische beslissingen is de beslissingsruimte slecht gedefinieerd, zijn de beslissingsregels slecht gestructureerd, wordt gewerkt met geaggregeerde gegevens en wordt vaak gebruik gemaakt van externe gegevens.
In een dergelijke situatie ontbreekt de tijd voor een volledige systeemontwikkelingscyclus. Doordat de programmatuur niet zeer frekwent zal draaien is het efficiency aspect van minder belang.
Het ligt derhalve voor de hand juist voor dit soort beslissingen systemen te ontwikkelen met behulp van vierde generatie applicatie generatoren.

Indien door ontwikkelingen in de computertechniek en door optimalisatie van de 4GL's in de toekomst de computersnelheid en -capaciteit geen belemmering meer vormen, is de relevantie van het hiervoor geschetste onderscheid achterhaald.

Een ervaring

Een van de weinige goed beschreven ervaringen met het gebruik van een 4GL in de ontwikkeling van een operationeel systeem betreft een bank-organisatie in de VS.
Nadat een project geïnitieerd is, volgt een toepasbaarheidsonderzoek, waarin vooral gekeken wordt naar de mate waarin het probleem reeds kan worden gespecificeerd en de te verwachten prestatie-eisen.
Op grond van het toepasbaarheidsonderzoek wordt beslist of het systeem op de traditionele wijze zal worden ontwikkeld, danwel dat eerst een prototyping fase zal worden doorlopen.
Wordt voor het laatste beslist dan dient door de toekomstige gebruikers reeds in de prototyping fase een manager fulltime uit hun organisatie voor het project te worden vrijgemaakt.
In de prototyping fase wordt iteratief een model van het toekomstige systeem ontwikkeld. Dit model wordt mede door de gebruiker ontwikkeld en voortdurend door hem getoetst. Gaat hij eenmaal akkoord met het model, dan worden op basis daarvan de definitieve specificaties voor het systeem opgesteld. Deze specificaties worden bevroren!
Het definitieve ontwerp gebeurt traditioneel, waarbij elk deel van het prototype, dat alsnog gebruikt kan worden, "meegenomen" is.

5. Enige attentiepunten

In dit hoofdstuk zal aandacht worden besteed aan een aantal punten welke uit de praktijk van het gebruik van 4GL's naar voren zijn gekomen. Het betreft aandachtspunten die betrekking hebben op de traditionele ontwikkelingsorganisatie enerzijds en punten die voortvloeien uit systeemrealisatie door gebruikers (end-user computing) anderzijds.

Ontwikkelingsorganisatie

- Doordat de gebruiker het prototype heeft zien werken, zal er weinig begrip zijn voor het feit dat het prototype ook niet meer is dan de term aangeeft (een voorafbeelding, een eerste afdruk). Hierdoor ontstaat het gevaar dat onrijpe systemen operationeel worden verklaard.

- Weliswaar neemt door het gebruik van 4GL's de totale geïnvesteerde men kracht af, de ervaring is dat de totale doorlooptijd per project niet spectaculair wordt bekort.
- Op elk project is tenminste een zeer vakbekwame programmeur nodig.
- Doordat de omvang van de projectgroep sterk terugloopt, is er meer afhankelijkheid van een sleutelfiguur. Ter waarborging van de continuïteit van het project zal dus zeer effectief moeten worden gedocumenteerd.
- Bij onvoldoende sterke gebruikers bestaat het gevaar dat de programmeur beslissingen neemt die het functionele ontwerp betreffen, waardoor een belangrijk voordeel van de mogelijkheid tot actieve gebruikersparticipatie dankzij 4GL's wordt weggegooid.
- Voor een goede voortgang is het van eminent belang dat de gebruikers snel (kunnen) reageren op de resultaten van een versie van het prototype.

End-user computing

- Wanneer gebruikers hun eigen applicaties realiseren zijn zij ook verantwoordelijk voor het onderhoud en kunnen derhalve in principe op ieder moment hun toepassing wijzigen. Indien echter hun toepassing gegevens aanlevert aan een ander proces, bestaat het gevaar van verkeerd bijwerken van de data base.
- De resultaten van de end-user computing worden gepresenteerd als resultaten van de standaard gegevensverwerking, ze zien er "echt" uit. De in de end-user computing gebruikte algoritmen worden vaak minder goed getest dan in de klassieke systeemontwikkeling, waardoor de resultaten minder betrouwbaar kunnen zijn.
  Wellicht is dit probleem op te lossen door op iedere pagina af te drukken "Pas op, door een gebruiker geprogrammeerd; kan onbetrouwbaar zijn". Dit laatste is als een grapje bedoeld, maar er zit een kern van waarheid in.
- De manier waarop de computer werkt onttrekt zich geheel aan de kennis van de gebruikers, waardoor zij de op te lossen problemen zo kunnen formuleren en ter verwerking aanbieden dat een uiterst inefficiënt gebruik wordt gemaakt van de computercapaciteit.
- De gebruikers zijn geen professionele automatiseerders en werken doorgaans betrekkelijk geisoleerd. Hierdoor loopt de organisatie het risico dat relatief dure mensen relatief eenvoudige programma's maken, wellicht ter oplossing van problemen die reeds elders in de organisatie zijn opgelost.

Verdere attentiepunten blijven:
- Continuïteit (backup/recovery/documentatie).
- Vertrouwelijkheid van de gebruikte gegevens en de gecreëerde informatie.

Een praktische oplossing die wel wordt aangetroffen is een scheiding tussen de gegevensverwerkende "fabriek" en het beslissingsondersteuningssysteem (decision support system) door deze laatste op verschillende computersystemen onder te brengen (tegenwoordig ook wel personal computers). De fabriek verwerkt gegevens ten behoeve van de operationele beslissingen en maakt gebruik van zo efficiënt mogelijke programmatuur en hulpprogrammatuur (data base management systeem en teleprocessing monitor).
Het decision support system maakt gebruik van gegevens die, veelal in samengevatte vorm, worden overgezonden vanuit de fabriek. Deze gegevens worden bewerkt met applicatiegeneratoren.
Het verdient aanbeveling maatregelen te nemen dat geen gegevens terug kunnen stromen naar de fabriek, zodat niet vanuit de minder controleerbare end-user computing omgeving vervuiling kan optreden van de basisgegevens.

Verder is het van belang dat goede, ook voor de gebruikers begrijpelijke beschrijvingen van de gegevens worden opgesteld, dat vastgelegd is wie welke bevoegdheden heeft ten aanzien van welke gegevens en dat organisatorische maatregelen worden genomen met betrekking tot het beschikbaar stellen van bestanden.
Het verdient aanbeveling te bevorderen dat standaardroutines voor standaardbewerkingen bij de gebruikers bekend zijn.

6. Controle

Dat het onderwerp "automatisering en accountant" als zodanig bestaat, komt voornamelijk voort uit het feit dat de gebruikers de automatisering niet zelf ter hand hebben genomen.
Door de complexiteit en weerbarstigheid van de automatiseringsmaterie is er een aparte bedrijfsfunctie automatisering ontstaan, waaraan de primaire bedrijfsfuncties (inkoop, verkoop, produktie etc.) hun gegevensverwerkende processen, zowel ontwerp, als bouw, als verwerking, hebben gedelegeerd. Zij - de primaire functies - staan daarmee voor het vraagstuk hoe dit te controleren.
Een groot deel van de automatiseringsproblematiek is terug te voeren tot de problemen die samenhangen met de delegatie van een belangrijk element (de gegevensverwerking) door alle bedrijfsfuncties aan een verbijzonderde afdeling (de Automatiseringsafdeling).
De 4GL's houden de belofte in dat een deel van die gedelegeerde activiteiten weer onder de hoede en invloed van de primaire bedrijfsfuncties kunnen worden gebracht.

4GL bij cliënt

Uiteindelijk wordt de controlerend accountant slechts geconfronteerd met de resultaten van de systeemontwikkeling, te weten:
- een toepassingssysteem, dat in een geautomatiseerde omgeving operationeel is;
- het gebruik van dat systeem door gebruikers en de betrokken automatiseerders (werkvoorbereiders, operators etc.);
- de resultaten van de gegevensverwerking.

De manier waarop het systeem tot stand is gekomen zou voor de accountant transparant kunnen zijn.
Het is vooral uit doelmatigheidsoverwegingen dat hij bij de systeembouw op kritische momenten gaat (respectievelijk laat) vaststellen dat in voldoende mate rekening is gehouden met aspecten van betrouwbaarheid, controleerbaarheid en continuïteit (zie laatste alinea van deze paragraaf).
Wat dat betreft verandert er weinig bij het gebruik van programmageneratoren.
Bij gebruik van applicatiegeneratoren ten behoeve van end-user computing is de belangstelling van de accountant gekoppeld aan het gebruik van de resultaten van de gebruikerstoepassing. Voorbeelden zijn:
- toepassingen, waarvan de resultaten in de jaarrekening worden verwerkt, als specificaties of in de vorm van prognoses.
  Voorop dient te staan dat de kwaliteit van het accountantsoordeel niet mag afhangen van de hulpmiddelen waarmee de specificaties en/of prognoses zijn gemaakt.
  Het is dan ook van het grootste belang dat het gehanteerde berekeningsmodel goed te volgen is en dat de stappen in de berekening zijn gemarkeerd door tussenresultaten, zodat het gehele proces goed controleerbaar is.
  Is dat niet het geval en de cliënt kan of wil het model niet aanpassen, dan zal de accountant zelfstandig de resultaten berekenen of benaderen. Hiervoor kan hij ofwel dezelfde hulpmiddelen gebruiken als zijn cliënt (mits zijn onafhankelijkheid daardoor niet in gevaar komt) of met eigen programmatuur (grote danwel microcomputer) de noodzakelijke berekeningen uitvoeren;
- toepassingen, waarvan de resultaten gebruikt worden bij het voorbereiden van beleidsbeslissingen, welke wellicht gevolg kunnen hebben voor de continuïteit van de onderneming.
  Hier zal de accountant zeker het management moeten wijzen op het belang van goede controles in het te hanteren model. Of hij ook, zonder daartoe een opdracht te hebben, zich een oordeel moet vormen over deze modellen is een vraag waarop hier niet wordt ingegaan. Als dat oordeel gevraagd wordt, geldt hetzelfde als hiervoor met betrekking tot specificaties/prognoses is gesteld.

4GL gebruikt door accountant

De door accountants uit te voeren bestandsonderzoeken zijn qua karakteristiek te vergelijken met geautomatiseerde systemen gericht op het voorbereiden van tactische en strategische beslissingen.
Het zou dus niet verbazen als die accountants ook gebruik zouden maken van 4GL's.
In feite wordt er reeds jaar en dag met (gespecialiseerde) reportwriters gewerkt, terwijl de programmatuur op de microcomputers grote gelijkenis vertoont met applicatiegeneratoren.
Voor de accountants die hiervan gebruik maken gelden derhalve dezelfde opmerkingen als in het vorige hoofdstuk zijn aangegeven!

COMPACT is een uitgave van de AC-groep van
KMG Klynveld Kraayenhof & Co.

INFORMATIE OVER (ELEKTRONISCHE) INFORMATIE "WERKEN MET EEN DATABASE"

door D. Boom

Inleiding

De laatste jaren ondervindt het gedrukte tijdschrift sterke concurrentie van de zogenaamde bibliografische databases, die de tijdschriftartikelen in elektronische vorm aanbieden. Hoe dat gebeurt, welke informatie, en tegen welke kosten dit geschiedt wordt in dit artikel uiteengezet.

Elektronische informatie

De snelle ontwikkeling van de automatiseringstechniek en de forse daling van de daaraan verbonden kosten hebben het mogelijk gemaakt de grote stroom publicaties ook in een andere vorm dan de gedrukte versie, aan te bieden.
De gang van zaken is als volgt: de door een auteur aangeboden tekst wordt getypt op een magneetband van een computergestuurde fotozetmachine.
Deze magneetbanden waarvan de inhoud overeenkomt met de inhoud van de gedrukte uitgaven, leveren een voor de computer leesbaar en manipuleerbaar bestand op. Als extra bewerking wordt de tekst voorzien van zoekgegevens.
De aldus ontstane magneetband wordt vervolgens geconverteerd en toegevoegd aan het gegevensbestand van een databank- of database-exploitant (elektronische uitgever).

Databanken (A) en databases (B)

Een database- (of databank)organisatie (ook wel host- of gastheerorganisatie genoemd) is een bedrijf of instelling die computercapaciteit exploiteert, waarin gecomputeriseerde gegevensbestanden worden ondergebracht.
We onderscheiden in principe twee soorten gegevensbestanden:
A. factuele gegevensbestanden (Databanken);
B. bibliografische gegevensbestanden (Databases).

A. Databanken, die ook wel factuele of feitelijke gegevensbestanden worden genoemd, bevatten concrete gegevens op een specifiek terrein.
Als voorbeeld kunnen wij de volgende gegevensbestanden noemen:
- Hoppenstedt (ABC voor handel & industrie) handelsinformatie;
- Centraal Beheer met de volledige teksten van Subsidies voor het Nederlandse bedrijfsleven;
- Vermande met de Nederlandse wetgeving;
- Predicast met marktinformatie;
- IBM met statistische landeninformatie (OECD);

Het voornaamste kenmerk van databanken is dat men op een vraag concrete gegevens ten antwoord krijgt, die vervolgens onmiddellijk gebruikt kunnen worden.

B. Databases, ook wel bibliografische gegevensbestanden genoemd, bevatten niet de concrete gegevens zelf, maar verwijzen naar de oorspronkelijke documenten, bijvoorbeeld: boeken, tijdschriften, kranten, conferentieverslagen, etc.
Verwijzingen naar de bestaande literatuur.
Als voorbeeld kunnen wij noemen:
- Ministerie van Economische Zaken met bedrijfskundige informatie en exportmogelijkheden/landeninformatie;
- Management Contents met verwijzingen naar bedrijfskundige of management literatuur;
- Computer Contents over automatiseringsonderwerpen;
- PARAC. Het parlementaire automatiseringscentrum met informatie over onder andere kamerstukken, handelingen en de parlementaire persdocumentatie;
- Kluwer's Juridische Databank met Burgerlijke en Strafzaken Jurisprudentie als ook Beslissingen in Belastingzaken.
- IEEE: de Amerikaanse elektronische ingenieursorganisatie met veel technische onderwerpen over onder andere computers, toepassingen, etc.

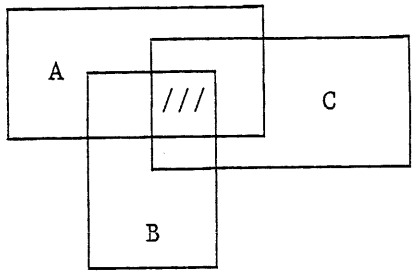
(Er is overigens een tendens gaande dat bibliografische databases ook de volledige tekst gaan leveren.)

Op basis van deze hiervoor genoemde bestanden kunnen de database-organisaties drie soorten diensten verlenen:
1. On-line;
2. Selective Dissemination of Information (SDI);
3. Down-loading.

On-line

Via een telefoonlijnverbinding en een daarop aangesloten terminal kan men de gewenste informatie krijgen. Vooraf moet een keuze gemaakt worden welke database benaderd wordt.

Ten einde toegang tot de gekozen database te verkrijgen, dient men wel de noodzakelijke toegangswoorden (passwords) te bezitten voor het telecommunicatienetwerk (PTT) en de betreffende host.
Na diverse "in-log"-procedures kan het zoekproces beginnen door het intoetsen van trefwoorden, auteursnamen of titels, de zogenaamde zoekgegevens.
Een karakteristieke eigenschap van het "on-line" zoeken is, dat er op combinaties van woorden en zoektermen gezocht kan worden.
Door middel van operatoren "and", "or", "not", kunnen terugzoekkenmerken met elkaar worden gecombineerd tot de gewenste publicatie(s) is/zijn gevonden.
Een voorbeeld moge dit verduidelijken:

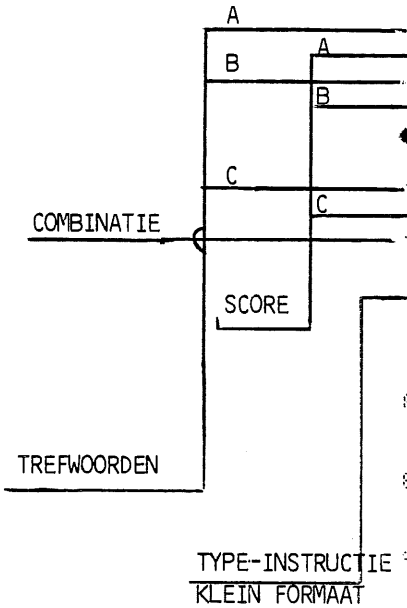


Vraag: Literatuur over beheer en beveiliging van de programmabibliotheek bij voorkeur in het Nederlands geschreven.
Allereerst wordt de host-organisatie gekozen waarvan men denkt dat deze de gezochte informatie kan leveren.

A. Bevat alle artikelen over EDP-Management (4508 artikelen);
B. Bevat alle artikelen over Security of data (3455 artikelen);
C. Bevat alle artikelen in de Nederlandse taal geschreven (5082 artikelen).

Waar de vlakken elkaar overlappen (gearceerde gedeelte) bevinden zich de artikelen die aan alle drie zoekgegevens voldoen. In het voorbeeld (figuur 1) kunt u het scoreverloop volgen. In dit voorbeeld voldoen maar 5 artikelen aan alle drie criteria. Vervolgens een "print"opdracht op titelniveau en één op "abstract" niveau. Toevallig een KKC-artikel!!

Lente 1985




```
? BB
----------18JUN85  11:30:02  USER3715---
     0.21 AU 0.27 MINUTES IN FILE  32
     0.21 AU APPROX TOTAL
FILE   8:INSPEC:1971-85,13
SET ITEMS DESCRIPTION (+=OR;*=AND;-=NOT)
--- ----- ------------------------------
? S E.D.P. MANAGEMENT
1   4508 E.D.P. MANAGEMENT
? S SECURITY OF DATA
2   3455 SECURITY OF DATA
? S LA=DUTCH
4   5082 LA=DUTCH
? C 1AND2AND4
5      5  1AND2AND4
? T5/6/1-5

                TYPE 5/6/1-5
80C019582  INSPEC  JOURNAL PAPER  80C19582
  CONTINUITY IN THE STATE COMPUTER CENTRE FOR DISASTERS

80C019797  INSPEC  JOURNAL PAPER  80C19797
  MANAGEMENT OF THE PROGRAM LIBRARY

80C019796  INSPEC  JOURNAL PAPER  80C19796
  THE SECURITY RULES OF A COMPUTER CENTRE AS APPLIED IN PRACTICE

80C019795  INSPEC  JOURNAL PAPER  80C19795
  ORGANISATIONAL ASPECTS OF DATA SECURITY

79C014176  INSPEC  JOURNAL PAPER  79C14176
  OWNERSHIP AND CONTROL OF INFORMATION SYSTEMS
```

TYPE-INSTRUCTIE
GROOT FORMAAT

```
? T5/4/2

                TYPE 5/4/2
80C019797  INSPEC  JOURNAL PAPER  80C19797
  MANAGEMENT OF THE PROGRAM LIBRARY
  KAMSTRA, A.; MOSCH, P.T.; WIEGERS, H.; VAN ZANEN, P.
  KLYNVELD KRAAYENHOF & CO., AMSTERDAM, NETHERLANDS
  INFORMATIE (NETHERLANDS), VOL.22, NO.3 P.189-97, 5 REFS, MARCH
1980 . CODEN: INFTC
  . IN DUTCH
  TREATMENT GENERAL

  A FLOW DIAGRAM IS GIVEN, SHOWING THE STEPS TO BE TAKEN TO ACTIVATE
A GIVEN PROGRAM. SECURITY REQUIREMENTS DEMAND THE PRODUCTION OF
APPROPRIATE AUTHORISATION AT SEVERAL STAGES. BY STORAGE OF THE
APPROPRIATE STOCK AND FINANCIAL DATA, THE ACCOUNTANT CAN MAINTAIN A
CONSTANT CHECK ON THE PROGRESS OF A BUSINESS UNDERTAKING

  CLASSIFICATION CODES: C0210
  CONTROLLED TERMS: E.D.P. MANAGEMENT ; SECURITY OF DATA
  UNCONTROLLED TERMS: PROGRAM LIBRARY ; FLOW DIAGRAM ; SECURITY
REQUIREMENTS ; AUTHORISATION ; FINANCIAL DATA ; ACCOUNTANT ; EDP
MANAGEMENT

? LOGOFF
----------18JUN85  11:34:16  USER3715---
     7.23 AU 4.25 MINUTES IN FILE   9
     0.28 AU 1 ONLINE PRINT
```

KOSTEN

Selective Dissemination of Information
Berust min of meer op hetzelfde principe als hiervoor beschreven. De abonnee of gebruiker moet de door hem gekozen trefwoorden op een meer permanente basis opgeven, waarna de "host" na iedere actualisering met nieuwe literatuur een zoekopdracht uitvoert en de resultaten rechtstreeks (per post) naar de gebruiker stuurt.
SDI is goed te gebruiken om op een specifiek terrein bij te blijven. Uiteraard is het mogelijk de trefwoorden te wijzigen en zodoende de interessegebieden te verbreden, versmallen of veranderen.

Down-loading
Een betrekkelijk nieuw fenomeen is "down-loading" of wel het (toegestaan) kopiëren van een gegevensbestand. Dit kan zinvol zijn als een gebruiker of groep van gebruikers binnen een onderneming frequent een bepaald gegevensbestand raadpleegt.
Men kopieert dan dat bestand op een eigen computersysteem. Hiervoor wordt er door de "host" een eenmalige vergoeding berekend. De gebruiker is dan echter vrij in het gebruik van dat bestand.
Een leuke eigenschap van "down-loading" is dat de gebruiker de mogelijkheid heeft zijn eigen bibliotheek of documentatie in het "grotere" bestand kan onderbrengen.

Kosten

Vanuit Nederland zijn ongeveer een tiental database-organisaties aanspreekbaar die te zamen zo'n 400 bestanden aanbieden. Ze zijn verspreid over Europa en de USA. Sommige host-organisaties zijn gespecialiseerd in een bepaald vakgebied.

Het contract of abonnement kost op zich in vele gevallen niets.
Betaald wordt het aantal opvragingen en de tijd dat de aanvrager en database-organisatie met elkaar verbonden zijn.
De kosten hiervan bedragen ongeveer f 75,-- tot f 250,-- per uur online.

De ervaring heeft ons geleerd dat de kosten van een gemiddelde zoekprocedure zo'n f 100,-- bedragen.

Apparatuurkosten bedragen ongeveer f 4.000,-- voor een eenvoudige terminal zonder beeldscherm tot f 15.000,-- voor een complete microcomputer. De kosten voor het gebruik van het telecommunicatienetwerk, waarvan de prijs wordt bepaald door de telefoonkosten tussen abonnee en exploitant van de desbetreffende database (bijvoorbeeld Apeldoorn voor Centraal Beheer en Deventer voor Kluwer) moeten nog hierbij worden gerekend.

De buitenlandse database-organisaties zijn te benaderen gebruik makend van Datanet. Het knooppunt van deze netwerken bevindt zich in Amsterdam en Den Haag.

Slot

Het spreekt vanzelf dat bij het besluit om van een databank gebruik te maken, afgewogen moet worden of de kosten verantwoord zijn tegenover het nut van de informatie die ter beschikking komt.
Bij het vragen van de informatie moet goed worden gedefinieerd (keuze trefwoorden) welke gegevens gewenst zijn. Daarom is het ook noodzakelijk dat de gebruiker een opleiding volgt over hoe men de literatuur in het gegevensbestand benadert en hoe met deze te manipuleren.

Elektronische bibliotheken zullen de traditionele bedrijfsbibliotheek (nog) niet verdringen maar een welkome aanvulling zijn op het boekenfront. Het gebruik van een databank weegt vaak op tegen het dagenlang zoeken naar informatie in tijdschriften en boeken.
Het Bureau Documentaire Informatieverstrekking (BDI) is aangesloten op enkele database-bestanden.
Indien u vragen heeft aarzel dan niet deze te stellen.

COMPACT is een uitgave van de AC-groep van
KMG Klynveld Kraayenhof & Co.

TIJDSCHRIFTEN

Lente 1985

COMPUTERS THAT ARE "NEVER" DOWN

IEEE Spectrum, April 1985, G. Zorpette

door W.C. Bakker

In dit artikel worden aspecten behandeld ten aanzien van fault-tolerant computersystemen, welke van belang zijn voor de technisch-geïnteresseerde EDP-auditors.

Steeds meer valt te bespeuren dat computers het hart vormen van de bedrijfsvoering, waardoor de computers voortdurend beschikbaar moeten zijn.
Fault-tolerant houdt in dat een computer correct moet kunnen blijven werken, ondanks een opgetreden systeemfout in hard- dan wel software.

Momenteel is de ontwikkeling van fault-tolerant computersystemen gebaseerd op gebruik van meerdere microprocessoren in de architectuur daarvan.

Architectuur

In de architectuur van fout-tolerant computersystemen kan het volgende onderscheid worden gemaakt:

Loosely-coupled processoren → de processoren zijn onderling min of meer onafhankelijk van elkaar; ze hebben ieder hun eigen geheugen en kopie van het operating system.

Tightly-coupled processoren → de processoren hebben een gemeenschappelijke clock en vaak ook gemeenschappelijk geheugen en operating system. (Deze architectuur wordt momenteel het meest gebruikt.)

Een superieure architectuur is niet aan te geven; toch kunnen wel voor- en nadelen van beide worden opgesomd.

	Loosely-coupled	Tightly-coupled
Voordelen	relatief eenvoudig is een hoge graad van fout-isolatie te implementeren.	werkt efficiënter dan loosely-coupled systeem.
Nadelen	minder efficiënt dan tightly-coupled systeem in verband met extra coördinatie-arbeid.	memory blijft non-redundant, hetgeen een kans op fouten geeft, die het hele systeem beïnvloedt.

Hierboven is de term fout-isolatie genoemd; dit is de functie om te voorkomen dat een opgetreden systeemfout in een processor of in een incorrect memory-segment vernielingen aanricht in systeemelementen of databases in het computersysteem.

Het gemeenschappelijk geheugen bij tightly-coupled systemen en het daarmee verband houdende nadeel van het ontstaan van bottle-necks, is bij veel systemen ondervangen door aan iedere processor een stuk cache-geheugen (dat kleine, vaak te gebruiken programma's bevat) te "plakken", zodat minder toegang tot het hoofdgeheugen nodig is.

Hoewel bij on-line transactieverwerking in financiële ondernemingen een minimale hoeveelheid computer-downtime nog getolereerd kan worden, moet de integriteit van de financiële gegevens altijd verzekerd zijn. Bij tightly-coupled systemen wordt dit gewaarborgd door het hoofdgeheugen dubbel uit te voeren op verschillende print-boards, waarbij iedere page tweemaal op verschillende geheugenstukken wordt opgeslagen. Indien de ene versie wordt vernietigd, dan is de andere nog beschikbaar. (Uiteraard worden de twee geheugengebieden benaderd met behulp van verschillende bussen en poorten.)

Foutontdekking

Fouten worden ontdekt middels het vergelijken van de resultaten van onafhankelijk van elkaar uitgevoerde identieke processen, error-detecting microcode en pariteitstechnieken.

Fout-isolatie bij systemen met tightly-coupled processoren

Bij fault-tolerant computersystemen met tightly-coupled processoren kan zowel hardware als software worden gebruikt om systeemfouten te isoleren.

Als maatregelen in de hardware ten behoeve van fout-isolatie kan men denken aan het ontkoppelen van een processor. Om erachter te komen of in een processor een systeemfout is opgetreden, laat men meerdere processoren simultaan een activiteit verrichten. De uitkomsten worden dan vergeleken door "comparatoren". Bij gelijkheid wordt de gegeven uitkomst als zijnde juist geaccepteerd (onderstelling hierbij is dat een fout nooit tweemaal tegelijk zal voorkomen!?); bij ongelijkheid zal een andere processor uitkomst moeten brengen, waarna de foutieve processor wordt bestraft met uitsluiting van verdere acties.
Wat er gebeurt als steeds meer processoren worden bestraft, wordt niet vermeld.

Als maatregelen in de software ten behoeve van fout-isolatie kan men denken aan het ongeldig verklaren van een stuk data in het cache-geheugen van een processor, nadat een andere processor (na een modificatieverzoek) eigenaar van een kopie van die data is geworden en het heeft gemodificeerd.

Fout-isolatie bij systemen met loosely-coupled processoren

De meeste loosely-coupled fault-tolerant computersystemen leggen de nadruk op software en vertrouwen op de checkpointing-strategie van Tandem.
Bij deze strategie wordt een applicatie uitgevoerd door 2 CPU's; echter op de ene als primair proces en op de andere als passief back-up-proces. Het primaire proces wordt daadwerkelijk uitgevoerd en geeft van tijd tot tijd checkpoint-informatie door aan de andere processor, die daarmee het back-up-proces bijwerkt.
Indien nu de primaire processor faalt, dan neemt de back-up processor de uitvoering over vanaf het laatste checkpoint.

Praktijkgeval

AT&T had als doelstelling geformuleerd dat de fault-tolerant switchingsystemen max. 1 minuut per jaar down mocht gaan en de computer-controller max. 2 minuten.
Het Electronic Switching System voldeed aan de eerste doelstelling; de 3B20D-computer echter nog niet, zij was gemiddeld 6 minuten per jaar down. Wel wordt verwacht dat ook die doelstelling gehaald gaat worden (wanneer en hoe is niet vermeld.)

Leveranciers

Leveranciers van computersystemen met loosely-coupled processoren zijn onder andere:
- Tandem Computers Inc.
  (oudste en meest succesvolle fabrikant van fault-tolerant computersystemen, zoals de Nonstop TXP computers)
- Auragen Systems Corp.
- Tolerant Systems Inc.
- Computer Consoles Inc.

Leveranciers van computersystemen met tightly-coupled processoren zijn onder andere:
- Synapse Computer Corp.
- Stratus Computer Inc.
- Sequoia Systems Inc.

Lezing van het gehele artikel wordt sterk aanbevolen.

COMPACT is een uitgave van de AC-groep van
KMG Klynveld Kraayenhof & Co.