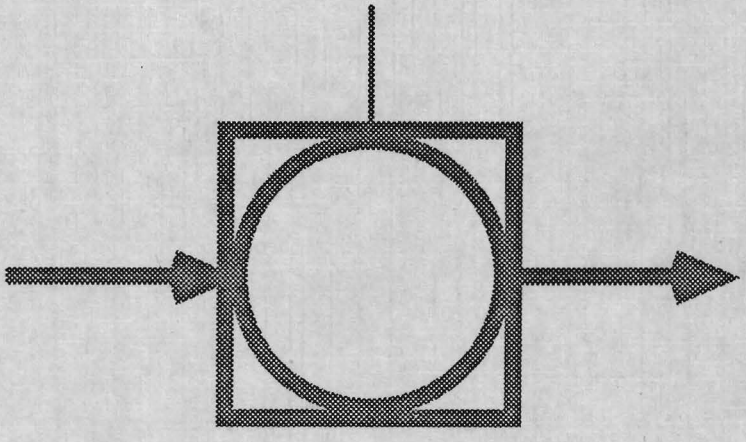
Modelleren en specificeren van informatiesystemen

MODELLEREN
EN
SPECIFICEREN
VAN
INFORMATIESYSTEMEN



JAN DIETZ

Modelleren en Specificeren van Informatiesystemen

MODELLEREN

EN

SPECIFICEREN

VAN

INFORMATIESYSTEMEN

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus, Prof. dr. F.N. Hooge, voor een commissie aangewezen door het college van decanen in het openbaar te verdedigen op dinsdag 27 oktober 1987 te 16.00 uur

door

JEAN LEONARDUS GERARDUS DIETZ

geboren te Brunssum

Dit proefschrift is goedgekeurd door de promotoren

Prof. dr. K.M. van Hee

en

Prof. dr. T.M.A. Bemelmans

aan Wies

INHOUD

Deel I Probleemstelling

Digitale electronische rekenmachines, meestal computers genoemd, worden thans op grote schaal geproduceerd en gekocht. Het merendeel van de kopers zijn organisaties: ondernemingen, non-profit organisaties en gemeentelijke, provinciale en nationale overheden.
De meeste computers, die door organisaties worden gekocht, zijn bedoeld om de operationele en bestuurlijke processen in die organisaties beter, dat wil zeggen effectiever en efficiënter, te laten verlopen.
Hoewel computers onmiskenbaar aanleg hebben tot het spelen van die rol, is het geen sine cure hen dat ook met succes te laten doen. Om de beeldspraak nog even voort te zetten: dat vereist nauwgezet en langdurig instuderen en repeteren.

De aanschafkosten van een computer zijn in veel gevallen slechts een fractie van de kosten, die gemoeid zijn met het laten vervullen van de gewenste rol.
Om die reden zijn niet meer computers, maar informatiesystemen de belangrijkste aandachtsobjecten bij de automatisering in organisaties. De computer vervult, vergeleken met het belang van het (omvattende) informatiesysteem, daarin nog maar een figurantenrol.
De opkomst van de computerindustrie heeft dan ook geleid tot het ontstaan van een even gigantische 'hulpverlenings'-industrie, meestal software-industrie genoemd. Daartoe rekenen we niet alleen software-bureaus, maar ook systeemhuizen, opleidingsinstituten, adviesbureaus etc.
De 'hulpverlening' betreft het ontwerpen, bouwen en gebruiken van informatiesystemen, die effectief en efficiënt zijn, en die in een symbiose met de rest van de organisatie hun rollen vervullen.

Informatiesystemen zijn ook het hoofdonderwerp van dit boek. In het bijzonder is de aandacht gericht op het ontwerpen van die systemen, en daarbinnen weer op geformaliseerde technieken, die de uitvoering van ontwerptaken ondersteunen.
In dit eerste deel zetten we de probleemstelling uiteen.

Allereerst wordt (in hoofdstuk 1) een schets gegeven van het ontwikkelingsproces van geautomatiseerde informatiesystemen en worden klassen van fouten, die daarbij een rol spelen, genoemd. Als een belangrijke oorzaak voor het ontstaan van fouten wordt het informele karakter van de "requirements" en de specificaties aangewezen. Door middel van conceptuele modellen kan aan de "requirements" en de specificaties een formelere grondslag worden gegeven.

In hoofdstuk 2 wordt het onderwerp conceptueel model uitgediept. Er blijken diverse betekenissen aan zulke modellen te worden gegeven. Door de kennisweten-

schappen en door de modelmethodologie worden bijdragen geleverd aan de ontwikkeling van een kader waarbinnen die verschillende betekenissen met elkaar in verband kunnen worden gebracht.

Hoofdstuk 3 spitst de probleemstelling nader toe: het gaat om het formeel beschrijven van het gedrag van een speciaal soort systemen, discrete dynamische systemen genoemd. Daarmee is een brug geslagen naar deel II waarin een metamodel voor het formeel beschrijven van deze soort systemen wordt behandeld.

De samenhang tussen de onderscheiden delen van het boek wordt steeds in die delen zelf nader verklaard.

1. Inleiding

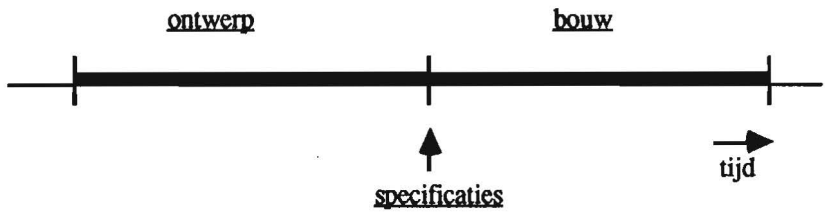
1.1 Het ontwikkelen van informatiesystemen

Het vaak fraaie gezicht, in letterlijke en in figuurlijke zin, van een geautomatiseerd informatiesysteem wordt niet zelden ontsierd door schoonheidsfouten. Deze fouten zijn in twee soorten te verdelen. De ene soort betreft het ontbreken van gewenste functies en het voorkomen van ongewenste functies. Men zegt in dat geval dat het informatiesysteem niet voldoet aan de functionele behoeften van de gebruikers.
De andere soort van fouten betreft het prestatieniveau waarop de gewenste functies worden uitgeoefend. Een voorbeeld van een fout in deze klasse is dat de antwoordtijd op bepaalde vragen te lang is. Zulke fouten kunnen zich uiten als systematische gebreken, maar ook als toevallige storingen in de werking van het systeem: een eerst goed werkend systeem laat het onverwacht afweten.

Bij het opsporen van foutoorzaken is de klassieke verdeling die in programmatuurfouten en apparatuurfouten.
Hoewel apparatuurfouten wel degelijk de oorzaak kunnen zijn van storingen en zelfs van systematische gebreken, maken zij van het totale foutenpakket waarmee een informatiesysteem is behept, meestal een verwaarloosbare kleine fractie uit.
Om die reden laten we apparatuurfouten verder buiten beschouwing.

Programmatuurfouten worden geïntroduceerd tijdens de ontwikkelingsfase van het informatiesysteem. Met de *ontwikkelingsfase* worden alle aktiviteiten bedoeld, die te maken hebben met het tot stand brengen van de programmatuur, en met de organisatorische inbedding van het informatiesysteem. (Aan het tot stand brengen c.q. het verwerven van de benodigde apparatuur wordt in dit boek geen aandacht geschonken).
Na de ontwikkelingsfase volgt de *gebruiksfase* van het informatiesysteem.
Hoewel dat niet zo gangbaar is, zullen we het onderhoud van een systeem opvatten als (her-)ontwikkeling, zodat de levenscyclus van een informatiesysteem bestaat uit elkaar afwisselende ontwikkelings- en gebruiksfasen.

Een ontwikkelingsfase is vereenvoudigd voor te stellen als een *ontwerpfase* gevolgd door een *bouwfase*. Het eindresultaat van de ontwerpfase wordt de *specificaties* van het informatiesysteem genoemd. Dit is tevens het uitgangspunt van de bouwfase (zie figuur 1.1).



Figuur 1.1: Een ontwikkelingsfase van een informatiesysteem

In de specificaties is vastgelegd welke onderscheiden soorten informatie het systeem moet kunnen leveren (men noemt dat meestal de functionele specificaties), en op welke wijze en onder welke voorwaarden dat dient te gebeuren (de niet-functionele eisen of prestatie-eisen geheten). De vaststelling van de informatiebasis waaruit de te leveren informatie kan worden verkregen en van de wijze waarop de actualiteit van die informatiebasis wordt gewaarborgd rekent men veelal ook tot de specificaties van een informatiesysteem.

Fouten, die tijdens de ontwerpfase worden geïntroduceerd, zullen we *ontwerpfouten* noemen; fouten, die tijdens de bouwfase worden geïntroduceerd, heten *bouwfouten*. Op bouwfouten zullen we niet verder ingaan: het bouwen van geautomatiseerde informatiesystemen behoort tot het terrein van de "Software Engineering". Onze aandacht is gevestigd op het ontwerpen van informatiesystemen.
Het is niet zo gemakkelijk de "engineering"-discipline aan te duiden, die dit tot haar verantwoordelijkheid mag rekenen. In Nederland worden daarvoor twee namen het vaakst gehoord: *informatica* en *informatiekunde*. We zullen in dit boek de gunst verlenen aan de informatiekunde. Deze keuze betekent niet dat we een standpunt innemen in een kwestie, die sommige gemoederen nog wel eens wil bezighouden. Evenmin betekent het dat we pretenderen precies te weten wat het vakgebied informatiekunde omvat.
Een belangrijk deelgebied binnen de informatiekunde is de *bestuurlijke informatiekunde*. Het object van aandacht in dit deelgebied zijn de informatiesystemen, die dienen ter ondersteuning van de uitvoering en de besturing van de activiteiten in producerende en dienstverlenende organisaties. De problemen, die zich voordoen bij het ontwerpen van deze (bestuurlijke) informatiesystemen, zijn het uitgangspunt geweest voor het onderzoek, dat aan dit boek ten grondslag ligt.

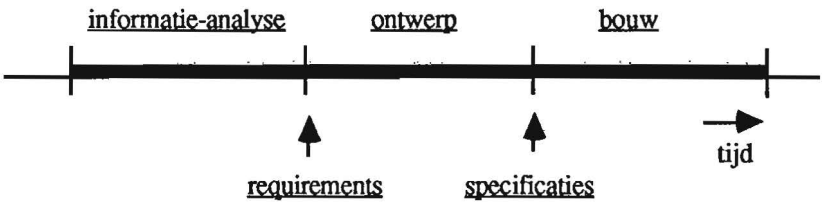
Door de gemaakte onderscheiding in ontwerpfouten en bouwfouten dreigt een derde categorie onbesproken te blijven, die we *specificatiefouten* zullen noemen. Hiermee bedoelen we elke vorm van misverstand tussen ontwerper en bouwer over de inhoud van de specificaties. Op de keper beschouwd is een specificatiefout een

ontwerpfout, immers het is de ontwerper die de specificaties opstelt. Hoe juist deze stellingname echter ook moge klinken, de fouten zelf worden er natuurlijk niet door opgelost of voorkomen.
Het vastleggen van een ontwerp in correcte specificaties is verre van gemakkelijk. Hoe moeilijk het al is voor eenvoudige gevallen om precieze, eenduidige en volledige specificaties op te stellen, wordt overtuigend gedemonstreerd in [Meyer 1985].
Er is een beproefde manier, afkomstig uit andere ontwerpdisciplines, om specificatiefouten zoveel mogelijk te vermijden. Deze is dat de ontwerper en de bouwer overeenkomen een *formele taal* te gebruiken voor het vastleggen van de specificaties. Indien nodig of gewenst, kunnen de formele specificaties worden vergezeld van toelichtingen.
Het belang van het gebruik van een formele taal wordt bijvoorbeeld in de constructieve disciplines al lange tijd onderkend: wie bij wijze van specificaties van een produkt met timmermanspotlood wat krassen op papier zet en daar in de kantlijn enkele terloopse opmerkingen aan toevoegt, krijgt misschien wel een technisch hoogstandje terug uit de fabriek, maar zeker niet wat men bedoelde te hebben.

De praktijk van het ontwikkelen van bestuurlijke informatiesystemen heeft helaas soms veel weg van deze als scherts bedoelde werkwijze.
Gelukkig echter wordt er al jarenlang, en in toenemende mate, door diverse onderzoekers gewezen op de wenselijkheid c.q. de noodzakelijkheid van het gebruik van geformaliseerde specificatietechnieken.

In de ontwerpfase van een informatiesysteem wordt meestal een tussenresultaat geproduceerd, dat veel overeenkomst vertoont met de specificaties. Dit zijn de zogeheten wensen en eisen, ofwel, in het meer gangbare Engels "requirements". De "*requirements*" zijn het resultaat van een analyse van de meestal vaag omschreven problematiek van de opdrachtgever en beogen vooral het informatieprobleem helder uit te drukken. Voor sommige software-systemen, men denke bijvoorbeeld aan compilers of tekstverwerkers, hoeft men nauwelijks onderscheid te maken tussen "requirements" en specificaties. De probleemstelling is in die gevallen namelijk niet alleen duidelijk, maar vaak ook al formeel beschreven.
Voor andere systemen, met name voor bestuurlijke informatiesystemen, ligt dat geheel anders. Men onderkent dan ook meestal bij het ontwikkelen van deze systemen een aparte fase, voorafgaand aan het ontwerp, waarvan de "requirements" het eindresultaat zijn. De Engelstalige benaming van deze fase is "requirements engineering". Het meest gebruikte Nederlands synoniem is *informatie-analyse*.
De meest algemene indeling van een ontwikkelingsfase van een informatiesysteem is dan ook een indeling zoals getekend in figuur 1.2.
Men zou de ontwerpfase in figuur 1.2 kunnen opvatten als een deelfase van de

ontwerpfase in figuur 1.1. Onze voorkeur gaat er echter naar uit de fase informatie-analyse als een toevoeging te zien.



Figuur 1.2: Een ontwikkelingsfase van een bestuurlijk informatiesysteem

Het bijzondere karakter van de informatie-analyse, in vergelijking met het ontwerp van een systeem, komt vooral tot uiting indien men het ontwikkelen van een bestuurlijk informatiesysteem plaatst binnen het bredere kader van "systems engineering". Dit is de naam van een op de systeemtheorie gebaseerde methodiek voor het aanpakken van praktische problemen. De methodiek is ontstaan in de jaren vijftig, en heeft met name vorm gekregen door het werk binnen Bell Telephone Laboratories ([Hunt 1954]). De klassieke definitie van "systems engineering" is die van [Hall 1962], en is het resultaat van de ervaringen in dat onderzoeksinstituut. De stappen, die in deze definitie aan het "systems engineering"-proces worden onderscheiden zijn de volgende (met de originele engelstalige benamingen):

1: "Problem Definition" : dit is in essentie de onderkenning van de behoefte tot het veranderen van een situatie

2: "Choice of Objectives" : de definitie van de behoeften en van het kader waarbinnen daaraan tegemoet moet worden gekomen

3: "Systems Synthesis" : het creëeren van mogelijke alternatieve systemen

4: "Systems Analysis" : de analyse van die hypothetische systemen in het licht van de doelstellingen ("objectives")

5: "Systems Selection" : de keuze van het meest belovende alternatief

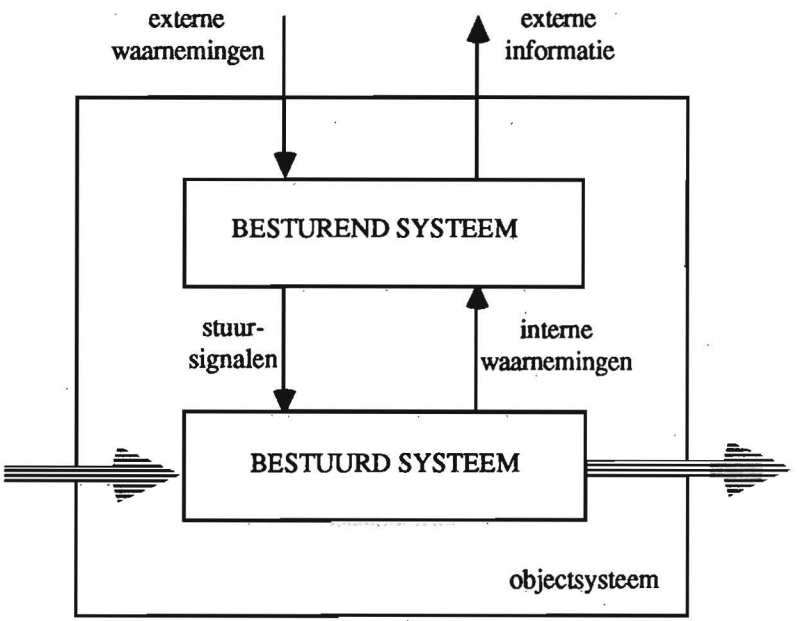
6: "Systems Development" : het (verder) ontwikkelen van het gekozen

alternatief tot en met de prototype-fase

7: "Current Engineering" : de daadwerkelijke realisering van het systeem

(N.B. In het Nederlands is, ongelukkigerwijs, geen goed onderscheid te maken tussen "engineering" en "development". Voor beide woorden is de vertaling: ontwikkeling. "Engineering" is in deze context echter veel meer omvattend dan "development". In elk geval behoren tot "Engineering" het ontwerpen zelf en het plannen en beheersen van het ontwerpproces.)

We zullen het in deze stappen genoemde "system" *objectsysteem* noemen. Vanuit het oogpunt van besturing kan een objectsysteem worden verdeeld in een *bestuurd systeem* en een *besturend systeem* (zie figuur 1.3).

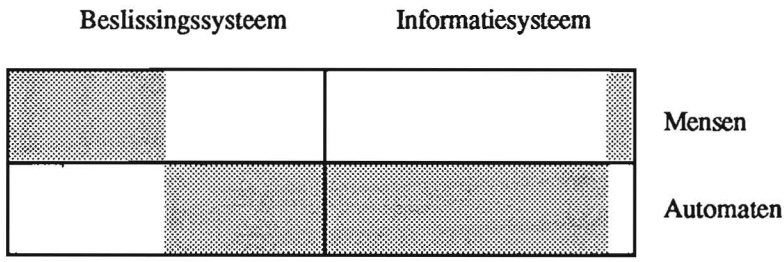


Figuur 1.3: Het besturingsmodel van een systeem

Binnen een besturend systeem kan men vervolgens een *beslissingssysteem* en een *informatiesysteem* onderscheiden. Met het beslissingssysteem worden de, al of niet menselijke, beslissers en de beslissingsregels bedoeld. Het verzamelen, verwerken,

bewaren en beschikbaar stellen van de voor het nemen van beslissingen benodigde informatie zijn taken van het informatiesysteem. Omdat we onder een informatiesysteem altijd een systeem met deze functie verstaan, laten we het adjectief bestuurlijk voortaan weg.
Een informatiesysteem heeft twee gegevensbronnen: (interne) waarnemingen van het primaire proces en waarnemingen van de omgeving van het objectsysteem, externe waarnemingen geheten.

Een andere indeling van een besturend systeem ontstaat wanneer men een onderscheid maakt tussen mensen en automaten als actieve elementen.
De combinatie van deze onderscheiding met de bovengenoemde geeft een vierdeling zoals getoond in figuur 1.4.



Figuur 1.4: Aspecten van een besturend systeem

De aandacht in dit boek is vooral gevestigd op besturende systemen, waarvan het informatiesysteem vrijwel geheel is geautomatiseerd, en waarbij dat voor het beslissingssysteem in belangrijke mate het geval is (vergelijk de grijze gebieden in figuur 1.4). Het is gebruikelijk de geïntegreerde samenstelling van een geautomatiseerd informatiesysteem en het geautomatiseerde deel van een beslissingssysteem weer een informatiesysteem te noemen. (Denk bijvoorbeeld aan een ordersysteem of een voorraadbeheerssysteem). We zullen ons aan dit gebruik conformeren.
Als het beslissingssysteem helemaal is geautomatiseerd noemt men het besturend systeem een *regelsysteem* (Engels: "control system").

Het ontwerpen en bouwen van een informatiesysteem moet men zien als één van de bijdragen (naast andere) aan het tot stand brengen van het in stap 5 ("systems selection") gekozen alternatief van een objectsysteem.

Waar het primair om gaat is dus de realisering van een gewenst objectsysteem.
Een voor het functioneren van een objectsysteem benodigd informatiesysteem dient afgestemd te zijn op de behoeften van het beslissingssysteem, terwijl het

besturend systeem als geheel weer afgestemd dient te zijn op het bestuurd systeem. Men noemt deze zienswijze op informatiesystemen wel het *systelogisch perspectief* ([Welke 1975]). Een uitvoeriger bespreking van de samenhang tussen bestuurd systeem, beslissingssysteem en informatiesysteem vindt men in [Bemelmans 1986].
De activiteit informatie-analyse begint daarom al in stap 1 en loopt door tot (en met) stap 5. De meeste ontwikkelingsmethoden voor informatiesystemen onderkennen evenwel deze positie van de informatie-analyse niet. Een bekende goede uitzondering is de ISAC-methodiek [Lundeberg, Goldkuhl, Nilsson 1979].
De meeste methoden dekken feitelijk slechts de stappen 6 en 7. Anders gezegd, ze gaan ervan uit dat de "requirements" bekend zijn, en alleen maar hoeven te worden geïnventariseerd, een benadering die in [Bemelmans 1987] de oberstrategie wordt genoemd.

Ook voor de "requirements" is het uiteraard wenselijk dat ze op formele wijze worden uitgedrukt ([Roman 1985]). Ten behoeve hiervan worden al enige tijd "requirements modeling languages" ontwikkeld. De bedoeling van deze talen is de formele beschrijving van een objectsysteem mogelijk te maken. Omdat zo'n beschrijving van een objectsysteem vaak een conceptueel model van dat systeem wordt genoemd, spreek men ook wel van "conceptual modeling languages" ([Borgida 1985]).
Met een conceptueel model van een objectsysteem wordt in dit verband vooral de beschrijving van het besturend systeem bedoeld. Het bestuurd systeem is in het algemeen slechts van belang voor zover het de interactie met het besturend systeem betreft (de stromen "interne waarnemingen" en "stuursignalen" in figuur 1.3).

Men kan een conceptueel model van een objectsysteem in principe op twee manieren beschouwen.
De ene manier is dat men het opvat als de beschrijving van het feitelijk functioneren van een reëel systeem, waardoor men dat functioneren beter kan begrijpen. Deze zienswijze is typisch voor *systeemanalyse*, en is dus bijvoorbeeld van toepassing op het conceptuele model dat de basis vormt voor het vaststellen van de "requirements".
Bij de andere gebruikswijze vat men het conceptuele model op als de beschrijving van een te implementeren systeem. Deze zienswijze is typisch voor *systeemsynthese*, en past dus bijvoorbeeld bij het conceptuele model dat de basis vormt van de specificaties van een informatiesysteem.

Tot de belangrijkste bijdragen op het gebied van "conceptual modeling languages" moeten, naast het pionierswerk van Teichroew ([Teichroew, Hershey 1977]) ook onder andere worden gerekend: [Bodart, Pigneur 1983], [Solvberg, Kung 1985], [Dubois e.a. 1986], [Greenspan, Borgida, Mylopoulos 1986] en [Gustafsson, Karlsson, Bubenko 1982]).

1.2 Aspecten van informatie

Betere technieken en hulpmiddelen kunnen een aanzienlijke verhoging van de kwaliteit van de informatie-analyse en het systeemontwerp met zich meebrengen, diepgewortelde vakkennis blijft echter een sine qua non. Anders gezegd, de tegenwoordig voorhanden, geïntegreerde, ondersteuning van de informatie-analist/systeemontwerper in de vorm van zogeheten "work benches", biedt alle mogelijkheden tot het met weinig inspanning en in minimale tijd voortbrengen van goede specificaties, maar ook van slechte. Het gebruik van "work benches" is dus geen garantie voor betere analyses en ontwerpen.

Tot de algemene vakkennis van de informatie-analist dienen in elk geval te behoren: systeemkennis, met name kennis van het modelleren van systemen, en kennis van het thema informatie. Het eerste element komt uitvoerig aan de orde in de volgende hoofdstukken van dit boek. Wat het tweede betreft zullen we de hoofdzaken beknopt hieronder bespreken.

Daartoe maken we een kort uitstapje naar de semiotiek, het vakgebied dat zich bij uitstek bezig houdt met de bestudering van het thema informatie. Enkele belangrijke, klassieke, bijdragen zijn [Peirce 1960] en [Morris 1955], terwijl van de meer recente studies [Nauta 1972] en [Stamper 1973] vermeldenswaard zijn. Zeer toegankelijke en Nederlandstalige werken zijn [Peursen 1968] en [Nielen 1976].

De voor ons doel interessante aspecten, die men kan onderscheiden aan informatie zijn:

- het syntactisch aspect,
- het semantisch aspect en
- het pragmatisch aspect.

Het object, waaraan deze aspecten worden onderscheiden wordt in de semiotiek niet met informatie maar met de algemene term teken aangeduid.
Een teken is gedefinieerd als een fysisch waarneembaar fenomeen, waarvan de belangrijkste functie is dat het verwijst naar iets anders. Dezelfde onderscheiden aspecten treft men, niet verwonderlijk, aan in de linguistiek.

Onder het *syntactisch* aspect wordt de vorm van een teken verstaan. Indien men de gehele zin "De voorraad van artikel x is y" opvat als een teken, is het syntactisch aspect van dat teken de bepaalde grammaticale vorm, waarin de zin is gegoten. Een andere zin is "Er is een hoeveelheid y van artikel x op voorraad".

Met het *semantisch* aspect wordt de concrete of abstracte zaak bedoeld waarnaar een teken verwijst. Voor de eerste zin zou dat kunnen zijn dat er in het magazijn y stuks aanwezig zijn van het artikelsoort x.
De (semantische) betekenis van een zin in een taal berust op pure afspraak. Men zou dus aan de tweede zin dezelfde betekenis kunnen hechten. (In de Nederlandse taal is dat ook zo).

Het *pragmatisch* aspect betreft de invloed van het ontvangen van een teken op de ontvanger. Dit dekt een breed gebied van effecten, waaronder rationele (als er een spoorwegstaking is afgeroepen neem ik de auto), maar bijvoorbeeld ook fysiologische (zoals verhoging van de hartslag vlak voor de ontknoping van een verhaal).

In de informatie-analyse zijn vooral het semantisch aspect en het pragmatisch aspect van belang. In de ontwerpfase speelt het semantisch aspect in combinatie met het syntactisch aspect de hoofdrol.
Een complete beschrijving van een informatiesysteem dient alle drie aspecten te omvatten.

Wat het syntactisch aspect betreft, houdt dit in dat men geschikte vormen kiest voor de tekens, die moeten worden ingevoerd, uitgevoerd en bewaard. De tekens die worden ingevoerd mogen daarbij verschillen van de tekens die worden bewaard, en de tekens die worden uitgevoerd kunnen weer geheel anders zijn. Dit is een onderdeel van de beschrijving van het informatiesysteem op het zogeheten *datalogisch niveau* ([Langefors 1980], [Sundgren 1978]), waarbij het er vooral om gaat vast te leggen hoe tekens worden bewaard, ontvangen en verzonden, en hoe het bewerken van gegevens gebeurt.

> Men kan hierbij bijvoorbeeld denken aan de inrichting van een kantoor met opbergkasten voor formulieren en aan de vaststelling van werkprocedures voor het administratief personeel.
> Een geheel andere beschrijving is die waarin computerbestanden en programma- modules voorkomen.

Aan het datalogisch niveau wordt in dit boek geen aandacht geschonken. Omdat informatie onvermijdelijk een syntactisch aspect heeft, komt dit erop neer dat we een bepaalde, geschikte vormafspraak kiezen en aannemen dat deze naar een andere kan worden getransformeerd. De gekozen syntactische structuren zijn de in hoofdstuk 6 gepresenteerde notaties van formules in een eerste orde logische taal.

Het richten van de aandacht op het semantisch aspect noemt men wel het bekijken van een informatiesysteem op het *infologisch niveau* ([Langefors 1980], [Sundgren 1978]). De doelstelling daarbij is het vastleggen van de (semantische)

betekenis van alle tekens of gegevens. Essentieel daarvoor is dat men de werking van het objectsysteem goed begrijpt.
Een operationele vertaling van "goed begrijpen" is het beschikken over een helder en precies conceptueel model van het objectsysteem.

Op de constructie van een conceptueel model en op het gebruik ervan is de aandacht in dit boek gericht. De uiteindelijke doelstelling daarbij is een verhoging van het kwaliteitsniveau waarop het ontwikkelen van informatiesystemen plaatsvindt. De reikwijdte is echter groter. Vandaar dat we, de gebruiksmogelijkheden van het in deel II ontwikkelde raamwerk beschouwend, als toepassingsgebied liever de algemene termen *systeemanalyse* en *systeemsynthese* noemen.

2. Conceptuele Modellen

In hoofdstuk 1 is, zonder verdere uitleg, de term conceptueel model geïntroduceerd. De bedoeling van dit hoofdstuk is de betekenis en het gebruik van deze term nader te onderzoeken.
Dat gebeurt in stappen. Allereerst wordt de betekenis van het adjectief conceptueel bestudeerd. Daarna bespreken we een aantal verschillende betekenissen van het modelbegrip. Tevens zullen we dan in een samenvattend kader de betekenis van "conceptueel model" proberen te verklaren.

2.1 De semantiek van informatie

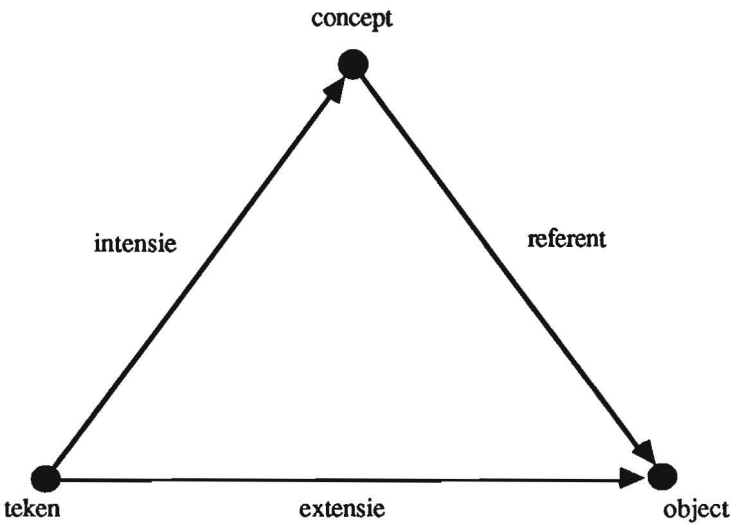
Van de aspecten, die in het vorige hoofdstuk aan informatie werden onderscheiden, zullen we het semantisch aspect nu wat uitgebreider bestuderen. Daartoe maken we een kort uitstapje naar het vakgebied van de kenniswetenschappen.
In de kenniswetenschappen wordt de wijze waarop mensen de wereld observeren bestudeerd, alsmede de wijze waarop observaties worden verwerkt en ingepast in de aanwezige kennisstructuren. Een algemeen aanvaarde hypothese is dat kennis is vastgelegd in concepten en conceptuele relaties die tezamen een netwerk vormen ([Sowa 1984]). Zo'n netwerk noemt men wel een *semantisch netwerk.*
Een conceptuele relatie is een, meestal binaire, associatie tussen concepten.

Een concept ontleent zijn *betekenis* aan de verbanden die het heeft, via conceptuele relaties, met andere concepten. Anders gezegd, de plaats van een concept in het semantisch netwerk bepaalt de betekenis van het concept. Voor veel concepten geldt dat ze ook een relatie met een (al of niet waarneembare) werkelijkheid of wereld hebben. Zulke concepten worden *concrete concepten* genoemd. Voorbeelden van concrete concepten zijn: een persoon, een auto, een order. Concepten, waarvoor zo'n relatie niet bestaat, heten *abstracte concepten.* Voorbeelden van abstracte concepten zijn: een gewicht, een temperatuur, een natuurlijk getal.
In hoofdstuk 8 komen we op dit onderscheid terug. Voor de rest van dit hoofdstuk wordt met een concept steeds een concreet concept bedoeld.

Een concept is zelf iets immaterieels. Om het te kunnen 'bewaren' of 'mededelen' moet aan een concept een teken worden toegewezen. Een *teken* is een afgesproken en herkenbare vorm van een fysisch waarneembaar object, dat als de drager van het teken fungeert. De geschreven of gedrukte voorstellingen van de woorden van de Nederlandse taal zijn voorbeelden van tekens. Bekende fysische fenomenen voor de realisatie van tekens zijn chemische processen in hersenweefsel, potloodstrepen op

papier en electrische en magnetische grootheden in computers.

Het verband tussen een teken, het bijbehorend concept, en het object in de wereld waarnaar het teken verwijst wordt uitgebeeld door de betekenisdriehoek ([Ogden, Richards 1923]):



Figuur 2.1: De betekenisdriehoek

Het object, of de verzameling objecten, waarvoor een teken staat heet de *extensie* van het teken. Het concept zelf, in het bijzonder de betekenis van het concept, wordt de *intensie* van het teken genoemd. De extensie van een teken heet de *referent* van het bijbehorend concept.

> Laten we ter illustratie het concept paard nemen. Het woord "paard" is het teken dat het concept paard representeert. Omgekeerd is dus het concept paard de intensie van het woord "paard". Het woord staat voor iets in de werkelijkheid, in dit geval alle paarden. De verzameling van alle paarden is de extensie van het woord "paard". Deze verzameling is ook de referent van het concept paard. Bij het concept zelf moet men vooral denken aan algemene eigenschappen van paarden, waardoor deze zich onderscheiden van bijvoorbeeld honden en geraniums.

Het is gebruikelijk concepten, die een verzameling als referent hebben, *generieke concepten* te noemen. Het concept paard is dus een generiek concept. Daarnaast onderscheidt men *individuele concepten*. Deze hebben steeds één bepaald individu

als referent.

> Indien iemand bijvoorbeeld het paard met de naam "Bertje" kent, behoort tot de kennis van die persoon een individueel concept, dat het bestaan van het (unieke) paard "Bertje" voorstelt.

Kennis met betrekking tot generieke concepten heet *generieke kennis*. Deze betreft de onderscheiden generieke concepten zelf, maar ook de samenhang tussen de concepten onderling. Generieke kennis kan worden voorgesteld als een verzameling algemene feiten. Voorbeelden van *algemene feiten* zijn:

- er zijn mensen (ofwel: er is een generiek concept mens);
- ieder mens heeft een vader en een moeder;
- iemand die de burgerlijke staat "gescheiden" heeft, heeft daarvoor de status "getrouwd" gehad;
- een geregistreerde auto heeft een kenteken.

Kennis met betrekking tot individuele concepten heet *individuele kennis*. Deze betreft de onderscheiden concepten zelf en de individuele conceptuele relaties met andere individuele concepten. Individuele kennis kan worden voorgesteld als een verzameling elementaire feiten. Voorbeelden van *elementaire feiten* zijn:

- er is een persoon Nick;
- er is een persoon Sanne;
- Nick is een neef van Sanne;
- er is een auto met kenteken HL-98-YD.

2.2 Systemen en modellen

Met betrekking tot de ontwikkeling en het gebruik van geautomatiseerde informatiesystemen zijn vooral de volgende betekenissen van het woord model relevant:

1. *simulatie*: Een informatiesysteem wordt in deze betekenis een model genoemd van het bijbehorend bestuurd systeem. Hiermee bedoelt men uit te drukken dat het informatiesysteem op elk moment een afspiegeling is van de stand van zaken in het bestuurd systeem, en dat het de veranderingen daarin volgt.
   Men treft deze modelbetekenis vooral aan in de database-wereld (zie bijvoorbeeld [Abrial 1974]).

2. *abstractie*: Onder deze betekenis van het modelbegrip verstaat men de algemene wetmatigheden en beslissingsregels die gelden voor (de besturing van) een objectsysteem. Deze zijn op een of andere wijze uitgebeeld in het informatiesysteem. In de database-wereld noemt men deze modelbetekenis het (conceptuele) schema van het informatiesysteem ([ISO 1982]). De abstracte voorstelling die men heeft van een systeem, kan heel anders zijn dan de feitelijke werking ervan. Men vergelijke bijvoorbeeld zijn (conceptueel) model van een koffieautomaat met de wirwar van slangen, vaten en kranen in de kast zelf ([Checkland 1981]).

3. *prototype* : In de taal van alle dag is deze betekenis het meest verwant met ideaal of standaard (bijvoorbeeld modelkeuken of fotomodel). In de systeemontwikkeling wordt met prototype een systeem bedoeld dat de belangrijkste kenmerken van het te ontwikkelen systeem toont ([Floyd 1984]).

4. *realisatie* : De elementaire en algemene feiten, die zijn bevat in een informatiesysteem kunnen worden gerepresenteerd door logische formules. Een model van een logische formule is elke interpretatie van de formule die een waar feit uitdrukt.
Bijvoorbeeld: de verzameling der natuurlijke getallen is een model van de formule $\forall x: \exists y: p(x, y)$, indien men voor p de binaire operator "kleiner dan" kiest ([Bertels, Nauta 1974]).

We zullen nu proberen deze nogal verschillende modelbetekenissen op één noemer te brengen. We beginnen met het geven van een algemeen bruikbare, pragmatische, definitie van *model*, ontleend aan [Apostel 1960]. Vrij vertaald luidt deze:

> "Iemand, die een systeem B, dat onafhankelijk is van een systeem A, benut met de bedoeling kennis te verwerven over A, gebruikt B als een model van A".

Deze modeldefinitie is zeer goed toepasbaar op de eerste drie modelbetekenissen, zoals men gemakkelijk zelf kan verifiëren. De vierde betekenis is iets moeilijker daarmee in overeenstemming te brengen. Het beste is een model in de realisatie-betekenis te zien als een voorbeeld van iets, dat voldoet aan bepaalde eisen of voorschriften. Vervolgens dient men te bedenken dat een logische formule van zichzelf geen betekenis heeft, het is een puur syntactisch begrip. Wanneer men nu aan de symbolen van de formule (zelf gekozen) betekenissen hecht, kan men vervolgens uitzoeken of er situaties te vinden zijn die aan de gekozen betekenis

voldoen. Bijvoorbeeld: een model voor p(x) is dat p betekent "wielrenner", en x: "Joop Zoetemelk".

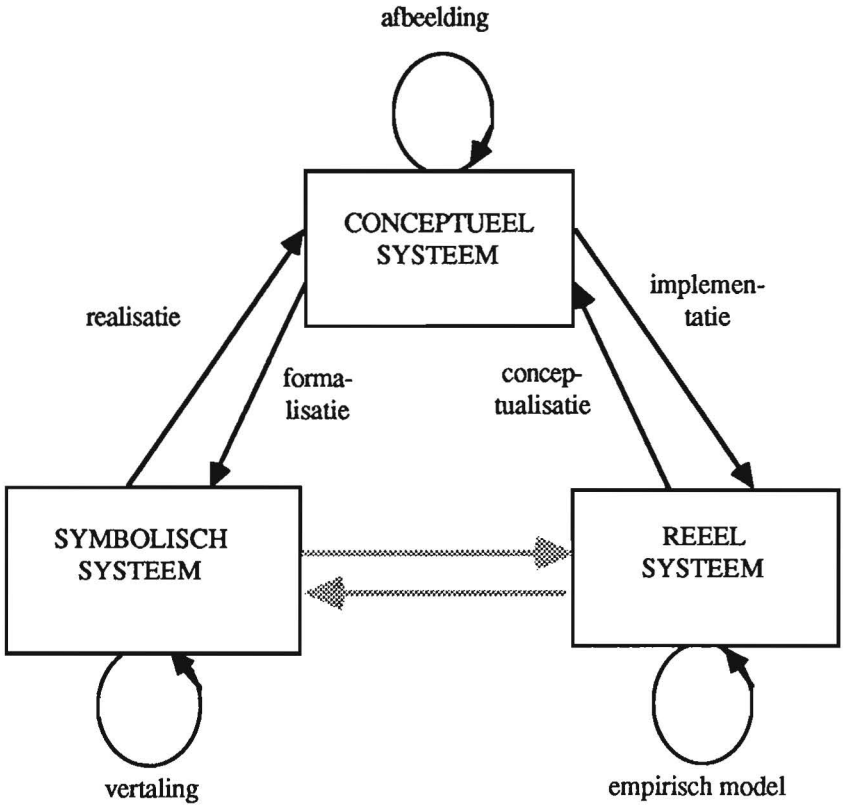
Een model is dus een systeem, beter gezegd, het is een rol die een systeem ten opzichte van een ander systeem kan vervullen.
We veronderstellen een intuïtief systeembegrip bekend, (zie bijvoorbeeld [Checkland 1981] of [Emery 1969]). De in dit boek centraal staande klasse van discrete dynamische systemen wordt in de komende hoofdstukken nauwkeurig beschreven.
Voor het vervolg van dit hoofdstuk onderscheiden we drie soorten systemen: reële systemen, conceptuele systemen en symbolische systemen.

Een *reëel* systeem is een systeem, waarvan de elementen reële dingen zijn. Voorbeelden van reële systemen zijn: een kristal, een produktie-organisatie, een draaimolen.
Onder reëel verstaan we overigens niet alleen "werkelijk bestaand" maar ook "voorstelbaar bestaand" zoals bijvoorbeeld feëen en eenhoorns.

Een *conceptueel systeem* is een systeem, waarvan de elementen begripsmatige dingen zijn.
Voorbeelden van conceptuele systemen zijn: de verzamelingenleer, een meetkundige figuur, een dds (zie hoofdstuk 4 en hoofdstuk 5).

Een *symbolisch systeem* is een systeem, waarvan de elementen ongeïnterpreteerde syntactische symbolen zijn. In het bijzonder bedoelen we met een symbolisch systeem een deelverzameling van de formules van een formele taal (zie hoofdstuk 6).

In principe kan een systeem van een van deze soorten model zijn van een systeem van elk ander soort, maar ook van systemen van zijn eigen soort. Daardoor kunnen er negen verschillende modelsoorten worden onderscheiden. We zullen deze modelsoorten allemaal kort bespreken. Figuur 2.2 toont het verband tussen de modelsoorten en de systeemsoorten. Deze figuur is een variant van een figuur in [Bertels, Nauta 1974]. Ter onderstreping van de analogie met figuur 2.1 zullen we figuur 2.2 de modellendriehoek noemen.
Het algemeen spraakgebruik volgend, zullen we een systeem van soort x dat gebruikt wordt als een model van een ander systeem een model van soort x noemen.



Figuur 2.2: De modellendriehoek

Een reëel model van een reëel systeem wordt een *empirisch model* genoemd. Een voorbeeld van een empirisch model is een schaalmodel van een vliegtuig. Het dynamisch gedrag van een schaalmodel heeft een verwantschap met het dynamisch gedrag van het vliegtuig waarvan het een model is. Deze verwantschap heet simulatie. Simulatie van dynamisch gedrag is een belangrijk toepassingsgebied van empirische modellen.

Een reëel model van een conceptueel systeem heet een *implementatie* van het conceptuele systeem. Bijvoorbeeld, de piramide van Gizeh is een implementatie van de stereometrische figuur, die piramide heet.
Goede synoniemen voor implementatie zijn realisatie en concretisatie. Omdat deze woorden echter in dit boek al voor andere betekenissen worden gebruikt, vermijden we ze liever.

Een conceptueel model van een reëel systeem heet een *conceptualisatie* van het reële systeem.
Een conceptueel model kan een mathematisch model zijn, maar dat hoeft niet.
Voorbeelden van conceptualisaties zijn de terugkoppelmachine als model van allerlei biologische processen, en het dds als model van discrete dynamische systemen (zie hoofdstuk 4).

Een conceptueel model van een conceptueel systeem noemen we een *afbeelding* van dat conceptuele systeem. Afbeeldingen zijn vaak het resultaat van mathematisering van conceptuele systemen.
Voorbeelden van afbeeldingen zijn de algebraïsche vergelijking van een cirkel als model van de meetkundige figuur met die naam, en het tupel <**S, M, A, R, T, I**,> als mathematisch model van een dds (zie hoofdstuk 5).

Een symbolisch model van een conceptueel systeem heet een *formalisatie* van het conceptuele systeem. Een voorbeeld van een formalisatie is het axiomastelsel van Euclides als formeel model van de Euclidische ruimte.

Een conceptueel model van een symbolisch systeem heet een *realisatie* van het symbolische systeem. Het is in de logica gebruikelijk een realisatie kortweg een model te noemen.
Een, al eerder gegeven, voorbeeld van een realisatie is de verzameling der natuurlijke getallen als model van de formule $\forall x: \exists y: p(x, y)$, indien aan p de relatie "kleiner dan" wordt toegekend. Er staat dan dat voor elke x een y te vinden is zodanig dat $x < y$. Als x en y natuurlijke getallen zijn, is dat waar.

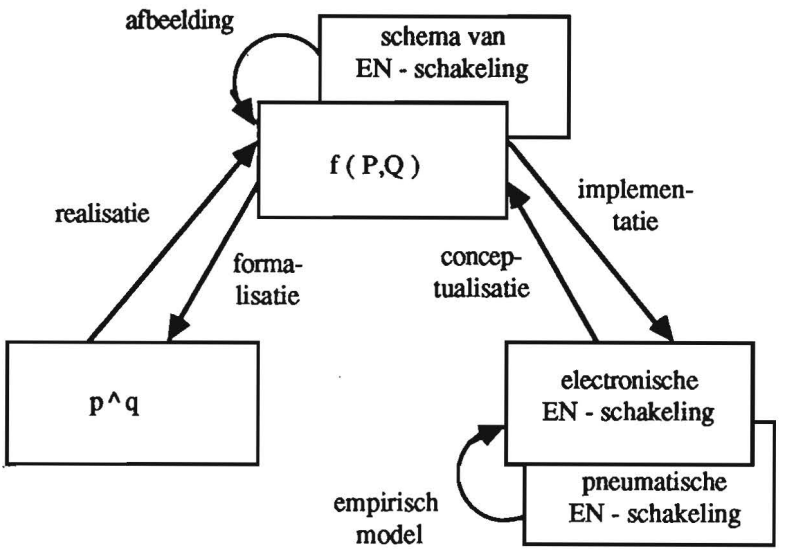
Een symbolisch model van een symbolisch systeem heet een *vertaling* van dat symbolisch systeem. Een voorbeeld van een vertaling is de omzetting van tekens in de EBCDIC-code naar tekens in de ASCII-code.

Er resteren nog twee modelsoorten: een reëel model van een symbolisch systeem en een symbolisch model van een reëel systeem. Beide bestaan echter niet rechtstreeks, maar slechts via de tussenstap van een conceptueel systeem. Zo kan men spreken van de implementatie van een realisatie en van de formalisatie van een conceptualisatie.

Aan de hand van het voorbeeld van een EN-schakeling, ontleend aan [Bertels, Nauta 1974], kunnen de meeste van de onderscheiden modelsoorten eenvoudig worden geïllustreerd.
Een gebruikelijke presentatie van het schema van een EN-schakeling is de volgende:

Figuur 2.3: Schema van een EN-schakeling

Hierin stellen P en Q schakelaars voor.
Figuur 2.4 toont het verband tussen de diverse modelsoorten, die met betrekking tot de EN-schakeling kunnen worden onderscheiden.



Figuur 2.4: Toepassing van de modellendriehoek op een EN-schakeling

Uitgaande van het schema van de EN-schakeling (dat een conceptueel systeem voorstelt) kan bij deze figuur de volgende toelichting worden gegeven.

Een afbeelding van het schema is de algebraïsche functie f(P,Q). De definitie van f is: P * Q, waarbij geldt dat P en Q slechts de waarden 0 en 1 kunnen aannemen (0 betekent "open" en 1 betekent "gesloten").
De logische formule p Λ q is een formalisatie van de functie f (de logische waarden "waar" en "onwaar" komen overeen met "1", respectievelijk "0").
De functie f is een realisatie van p Λ q. (Er zijn vele andere realisaties van p Λ q mogelijk).
Het schema is een conceptualisatie van elke reële EN-schakeling en elke reële EN-

schakeling kan worden opgevat als een implementatie van het schema. Tenslotte kan men een electronische EN-schakeling opvatten als een empirisch model van een pneumatische EN-schakeling.

2.3 Informatiesystemen

Komen we dan nu toe aan een poging de verschillende betekenissen van het modelbegrip en van de toevoeging conceptueel met elkaar in verband te brengen.

Om te beginnen is daarvoor het onderscheid tussen het datalogisch en het infologisch niveau relevant (zie hoofdstuk 1). Op datalogisch niveau is een informatiesysteem een symbolisch systeem. Dit systeem heet het *datalogisch model* van een informatiesysteem. Op het infologisch niveau is het een conceptueel systeem, het *infologisch model* van een informatiesysteem geheten.
N.B. De betekenis van het woord model in "datalogisch model" en "infologisch model" is een andere dan de in paragraaf 2.2 gehanteerde. Het is nu eerder een aspectsysteem dan een systeem dat de rol van model vervult conform de definitie van Apostel.

Het is van belang in het infologisch model en in het datalogisch model de in paragraaf 2.1 behandelde twee kennisniveau's te onderscheiden. Het infologisch model bevat dus algemene feiten én elementaire feiten, en het datalogisch model bevat notaties van beide soorten.
De in paragraaf 2.1 onderscheiden betekenissen van het modelbegrip kunnen nu als volgt worden verklaard.

"Een informatiesysteem is een *simulatie* van het bestuurd systeem" betekent dat de elementaire feiten in het infologisch model conceptualisaties zijn van individuele dingen en gebeurtenissen in het bestuurd systeem.

"Een informatiesysteem is een *abstractie* van het objectsysteem" betekent dat het infologisch model conceptualisaties bevat van algemene feiten in het bestuurd systeem, alsmede (eventueel) beslissingsregels, die gelden voor het beslissingssysteem.

"Informatiesysteem B is een *prototype* van informatiesysteem A" betekent onder andere dat het infologisch model van B een afbeelding is van het infologisch model van A.
Meestal stelt men aan een prototype B van A de eis, dat alle functionele specificaties van B, dus ook die betreffende de invoer en de uitvoer van gegevens,

een bepaalde afbeelding zijn van die van A.

Wat de realisatie-betekenis betreft, kan men stellen dat het infologisch model een *realisatie* is van het datalogisch model (die dus een formalisatie is van het infologisch model).

We sluiten dit hoofdstuk over conceptuele modellen af met een korte bespreking van de term "metamodel".
Een conceptueel model is, zoals we hebben gezien, een conceptualisatie van één reëel systeem. Wanneer men een klasse van, op een of andere wijze gelijksoortige, reële systemen beschouwt, zullen de conceptuele modellen daarvan ook gemeenschappelijke trekken vertonen. Van deze eigenschap kan men gebruik maken door één algemeen raamwerk te ontwerpen voor de constructie van alle conceptuele modellen van dezelfde klasse. Elk afzonderlijk conceptueel model is een bepaalde invulling van het raamwerk.
In de literatuur wordt het woord model meestal gebruikt ter aanduiding van zowel een individueel model als een raamwerk (zie bijvoorbeeld [Tsichritzis, Lochovsky 1982]). Dit werkt ons inziens verwarrend. Om die reden zullen we een model in de betekenis van een raamwerk steeds een *metamodel* noemen.
Een implementatie van een metamodel is dus een klasse van reële systemen.

3. Discrete dynamische systemen

De systeembenadering (zie bijvoorbeeld [Checkland 1981] of [Emery 1969]) is een beschouwingswijze, volgens welke men een deel van een groter geheel afbakent en dat deel als een zelfstandig geheel opvat. De eis daarbij is, dat dit deel uit een verzameling samenhangende elementen bestaat. Zo'n afgebakend deel heet een systeem, de rest van het oorspronkelijke geheel wordt de *omgeving* van het systeem genoemd. Op grond van overeenkomstige eigenschappen kunnen systemen in klassen worden ingedeeld. Zo onderscheidt men bijvoorbeeld open systemen, gesloten systemen, statische systemen en dynamische systemen.
De soort systemen, waarop in dit boek de aandacht is gericht, zullen we discrete dynamische systemen noemen.

Een *discreet dynamisch systeem* is een open systeem, dat wil zeggen er is sprake van een wederzijdse beïnvloeding tussen het systeem en zijn omgeving. De wederzijdse beïnvloeding verandert in de tijd. Om die reden heet het systeem dynamisch. (De toevoeging "discreet" wordt verderop verklaard). Met omgeving worden daarbij in het bijzonder die delen van de omgeving bedoeld, waarmee die wederzijdse beïnvloeding feitelijk bestaat. Zo zullen we bijvoorbeeld de klanten en leveranciers van een winkel wel tot de omgeving van die winkel rekenen, maar de gasten van het belendend café niet. De systeembenadering houdt altijd een keuze van bepaalde aspecten in met uitsluiting van alle andere.
Produktie-organisaties, informatiesystemen, telefooncentrales, fruitmachines en koffie-automaten zijn voorbeelden van discrete dynamische systemen.
We onderscheiden twee soorten van wederzijdse beïnvloeding: interactie en communicatie.

Van *interactie* tussen twee systemen is sprake indien zij elkaar aanzetten tot activiteit door middel van over en weer verzonden onderscheidbare stimuli. Deze stimuli zijn voor het ontvangende systeem invoerstimuli en voor het verzendende systeem uitvoerstimuli. De stroom van invoerstimuli, die een systeem ontvangt heet de *invoer* van het systeem. De stroom van uitvoerstimuli, die een systeem afgeeft heet de *uitvoer* van het systeem.
De ontvangst van een enkele stimulus heet een *invoergebeurtenis*, de afgifte van een enkele stimulus heet een *uitvoergebeurtenis*.

Voorbeelden van invoerstimuli zijn:

- het plaatsen van een klantorder;
- de ontvangst van een giro-opdracht;
- de opname van een patiënt.

Voorbeelden van uitvoerstimuli zijn:

- de levering van een klantorder;
- de bevestiging van uitvoering van een giro-opdracht;
- het ontslag van een patiënt.

Naast de invoer en de uitvoer onderscheiden we een derde systeemgrootheid, de *toestand* geheten.
De functie van deze grootheid is het bewaren, meestal in een samengevatte vorm, van de systeemhistorie (dat is de invoer en de uitvoer in het verleden).
De toestand op enig moment wordt voorgesteld door een verzameling. De elementen van deze verzameling noemen we toestandselementen.
Een *toestandselement* representeert een elementair feit (of propositie, zie [ISO 1982]) betreffende de stand van zaken in de voor het systeem relevante werkelijkheid. Een toestand is een deelverzameling van een universum van elementaire feiten.

Het veranderen van de toestand bestaat uit het toevoegen of verwijderen van toestandselementen. We zullen het toevoegen of het verwijderen van een element een *mutatie* noemen, en de toevoeging of verwijdering van een element op een bepaald tijdstip een *mutatie-gebeurtenis.*

Van *communicatie* tussen twee systemen is sprake indien zij een deel van hun toestanden gemeenschappelijk hebben (in wiskundige terminologie: indien de doorsnede van hun toestanden niet leeg is).
Een voorbeeld hiervan zijn de posttarieven. Deze worden gedeeld door de (toestand van de) PTT en (de toestand van) iedereen, die wel eens wat per post verzendt. De communicatie is in dit geval eenzijdig, dat wil zeggen, slechts één van beide systemen kan (het gemeenschappelijke deel van) de toestand veranderen.
Een voorbeeld van communicatie, waarbij beide systemen veranderingen in het gemeenschappelijke deel van de toestand aanbrengen, is een buffer tussen twee systemen: het ene systeem voegt daar dingen aan toe, en het andere haalt er dingen uit.

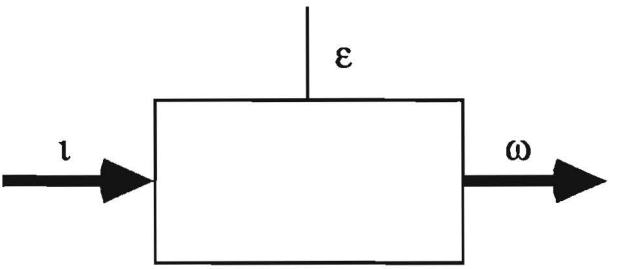
Aan de invoergebeurtenissen, de uitvoergebeurtenissen en de mutatiegebeurtenissen leggen we de beperking op, dat hun aantal in elk eindig tijdinterval eindig is. Een dynamisch systeem waarbij aan die eis is voldaan, heet een *discreet dynamisch systeem.*
Continue systemen, dat zijn systemen waarbij gebeurtenissen zich kunnen ophopen in de tijd (men spreekt dan van verdichting van gebeurtenissen), vallen dus buiten het bestek van dit boek. Gediscretiseerde modellen van deze systemen vallen er uiteraard weer binnen.
Omdat we het verder alleen maar over discrete dynamische systemen hebben, zullen we de adjectieven gemakshalve vaak weglaten.

Het begrip toestand, zoals hier gedefinieerd, is ruimer dan in de systeemtheorie gewoonlijk wordt gehanteerd, maar sluit deze wel in. Het verschil heeft te maken met het onderkennen van communicatie als een afzonderlijke vorm van beïnvloeding. Een gebruikelijke definitie is namelijk dat de toestand van een systeem de vastlegging is van de waarden van een minimale verzameling variabelen (dat wil zeggen: minimale kennis over de gedragshistorie), nodig om het toekomstig gedrag voor elke invoer te kunnen bepalen (zie bijvoorbeeld [Elgerd 1967]).

De centrale vraagstelling in dit boek is die naar de beschrijving van het gedrag van systemen. Daaronder verstaat men meestal het verband tussen de invoer en de uitvoer van een systeem. Omdat we voor de gedefinieerde klasse van discrete dynamische systemen ook communicatie hebben onderkend, moet deze definitie van het gedrag van een systeem worden aangepast. Onder gedrag zullen we daarom verstaan het verband tussen de invoer en de stroom van externe mutaties enerzijds en de uitvoer anderzijds.
Externe mutaties zijn toestandsveranderingen, die worden aangebracht door de omgeving.
We nemen aan dat met betrekking tot dat verband het causaliteitsbeginsel geldt. Dit houdt in, dat een uitvoergebeurtenis, direct of indirect, het gevolg is van één of meer invoergebeurtenissen en dat de afgegeven stimulus mede afhangt van eerdere mutatiegebeurtenissen.

We zullen het voorgaande thans in een wat formelere vorm gieten, om een gemakkelijke aansluiting te verkrijgen met de behandeling in deel II.

In figuur 3.1 is het "black-box"-model van een discreet dynamisch systeem weergegeven.



Figuur 3.1: "Black-box"- model van een discreet dynamisch systeem

De griekse letters ι, ρ en ε stellen respectievelijk de invoer, de uitvoer en de

stroom van externe mutaties voor. Formeel kunnen deze stromen als volgt worden gekarakteriseerd:

IA: een verzameling, de *invoerbasis* geheten. Deze bevat alle elementen, die door het systeem uit de omgeving kunnen worden ontvangen.

OR: een verzameling, de *uitvoerbasis* geheten. Deze bevat alle elementen, die door het systeem aan de omgeving kunnen worden afgegeven.

IS: een verzameling, de *externe-mutatiebasis* geheten. Deze bevat alle toestandselementen, die door de omgeving kunnen worden toegevoegd aan, of verwijderd uit de toestand van het systeem.

R : de verzameling der reële getallen. Deze verzameling wordt gebruikt als het tijddomein bij de beschrijving van het systeemgedrag.

$\iota \in R \rightarrow P$ (IA);

Voor $t \in R$ heet $\iota(t)$ de *invoer* op tijdstip t, dat is de verzameling invoerstimuli, die door het systeem tegelijkertijd op tijdstip t uit de omgeving worden ontvangen.
(Merk op dat vanwege het discrete karakter van de invoer, op de meeste tijdstippen t zal gelden : $\iota(t) = \varnothing$).
Een *invoergebeurtenis* is een paar $<a, t>$ met $a \in$ IA en $t \in R$.

(N.B.P(A) staat voor de machtsverzameling van verzameling A. In dit boek wordt daaronder steeds de verzameling van eindige deelverzamelingen van A verstaan.)

$\omega \in R \rightarrow P$ (OR);

Voor $t \in R$ heet $\omega(t)$ de *uitvoer* op tijdstip t, dat is de verzameling uitvoerstimuli, die op tijdstip t tegelijkertijd aan de omgeving worden afgegeven.
(Ook hiervoor geldt meestal: $\omega(t) = \varnothing$).
Een *uitvoergebeurtenis* is een paar $<r, t>$ met $r \in$ OR en $t \in R$;

$\varepsilon \in R \rightarrow P$ (IS);

Voor $t \in R$ heet $\varepsilon(t)$ de *externe mutaties* op tijdstip t, dat is de verzameling mutaties, die op tijdstip t door de omgeving worden

aangeboden.
(Ook hier zal meestal gelden: $\varepsilon(t) = \varnothing$).
Een *mutatiegebeurtenis* is een paar $<x, t>$ met $x \in IS$ en $t \in R$.

De nieuwe toestand definiëren we als het symmetrisch verschil van de oude toestand en de mutaties (zie appendix A).

In formelere termen uitgedrukt, houdt het causaliteitsbeginsel allereerst in, dat de uitvoerfunctie ω functioneel afhankelijk is van de invoerfunctie ι en de invoer-mutatiefunctie ε. Bij een bepaalde ι en ε hoort dus een bepaalde ω. Zo'n bij elkaar horend drietal noemen we een *individueel gedrag* van het systeem. In principe is elk systeem in staat zeer veel verschillende individuele gedragingen te vertonen. Men is immers vrij in de keuze van ι en ε.
We vatten daarom een individueel gedrag op als een element van een verzameling, die de *gedragsruimte* van het systeem heet. Deze verzameling denken we ons vervolgens bepaald door een functie B, de *gedragsfunctie* geheten, die zodanig is gedefinieerd, dat voor elk individueel gedrag geldt:

$$\omega = B(\iota, \varepsilon)$$

Het causaliteitsbeginsel houdt tevens in dat voor twee gegeven argumentwaarden $(\iota_1, \varepsilon_1,)$ en (ι_2, ε_2) het volgende geldt:

Voor elke $t' \in R$ is er een $h \geq 0$ zodanig, dat indien voor elke $t \leq t'$ geldt:
$\iota_1(t) = \iota_2(t)$ en $\varepsilon_1(t) = \varepsilon_2(t)$, dan is het ook zo dat voor elke s,
met $t' \leq s \leq t' + h$, geldt: $\omega_1(s) = \omega_2(s)$,
aangenomen dat de begintoestand in beide gevallen dezelfde is.

Bovendien leggen we aan B de eis van *stationariteit* op. Formeel uitgedrukt wil dat zeggen:

indien $(\iota_1, \varepsilon_1,)$ en (ι_2, ε_2) twee argumentwaarden van B zijn, en indien voor elke t en een bepaalde vaste k geldt:
$\iota_1(t) = \iota_2(t + k)$ en $\varepsilon_1(t) = \varepsilon_2(t + k)$, dan moet ook gelden:
$\omega_1(t) = \omega_2(t + k)$

Een specificatie van B wordt een *functionele specificatie* van het gedrag van een systeem genoemd.
In het algemeen is het niet mogelijk een simpele, extensionele of intensionele,

specificatie van B te geven. Een extensionele definitie is praktisch gesproken uitgesloten omdat de gedragsruimte doorgaans een oneindige verzameling is. De enige haalbare mogelijkheid is dus een intensionele definitie.

Ter illustratie van de potentiële complexiteit van de functie B gebruiken we als voorbeeld een deel van de operaties van een postorderbedrijf. Dit voorbeeld wordt in hoofdstuk 9 volledig uitgewerkt.

> Een postorderbedrijf ontvangt orders van klanten en verzendt leveringen aan klanten. Daarnaast plaatst het bestellingen bij leveranciers. De levering van zo'n bestelling is voor het bedrijf een aanvulling van de voorraad goederen. Het beschreven deel van het bedrijf communiceert alleen met de directie. De communicatie heeft betrekking op de regels voor het accepteren en het leveren van orders en voor het plaatsen van bestellingen.
>
> De verzameling IA bestaat uit twee soorten elementen:
>
> - orders; een order heeft drie parameters:
> - de klant, die de order plaatst;
> - het artikelsoort, waarvan men goederen wenst;
> - de gevraagde hoeveelheid goederen.
>
> - aanvullingen; een aanvulling heeft drie parameters:
> - de bestelling waarop de aanvulling betrekking heeft;
> - het artikelsoort van de goederen;
> - de hoeveelheid goederen.
>
> De verzameling OR bestaat ook uit twee soorten elementen:
>
> - leveringen; de parameters van een levering zijn dezelfde als de parameters van een order;
> - bestellingen; de parameters van een bestelling zijn dezelfde als de parameters van een aanvulling.
>
> De verzameling IS bestaat uit elementen van de volgende soorten:
>
> - artikelen; hiermee wordt het leverbare assortiment bepaald;
> - klanten; dat zijn de personen aan wie wordt geleverd;
> - datum; op grond hiervan wordt de orderdatum vastgesteld;
> - minimale voorraad per artikel;
> - de normatieve bestelhoeveelheid per artikel;
> - de bestelperiode per artikel; dit is het tijdinterval tussen de momenten waarop wordt gekeken of men moet bijbestellen;

- de uitleverperiode per artikel; dit is het tijdinterval tussen de momenten waarop (standaard) wordt gekeken of er kan worden geleverd;
- de maximum orderhoeveelheid per artikel.

Door de gegeven toelichtingen bij de beschrijving van IS wordt het een en ander gesuggereerd over het interne functioneren van het systeem, waaruit men al enige kennis over het gedrag van het systeem zou kunnen destilleren. Om het gedrag volledig te kennen moet men echter bijvoorbeeld ook weten welke regels en procedures er gelden voor de acceptatie en levering van orders en voor het plaatsen en ontvangen van bestellingen.

Het achterhalen van al deze zaken is een taak van de systeemanalyse. Het met elkaar in verband brengen van deze brokjes systeemkennis zodanig, dat men tot een precieze, intensionele, definitie van B geraakt blijkt, althans in de praktijk van de informatie-analyse, zo lastig te zijn, dat men het meestal maar achterwege laat, en de analysefase dus met een onvolledig resultaat afsluit.
We zullen in de navolgende twee delen van het boek een weg aangeven, waarlangs dat doel wel praktisch haalbaar wordt. In hoofdlijnen bestaat die uit het construeren van een conceptueel model van een reëel systeem, waarin statische én dynamische aspecten als een geïntegreerd geheel tot uitdrukking worden gebracht.
Deel II behandelt de theorie die aan zulke modellen ten grondslag ligt. In deel III worden daaraan twee schematechnieken toegevoegd, die de constructie van conceptuele modellen ondersteunen.

Deel II: Theorie

Dit deel, de kern van het boek, beslaat de hoofdstukken 4, 5 en 6. Het behandelt een metamodel en een specificatietaal voor het maken van conceptuele modellen van discrete dynamische systemen.

In hoofdstuk 4 wordt het metamodel op informele wijze gepresenteerd. Hoofdstuk 5 bevat een formalisering ervan. Een discreet dynamisch systeem wordt daarin gedefinieerd als een tupel <**S**, **M**, **A**, **R**, **T**, **I**>, waarvan de componenten verzamelingen en functies zijn, die tezamen een systeem volledig en precies beschrijven. Door toevoeging van een component **E**, de omgevingsinvloeden, wordt een proces bepaald.

De specificatie van deze componenten is het onderwerp van hoofdstuk 6. Daarin wordt een eerste orde taal gedefinieerd voor het specificeren van toestanden. De taal is uitgebreid met de mogelijkheid tot het specificeren van mutaties en responsies. Het metamodel en de specificatietaal zijn eerder, in iets andere vormen, behandeld in [Dietz, Hee 1987] en [Hee, Houben, Dietz 1987].

Het metamodel uit hoofdstuk (4 en) 5, en de specificatietaal uit hoofdstuk 6 vormen de theorie achter de conceptuele modellen waarvan in deel III een voorbeeld wordt gegeven, en waarvan in datzelfde deel ook praktische aspecten worden behandeld.

4. Conceptualisering

4.1 Inleiding

In dit hoofdstuk zal een raamwerk worden ontwikkeld voor het modelleren van discrete dynamische systemen. Als een inleiding daarop volgt hieronder een beschrijving van de mogelijke werking van het postorderbedrijf, waarvan in hoofdstuk 3 het "black-box"-model werd gepresenteerd.

We beschouwen de volgende vier functies van het systeem, die door gelijknamige afdelingen worden vervuld: orderacceptatie, orderlevering, bestellen en ontvangst.

De afdeling *orderacceptatie* ontvangt schriftelijke orders van klanten. Deze worden een of twee keer per dag bezorgd door postdiensten.
Elke medewerker heeft een bureau waarop een bak staat met nog te verwerken orders en een terminal via welke hij toegang heeft tot de bestanden "artikelen", "klanten" en "klantorders".
Voorts is elke medewerker bekend met de acceptatieregels. De belangrijkste daarvan zijn dat het gevraagde artikel tot het leverbare assortiment moet behoren, dat de gewenste hoeveelheid een, artikel-afhankelijk, maximum niet overschrijdt en dat de klant niet op de zwarte lijst voorkomt. De zwarte lijst is een deelverzameling van het klantenbestand.
Er is een voor iedere medewerker zichtbare klok waarop ook de datum is af te lezen. Een geaccepteerde order wordt van die datum voorzien en vervolgens ingevoerd in het klantorderbestand.
De medewerkers van de afdeling orderacceptatie kunnen geen wijzigingen aanbrengen in het artikelbestand en het klantenbestand. Dat doen andere afdelingen. Om uitweiding over de rest van de organisatie van het bedrijf te vermijden, zullen we zeggen, dat de directie in die bestanden wijzigingen kan aanbrengen.
De directie bepaalt ook de datum op de klok. Indien er bijvoorbeeld op zaterdag wordt overgewerkt om een achterstand van vrijdag in te lopen, is de datum op de klok die van vrijdag.
Tot zover de beschrijving van de werkzaamheden op de afdeling orderacceptatie.
De taken van de andere drie kunnen beknopt als volgt worden omschreven.

De afdeling *orderlevering* bekijkt eenmaal per dag welke klantorders kunnen worden geleverd. Van elk artikel wordt dan zoveel mogelijk geleverd. De

medewerkers van deze afdeling hebben toegang tot de onderhanden klantorders en de voorraadgegevens.
De parameterwaarden van de regels voor het bepalen van de volgende te leveren order worden door de directie vastgesteld. Naast deze dagelijkse procedure wordt er ook geleverd naar aanleiding van een lever-signaal, afkomstig van de afdeling ontvangst.

De afdeling *bestellen* inspecteert periodiek de voorraadgegevens en bepaalt of er voor een artikel bestellingen moeten worden geplaatst. De parameterwaarden van de bestelregels zijn, evenals de inspectie-periode, per artikel vastgesteld (door de directie).
Geplaatste bestellingen worden genoteerd in het bestand "bestelorders".

De afdeling *ontvangst* tenslotte, krijgt op willekeurige tijdstippen aanvullingen binnen, afkomstig van leveranciers. Een aanvulling wordt geaccepteerd als er een corresponderende uitstaande bestelling is. Indien dat het geval is, wordt de bestelorder verwijderd uit het bestand "bestelorders", worden de voorraadgegevens bijgewerkt en wordt er een leversignaal naar de afdeling orderlevering gestuurd, zodat eventueel wachtende klantorders meteen kunnen worden geleverd.

In de beschreven werking van het postorderbedrijf kan men actieve en passieve systeemcomponenten onderkennen. Met *actieve componenten* bedoelen we componenten die veranderingen kunnen aanbrengen in de stand van zaken op enig moment. Voorbeelden hiervan zijn de medewerkers van elke afdeling, de directie, de klanten en de leveranciers.
Met *passieve componenten* bedoelen we bewaarplaatsen van allerlei zaken. Voorbeelden hiervan zijn de diverse bestanden, het magazijn en de klok.
Een actieve component opereert volgens bepaalde regels of procedures. Hij inspecteert en/of verandert daarbij de inhoud van een aantal bewaarplaatsen (zie bijvoorbeeld de beschrijving van de werkzaamheden binnen de afdeling orderacceptatie).
Het actief worden van een actieve component hangt samen met een *gebeurtenis*. Voorbeelden van gebeurtenissen zijn: de binnenkomst van een order en het verstreken zijn van de bestelperiode van een artikel. Gebeurtenissen zijn zelf weer het gevolg van de operaties van actieve componenten.
Elke gebeurtenis zullen we zo opvatten; een toevallige gebeurtenis beschouwen we als een gebeurtenis waarvan we de oorzaak niet kennen. Bijvoorbeeld, een order kan alleen binnenkomen als hij door een klant is verzonden.
Ook voor periodieke activiteiten gaat dit beeld op. Bijvoorbeeld, het leveren op een bepaalde dag kan men opvatten als een activiteit, die is veroorzaakt door de overeenkomstige activiteit op de dag tevoren.

Tussen de onderscheiden componenten spelen twee soorten verbanden een rol, zoals uit het voorgaande is af te leiden. De ene soort zijn de toegangsverbanden tussen de actieve en de passieve componenten. Bijvoorbeeld, de medewerkers van de afdeling orderacceptatie hebben toegang tot het artikelbestand, het klantenbestand, de klok, het klantorderbestand en de bak te verwerken orders op hun bureau.
De andere soort verbanden zijn de activeringsverbanden tussen actieve componenten. Bijvoorbeeld, de afdeling ontvangst stuurt lever-signalen naar de afdeling orderlevering.
Tenslotte merken we op, dat men niet alleen de afzonderlijke medewerkers, maar ook gehele afdelingen als actieve componenten kan opvatten.

4.2 De conceptualisering van een gesloten systeem

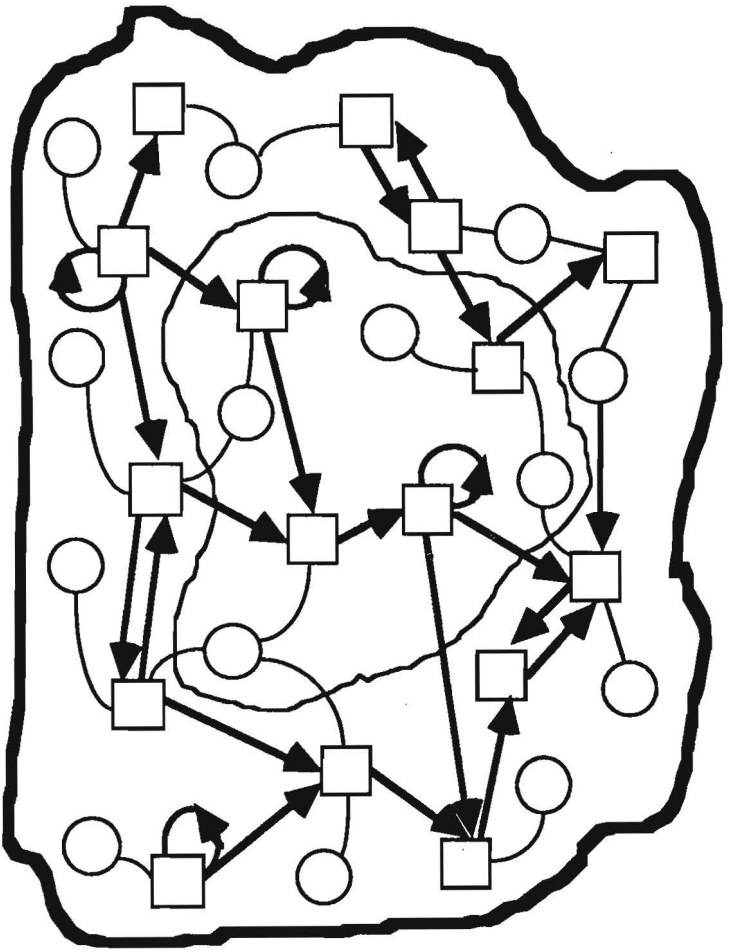
De voorgaande analyse is een opstapje naar de hieronder volgende conceptualisering van een discreet dynamisch systeem.
Beschouw een netwerk zoals getekend in figuur 4.1 (omsloten door de dik getekende kromme).
Het netwerk bestaat uit twee soorten componenten, processoren en geheugens genoemd, en twee soorten verbindingen tussen componenten, transactiekanalen en interconnecties geheten. (De betekenissen van de gebruikte symbolen zijn verklaard in paragraaf 7.1).
Het netwerk is een gesloten systeem. Dat houdt dus in, dat er geen enkele relatie bestaat tussen de componenten van het netwerk en de omgeving ervan.

Om de dynamica van het netwerk te kunnen beschrijven is een *tijddomein* nodig. Hiervoor kiezen we (zoals ook in hoofdstuk 3 is gedaan) de verzameling der reële getallen. De eigenschappen van deze verzameling komen goed overeen met de intuïtieve notie van tijd, die de meeste mensen bezitten.

Een *geheugen* is een passieve component, die de functie van bewaarplaats heeft. De inhoud van een geheugen op tijdstip t noemen we de *toestand van het geheugen* op tijdstip t. Een toestand is een deelverzameling van een verzameling, die we de *toestandsbasis* van het geheugen noemen. Deze bevat dus alle elementen die ooit tot de toestand kunnen behoren.
De toestandsbases van de geheugens in het netwerk zijn disjunct.
Een elementaire verandering van een toestand (dat wil zeggen: het toevoegen of verwijderen van een element) heet een *mutatie*.
Een verbinding tussen een processor en een geheugen heet een *interconnectie*. De toestanden van alle geheugens, waarmee een processor is verbonden, vormen

tezamen de *toestand van de processor.*
Het geheel van de interconnecties van het netwerk heet de *communicatiestructuur* van het netwerk.

Een *processor* is een actieve component. De activering van een processor heeft twee soorten effecten op de situatie in het netwerk: veranderingen van de toestand van de processor en toekomstige activeringen. Die activeringen kunnen zowel andere processoren als zichzelf betreffen.



Figuur 4.1: Netwerkvoorstelling van een systeem

Een processor wordt geactiveerd door invoergebeurtenissen. Een *invoergebeurtenis* wordt gekarakteriseerd door een actie en een tijdstip, het *gebeurtenistijdstip* geheten. We noteren daarom een invoergebeurtenis als een paar <a, t>, waarin a een actie en t een tijdstip, het gebeurtenistijdstip, voorstelt.
Gebeurtenissen kunnen coïncideren. Dit houdt in dat twee of meer processoren op hetzelfde moment kunnen worden geactiveerd, maar ook dat een processor wordt geactiveerd door een aantal gebeurtenissen tegelijkertijd.
Een *actie* is een element van een verzameling, de *actiebasis* van de processor geheten. Deze bevat alle elementen, die een actie voor de processor kunnen zijn. De actiebases van de processoren in het netwerk zijn disjunct.

Het effect van een activering van een processor is afhankelijk van de actie van de invoergebeurtenis (of acties bij coïncidentie van twee of meer gebeurtenissen) en van de toestand van de processor op het gebeurtenistijdstip. Het effect wordt teweeggebracht door een mechanisme, dat de *motor* van de processor heet. Die bestaat uit twee delen: het mutatiemechanisme en het reactiemechanisme.

Het *mutatiemechanisme* produceert bij activering instantaan een aantal, mogelijkerwijs nul, mutaties van de toestand van de processor. (Omdat de toestandsbases van de geheugens disjunct zijn, heeft een mutatie altijd betrekking op de toestand van precies één geheugen).

Het *reactiemechanisme* produceert bij activering instantaan een aantal, mogelijkerwijs nul, responsies. Een *responsie* is gekarakteriseerd door een reactie en een vertragingstijd. We noteren daarom een responsie als een paar <r, d>, waarin r een reactie voorstelt en d de vertragingstijd (dit is een positief reëel getal). Een responsie <r, d> die is geproduceerd op tijdstip t heeft een *uitvoergebeurtenis* <r, t + d> tot gevolg. Anders gezegd, een responsie <r, d> heeft de afgifte van de reactie r tot gevolg, precies d tijdeenheden later.
Een *reactie* is een element van een verzameling, de *reactiebasis* van de processor geheten. Deze verzameling bevat alle elementen, die ooit een reactie van de processor kunnen zijn. De reactiebases van de processoren in het netwerk zijn disjunct.

Tot welke toekomstige activeringen van processoren een responsie aanleiding geeft, wordt bepaald door de transactiekanalen tussen processoren. Het *transactiekanaal* van processor i naar processor k legt het verband vast tussen reacties van i en acties van k. Indien zo'n verband bestaat voor een reactie r van processor i en een actie a van processor k, dan betekent dat, dat een uitvoergebeurtenis <r, t> van processor i, een invoergebeurtenis <a, t> voor processor k is.
We zeggen dat er een transactiekanaal bestaat van processor i naar processor k indien er tenminste één reactie van i wordt omgezet in een actie voor k.

Indien i=k, wordt het transactiekanaal het *terugkoppelkanaal* van processor i genoemd.
Het geheel van de transactiekanalen van het netwerk heet de *interactiestructuur* van het netwerk.

De reeks van gebeurtenissen, die in de loop van de tijd plaatsvinden, en de toestanden van de geheugens als functie van de tijd, noemen we tezamen het *proces* van het netwerk.
Het proces kan op elk punt in het tijddomein worden uitgedrukt in de waarde van een aantal *procesvariabelen*. Een van de procesvariabelen is de reeds genoemde toestand van elk geheugen. Andere procesvariabelen zijn de agenda en de reagenda van elke processor.
Onder de *agenda* van een processor verstaan we de toekomstige invoergebeurtenissen voor de processor.
De toekomstige uitvoergebeurtenissen van een processor noemen we de *reagenda* van de processor.
Voor elk paar <r, t> in de reagenda van processor i is er een paar <a, t> in de agenda van processor k, indien er een transactiekanaal is van processor i naar processor k, die een reactie r omzet in een actie a.

Met betrekking tot de agenda en de reagenda van een processor eisen we dat het aantal gebeurtenissen, waarvan het gebeurtenistijdstip in een eindig interval ligt, ook eindig is. Dit kan worden gegarandeerd door een vaste minimale waarde te kiezen voor de vertragingstijd van responsies. (Vergelijk hiermee de definitie van "discreet" in hoofdstuk 3).

4.3 De conceptualisering van een open systeem

Beschouw vervolgens een deelnetwerk, zoals uitgebeeld in figuur 4.1 door de dun getekende gesloten kromme. De afbakening is zodanig, dat elk geheugen, waarmee een processor in het deelnetwerk is verbonden, ook tot het deelnetwerk behoort.
Het deelnetwerk is een open systeem. Tussen het deelnetwerk en de rest van het netwerk, de *omgeving* geheten, is er in principe sprake van interactie en van communicatie.
We spreken van *interactie* tussen het deelnetwerk en de omgeving, indien er transactiekanalen zijn tussen processoren in het deelnetwerk en processoren in de omgeving.
We spreken van *communicatie* tussen het deelnetwerk en de omgeving, indien er interconnecties zijn tussen geheugens in het deelnetwerk en processoren in de omgeving.

Een aldus afgebakend deelnetwerk noemen we een *dds* (discreet dynamisch systeem). Een dds is een conceptueel systeem.

Een transactiekanaal van een processor in de omgeving van een dds naar een processor in het dds heet een *invoerkanaal* van het dds. Via deze kanalen worden dus reacties van processoren in de omgeving van een dds omgezet in acties voor processoren in het dds. De stroom van acties, die op deze wijze worden aangeboden aan processoren in het dds, heet de *invoer* van het dds (vergelijk definitie 3.1).

Een transactiekanaal van een processor in een dds naar een processor in de omgeving heet een *uitvoerkanaal* van het dds. Via deze kanalen worden dus reacties van processoren in het dds omgezet in acties voor processoren in de omgeving. De stroom van reacties, die op deze wijze worden omgezet in acties voor processoren in de omgeving, heet de *uitvoer* van het dds (vergelijk definitie 3.1).
(N.B. We beschouwen dus niet de hiermee corresponderende invoer voor de omgeving).

Processoren in de omgeving van een dds, die zijn verbonden met geheugens in het dds, kunnen in principe mutaties produceren voor die geheugens. We zullen deze mutaties *externe mutaties* noemen (vergelijk definitie 3.1).

We willen een dds zo onafhankelijk mogelijk van zijn omgeving beschrijven. De motivering van deze stellingname is dat men meestal toch geen of weinig kennis heeft van de omgeving (dat is het geval bij systeemanalyse), ofwel zo weinig mogelijk eisen aan de omgeving wil opleggen (dat is het geval bij systeemsynthese). Daarom zullen we de invoerkanalen, de uitvoerkanalen en de interconnecties met processoren in de omgeving niet tot de definitie van een dds rekenen.

Wat de invoer betreft introduceren we per processor een verzameling invoergebeurtenissen, die door processoren in de omgeving worden geproduceerd. Deze verzameling noemen we de *externe agenda* van de processor.
De uitvoer van een dds is op elk moment een deelverzameling van de reacties van processoren in het dds op dat moment, namelijk die reacties, die ook tot de uitvoerbasis behoren (zie definitie 3.1).
Wat de externe mutaties betreft introduceren we per geheugen een verzameling *externe mutanda*, dat is de verzameling mutatiegebeurtenissen, die door processoren in de omgeving worden geproduceerd.
Tevens introduceren we voor elk geheugen in het dds een begintoestand.

Zoals we dat ook met betrekking tot het gedrag van een systeem hebben gedaan in hoofdstuk 3, maken we een onderscheid tussen de procesruimte van een dds en een (individueel) proces.

De *procesruimte* wordt bepaald door de definitie van het netwerk (de geheugens, de processoren, de transactiekanalen en de interconnecties). Afhankelijk van de omgevingsinvloeden, zoals de externe agenda en de externe mutanda, wordt een bepaald *proces* gevolgd.
Een proces is volledig bepaald door een begintijdstip, zeg t_0, de (begin)toestanden van de geheugens op t_0, de externe agenda op t_0 en de externe mutanda op t_0.

Een dds is, conform de definities van hoofdstuk 2, een *conceptueel systeem*. We zullen zo'n systeem gebruiken als *conceptueel model* van een discreet dynamisch systeem (dat een reëel systeem is). De wijze waarop dat gebeurt, zullen we demonstreren aan de hand van het voorbeeld van het postorderbedrijf.

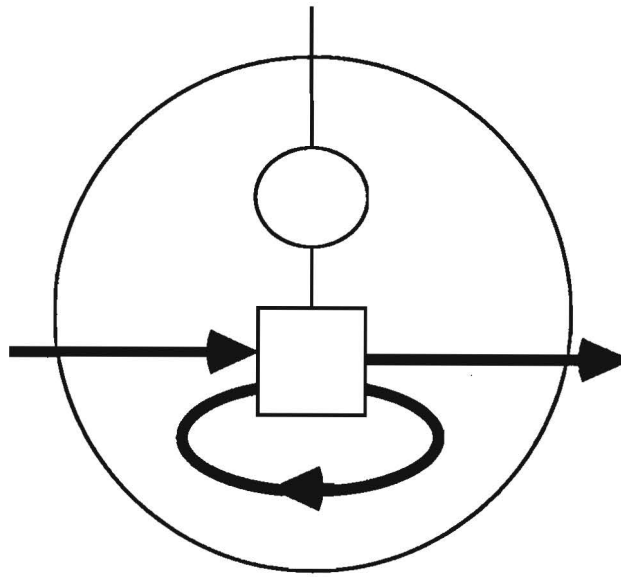
Een medewerker van de afdeling orderacceptatie wordt gemodelleerd als een processor. De geheugens waarmee deze processor via interconnecties is verbonden corresponderen met: de bak met nog te verwerken orders, het artikelbestand, het klantenbestand, de klok en het klantorderbestand.

De activerings-actie is "verwerk de volgende order". Het effect van zo'n activering op tijdstip t hangt onder andere af van de toestand van de bak met nog te verwerken orders. Stel dat die bak niet leeg is. Dan wordt er een order gekozen (bijvoorbeeld de "bovenste"). Als van deze order het gevraagde artikel bestaat, de klant niet voorkomt op de zwarte lijst en de bestelhoeveelheid niet groter is dan het toegestane maximum, wordt de order geaccepteerd. Dat houdt in dat er een mutatie voor het klantorderbestand wordt geproduceerd. Een van de parameters van die mutatie is de datum, die is bevat in het geheugen dat de klok voorstelt. Tevens wordt de order verwijderd uit de bak nog te verwerken orders. Als responsie produceert de processor een paar <r, d>.
Er is een terugkoppelkanaal, dat een reactie r omzet in een actie "verwerk de volgende order". Dit is de actie van een gebeurtenis, die d tijdseenheden later plaatsvindt. Anders gezegd, aan de agenda van de processor is de gebeurtenis <"verwerk de volgende order", t+d> toegevoegd. De vertragingstijd d stelt de verwerkingstijd van de onderhanden order voor.

Het toevoegen van orders aan de bak met nog te verwerken orders op elk bureau kan worden gemodelleerd door een aparte processor (deze stelt bijvoorbeeld een loopjongen voor, die de orders over de bakken verdeelt).
Deze processor wordt geactiveerd door een verzameling gebeurtenissen, die tegelijkertijd plaatsvinden. Elk van die gebeurtenissen stelt een binnengekomen order voor (de actie) met het tijdstip van bezorging door de postdienst.

4.4 De aggregatie van een systeem

Al eerder werd opgemerkt dat bijvoorbeeld de gehele afdeling orderacceptatie als één processor met geheugen kan worden opgevat. We zullen dit nu nader bestuderen. Daartoe introduceren we het begrip *basis-dds*. Onder een basis-dds wordt een dds verstaan dat bestaat uit één processor, één geheugen, de interconnectie tussen de processor en het geheugen, en, in principe ook, een terugkoppelkanaal (zie figuur 4.2).



Figuur 4.2: Netwerkvoorstelling van een basis-dds

Een basis-dds wordt de *aggregatie* genoemd van een dds, indien het daaruit kan worden verkregen door het samenvoegen van gelijksoortige componenten en verbindingen op de volgende wijze:

a. de processoren worden samengevoegd door:
   - het verenigen van de actiebases
   - het verenigen van de reactiebases
   - het 'parallel schakelen' van de motoren;

b. de geheugens worden samengevoegd door het verenigen van de

toestandsbases;

c. de transactiekanalen worden samengevoegd door ze te 'bundelen'; het resulterend kanaal is het terugkoppelkanaal van de aggregaat-processor;

d. de interconnecties worden samengevoegd door ze te 'bundelen' tot één interconnectie tussen de (aggregaat-) processor en het (aggregaat-) geheugen;

Deze definitie van de aggregatie van een dds is erg informeel en roept wellicht allerlei vragen op. In hoofdstuk 5 wordt het begrip aggregatie precies gedefinieerd, alsmede de *isomorfie* van twee aggregaties.
Indien de aggregaties van twee dds-en isomorf zijn, noemen we de dds-en *equivalent*. Twee equivalente dds-en zijn uitwisselbaar; ze vertonen ten opzichte van de omgeving hetzelfde gedrag.

De aggregatie-techniek maakt het mogelijk een systeem op een willekeurig niveau van detail te beschouwen. Immers, een (deel-) basis-dds van een dds kan men vervangen door elk dds, dat daaraan equivalent is. Het gebruik van de aggregatietechniek op deze wijze heet *hiërarchische compositie/decompositie*.
De (intensionele) definitie van de gedragsfunctie B (zie hoofdstuk 3) kan zodoende stapsgewijs worden ontwikkeld dan wel verklaard.

5. Formalisering

In dit hoofdstuk wordt een formele basis gegeven aan hetgeen in hoofdstuk 4 op informele wijze is gepresenteerd.
Dit gebeurt door de definitie van mathematische modellen voor een dds, het proces en het gedrag van een dds, en voor de aggregatie van een dds.
Op grond van de isomorfie van hun aggregaties wordt tevens de equivalentie van twee systemen formeel gedefinieerd.

Definitie 5.1

Een *dds* is een tupel <**S, M, A, R, T, I**>, waarvan de componenten als volgt zijn gedefinieerd:

S: een verzamelingswaardige functie;
dom (**S**) heet de verzameling van *geheugen indices*; dom (**S**) $\neq \emptyset$;
voor $j \in$ dom (**S**) heet $\mathbf{S}_j$ *) de *toestandsbasis van geheugen j*;
$\forall i, j \in$ dom (**S**) : $i \neq j \rightarrow \mathbf{S}_i \cap \mathbf{S}_j = \emptyset$.

M: een functie;
dom (**M**) heet de verzameling van *processor indices*, en is een eindige verzameling; dom (**M**) $\neq \emptyset$;
voor $i \in$ dom (**M**) is $\mathbf{M}_i = <\mathbf{MM}_i, \mathbf{MR}_i>$, de *motor* van processor i geheten, waarin $\mathbf{MM}_i$ en $\mathbf{MR}_i$ functies zijn:
$\mathbf{MM}_i$ heet het *mutatiemechanisme van processor i*;
$\mathbf{MR}_i$ heet het *reactiemechanisme van processor i*;
laat $\mathbf{MS}_i = (\cup j \in \mathbf{I}_i : \mathbf{S}_j)$, de *toestandsbasis van processor i* geheten;

$$\mathbf{MM}_i \in P(\mathbf{A}_i) * P(\mathbf{MS}_i) \rightarrow P(\mathbf{MS}_i);$$
$$\mathbf{MR}_i \in P(\mathbf{A}_i) * P(\mathbf{MS}_i) \rightarrow P(\mathbf{R}_i * R^{\varepsilon});$$

waarin $R^{\varepsilon} = \{x \mid x \in R \wedge x \geq \varepsilon\}$ voor een vaste $\varepsilon > 0$.

*) In plaats van S(j) noteren we $\mathbf{S}_j$; dit geldt ook voor de overige functies die als domein dom (**S**) of dom (**M**) hebben.

A: een verzamelingswaardige functie met dom (**A**) = dom (**M**);
voor i ∈ dom (**A**) heet $\mathbf{A}_i$ de *actiebasis van processor i* ;
$\forall i, j \in \text{dom } (\mathbf{A}) : i \neq j \rightarrow \mathbf{A}_i \cap \mathbf{A}_j = \emptyset$.

R: een verzamelingswaardige functie met dom (**R**) = dom (**M**);
voor i ∈ dom (**R**) heet $\mathbf{R}_i$ de *reactiebasis van processor i* ;
$\forall i, j \in \text{dom } (\mathbf{R}) : i \neq j \rightarrow \mathbf{R}_i \cap \mathbf{R}_j = \emptyset$.

T: een functie met dom (**T**) = dom (**M**) * dom (**M**)
voor <i, j> in dom (**T**) geldt:
$\mathbf{T}_{ij} \subset \mathbf{R}_i * \mathbf{A}_j$;
Indien $\mathbf{T}_{ij} \neq \emptyset$ heet $\mathbf{T}_{ij}$ het *transactiekanaal van processor i naar processor j*;

I: een functie met dom (**I**) = dom (**M**);
voor i ∈ dom (**I**) heet $\mathbf{I}_i$ de *interconnecties van processor i*;
$\forall i \in \text{dom } (\mathbf{I}) : \mathbf{I}_i \subset \text{dom } (\mathbf{S})$;

(einde definitie 5.1)

Zoals uit bovenstaande definitie blijkt is het aantal geheugens van een dds onbeperkt. Wel is het zo dat er minstens één geheugen moet zijn en dat de toestandsbases disjunct dienen te zijn. Deze eisen houden verband met de definitie van de aggregatie van een dds. Het bijzondere geval van een systeem zonder toestand (dat dus strikt genomen geen discreet dynamisch systeem is) kan worden gemodelleerd door een geheugen te definiëren waarvan de toestandsbasis leeg is.

Het aantal processoren van een dds moet wel eindig zijn. Deze eis staat er, tezamen met R^{ε} als domein van de tijdvertragingen, garant voor dat er zich geen verdichtingen van gebeurtenissen voordoen. (Dit is te bewijzen).
Verder dient er tenminste één processor te zijn.

Ook de actiebases en de reactiebases moeten disjunct zijn. De argumentatie van deze eis is dezelfde als bij de disjunctie-eis van de toestandsbases werd gegeven.

Definitie 5.2

Een proces is bepaald door aan een dds een component **E**, die de omgevingsinvloeden representeert, toe te voegen zoals hieronder gedefinieerd:

E: een tupel <**ET, ES, EM, EA**> waarin:

ET : een element van R, het *begintijdstip* van het proces geheten;

ES: een functie met dom (**ES**) = dom (**S**);
voor j $\in$ dom (**ES**) is $\mathbf{ES}_j \in P\ (\mathbf{S}_j)$, de *begintoestand van geheugen j* geheten;

EM: een functie met dom (**EM**) = dom (**S**);
voor j $\in$ dom (**EM**) is $\mathbf{EM}_j \in P\ (\mathbf{S}_j * R)$, de
externe mutanda voor geheugen j geheten;

EA: een functie met dom (**EA**) = dom (**M**);
voor i $\in$ dom (**EA**) is $\mathbf{EA}_i \in P\ (\mathbf{A}_i * R)$,
de *externe agenda van processor* i geheten;

Het *proces* van een dds, gedefinieerd als een tupel <**S, M, A, R, T, I**> conform definitie 5.1, waaraan een component **E**, zoals hierboven gedefinieerd, is toegevoegd, is gedefinieerd als een tupel <σ, α, ρ>, waarin:

σ : een functie met dom (σ) = dom (**S**);
voor j $\in$ dom (σ) is $\sigma_j \in R \rightarrow P(\mathbf{S}_j)$;
voor t $\in R$ heet $\sigma_j(t)$ de *toestand van geheugen j* op tijdstip t;

α : een functie met dom (α) = dom (**M**);
voor i $\in$ dom (α) is $\alpha_i \in R \rightarrow P(\mathbf{A}_i)$;
voor t $\in R$ heet $\alpha_i(t)$ de *acties voor processor i* op tijdstip t;

ρ : een functie met dom (ρ) = dom (**M**);
voor i $\in$ dom (ρ) is $\rho_i \in R \rightarrow P(\mathbf{R}_i)$;
voor t $\in R$ heet $\rho_i(t)$ de *reactie van processor i* op tijdstip t;

Ten behoeve van de verdere definitie van σ, α en ρ worden de volgende hulpfuncties ingevoerd.

$\tau \in N \rightarrow R$;

de tijdstippen $\tau(n)$, $n \in N$ zijn de enige tijdstippen, waarop één of meer procesvariabelen van waarde veranderen; $\tau(n)$ wordt genoteerd als τ_n;

ψ : een functie met dom (ψ) = dom (**M**);
voor $i \in$ dom (ψ) is $\psi_i \in R \rightarrow P(\mathbf{R}_i * R)$;
voor $t \in R$ heet ψ_i (t) de *reagenda van processor i* op tijdstip t.

ϕ : een functie met dom (ϕ) = dom (**M**);
voor $i \in$ dom (ϕ) is $\phi_i \in R \rightarrow P(\mathbf{A}_i * R)$;
voor $t \in R$ heet ϕ_i (t) de *agenda van processor i* op tijdstip t;

γ : een functie met dom (γ) = dom (**M**);
voor $i \in$ dom (γ) is $\gamma_i \in R \rightarrow P(\cup_j \in \mathbf{I}_i : \mathbf{S}_j)$;
voor $t \in R$ heet γ_i(t) de *toestand van de processor i* op tijdstip t;

δ : een functie met dom (δ) = dom (**M**);
voor $i \in$ dom (δ) is $\delta_i \in R \rightarrow P(\cup j \in \mathbf{I}_i : \mathbf{S}_j)$;
voor $t \in R$ heet δ_i (t) de *mutaties door processor i* op tijdstip t.

μ : een functie met dom (μ) = dom (**S**);
voor $j \in$ dom (μ) is $\mu_j \in R \rightarrow P(\mathbf{S}_j)$;
voor $t \in R$ heet μ_j (t) de *mutaties voor geheugen j* op tijdstip t.

λ : een functie met dom (λ) = dom (**M**);
voor $i \in$ dom (λ) is $\lambda_i \in R \rightarrow P(\mathbf{R}_i * R^{\varepsilon})$;
voor $t \in R$ heet λ_i (t) de *responsies van processor i* op tijdstip t.

Voor $n \in N$, $i \in$ dom (M) en $j \in$ dom (S) geldt dan:

$$\tau_0 = \mathbf{ET};$$

$$\phi_i(\tau_0) = \mathbf{EA}_i;\ \psi_i(\tau_0) = \varnothing;\ \sigma_j(\tau_0) = \mathbf{ES}_j;\ \gamma_i(\tau_0) = \cup\, j \in \mathbf{I}_i\colon \sigma_j(\tau_0);$$

$$\begin{aligned}\tau_{n+1} = \min\ \{t \mid t > \tau_n \wedge\ & ((\exists i \in \text{dom (M)}\colon \exists a \in \mathbf{A}_i\colon\ \langle a, \triangleright \in \phi_i(\tau_n)) \vee \\ & (\exists i \in \text{dom (M)}\colon \exists r \in \mathbf{R}_i\colon\ \langle r, \triangleright \in \psi_i(\tau_n)) \vee \\ & (\exists j \in \text{dom (S)}\colon \exists x \in \mathbf{S}_j\colon\ \langle x, \triangleright \in \mathbf{EM}_j))\};\end{aligned}$$

$$\begin{aligned}\psi_i(\tau_{n+1}) = \{\ \langle r, \triangleright \mid (\ \langle r, \triangleright \in \psi_i(\tau_n) \wedge t > \tau_{n+1}) \vee \\ (\ \langle r, t - \tau_{n+1} \rangle \in \lambda_i(\tau_{n+1})\ ]\end{aligned}$$

$$\begin{aligned}\phi_i(\tau_{n+1}) = \{\ \langle a, \triangleright \mid (\langle a, \triangleright \in \phi_i(\tau_n) \wedge t > \tau_{n+1}) \vee \\ (\exists k \in \text{dom (M)}\colon \exists r \in \mathbf{R}_k\colon \\ \langle r, \triangleright \in \psi_k(\tau_{n+1}) \wedge \langle r, a \rangle \in \mathbf{T}_{ki})\};\end{aligned}$$

$$\alpha_i(\tau_{n+1}) = \{a \mid \langle a, \tau_{n+1} \rangle \in \phi_i(\tau_n)\};$$

$$\delta_i(\tau_{n+1}) = \mathbf{MM}_i(\alpha_i(\tau_{n+1}), \gamma_i(\tau_n));$$

$$\begin{aligned}\mu_j(\tau_{n+1}) = ((\Delta\, i \in \text{dom (M)} : \delta_i(\tau_{n+1})) \cap \mathbf{S}_j)\ \Delta \\ \{x \mid \langle x, \tau_{n+1} \rangle \in \mathbf{EM}_j\};\end{aligned}$$

$$\sigma_j(\tau_{n+1}) = \sigma_j(\tau_n)\ \Delta\ \mu_j(\tau_{n+1});$$

$$\gamma_i(\tau_n) = \cup\, j \in \mathbf{I}_i : \sigma_j(\tau_n);$$

$$\lambda_i(\tau_{n+1}) = \mathbf{MR}_i(\alpha_i(\tau_{n+1}), \gamma_i(\tau_n));$$

$$\rho_i(\tau_{n+1}) = \{r \mid \langle r,\ \tau_{n+1} \rangle, \in \psi_i(\tau_n)\};$$

Voor $\tau_n < t < \tau_{n+1}$, $i \in$ dom (M) en $j \in$ dom (S) geldt:

$\phi_i(t) = \phi_i(\tau_n)$;
$\psi_i(t) = \psi_i(\tau_n)$;
$\sigma_j(t) = \sigma_j(\tau_n)$;
$\gamma_i(t) = \gamma_i(\tau_n)$;
$\alpha_i(t) = \delta_i(t) = \lambda_i(t) = \rho_i(t) = \varnothing$;
$\mu_j(t) = \varnothing$;

(einde definitie 5.2)

De volgende opmerkingen zijn een toelichting bij deze definitie.

De τ-tijdstippen zijn alle tijdstippen waarop er een externe invoergebeurtenis of een uitvoergebeurtenis of een externe mutatiegebeurtenis plaatsvindt. Andere gebeurtenistijdstippen zijn er niet.

De reagenda van een processor worden op elk τ-tijdstip uitgebreid met een verzameling nieuw geproduceerde uitvoergebeurtenissen. Tegelijkertijd worden de agenda uitgebreid van elke processor, waarnaar een transactiekanaal bestaat.

Een nieuwe toestand van een geheugen is de oude toestand verschild met de verzameling mutaties. Door deze definitie is het niet nodig van een mutatie aan te geven of het een toevoeging of een verwijdering betreft. Door de verschiloperatie is dit eenduidig bepaald:

- als het mutatie-element tot de toestand behoort, wordt het daaruit verwijderd;
- als het mutatie-element niet tot de toestand behoort wordt het eraan toegevoegd.

De verzameling mutaties bestaat zelf weer uit het symmetrisch verschil van de interne en de externe mutaties. Onder de interne mutaties verstaan we daarbij de verschilling van de bijdragen van alle processoren (dat zijn de mutaties van de processoren voorzover ze betrekking hebben op een bepaald geheugen, vandaar de doorsnijding met S_j).

(N.B. Analoog aan de vereniging over een verzameling van verzamelingen spreken we van de verschilling over een verzameling van verzamelingen.)

Door definitie 5.1 is de procesruimte van een dds vastgelegd, dat is dus de verzameling van alle mogelijk (individuele) processen. Door toevoeging van een component **E** wordt hieruit een bepaald proces gekozen.

Definitie 5.3

Een *basis-dds* is een tupel <**S, M, A, R, T, I**>, gedefinieerd conform definitie 5.1 waarvoor geldt dat card (dom (**M**)) =1 en card (dom (**S**)) =1.
(De functie card stelt de cardinaliteit van een verzameling voor).

(einde definitie 5.3).

Elke functie **F** met dom (**M**) of dom (**S**) als domein heeft voor een basis-dds dus een singleton als domein. We zullen $\mathbf{F}_i$ met $i \in$ dom (**M**) of $i \in$ dom (**S**) gemakshalve noteren als F (dat dus het beeld van het enige element in dom (**M**) of dom (**S**) onder **F** voorstelt). Het tupel dat een basis-dds definieert zullen we daarom noteren als <**S, M, A, R, T, I**>.

T heet het *terugkoppelkanaal* van de processor.

Op overeenkomstige wijze worden de externe invloeden genoteerd als E. Een proces van een basis-dds wordt genoteerd als een tupel $<\underline{\sigma}, \underline{\alpha}, \underline{\rho}>$.

Definitie 5.4

De *aggregatie* van een dds, gedefinieerd conform definite 5.1 als een tupel <**S, M, A, R, T, I**> is een basis-dds, gedefinieerd conform definitie 5.3 en genoteerd als een tupel <**S, M, A, R, T, I**>. De componenten hiervan zijn als volgt gedefinieerd:

$S = \cup j \in$ dom (S) : S_j;

M = <MM, MR>, zodanig dat voor

$\bar{s} \in P(S), \bar{a} \in P(A), i \in$ dom (**M**), en met

$s_i = \bar{s} \cap (\cup j \in I_i : S_j)$ en $a_i = \bar{a} \cap A_i$ geldt:

$MM\ (\bar{a},\bar{s}) = \cup\, i \in \text{dom}\ (\mathbf{M}) : \mathbf{MM}_i\ (a_i, s_i)$,

$MR\ (\bar{a},\bar{s}) = \cup\, i \in \text{dom}\ (\mathbf{M}) : \mathbf{MR}_i\ (a_i, s_i)$;

$A = \cup\, i \in \text{dom}\ (\mathbf{A}) : \mathbf{A}_i$;

$R = \cup\, i \in \text{dom}\ (\mathbf{R}) : \mathbf{R}_i$;

$T = \cup\, \langle i,j \rangle \in \text{dom}\ (\mathbf{T}) : \mathbf{T}_{ik}$;

I = {*}, waarin * het element in dom (S) is;

De aggregatie van de externe invloeden van een dds is een component E, die als volgt is gedefinieerd:

E = <ET, ES, EM, EA>, met
ET = **ET**,
$ES = \cup\, j \in \text{dom}\ (\mathbf{ES}) : \mathbf{ES}_j$;
$EM = \cup\, j \in \text{dom}\ (\mathbf{EM}) : \mathbf{EM}_j$;
$EA = \cup\, i \in \text{dom}\ (\mathbf{EA}) : \mathbf{EA}_i$;

(einde definitie 5.4)

Het is met behulp van inductie eenvoudig af te leiden dat tussen een proces van de aggregatie van een dds, genoteerd als $\langle \underline{\sigma}, \underline{\alpha}, \underline{\rho} \rangle$, en het corresponderende proces van het dds dan het volgende verband bestaat:

$\underline{\sigma}\,(t) = \cup\, j \in \text{dom}\ (\mathbf{S}) : \sigma_j\ (t)$, en
$\forall\, j \in \text{dom}\ (\mathbf{S}) : \sigma_j\ (t) = \underline{\sigma}\,(t) \cap \mathbf{S}_j$;

$\underline{\alpha}\,(t) = \cup\, i \in \text{dom}\ (\mathbf{M}) : \alpha_i\ (t)$, en
$\forall\, j \in \text{dom}\ (\mathbf{M}) : \alpha_i\ (t) = \underline{\alpha}(t) \cap \mathbf{A}_i$;

$\underline{\rho}\,(t) = \cup\, i \in \text{dom}\ (\mathbf{M}) : \rho_i\ (t)$, en
$\forall\, j \in \text{dom}\ (\mathbf{M}) : \rho_i\ (t) = \underline{\rho}(t) \cap \mathbf{R}_i$;

Soortgelijke eigenschappen gelden voor de overige procesvariabelen.

Definitie 5.5

Het gedrag van een dds, gedefinieerd als een tupel <S, M, A, R, T, I> conform definitie 5.4, waaraan een component E is toegevoegd, eveneens conform definitie 5.4, is gedefinieerd als een tupel $<\iota, \varepsilon, \omega>$ waarvan de componenten zijn gedefinieerd conform definitie 3.1, zodanig dat:

$IA \subset A;$
$IS \subset S;$
$OR \subset R;$

Het verband tussen het gedrag en het proces van een dds is nu als volgt gedefinieerd (voor $t \in R$):

$\iota(t) = \{a \mid <a, \triangleright> \in EA\};$
$\varepsilon(t) = \{x \mid <x, \triangleright> \in EM\};$
$\omega(t) = \varrho(t) \cap OR;$

(einde definitie 5.5)

Door deze definitie is de gedragsfunctie B uit hoofdstuk 3 nu formeel gedefinieerd.

Voor de compositie en decompositie van een dds is het van belang te kunnen vaststellen of twee dds-en elkaar kunnen vervangen. We stellen dat zulks het geval is als de twee systemen equivalent zijn in de hieronder gedefinieerde zin.
Allereerst wordt de isomorfie van twee basis-dds-en gedefinieerd.

Definitie 5.6

Twee basis-dds-en $<S_1, M_1, A_1, R_1, T_1, I_1>$ en $<S_2, M_2, A_2, R_2, T_2, I_2>$ heten isomorf als er bijecties f_1, f_2, f_3 bestaan, die als volgt zijn gedefinieerd:

$f_1 \in S_1 \rightarrow S_2;$
$f_2 \in A_1 \rightarrow A_2;$
$f_3 \in R_1 \rightarrow R_2;$

zodanig dat $\forall\, a \subset A_1$ en $\forall\, s \subset S_1$ geldt:

$MM_2 (f_2(a), f_1(s)) = MM_1 (a, s)$
$MR_2 (f_2(a), f_1 (s)) = MR_1 (a, s)$
$T_2 = \{ < f_3 (r), f_2 (a) > | <r, a> \in T_1 \}$

(N.B. Van een functie $F \in A \rightarrow B$ is ook de uitbreiding $\bar{F} \in P(A) \rightarrow P(B)$ gedefinieerd, namelijk als volgt: $\bar{F}(a) = \{ y \mid \exists x \in a: F(x) = y \}$. Waar F staat moet men dus $\bar{F}$ lezen).

Twee bijbehorende omgevingsinvloeden E_1 en E_2 heten isomorf als:

$ET_1 = ET_2$,
$f_1 (ES_1) = ES_2$,
$EA_2 = \{ < f_2 (a), t> | <a, t> \in EA_1 \}$;
$EM_2 = \{ < f_1 (x), t> | <x, t> \in EM_1 \}$;

(einde definitie 5.6).

De aldus gedefinieerde isomorfie is een equivalentierelatie.
Indien twee basis-dds-en isomorf zijn, dan is het eenvoudig in te zien dat, bij isomorfe omgevingsinvloeden, ook de processen isomorf zijn.

Definitie 5.7

Twee dds-en heten *equivalent* indien hun aggregaties isomorf zijn.
Indien de aggregaties van de omgevingsinvloeden van de dds-en isomorf zijn, heten de daardoor bepaalde processen ook *equivalent*.

(einde definitie 5.7).

Deze relatie is een equivalentierelatie.

6. Specificatie

Onder de specificatie van een dds wordt de formele beschrijving verstaan van de componenten van het tupel <S, M, A, R, T, I> conform definitie 5.1. De verzamelingen dom (S) en dom (M) worden op de gebruikelijke wijze gespecificeerd. Meestal zal dat door middel van enumeratie gebeuren.
Een formele beschrijving van een dds is, volgens de definities in hoofdstuk 2 een symbolisch systeem, waarvan het beschreven dds een (mogelijke) realisatie is.

Om een bepaald proces te "creëren" is ook de specificatie van de omgevingsinvloeden, een component E dus, nodig. Voor zo'n component E komen verschillende bronnen in aanmerking. Eén van de mogelijkheden is dat men het model voedt met waarnemingen die worden verricht aan een reëel proces. Een variant hierop is dat men historische waarnemingen van een proces gebruikt. Ook valt er tenslotte te denken aan het genereren van omgevingsinvloeden. In het algemeen zal men van deze mogelijkheid gebruik maken bij simulatie-experimenten.

Er is in de databasewereld een toenemende belangstelling waarneembaar voor het gebruik van eerste orde logica om de semantiek van databases te beschrijven (zie bijvoorbeeld [Gallaire 1986] en [Reiter 1984]). Deze, ons inziens toe te juichen, belangstelling is mede een argument geweest voor de keuze van eerste orde logica voor de specificatie van een dds.
De definities van de concepten, die aan eerste orde talen ten grondslag liggen, en de notaties daarvan zoals in dit hoofdstuk gebruikt, zijn hoofdzakelijk ontleend aan [Lloyd 1984]. Daarin wordt overigens naast de database-interpretatie van eerste orde logica ook, en vooral, aandacht geschonken aan de procedurele interpretatie ervan, waardoor logica een effectieve programmeertaal wordt.

In dit hoofdstuk wordt een specificatietaal, SL geheten (Specification Language), gepresenteerd.
De kern van SL is de mogelijkheid tot het definiëren van een eerste orde taal uit een bepaalde klasse, die we SL-talen zullen noemen.
Een *SL-taal* is een eerste orde taal, waarvan het alfabet is gedefinieerd conform de regels, die zijn beschreven in paragraaf 6.1. Elk dds heeft zijn eigen SL-taal.
Om de component M te kunnen beschrijven is daaraan toegevoegd de mogelijkheid tot het specificeren van mutaties en responsies en van de condities waaronder die worden geproduceerd.
De specificatie van een conditie en van de daarbij te produceren mutaties en responsies heet een *motorregel.*

6.1. De definitie van een SL-taal

De definitie van een SL-taal valt in twee delen uiteen: de syntaxis en de semantiek. Syntactisch gezien bestaat een eerste orde taal uit alle zinnen, formules geheten, die met behulp van het alfabet van de taal kunnen worden geconstrueerd. Het alfabet bevat dus alle grammaticale regels.
Onder de semantiek van een eerste orde taal verstaat men de vaststelling van de waarheidswaarde van de formules. Daarvoor is het nodig dat men eerst een bepaalde context defineert waarbinnen het mogelijk is voor elke formule te beslissen of hij waar is of onwaar. We zullen de semantiek van een SL-taal slechts informeel behandelen. De formele definitie van de semantiek van eerste orde talen is onder andere beschreven in [Lloyd 1984].
Het alfabet van een SL-taal definiëren we wel formeel.

Het *alfabet* van een SL-taal bestaat uit:

- een verzameling C van constanten;
- een verzameling V van variabelen;
- een verzameling F van functiesymbolen; F bestaat uit deelverzamelingen F^n (van zogeheten n-aire functie-symbolen ($n \in N$ en $n > 0$) ;
- een verzameling P van predicaatsymbolen; P bestaat uit deelverzamelingen P^n van zogeheten n-aire predicaatsymbolen ($n \in N$);
- p^0 bevat standaard de volgende symbolen: Θ, Φ;
- P^1 bevat standaard de volgende symbolen: concreet, abstract, num, int;
- de kwantorsymbolen $\exists$, $\forall$, Σ en #;
- de logische-operator-symbolen $\vee$, $\wedge$, $\neg$, $\rightarrow$ en $\leftrightarrow$;
- de relationele-operator-symbolen $<$, $>$, $\leq$, $\geq$, $=$ en $\neq$;
- de rekenkundige-operator-symbolen +, -, *, +, mod en div;
- de interpunctie-symbolen (,), :, { , }, |, en ,

Een *term* is als volgt gedefinieerd:

- een constante is een term;
- een variabele is een term;
- als $t_1, ..., t_n$ termen zijn, en $f \in F^n$, dan is $f(t_1, ..., t_n)$ een term;
- als t_1 en t_2 termen zijn, dan zijn $(t_1 + t_2)$, $(t_1 - t_2)$, $(t_1 * t_2)$, $(t_1 + t_2)$, $(t_1$ mod $t_2)$ en $(t_1$ div $t_2)$ termen;
- als x een variabele is en Q een formule, dan is (#x : Q) een term;
- als x een variabele is, Q een formule en t een term, dan is (Σx : Q : t) een term.

Een *atoom* is als volgt gedefinieerd:

- als $p \in P^0$, dan is p een atoom;
- als $t_1, ..., t_n$ termen zijn en $p \in P^n$, dan is $p(t_1, ..., t_n)$ een atoom ($n > 0$);
- als t_1 en t_2 termen zijn, dan zijn $(t_1<t_2)$, $(t_1>t_2)$, $(t_1 \leq t_2)$, $(t_1 \geq t_2)$, $(t_1=t_2)$ en $(t_1 \neq t_2)$ atomen.

Een *formule* is als volgt gedefinieerd:

- een atoom is een formule;
- als Q en R formules zijn, dan zijn $(Q \vee R)$, $(Q \wedge R)$, $(\neg Q)$, $(Q \rightarrow R)$ en $(Q \leftrightarrow R)$ formules;
- als Q een formule is en x een variabele, dan zijn $(\exists x : Q)$ en $(\forall x : Q)$ formules.

(Einde definitie)

Om de waarheidswaarde van de formules van een SL-taal, behorend bij een dds, te kunnen vaststellen, is het nodig af te spreken naar welke dingen of zaken in het reële systeem door de symbolen van de taal wordt verwezen.
Het maken van zo'n afspraak heet in de terminologie van eerste orde talen het definiëren van de *interpretatie* van een taal ([Lloyd 1984]).
Het interpretatiedomein is in ons geval het reële systeem, waarvan het dds de conceptualisatie is.
We onderscheiden binnen een reëel systeem twee soorten objecten, concrete en abstracte. In hoofdstuk 8 wordt dit onderscheid verklaard.
Voorbeelden van concrete objecten zijn personen, auto's en orders. Voorbeelden van abstracte objecten zijn lengtematen, gewichten en temperatuurwaarden.
De interpretatie van een SL-taal kan, informeel, als volgt worden gedefinieerd:

- Een constante stelt een concreet of abstract object voor. In plaats van te zeggen dat een constante c de voorstelling is van een object, zullen we gemakshalve vaak zeggen dat c een object is.

- Een variabele is een symbool, dat in de plaats staat van een object.

- Een functiesymbool $f \in F^n$ stelt een functie $C \rightarrow C^n$ voor (C^n is het cartesisch produkt van n maal C). Een voorbeeld van een functie in F^1 is: leeftijd (dit stelt bijvoorbeeld de leeftijd van een persoon voor).

- Een predicaatsymbool $p \in P^n$ stelt een predicaat voor in C^n. Met de notatie $p(c_1, c_2, \ldots c_n)$ wordt uitgedrukt dat $<c_1, c_2, \ldots c_n>$ tot het predicaat behoort. De ariteit o is een speciaal geval; een $p \in P^0$ stelt een waarheidswaarde (waar of onwaar) voor.
  De standaardsymbolen hebben de volgende betekenis:

  Θ stelt de waarheidswaarde waar voor;
  Φ stelt de waarheidswaarde onwaar voor;
  concreet (x) betekent dat x een concreet object is;
  abstract (x) betekent dat x een abstract object is;
  num (x) betekent dat x een rationaal getal is.
  int(x) betekent dat x een geheel getal is.

De verzamelingen concrete objecten en abstracte objecten zijn disjunct. De verzameling der rationale getallen is gekozen omdat een rationaal getal een eindige representatie heeft, terwijl de verzameling toch dicht is (dat wil zeggen voor elke twee getallen a en b is er een getal c te vinden zodanig dat $a < c < b$).
Door deze eigenschap vormen de rationale getallen een deelverzameling van *R*, die geschikt is om er tijdvertragingen van responsies in uit te drukken.

De betekenis van de kwantoren en de operatoren is de gebruikelijke en wordt daarom niet nader toegelicht. Een uitzondering hierop vormen de speciale kwantoren Σ (sommatie) en # (cardinaliteit).
Een term van de vorm $(\Sigma x : Q : t)$ levert een rationaal getal op, dat gelijk is aan de som van alle waarden van de term t (waarin x als vrije variabele kan voorkomen), die men verkrijgt door voor x een constante te substitueren, zodanig dat de formule Q waar is. De voorwaarde daarbij is dat x over een eindige verzameling varieert, en dat de term getalwaardig is.
Een voorbeeld moge dit toelichten.

> Een storting op een bankrekening wordt voorgesteld door een grondatoom van het soort stort (i, x), waarin i de rekening en x het bedrag van de storting is.
> Voor een bepaalde rekening, zeg r387, is het totaalbedrag van alle stortingen dan gelijk aan $(\Sigma x : \text{stort}(r387, x) : x)$.

Een term van de vorm (#x : Q) levert een natuurlijk getal op, dat gelijk is aan het aantal constanten dat men voor x kan substitueren, zodanig dat de formule Q waar is.

> Bijvoorbeeld: het aantal stortingsopdrachten voor rekening 387 is gedefinieerd als (#x : stort (r387, x)).

Met betrekking tot de notatie van de elementen van C, V, F en P maken we de volgende afspraken.

- Een constante c , wordt genoteerd als een string van letters en cijfers, die minstens twee tekens bevat. Als de string begint met een cijfer, moet hij tenminste één letter bevatten. Zowel hoofdletters als kleine letters zijn toegestaan. Ter bevordering van de leesbaarheid mogen koppeltekens ("-") worden tussengevoegd; deze worden formeel niet tot de string gerekend.

  Voorbeelden: b12, rood, Janssen, HL-98-YD;

- Een rationaal getal, dat is een constante c waarvoor geldt num (c), wordt in afwijking van de vorige afspraak op de gebruikelijke wijze genoteerd.

  Voorbeelden: 3, 19/3, - 26;

  (N.B. Een getal kan meer dan één notatie hebben; bijvoorbeeld, de notaties 19/3 en 38/6 stellen hetzelfde getal voor).

- Een variabele wordt genoteerd als een kleine letter, eventueel voorzien van een index.

  Voorbeelden: a, i, x, y, c_1, c_2;

- Een functiesymbool en een predicaatsymbool worden genoteerd als een string van letters. Ter bevordering van de leesbaarheid mogen koppeltekens ("-") worden tussengevoegd; deze worden formeel niet tot de string gerekend.

  Voorbeelden: persoon, AUTO, leeftijd, check-in;

Een SL-taal bestaat uit alle formules, die conform de definitie van de taal kunnen worden geconstrueerd.
Onze interesse met betrekking tot het specificeren van een dds gaat vooral uit naar een speciale klasse van formules, gesloten formules geheten.
Een *gesloten formule* is een formule zonder vrije variabelen. Voorbeelden van gesloten formules zijn:

artikel (fiets);
voorraad (fiets, 33);
$\forall x : \forall y$: voorraad (x, y) $\rightarrow$ artikel (x) $\wedge$ int (y) $\wedge$ y $\geq$ 0;

De eerste twee formules stellen, informeel gesproken, elementaire feiten voor, namelijk "fiets is een artikel" en "de voorraad fietsen is 33".
Het zijn voorbeelden van grondatomen. Een *grondatoom* is een atoom zonder vrije variabelen.
De derde formule stelt een algemeen feit voor, namelijk "voor elk voorraadgegven <x, y> is het zo, dat x een artikel is en y een natuurlijk getal".

6.2 De specificatie van S, A, R, T en I

Een basis, bijvoorbeeld de toestandsbasis van een geheugen, is gedefinieerd als de verzameling van alle grondatomen, waarvan de predicaatsymbolen tot een bepaalde deelverzameling van P behoren.
Voor de toestandsbasis S_j van geheugen j noteren we die deelverzameling als PS_j. De toestandsbasis S_j is dus volledig bepaald door PS_j. De verzamelingen PS_j zijn onderling disjunct.

> Bijvoorbeeld: stel dat S de zwarte lijst voorstelt uit het voorbeeld van het postorderbedrijf; PS zou men dan kunnen definiëren als {zwart (.)}, waarbij de punt tussen haakjes aangeeft dat de ariteit van zwart 1 is. S bestaat dan uit evenzoveel grondatomen zwart (x) als er constanten in C zijn.

Op analoge wijze onderscheiden we voor elke processor i de verzameling PA_i en PR_i, die respectievelijk de actiebasis A_i en de reactiebasis R_i volledig bepalen. De verzamelingen PA_i zijn onderling disjunct, evenals de verzamelingen PR_i.

De toestandsbasis van een processor i is bepaald door een verzameling predicaatsymbolen PM_i, die als volgt is gedefinieerd:

$$PM_i = \cup\, j \in I_i : PS_j ;$$

Voor elke processor i specificeren we bovendien een, mogelijkerwijs lege, verzameling $PD_i \subset P$. Deze predicaatsymbolen worden gebruikt als *kortschriften* in de specificatie van de motor. PD_i is disjunct met PM_i, PA_i en PR_i.

Een transactiekanaal T_{ik} is een verzameling paren <r, a> waarin r een grondatoom is uit R_i en a een grondatoom uit A_k. In het algemeen is het mogelijk T_{ik} als de vereniging van een aantal deelverzamelingen te specificeren, waarbij de definitie

van elke deelverzameling de volgende vorm heeft:

$$\{ < p (x_1, \ldots x_n), q (y_1, \ldots y_m) > \mid \text{formule} \}$$

Hierin is $p \in PR_i$, $q \in PA_k$, en zijn $x_1, \ldots x_n$ en $y_1, \ldots y_m$ vrije variabelen, die gebonden worden door de formule. Een eis daarbij is dat voor elke r het aantal paren <r, a> in T_{ik} eindig is.
Een voorbeeld van een definitie van bovenstaande vorm is:

$$\{ < p (x), q (x, y) > \mid x > 0 \wedge y = 3 \}$$

Dit is dus de verzameling paren < p (x), q (x, 3) > met x > 0.

Elke I_i wordt gespecificeerd door de verzameling indices van de betreffende geheugens te definiëren. Dit gebeurt door enumeratie.

We zullen de specificatie van **S, A, R, T** en **I** toelichten aan de hand van een deel van het postorderbedrijf.

De afdeling levering wordt gemodelleerd als één processor "levering". De geheugens, waarmee de processor "levering" is verbonden zijn "leverparameters", "klantorders" en "voorraad". De indices van de geheugens zijn respectievelijk 1, 2 en 3. We kunnen dan de PSj als volgt specificeren (waarbij de ariteit is aangegeven door een aantal punten, gescheiden door komma's, tussen haakjes):

PS1 = {leverperiode (. , .)};
leverperiode (a, p) betekent bijvoorbeeld dat er elke p dagen moet worden gekeken of orders op artikel a kunnen worden geleverd;

PS2 = {klantorder (.), orderklant (. , .),
orderartikel (. , .), orderq (. , .),
orderdatum (. , .)};

de mogelijke betekenis van hiermee gevormde predicaten veronderstellen we voldoende gesuggereerd door de naamgeving, wellicht met uitzondering van orderq; orderq (r, q) stelt de gewenste hoeveelheid q van artikel r voor.

PS3 = {voorraad (. , .)};
voorraad (a, v) betekent dat er zich een hoeveelheid v van artikel a in het magazijn bevindt.

De actiebasis en de reactiebasis zijn als volgt bepaald:

PA = {lever (.), probeer (.)};
lever (a) is een actie om te kijken of orders op artikel a kunnen worden geleverd;
probeer (a) is een actie met hetzelfde effect als lever (a); het verschil blijkt bij de specificatie van **M** (zie paragraaf 6.4).

PR = {levering (. , . , .), probeer (.), lever (.)}
levering (k, a, h) stelt de levering van h stuks van artikel a aan klant k voor;
probeer (a) en lever (a) zijn terugkoppelreacties.

Als kortschriften specificeren we tevens:

PD = {volgende (., .)};
volgende (a, r) betekent dat order r de eerstvolgende order op artikel a is, die moet worden geleverd.

Het terugkoppelkanaal wordt als volgt gespecificeerd:

T = {< lever (a), lever (a) > | artikel (a) } ∪ {<probeer (a), probeer (a)> | artikel (a)};

De interconnecties van de processor, tenslotte, zijn:

I = {1, 2, 3};

Aan de specificatie van de toestandsbasis, de actiebasis en de reactiebasis van een processor i voegen we een verzameling X_i van niet-atomaire gesloten formules toe, de *axioma's* van processor i geheten. De predicaatsymbolen in deze verzameling komen uit $PM_i \cup PA_i \cup PR_i \cup PD_i$.
Een deelverzameling van de axioma's wordt gevormd door de definities van kortschriften. Een *definitie* is een formule van de vorm:

$$p\ (x_1, ..., x_n) \leftrightarrow Q$$

waarin $p \in PD_i$ en waarin Q een formule is, die predicaatsymbolen bevat uit $PM_i \cup PA_i \cup PR_i \cup PD_i$, en die ten hoogste $x_1, ..., x_n$ als vrije variabelen bevat.

(N.B. Voor deze formule en voor alle volgende formules in het boek geldt dat op elke vrije variabele universele quantificatie moet worden toegepast.)

Alle andere axioma's in X_i worden tezamen de *integriteitsregels* van processor i genoemd. De integriteitsregels representeren algemene feiten, die altijd waar (moeten) zijn. Een belangrijke praktische functie van deze axioma's is dat ze voor de specificator van de motor van processor i een leidraad zijn bij het bewaken van de integriteit van de toestand.
Men kan het ook als volgt stellen: de integriteitsregels definiëren de *toestandsruimte* van processor i, dat is de verzameling van mogelijke of toelaatbare toestanden.

Op overeenkomstige wijze worden door de integriteitsregels de actieruimte en de reactieruimte van processor i gedefinieerd. Dat zijn respectievelijk de verzameling van mogelijke of toelaatbare verzamelingen gelijktijdig optredende acties, en de verzameling van mogelijke of toelaatbare verzamelingen gelijktijdig optredende reacties.
De praktische betekenis van de actieruimte en de reactieruimte is in het algemeen echter lang niet zo groot als die van de toestandsruimte.

Tot de axioma's X_i van processor i behoren standaard de volgende formules:

$$(\text{concreet}(x) \vee \text{abstract}(x)) \wedge (\text{concreet}(x) \leftrightarrow \neg\, \text{abstract}(x));$$
$$\text{num}(x) \rightarrow \text{abstract}(x);$$
$$\text{int}(x) \rightarrow \text{num}(x);$$

De eerste formule drukt uit dat elke constante een concreet object of een abstract object voorstelt (niet beide). De tweede formule drukt uit dat een rationaal getal een abstract object is. We nemen aan dat de rationale getallen in elke specificatie al 'aanwezig' zijn, evenals een verzameling formules, die hiervan op adequate wijze de deelverzameling der gehele getallen definieert, dat is de verzameling abstracte objecten waarvoor geldt: int(x);

Voor de hierboven gepresenteerde specificatie van de processor "levering" zouden de navolgende *integriteitsregels* van toepassing kunnen zijn:

voorraad (a, v) $\rightarrow$ num (v) $\wedge$ v $\geq$ 0;

de voorraad van een artikel is een niet-negatief getal.

voorraad (a, x) $\wedge$ voorraad (a, y) $\rightarrow$ x = y;

elk artikel heeft ten hoogste één hoeveelheid als voorraad;

leverperiode (a, p) $\rightarrow$ int (v) $\wedge$ v > 0;

de leverperiode is steeds een geheel getal > 0; dit getal stelt bijvoorbeeld een aantal dagen of uren voor;

klantorder (r) $\rightarrow$ ($\exists$k: orderklant (r, k)) $\wedge$ ($\exists$a: order-artikel (r, a)) $\wedge$
($\exists$q: orderq (r, q)) $\wedge$ ($\exists$d: orderdatum (r, d));

Deze formule drukt uit dat bij een klantorder altijd een klant, een artikel, een hoeveelheid en een datum horen.

orderklant (r, k) $\rightarrow$ klantorder (r);
orderartikel (r, a) $\rightarrow$ klantorder (r);
orderq (r, q) $\rightarrow$ klantorder (r);
orderdatum (r, d) $\rightarrow$ klantorder (r);

Deze formules drukken de, triviale, eisen uit dat indien k, a, q, en d respectievelijk de klant, het artikel, de hoeveelheid en de datum van r zijn, r dan ook een klantorder moet zijn.

orderklant (r, x) $\wedge$ orderklant (r, y) $\rightarrow$ x = y;
orderartikel (r, x) $\wedge$ orderartikel (r, y) $\rightarrow$ x = y;
orderq (r, x) $\wedge$ orderq (r, y) $\rightarrow$ x = y;
orderdatum (r, x) $\wedge$ orderdatum (r, y) $\rightarrow$ x = y.

Deze formules drukken uit dat bij elke klantorder ten hoogste één klant, artikel, hoeveelheid en datum hoort; samen met de eerste formule betreffende klantorder (r) betekent dat dus dat elke klantorder precies één klant, één artikel, één hoeveelheid en één datum heeft.

De *definitie* van het kortschrift zou als volgt kunnen luiden:

$$\begin{aligned}\text{volgende}(a, r) \leftrightarrow\ & \text{order}(r) \wedge \text{orderartikel}(r, a) \wedge \text{orderq}(r, q) \wedge \\ & \text{orderdatum}(r, d) \wedge \text{voorraad}(a, v) \wedge q \leq v \wedge \\ & \forall x: (\text{order}(x) \wedge \text{orderartikel}(x, a) \wedge \text{orderq}(x, y) \wedge \\ & \text{orderdatum}(x, z) \wedge y \leq v \rightarrow z > d \vee (z = d \wedge y > q) \vee \\ & (z = d \wedge y = q \wedge x \geq r)\end{aligned}$$

Deze formule drukt uit dat de eerstvolgende te leveren order op artikel a de order met de oudste orderdatum is. Als er twee of meer orders met dezelfde (oudste) orderdatum zijn wordt uit die verzameling de order met de kleinste orderhoeveelheid gekozen. Als ook dat criterium geen uitsluitsel biedt wordt er uit de overgebleven verzameling orders met dezelfde (kleinste) orderhoeveelheid een 'willekeurige' gekozen. Willekeurig houdt in: de order met het 'kleinste' object.

6.3 De definitie van motorregels

De functies $\mathbf{MM}_i$ en $\mathbf{MR}_i$ van een processor i worden gespecificeerd met behulp van motorregels.

Een *motorregel* is een taalconstructie in SL, waarmee mutaties en responsies kunnen worden gespecificeerd als het resultaat van de activering van een processor. De mutaties en responsies worden geproduceerd als aan de conditie van de motorregel is voldaan. Deze conditie is een bewering over de actuele toestand en de ontvangen verzameling acties.

We zullen eerst de syntaxis van een motorregel definiëren. Daarbij maken we gebruik van de EBNF-notatie. (EBNF staat voor "Extended Backus-Naur Form". De extensie betreft het symbolenpaar [ ], waarmee men aangeeft dat het gedeelte tussen de symbolen er niet hoeft te zijn, en het symbolenpaar { }, waarmee wordt aangegeven dat het gedeelte tussen de symbolen meer dan één keer kan voorkomen.)

In de definitie staat "a-set" voor "verzameling atomen" en "at-set" voor "verzameling paren <atoom, vertragingstijd>".
Voorts is "ext" de afkorting van "extensioneel gedefinieerd" en "int" die van "intensioneel gedefinieerd", en staan "M" en "R" voor "mutaties" respectievelijk "responsies".

Definitie

<motorregel> : := <conditie> [<M-deel>] [<R-deel>];

<conditie> : := |= <formule> ;

$\overset{M}{\Rightarrow}$

<M-deel> : := $\Rightarrow$ <a-set> ;

$\overset{R}{\Rightarrow}$

<R-deel> : := $\Rightarrow$ <at-set> ;

<a-set> : : = <ext-a-set> | <int-a-set> | < a-set > $\cup$ < a-set >

<ext-a-set>: := {< atoom > { , < atoom > } }

<int-a-set>: := {<atoom> | <formule> }

<at-set>: := <ext-at-set> | <int-at-set> | <at-set> $\cup$ <at-set>

<ext-at-set>: := {<at-paar> {, <at-paar> } }

<int-at-set>: := {<at-paar> | <formule> }

<at-paar>: := <<atoom> , <posrat> >

De symbolen " |=", "M", "R", "$\cup$", "{", "}", "<", ">", "|" en "," zijn in deze
$\Rightarrow$ $\Rightarrow$

definitie non-terminals.
De terminals <formule>, <atoom> en <posrat> zijn respectievelijk een formule, een atoom en een positief rationaal getal of "*" (dit symbool wordt gebruikt om een onbepaald positief rationaal getal aan te duiden).

(Einde definitie)

In de formule van de conditie, in de a-set van het M-deel en in de at-set van het R-deel van een motorregel mogen vrije variabelen voorkomen. De voorwaarde die daarbij hoort is, dat de vrije variabelen van de a-set en de at-set ook voorkomen in de formule van de conditie.
Een voorbeeld moge dit verduidelijken:

$\models$ lever(a) Λ leverperiode (a, p)

R
$\Rightarrow$ { < lever (a), p > }

De vrije variabelen a en p in het R-deel komen ook voor in de conditie. Ze worden gebonden door een binding van de variabelen in de conditie, zoals hieronder wordt verklaard.

(N.B. het M-deel is hier dus blijkbaar leeg).

Op het moment van activering van een processor i is er een bepaalde verzameling acties en een bepaalde toestand, die respectievelijk een deelverzameling zijn van de actiebasis en de toestandsbasis van de processor. De vereniging van de acties, de toestand en de axioma's noemen we de *referentieset*. De verzameling van de constanten, die voorkomen in de referentieset heet het *actieve domein* van de motor op dat moment. Deze verzameling is eindig.
De semantiek van een motorregel kan nu als volgt worden gedefinieerd.

Een binding van de vrije variabelen in de formule van de conditie bestaat uit de substitutie van die variabelen door constanten uit het actieve domein.
Voor elke binding van de vrije variabelen in de formule van de conditie wordt nagegaan of de formule waar is met betrekking tot de referentieset.
De formule is waar als hij een logische consequentie is van de referentieset. Daarbij is de volgende regel van toepassing:

De waarheidswaarde van elk grondatoom uit $\mathbf{A_i}$ of $\mathbf{MS_i}$ is waar als het grondatoom tot de referentieset behoort. Anders is de waarheidswaarde onwaar. Deze regel staat bekend als de "closed world assumption" ([Reiter 1978]).

Als de formule waar is worden de vrije variabelen in de a-set van het M-deel en de at-set van het R-deel vervangen door de constanten, waarmee die variabelen in de binding van de conditie zijn gebonden.
De quantificaties over gebonden variabelen in gesloten formules en in intensioneel gedefinieerde verzamelingen atomen en paren <atoom, posrat> worden daarbij ook berekend met betrekking tot het actieve domein. Deze quantificaties zijn dus berekenbaar.

De berekende a-set is een eindige verzameling grondatomen en de berekende at-set is een eindige verzameling paren <grondatoom, posrat>.

Laat nu r een motorregel zijn, laat b een bepaalde binding van de vrije variabelen in de conditie van r zijn, en laat $M_{r,b}$ en $R_{r,b}$ de berekende verzamelingen grondatomen van het M-deel respectievelijk het R-deel voorstellen.

Dan is de evaluatie van de motorfuncties, bij gegeven acties a en toestand s, als volgt bepaald:

$$MM(a,s) = \Delta r : (\cup_b : M_{r,b})$$

$$MR(a,s) = \cup r : (\cup_b : R_{r,b})$$

Hier staat dus dat voor elke motorregel van een motor eerst de vereniging van $M_{r,b}$ respectievelijk $R_{r,b}$ wordt genomen over alle bindingen. Het resultaat daarvan is een verzameling mutaties M_r en een verzameling responsies R_r. Vervolgens worden de verzamelingen M_r verschild en de verzamelingen R_r verenigd over alle motorregels.

Dat op de verzamelingen M_r de verschiloperatie van toepassing is, is als volgt in te zien.
Elke motorregel is op te vatten als een aparte processor; deze processoren opereren echter allemaal op dezelfde verzameling geheugens. Uit de definities in hoofdstuk 5 (met name van de functie μ in definitie 5.2) volgt dat de mutaties voor een bepaald geheugen, geproduceerd door verschillende processoren, moeten worden verschild.

Het is meestal gemakkelijk bij het specificeren van het M-deel een onderscheid te maken tussen de grondatomen, die moeten worden verwijderd en de grondatomen, die moeten worden toegevoegd.
Daartoe vervangen we in de definities van een motorregel [<M-deel>] door [<verwijderingen>] [<toevoegingen>] met:

<verwijderingen> : : = $\bar{\Rightarrow}$ <a-set>

<toevoegingen> : : = $\overset{+}{\Rightarrow}$ <a-set>

en zodanig dat de twee <a-set>'s disjunct zijn.

Laat nu $M^-_{r,b}$ de berekende verwijderingen voorstellen en $M^+_{r,b}$ de berekende toevoegingen. Dan geldt:

$$MM(a, s) = (\Delta r : M^-_r) \Delta (\Delta r : M^+_r), \text{ waarin}$$

$$M^-_r = \cup b: M^-_{r,b} \text{ en } M^+_r = \cup b : M^+_{r,b};$$

Deze definite is equivalent aan de eerder gepresenteerde. Namelijk, uit de eigenschappen van de verschiloperatie volgt:

$$MM(a,s) = \Delta r : (M_r^- \Delta M_r^+)$$

Omdat $M_r^- \cap M_r^+ = \emptyset$ geldt: $M_r^- \Delta M_r^+ = M_r^- \cup M_r^+ = M_r$

en dus:

$$MM(a,s) = \Delta r : M_r;$$

6.4 De specificatie van M

De specificatie van de motor M_i van een processor i bestaat uit de specificatie van een aantal motorregels. De specificator draagt daarbij twee verantwoordelijkheden. De eerste is dat hij ervoor dient te zorgen dat door de gespecificeerde condities alle mogelijke combinaties van een toelaatbare toestand en een toelaatbare verzameling acties zijn afgedekt.
De tweede verantwoordelijkheid is, dat er alleen toelaatbare nieuwe toestanden en toelaatbare verzamelingen reacties worden geproduceerd.
Zoals we al hebben gezien zijn een toestand, een verzameling acties en een verzameling reacties toelaatbaar als ze behoren tot respectievelijk de toestandsruimte, de actieruimte en de reactieruimte.

Ter illustratie van de syntaxis en de semantiek van de specificatie van een component M, geven we hieronder de specificatie van de motor van de processor "levering" van het postorderbedrijf.

$\models$ (lever (a) $\vee$ probeer (a)) $\wedge$ volgende (a, r) $\wedge$
orderklant (r, k) $\wedge$ orderartikel (r, a) $\wedge$
orderq (r, q) $\wedge$ orderdatum (r, d) $\wedge$ voorraad (a, v) $\wedge$ q < v

$\overset{-}{\Rightarrow}$ {klantorder (r), orderklant (r, k), orderartikel (r, a),
orderq (r, q), orderdatum (r, d), voorraad (a, v)}

$\overset{+}{\Rightarrow}$ {voorraad (a, v - q)}

R
$\Rightarrow$ {<levering (k, a, q), * >, <probeer (a), *>}

$\models$ lever(a) $\wedge$ leverperiode (a, p)

R
$\Rightarrow$ {<lever (a), p>}

Hierbij hoort de volgende toelichting:

- De specificatie bestaat uit twee motorregels. In de tweede regel is het M-deel leeg. De twee regels worden tegelijkertijd geëvalueerd indien er zich in de referentieset een actie "lever (a)" bevindt.
- Voor elk artikel, waarvoor er zich in de referentieset een actie "lever (a)" of een actie "probeer (a)" bevindt en waarvoor er nog een order te leveren is ("volgende (a,r)") wordt de eerste motorregel uitgevoerd. (N.B. voor elke a is er maar één "volgende (a, r)" en één "voorraad (a, v)"; evenzo is er voor elke r maar één "orderklant (r, k)", "orderartikel (r, a)", "orderq (r, q)" en "orderdatum (r, d)").
- "volgende (a, r)" is een kortschrift, zoals eerder gedefinieerd (zie paragraaf 6.2).
- Bij uitvoering van de eerste regel voor order r worden alle gegevens over r uit het klantorderbestand verwijderd en wordt de voorraad bijgewerkt. Er worden twee responsies geproduceerd. De eerste is de levering van de order. De vertragingstijd daarvan is onbepaald (aangegeven door het symbool "*"); de reden voor het niet specificeren van de vertragingstijd is bijvoorbeeld dat die tijd niet bekend is. De andere responsie betreft een reactie probeer (a)", dat door het terugkoppelkanaal wordt omgezet in een actie "probeer (a)". De vertragingstijd hierbij is ook onbepaald.

- Het leveren van orders is als volgt georganiseerd. Door een actie "lever (a)" wordt de processor periodiek geactiveerd. Per activering wordt er ten hoogste één order afgewerkt. De bedoeling van probeer (a) is om na elke geslaagde levering zo snel mogelijk te kijken of er nog een order kan worden geleverd.
- De tweede motorregel wordt periodiek, voor elk artikel met een eigen periode, uitgevoerd. Er is geen verandering van de toestand.
  De responsie van de tweede regel is een reactie "lever (a)", die door het terugkoppelkanaal wordt omgezet in een actie "lever(a)". Deze actie wordt een tijd p later ontvangen.

De in dit hoofdstuk gepresenteerde specificatietaal SL is niet de enige taal, die men voor de specificatie van een dds zou kunnen gebruiken. Elke andere specificatietaal is goed, mits de semantiek daarvan maar (even) formeel is gedefinieerd.

Deel III: Praxis

In dit deel worden praktische aspecten besproken van de toepassing van het metamodel en de specificatietaal uit deel II. In de constructie van een dds onderscheiden we twee activiteiten: modelleren en specificeren.

Onder modelleren wordt het ontwerp van een netwerkvoorstelling verstaan, zoals besproken in hoofdstuk 4. Dat wil dus zeggen, het benoemen van de processoren, de geheugens, de interconnecties en de transactiekanalen.
Formeel gesproken komt dat neer op de definitie van het domein van **S**, het domein van **M**, de verzamelingen $\mathbf{I}_i$ voor elke processor i, en de verzameling paren <i,k>, waarvoor geldt dat $\mathbf{T}_{ik}$ niet leeg is.
Ten behoeve hiervan wordt een schematechniek, het zogeheten N-schema, gepresenteerd. Ook wordt er aandacht geschonken aan de hiërarchische compositie en decompositie van netwerken. Dit gebeurt in hoofdstuk 7.

Het onderwerp van hoofdstuk 8 is het specificeren van een dds. Formeel gesproken betreft dat de specificatie van **S**, **A**, **R**, **T** en **M**.
De nadruk ligt echter op de specificatie van de toestandsruimte van een processor i, dat is dus de deelverzameling van $P(\mathbf{MS}_i)$, die alle mogelijke of toelaatbare toestanden bevat.
Om de betekenis, in het reële systeem, van een toestand en een toestandsruimte, gemakkelijker te kunnen begrijpen wordt een metamodel gedefinieerd voor het beschrijven van een toestandsruimte in meer intuïtieve begrippen zoals entiteiten en associaties. Tevens wordt er een grafische notatiewijze voor de meest voorkomende soorten integriteitsregels gepresenteerd, het S-schema geheten.

In hoofdstuk 9 wordt de praktische bruikbaarheid van de theorie van deel II en de hulpmiddelen uit hoofdstuk 7 en 8 gedemonstreerd aan de hand van het voorbeeld van het postorderbedrijf, waarvan al eerder gedeelten zijn besproken.

7. Het modelleren van systemen

Onder het modelleren van een systeem verstaan we het concipiëren van de structuur van een dds, dat is het netwerk van de onderscheiden processoren, geheugens, transactiekanalen en interconnecties. Het modelleren van een systeem gaat vooraf aan het specificeren. Met specificeren wordt, zoals we hebben gezien, de precieze beschrijving van elke component bedoeld. Bij het modelleren gaan we wat deze beschrijving betreft niet verder dan het benoemen van elke component. Eventueel wordt door middel van commentaar globaal aangegeven wat de functie van de component in het netwerk is.
De structuur wordt vastgelegd in een zogeheten N-schema, dat in paragraaf 7.1 wordt behandeld. Een N-schema is een structuurschema, te vergelijken met bijvoorbeeld een A-schema van de ISAC-methodiek [Lundeberg, Goldkuhl, Nilsson 1979] of een "Data Flow Diagram" uit de SA/SD-school [Gane, Sarson 1979].

Een belangrijke techniek bij het modelleren van systemen is de hiërarchische compositie/decompositie. Door deze techniek is het mogelijk willekeurig omvangrijke systemen in elke mate van detail te modelleren, terwijl toch de samenhang van een deel met het geheel duidelijk en eenvoudig kan worden aangegeven. Deze techniek komt in paragraaf 7.2 aan de orde.

Tenslotte wordt in paragraaf 7.3 een classificatie van systeemtypen gepresenteerd. De bedoeling hiervan is vooral de praktische bruikbaarheid van de in dit boek onderscheiden vormen van wederzijdse beïnvloeding tussen een systeem en zijn omgeving te illustreren.

7.1 Het N-schema

Een N-schema (Network-schema) is een grafische voorstelling van de processoren, geheugens, transactiekanalen en interconnecties waaruit een dds is opgebouwd. Formeel gesproken betreft dat de definitie van de volgende aspecten van een tupel **<S, M, A, R, T, I>**:

- dom (**M**): dit zijn de processoren; de indices worden genoteerd conform de hieronder volgende notatieafspraak;
- dom (**S**): dit zijn de geheugens; de indices worden genoteerd conform de hieronder volgende notatieafspraak;

- $\{<i, k> | i, k \in dom(\mathbf{T}) \wedge \mathbf{T}_{ik} \neq \emptyset\}$;
  dit zijn de transactiekanalen; het transactiekanaal van processor i naar processor k wordt alleen getekend als $\mathbf{T}_{ik} \neq \emptyset$ (zie ook definitie 5.1).
- $\{<i, j> | i \in dom(\mathbf{M}) \wedge j \in dom(\mathbf{S}) \wedge j \in \mathbf{I}_i\}$;
  dit zijn de interconnecties.

Notatie-afspraak

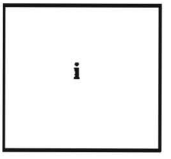
Een processor respectievelijk geheugen wordt aangeduid door een nummer, dat in BNF als volgt is gedefinieerd:

<nummer> : : = <nat. getal> | <nummer> . <nat. getal>

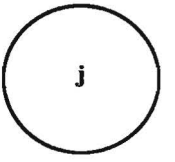
Door deze constructiewijze van de aanduiding van processoren en geheugens is het mogelijk aan te geven van welke aggregaat-processor een processor een deel is, en van welk aggregaat-geheugen een geheugen een deel is.
Indien van een processor of geheugen de index <nummer>.<nat. getal> is, is <nummer> de index van de aggregaatprocessor respectievelijk het aggregaat-geheugen, en is <nat. getal> een lokale suffix.

De symbolenset

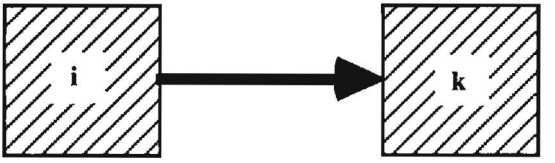
De elementaire symbolen van een N-schema zijn:



dit stelt processor i voor



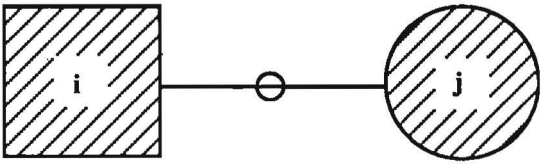
dit stelt geheugen j voor



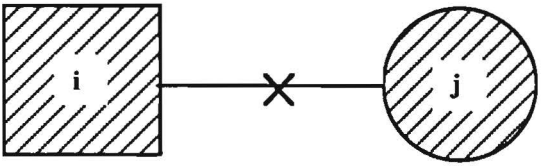
dit stelt het transactiekanaal van processor i naar processor k voor

dit stelt de interconnectie voor tussen processor i en geheugen j

De interconnecties kunnen als volgt nader worden gepreciseerd:

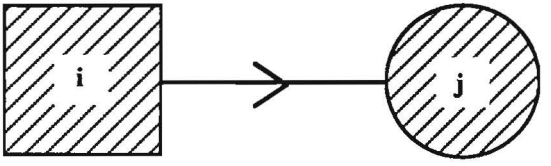


processor i mag geheugen j alleen *inspecteren*

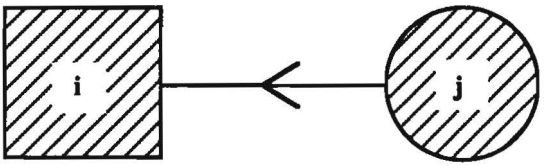


processor i mag geheugen j inspecteren en *muteren*

Het mutatierecht kan weer als volgt worden gepreciseerd:



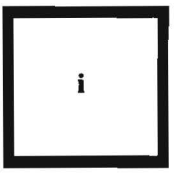
processor i mag alleen elementen *toevoegen* aan geheugen j



processor i mag alleen elementen *verwijderen* uit geheugen j

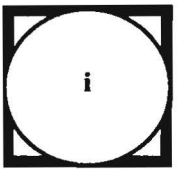
Van deze elementaire symbolen zijn de volgende samengestelde symbolen afgeleid:

1.



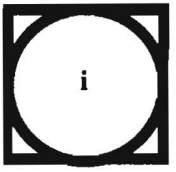
processor i met transactiekanaal <i, i>, het *terugkoppelkanaal* van i geheten; dit wordt uitgebeeld door de vet gedrukte rand van het processor-symbool.

2.



deel-dds i; dit is een *processor* met *privé-geheugen* en *terugkoppelkanaal*; de processor heeft ook interconnecties met tenminste één ander geheugen; er is geen andere processor, die toegang heeft tot het privé-geheugen.

3.

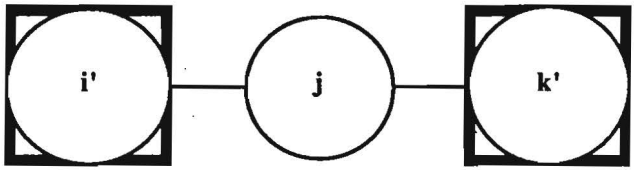


basis-dds i; dit bestaat uit een *processor*, een *geheugen* en een *terugkoppelkanaal*; de processor heeft geen interconnecties met andere processoren; wel kunnen er interconnecties zijn tussen het geheugen en andere processoren.

Voor symbool (1) geldt dat het terugkoppelkanaal ook feitelijk bestaat, dat wil dus zeggen: $T_{ii} \neq \emptyset$. Voor de symbolen (2) en (3) is het toegestaan dat zulks niet het geval is. De reden hiervoor is dat deze symbolen bij decompositie van een netwerk aggregaties of deel-aggregaties voorstellen van netwerken, die in het algemeen (nog) niet zijn gespecificeerd en waarvan dus de component T (nog) niet bekend is.

Het gebruik van symbool (2) kan als volgt worden toegelicht.
Stel dat op een bepaald aggregatieniveau twee basis-dds-en worden onderscheiden, i en k, waartussen communicatie bestaat. Deze dds-en worden als volgt gepresenteerd.

Om preciezer aan te geven waaruit de communicatie tussen deze dds-en bestaat, kan het gemeenschappelijke geheugen uit de twee dds-en worden 'uitgelicht' op de volgende wijze:



In plaats van symbool (3) wordt nu dus symbool (2) gebruikt. Het basis-dds i bestaat uit het deel-dds i en het gemeenschappelijk geheugen j, en het basis-dds k bestaat uit het deel-dds k en het gemeenschappelijk geheugen j.

Om tot uitdrukking te brengen dat in symbool (3) het gehele (aggregaat-) geheugen wordt uitgebeeld is het geheugensymbool vet gedrukt.

De in schema-symbolen toegevoegde woorden zijn puur commentaar. Ze beogen de functie van de component zo goed mogelijk toe te lichten. De woorden bij een invoerkanaal zijn predicaatsymbolen uit de verzameling PA van de betreffende processor. De corresponderende acties 'bereiken' de processor via dat kanaal.
De woorden bij een uitvoerkanaal zijn predicaatsymbolen uit de verzameling PR van de betreffende processor. De corresponderende reacties 'verlaten' de processor

via dat kanaal.
Indien een transactiekanaal geheel binnen de systeemgrens ligt worden alleen de actie-predicaatsymbolen vermeld. Dat geldt dus ook voor elk terugkoppelkanaal.

De systeemgrens

De systeemgrens van een dds wordt in een N-schema voorgesteld door een gesloten kromme, die precies alle componenten van het dds-netwerk omsluit. In het algemeen wordt de systeemgrens in de vorm van een rechthoek getekend.
De invoerkanalen, de uitvoerkanalen en de interconnecties van processoren buiten het netwerk met geheugens erbinnen zijn formeel geen onderdeel van het uitgebeelde dds.
Kennis van deze 'verbindingen' met de omgeving kan echter van belang zijn voor het inzicht in de functie van het dds ten opzichte van zijn omgeving. Waar die kennis aanwezig is, zullen we daarvan dan ook gebruik maken door de invoer- en uitvoerkanalen en de interconnecties met processoren in de omgeving wel te tekenen.

Als een voorbeeld van een N-schema is in figuur 7.1 het N-schema van het postorderbedrijf getekend. In hoofdstuk 9 wordt de modellering en specificatie van dit dds volledig behandeld.

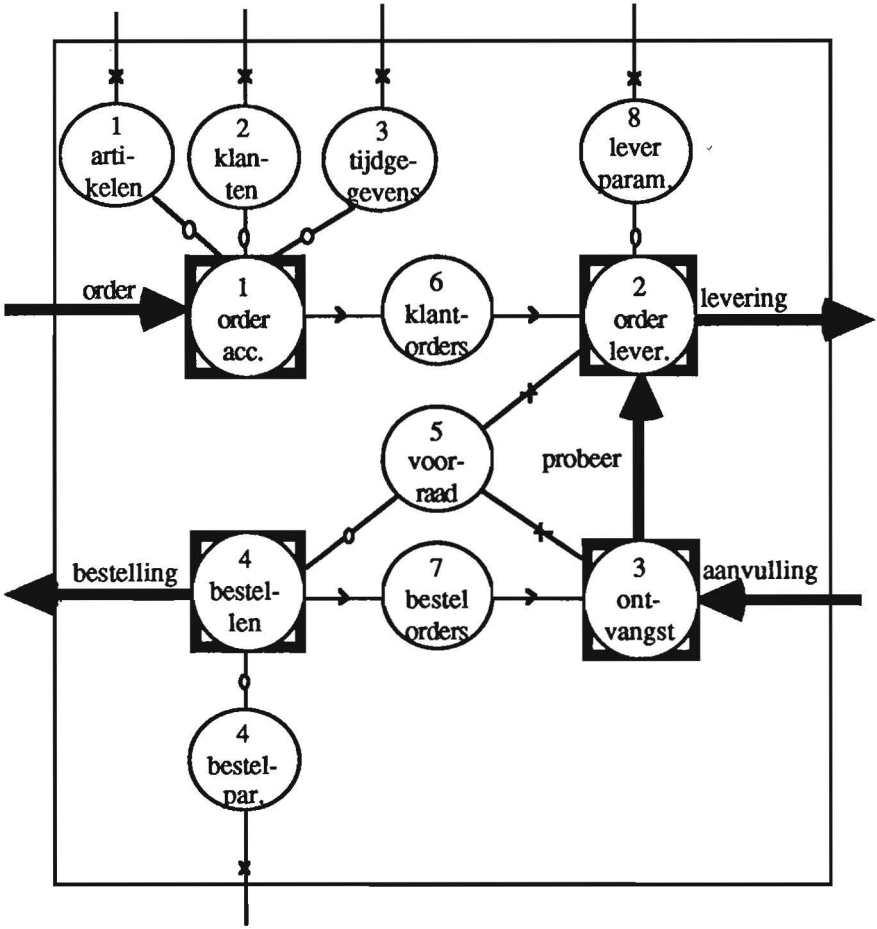
Het netwerk is een decompositie van een basis-netwerk. De indices van de processoren en geheugens zijn lokale suffixen. Er zijn vier deelsystemen te onderscheiden.

Het eerste deelsysteem bestaat uit deel-dds 1 en de geheugens 1, 2, 3 en 6. Er is een invoerkanaal vanuit de omgeving van het dds. Processor 1 mag de geheugens 1, 2 en 3 slechts inspecteren. Aan geheugen 6 mag hij elementen toevoegen.

Het tweede deelsysteem bestaat uit deel-dds 2 en de geheugens 5, 6 en 8. Er is een invoerkanaal vanuit processor 3 en een uitvoerkanaal naar de omgeving van het dds. Processor 2 mag geheugen 5 muteren, mag elementen uit geheugen 6 verwijderen en mag geheugen 8 alleen inspecteren.

Het derde deelsysteem bestaat uit deel-dds 3 en de geheugens 5 en 7. Er is een invoerkanaal vanuit de omgeving van het dds, en een uitvoerkanaal naar processor 2. Processor 3 mag geheugen 5 muteren, en mag elementen verwijderen uit geheugen 7.

Het vierde deelsysteem bestaat uit deel-dds 4 en de geheugens 4, 5 en 7. Er is een uitvoerkanaal naar de omgeving van het dds. Processor 4 mag de geheugens 4 en 5 alleen inspecteren, en mag aan geheugen 7 elementen toevoegen.



Figuur 7.1: N-schema van het postorderbedrijf

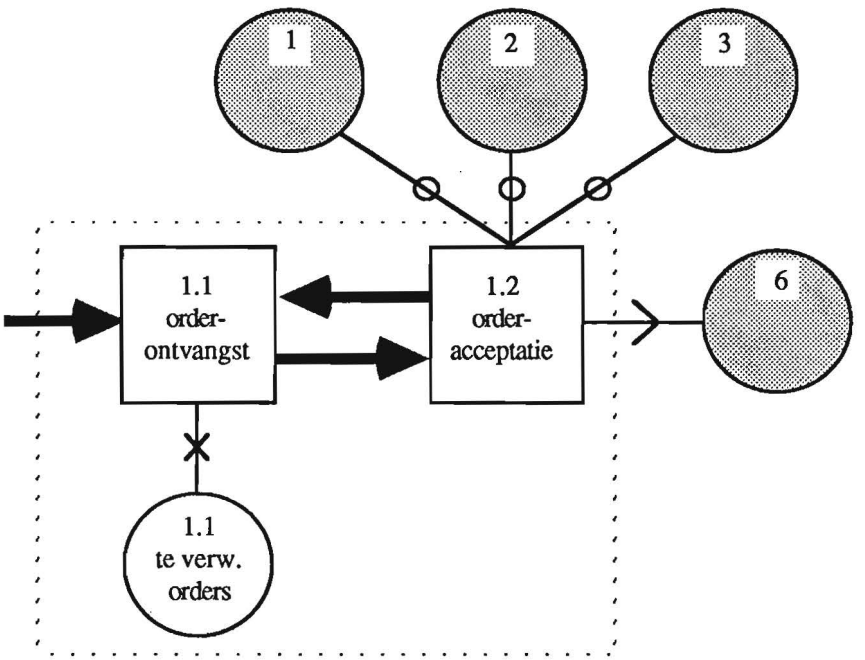
Er is sprake van communicatie tussen het eerste en het tweede deelsysteem, tussen het derde en het vierde, en tussen het tweede, het derde en het vierde onderling.

Er is sprake van (eenzijdige) interactie tussen het tweede en het derde deelsysteem.

Het dds als geheel kent interactie én communicatie met zijn omgeving. De omgeving heeft mutatierecht op de geheugens 1, 2, 3, 4 en 8.

7.2 Hiërarchische compositie/decompositie

In deel II is aangetoond dat voor elk dds een basis-dds kan worden gedefinieerd die de aggregatie is van het dds, en bovendien dat die twee modellen equivalent zijn. Deze eigenschap van een dds is de basis voor de techniek die bekend staat onder de naam hiërarchische compositie/decompositie.
Bij *hiërarchische compositie* is de procedure dat men een (deel-) systeem vervangt door zijn aggregatie. Door het herhaald toepassen van deze procedure ontstaat een hiërarchie of 'boom' van systemen. De bovengrens is daarbij de voorstelling van het gehele systeem als een basis-dds.
Bij *hiërarchische decompositie* is de procedure dat men een basis-dds (als deel van een netwerk) vervangt door een netwerk waarvan dat basis-dds de aggregatie is. Door het herhaald toepassen van deze procedure bouwt men ook een 'boom' op, maar nu in de andere richting.
Naarmate men 'stijgt' in de hiërarchische boom wordt een model *globaler* en naarmate men 'afdaalt' wordt het *gedetailleerder*.

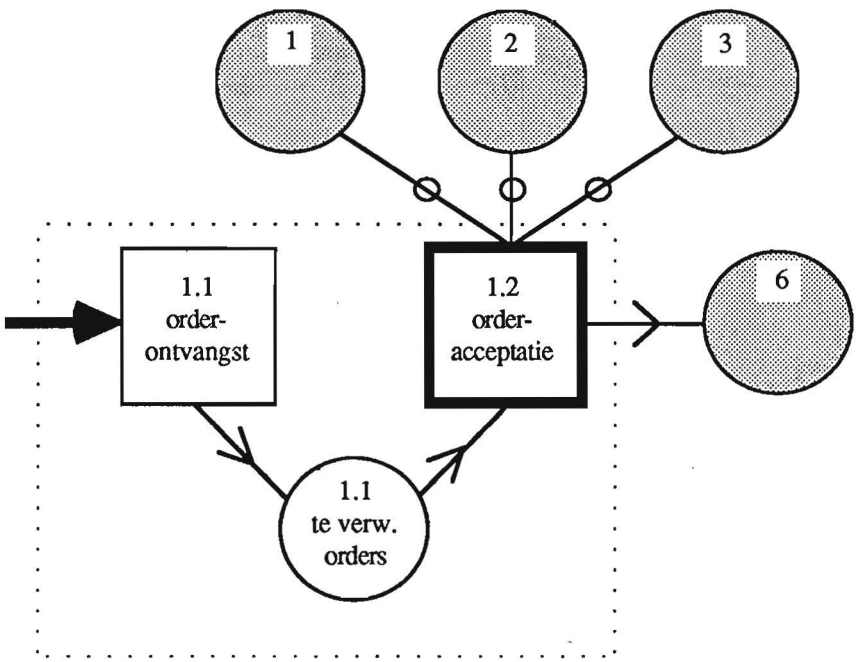


Figuur 7.2: Decompositie van een deel-dds

We zullen de techniek van hiërarchische compositie/decompositie demonstreren aan de hand van figuur 7.1.
Als voorbeeld van een decompositie beschouwen we de, mogelijke, vervanging van deel-dds 1, zoals getekend in figuur 7.2.

Processor 1.1 ontvangt de stapels orders, die door de postdienst worden aangeleverd en bewaart deze in geheugen 1.1.
Processor 1.2 is de eigenlijke acceptatie-processor, waarvan de werking als volgt is.
Na ontvangst van een te verwerken order via het transactiekanaal van processor 1.1 naar processor 1.2 wordt de acceptatieprocedure op die order toegepast. Zodra dat is gedaan stuurt processor 1.2 een verzoek aan processor 1.1 om een volgende te verwerken order. Dat gebeurt via het transactiekanaal van processor 1.2 naar processor 1.1.
Als bij ontvangst van een stapel orders door processor 1.1 het geheugen 1.1 leeg is wordt processor 1.2 van de nieuwe lading op de hoogte gesteld.

Een alternatieve oplossing is getekend in figuur 7.3.



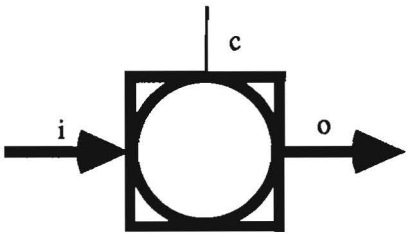
Figuur 7.3: Alternatieve decompositie van een deel-dds

De functie van processor 1.1 is nu alleen nog het ontvangen van orders en het opbergen ervan in het geheugen 1.1. Indien bij ontvangst van een stapel orders geheugen 1.1 leeg is, stuurt processor 1.1 een activeringssignaal naar processor 1.2. Processor 1.2 kijkt na elke verwerking van een order of er zich nog een te verwerken order in geheugen 1.1 bevindt. Als dat zo is, wordt er een order uitgehaald en verwerkt.
Deze modellering past het best bij de situatiebeschrijving van de afdeling orderacceptatie (zie hoofdstuk 4). Processor 1.2 is dan op te vatten als de aggregatie van de individuele medewerkers en geheugen 1.1 als de aggregatie van de bakjes met te verwerken orders op elk bureau. De medewerkers kijken zelf regelmatig in de bakjes om te zien of er nog orders zijn.

(N.B. De indicering van de processoren en geheugens in de figuren 7.1, 7.2 en 7.3 is niet geheel conform de afspraken in paragraaf 7.1. Geheugen 1.1 bijvoorbeeld is geen deel van geheugen 1. Misverstanden zijn echter uitgesloten door de toegevoegde commentaren).

7.3 Een classificatie van systeemtypen

Een discreet dynamisch systeem heeft in principe invoerkanalen, uitvoerkanalen en interconnecties met (processoren in) de omgeving. Niet al deze 'verbindingen' met de omgeving hoeven echter altijd aanwezig te zijn. Door het al of niet opnemen van de drie soorten 'verbindingen', te weten invoer, uitvoer en communicatie, ontstaan acht verschillende systeemtypen.
Deze typen worden hieronder kort besproken. Als voorstelling van een systeem gebruiken we het N-schema van het basis-dds (figuur 7.4).



Figuur 7.4: N-schema van een basis-dds

1. Een *ioc-systeem* is een systeem met invoer (i), uitvoer (o) en communicatie (c).

   Een ioc-systeem is het meest complete systeemtype. Een voorbeeld is het tweede deelsysteem in figuur 7.1.

2. Een *io-systeem* is een systeem met alleen invoer en uitvoer.

   Voorbeelden van een io-systeem zijn diverse automaten, zoals "realtime" procesregelaars en netwerkprocessoren in datanetwerken. Een voorwaarde daarbij is dat tijdens het operationeel zijn van het systeem geen 'parameters' via communicatie kunnen worden veranderd. Anders zou het een ioc-systeem zijn.

3. Een *ic-systeem* is een systeem met alleen invoer en communicatie.

   Door de in dit boek gekozen definities van interactie en communicatie vallen veel systemen in deze klasse, namelijk alle systemen, waarbij de 'reactie' geen stimulus is maar een extern 'zichtbare' toestandselement.
   Een zakrekenmachine is een voorbeeld van dit type systeem: de 'uitvoer' blijft zichtbaar op het schermpje; men kan erop kijken indien men wil en wanneer men wil.

4. Een *oc-systeem* is een systeem met alleen uitvoer en communicatie.

   Een voorwaarde voor het functioneren van een oc-systeem is dat het terugkoppelkanaal niet optioneel is, anders zouden er geen acties kunnen worden ontvangen. Voorbeelden van dit systeemtype zijn digitale monitorsystemen met een alarmfunctie, bijvoorbeeld een digitale klokschakelaar. Deze vergelijkt periodiek twee waarden en produceert onder bepaalde voorwaarden een reactie.
   Een ander voorbeeld is een handelaar in effecten, die toegang heeft tot de beursnoteringen en soms op grond daarvan middels zelf-activering iets onderneemt.

(N.B. Strikt genomen kan een systeem zonder invoer geen proces hebben. We zullen daarom bij deze soorten systemen veronderstellen dat er ooit een startsignaal is gegeven.)

5. Een *i-systeem* is een systeem met alleen invoer.

   Systemen van dit type worden meestal als weinig zinvol beschouwd. Voorbeelden zijn een koffie-automaat die nog wel munten accepteert maar

op geen enkel commando reageert, een bodemloze put, en sommige informatiesystemen.

6. Een *o-systeem* is een systeem met alleen uitvoer.

   Voorbeelden van o-systemen zijn pulsgeneratoren en mensen op praatstoelen.

7. Een *c-systeem* is een systeem met alleen communicatie.

   Tot deze klasse behoort een groot aantal systemen, namelijk alle systemen, die zichzelf activeren en die met andere systemen communiceren (via buffers).
   Een voorbeeld is het dds bestaande uit processor 1.2 en geheugens 1, 2, 3, 6 en 1.1, zoals getekend in figuur 7.3.

8. Een "niets"-systeem is een systeem zonder invoer, uitvoer of communicatie.
   Dit is een gesloten systeem, en dus niet discreet dynamisch tenzij men de eerder gemaakte uitzondering voor invoerloze systemen ook hier van toepassing verklaart. In dat geval is er een startactie, waarna periodieke zelfactivering volgt.

7.4 Het modelleren van eindige verwerkingscapaciteiten

Het instantaan produceren van mutaties en responsies door een processor betekent in de werkelijkheid dat men een oneindige verwerkingscapactieit veronderstelt. Een dds is daardoor vaak een niet-realistisch model.
Indien men uitsluitend in de 'logica' van de verwerking is geïnteresseerd, is dat geen probleem. Soms echter wil men juist laten uitkomen dat verwerkingscapaciteiten eindig zijn. In zulke gevallen dient men 'buffers' in het model toe te voegen.
We zullen dit toelichten aan de hand van het eerste deelsysteem van het postorderbedrijf (deel-dds 1 in figuur 7.1).
De specificatie van de motor van processor 1 wordt gegeven in hoofdstuk 9.

De verwerking van een binnengekomen stapel orders vindt in dit model instantaan plaats. Voor de processor is de tijd tussen twee activeringen dus zuivere wachttijd.

Een realistischer model van deze situatie zou een model zijn dat uitgaat van een beperkte verwerkingscapaciteit van processor 1.
Stel bijvoorbeeld dat er voor de acceptatie van orders één persoon beschikbaar is en dat de verwerking van elke order een bepaalde tijd (>0) duurt.

Een mogelijke modellering van deze situatie is de in figuur 7.3 geschetste.
Processor 1.1 ontvangt af en toe een stapel orders en bergt die op in geheugen 1.1.
Als processor 1.2 een activeringssignaal ontvangt kijkt hij of er een volgende te verwerken order is. Als dat zo is produceert de processor een klaar-signaal dat hij via het terugkoppelsignaal een bepaalde tijd later ontvangt. Deze tijd stelt de verwerkingstijd voor. Pas bij activering door het klaar-signaal wordt een geaccepteerde order opgeborgen in geheugen 6. Deze activering heeft weer een activeringssignaal enige tijd later tot gevolg. Die tijd stelt de wachttijd tussen twee verwerkingen voor.
Als bij ontvangst van een activeringssignaal geheugen 1.1 leeg is, wordt er een activeringssignaal geproduceerd, dat enige tijd later wordt ontvangen.
Voor processor 1.1 blijft wel de veronderstelling van een oneindige verwerkingscapaciteit gelden. Men bedenke daarbij echter dat een verzameling gebeurtenissen met exact hetzelfde gebeurtenistijdstip ook een niet realistische veronderstelling is. Dat geldt eveneens voor het instantaan opbergen in geheugen 1.1.

8. Het specificeren van een systeem

In dit hoofdstuk wordt aandacht geschonken aan enkele praktische aspecten bij het specificeren van een dds. We zullen ons daarbij concentreren op de specificatie van de toestandsruimte van een processor.
Een toestand is een verzameling grondatomen van de SL-taal, waarmee het dds is gespecificeerd. Elk grondatoom representeert een elementair feit in het reële systeem. De bij een processor behorende axioma's representeren algemene feiten in het reële systeem.
Voor een goed begrip van de betekenis van beide soorten van feiten heeft men in het algemeen behoefte aan een uitleg van die betekenis in meer intuïtieve begrippen zoals entiteiten, associaties en attributen. De uit de database-wereld afkomstige semantische of conceptuele datamodellen zijn voorbeelden van metamodellen, die beogen aan die behoefte te voldoen.
In dit hoofdstuk wordt een vergelijkbaar metamodel gepresenteerd, het *intuïtieve toestandsmodel* geheten. Na een informele uiteenzetting in paragraaf 8.1 wordt het model in paragraaf 8.2 geformaliseerd, zodat het verband met formules in de SL-taal precies en helder kan worden gedefinieerd.
In paragraaf 8.3 wordt een schematechniek besproken waarmee sommige soorten axioma's grafisch kunnen worden uitgedrukt.

8.1 Het intuïtieve toestandsmodel

Zoals we in hoofdstuk 2 hebben gezien bestaat iemands kennis uit een netwerk van concepten en conceptuele relaties. We hebben daarin twee niveau's onderscheiden: generieke concepten en individuele concepten.
Een generiek concept werkt in letterlijke zin als een scheppend concept bij het waarnemen van de wereld. Om dit te zien stelle men zich de wereld voor als een verzameling van objecten, die elk een eigen vorm hebben. Datgene wat men waarneemt aan een object noemen we de *vorm* van het *object* [Ross 1975]. De vorm van een object omvat dus alle afzonderlijke waarneembare eigenschappen. Door zijn vorm is een object uniek. Het vormbegrip is een primitief begrip, dat wil zeggen, het is moeilijk de betekenis ervan te definiëren. Een voorbeeld moge illustreren wat ermee wordt bedoeld.

> Stel dat er op een tafel een klomp klei ligt, die is gekneed, zodanig dat men er een konijn in kan herkennen. Met de vorm van de klomp klei bedoelen we alle fysische eigenschappen, in principe tot de ruimtelijke positie van elke klei-molecuul toe. Een andere klomp klei die, zover het

onderscheidingsvermogen van de waarnemer reikt, op precies dezelfde wijze is gekneed, heeft toch een eigen unieke vorm. Op zijn minst verschillen de twee vormen namelijk door hun positie in de ruimte.

Een generiek concept werkt als een vormvoorschrift voor de vorm van objecten. Elk object, waarvan de vorm 'conform' het vormvoorschrift is, is een element van de klasse van alle objecten waarvan de vorm 'conform' het vormvoorschrift is. Deze klasse is de referent van het concept. Omdat die klasse er zonder het concept niet zou zijn, zeggen we dat het concept de klasse creëert.

> We konden in het voorbeeld van de klompen klei alleen zeggen dat ze op een konijn leken omdat het generieke concept "konijn" tot onze kennis behoorde.

De vorm van een object kan voldoen aan verschillende vormvoorschriften. Daardoor behoort het object tot verschillende klassen.
Een element van een klasse wordt in het semantisch netwerk gerepresenteerd door een individueel concept. Een individueel concept kan dus worden genoteerd als een tweetal <g, o> waarin g de notatie is van een generiek concept en o de notatie is van een object.

> Indien de twee klompen klei worden genoteerd als object 1 en object 2, dan kunnen de twee individuele concepten worden genoteerd als
> <konijn, object1> en <konijn, object2>.

De concepten, die we tot nu beschouwden waren concrete concepten, dat zijn dus concepten, die een referent hebben (vergelijk figuur 2.1).
Op abstracte concepten kan men een analoge beschouwing toepassen, door het bestaan van abstracte objecten te veronderstellen. Daartoe stellen we een abstract object gelijk aan het teken waarvan het de extensie is. Om misverstanden te voorkomen noemen we de referent van een concreet concept een concreet object.

We zullen nu overstappen op een in de informatie-analyse meer gangbare terminologie. De definities die daarmee aan die termen wordt gegeven komen in hoofdlijnen overeen met de gangbare, intuïtieve, definities.
Een concreet individueel concept is een *entiteit*. Voorbeelden van entiteiten zijn: klant Jansen, order r387, artikel fiets.
Een concreet generiek concept is een *entiteittype*. Voorbeelden van entiteittypen zijn: klant, order, artikel.
Een abstract individueel concept is een *waarde*. Voorbeelden van waarden zijn: het getal 14, 14 kg, 14^{o} C.
Een abstract generiek concept is een *waardetype*. Voorbeelden van waardetypen zijn: getal, gewicht, temperatuur.

(N.B. Meestal spreekt men liever in termen van de referent van een waardetype: een getallenverzameling, een gewichtsschaal, een temperatuurschaal).

Een verandering in de wereld is een verandering van de vorm van een of meer concrete objecten. Het plaatsvinden van een bepaalde verandering op een bepaald tijdstip heet een gebeurtenis.
Doordat de vorm van een concreet object kan veranderen, kan vanaf een bepaald tijdstip de vorm niet meer 'conform' een bepaald vormvoorschrift zijn, maar wel 'conform' een ander. Entiteiten hebben dus een bestaansduur: ze 'ontstaan' als gevolg van een gebeurtenis en ze 'vergaan' ook weer als gevolg van een gebeurtenis. Tussen deze twee gebeurtenissen in 'bestaat' een entiteit.
We zullen dit aan de hand van een entiteit van het type persoon toelichten.

> De geboorte van een persoon is een gebeurtenis, waardoor de vorm van een bepaald object zodanig verandert dat het vanaf dat moment "lijkt op een persoon". Dit blijft zo tot de gebeurtenis, die de dood van diezelfde persoon voorstelt.
> Tijdens het bestaan van deze entiteit van het type persoon kan hetzelfde object zodanige vormveranderingen ondergaan dat het gedurende enkele tijdvakken bijvoorbeeld ook voldoet aan de intensie van het type student en/of het type patiënt. In elk van die tijdvakken bestaat er dan tevens een entiteit van het type student en/of een entiteit van het type patiënt. Dat zijn allemaal verschillende entiteiten met hetzelfde object.

Er is een alternatieve definitie van een entiteit, die soms beter aansluit bij het intuïtieve begrip ervan. Deze luidt dat een entiteit het plaatsvinden van een gebeurtenis in het verleden voorstelt, waarbij de referent van de entiteit in een bepaalde rol was betrokken.

> Bijvoorbeeld: als er nu een geleverde order bestaat, dan heeft er een levering plaatsgevonden, waarbij de referent een order was betrokken als referent van een te leveren order.

> Een ander voorbeeld: als er nu een patiënt bestaat, dan betekent dat, dat er in het verleden een opname heeft plaatsgevonden, waarbij de referent van de patiënt was betrokken in de rol van referent van de op te nemen persoon.

Abstracte objecten zijn onveranderlijk. Waarden 'bestaan' dus altijd. Anders gezegd, de referent van een waardetype is een vaste verzameling.

Een entiteit noteren we als een paar <et, co>, waarin et de notatie van een *entiteittype* is, en co de notatie van het concrete object dat de referent is van de entiteit.

Voorbeelden van entiteit-notaties zijn:

<persoon, Jansen>
<student, Jansen>
<order, r387>

Een waarde wordt genoteerd als een paar <vt, ao>, waarin vt de notatie van een *waardetype* is en ao de notatie van een abstract object.

Voorbeelden van waarde-notaties zijn:

<lengte-in-meters, 16>
<temperatuur-in-^{o}C, 16>
<leeftijd-in-jaren, 16>

N.B. De notatie van een abstract object is het object zelf.

Een conceptuele relatie tussen twee entiteiten noemen we een *associatie*. Voorbeelden van associaties zijn: de orderklant van order r387 is klant Janssen, Nick is een neef van Sanne.
Een conceptuele relatie tussen twee entiteittypen heet een *associatietype*. Voorbeelden van associatietypen zijn: orderklant, neef-van.

Een conceptuele relatie tussen een entiteit en een waarde noemen we een *attribuut*. Voorbeelden van attributen zijn: de temperatuur van het zeewater is 16^{o}, de voorraad fietsen is 33.
Een conceptuele relatie tussen een entiteittype en een waardetype heet een *attribuuttype*. Voorbeelden van attribuuttypen zijn: temperatuur, voorraad.

Conceptuele relaties tussen waarden en tussen waardetypen bespreken we niet. Voorbeelden van zulke relaties zijn valutakoersen en de omrekening van temperaturen in ^{o}C naar temperaturen in ^{o}F.

We beschouwen uitsluitend binaire conceptuele relaties. Een conceptuele relatie is een speciaal soort concept en heeft dus ook een referent.
De referent van een conceptuele relatie r tussen concept a en concept b is het paar objecten, die respectievelijk de referent van a en de referent van b zijn.

Een associatie wordt genoteerd als een tripel <rt, co1, co2>, waarin rt de notatie van een associatietype is en co1 en co2 de notaties van concrete objecten zijn.

Voorbeelden van associatie-notaties zijn:

<orderklant, r387, Jansen>
<neef-van, Sanne, Nick>

Een attribuut wordt genoteerd als een tripel <at, co, ao>, waarin at de notatie van een attribuuttype is, co de notatie van een concreet object en ao de notatie van een abstract object.

Voorbeelden van attribuut-notaties zijn:

<leeftijd, Sanne, 1>
<temperatuur, zeewater, 16>
<voorraad, fiets, 33>
<prijs, fiets, 769>

De associaties en attributen, waarbij een entiteit is betrokken zijn op te vatten als conceptualisaties van (delen van) de vorm van het object, dat de referent van de entiteit is. Daardoor hebben associaties en attributen evenals entiteiten een bestaansduur, en 'ontstaan' en 'vergaan' ze als gevolg van gebeurtenissen.

Ten behoeve van de eenvoud van het betoog spreken we, als dat niet tot onduidelijkheid leidt, de volgende vereenvoudigingen af:

- in plaats van "het object, dat wordt aangeduid door de notatie x" schrijven we "het object x";
- in plaats van "het type, dat wordt aangeduid door de notatie x" schrijven we "het type x";
- in plaats van "het object x is de referent van een entiteit van het type y" schrijven we "x is een y".
  Voorbeelden: Jansen is een klant
  fiets is een artikel
- in plaats van "het object x is de referent van een waarde van het type y" schrijven we eveneens "x is een y".
  Voorbeelden: 5 is een getal
  16^{o} C is een temperatuur.

8.2 Formalisering van het intuïtieve toestandsmodel

Om het verband te kunnen leggen met de formules van een SL-taal wordt het intuïtieve toestandsmodel op de hieronder beschreven wijze geformaliseerd.

CO: een verzameling; de elementen van CO heten *concrete objecten*;
AO: een verzameling; de elementen van AO heten *abstracte objecten*;
ET: een verzameling; de elementen van ET heten *entiteittypen*;
VT: een verzameling; de elementen van VT heten *waardetypen*;
BT: een verzameling; de elementen van BT heten *associatietypen*;
AT: een verzameling; de elementen van AT heten *attribuuttypen.*

De verzamelingen CO, AO, ET, VT, BT en AT zijn paarsgewijs disjunct.

E = ET * CO; de elementen van E heten *entiteiten*;
V = VT * AO; de elementen van V heten *waarden*;
B = BT * CO * CO; de elementen van B heten *associaties*;
A = AT * CO * AO; de elementen van A heten *attributen.*

Een toestand S is gedefinieerd als: $S \subset E \cup V \cup B \cup A$.

De *extensie* van een *entiteittype* et $\in$ ET in een toestand S wordt genoteerd als **et** en is als volgt gedefinieerd:

$$\mathbf{et} = \{ co \mid \langle et, co \rangle \in S \}$$

Voor entiteittypen zijn de volgende functies gedefinieerd:

$minc \in ET \rightarrow N$, de minimale cardinaliteit geheten,
$maxc \in ET \rightarrow N$, de maximale cardinaliteit geheten,

en de volgende bijbehorende eigenschap ($\forall$ et $\in$ ET):

$$minc(et) \leq card(\mathbf{et}) \leq maxc(et).$$

(N.B. De functie card is gedefinieerd voor elke verzameling en stelt de cardinaliteit van een verzameling voor.)

De *extensie* van een *waardetype* vt $\in$ VT in een toestand S wordt genoteerd als **vt** en is als volgt gedefinieerd:

$$\mathbf{vt} = \{ao \mid <vt, ao> \in S\}$$

De *extensie* van een *associatie-type* bt $\in$ BT in een toestand S wordt genoteerd als **bt** en is als volgt gedefinieerd:

$$\mathbf{bt} = \{ <co_1, co_2> \mid <bt, co_1, co_2> \in S \}$$

Voor associatietypen zijn de volgende functies gedefinieerd:

dom1 $\in$ BT $\rightarrow$ ET, het eerste domein geheten,
dom 2 $\in$ BT $\rightarrow$ ET, het tweede domein geheten,
$minc_1 \in$ BT $\rightarrow N$, de minimale cardinaliteit van het eerste domein geheten,
$maxc_1 \in$ BT $\rightarrow N$, de maximale cardinaliteit van het eerste domein geheten,
$minc_2 \in$ BT $\rightarrow N$, de minimale cardinaliteit van het tweede domein geheten,
$maxc_2 \in$ BT $\rightarrow N$, de maximale cardinaliteit van het tweede domein geheten,

en de volgende bijbehorende eigenschappen ($\forall$bt $\in$ BT):

$\forall et_1: \forall et_2: \forall co_1: \forall co_2:$
$dom_1(bt) = et_1 \wedge dom_2(bt) = et_2 \wedge <co_1, co_2> \in \mathbf{bt}$
$\rightarrow co_1 \in \mathbf{et_1} \wedge co_2 \in \mathbf{et_2}$;
$\forall co_1 \in \mathbf{et_1}:$
$minc_1(bt) \leq card(\{<x, y> \mid x = co_1 \wedge <x, y> \in \mathbf{bt}\}) \leq maxc_1(bt)$;
$\forall co_2 \in \mathbf{et_2}:$
$minc_2(bt) \leq card(\{<x, y> \mid y = co_2 \wedge <x, y> \in \mathbf{bt}\}) \leq maxc_2(bt)$;

De eerste eigenschap is, in woorden, dat voor elk paar objecten in de extensie van een associatietype geldt dat het eerste object een element is van de extensie van het eerste domein van het associatietype, en dat het tweede object een element is van de extensie van het tweede domein.

De *extensie* van een *attribuuttype* at $\in$ AT in een toestand S wordt genoteerd als **at** en is als volgt gedefinieerd:

$$\mathbf{at} = \{ \langle co, ao \rangle \mid \langle at, co, ao \rangle \in S \}$$

Voor attribuuttypen zijn de volgende functies gedefinieerd:

dom $\in$ AT $\rightarrow$ ET, het domein geheten;
rng $\in$ AT $\rightarrow$ VT, het bereik geheten;

Voor een attribuuttype at met dom (at) = et en rng (at) = vt moet nu gelden:

$$\mathbf{at} \in \mathbf{et} \rightarrow \mathbf{vt},$$

ofwel een attribuuttype is een zuivere functie.

Het verband tussen het intuïtieve toestandsmodel en een SL-taal wordt in paragraaf 8.3 behandeld.

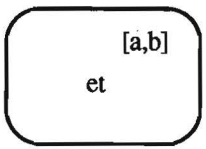
We sluiten deze paragraaf af met een opmerking over de *identificatie* van concrete objecten.
Het is in de praktijk meestal lastig objectnotaties te gebruiken ter aanduiding van concrete objecten. In plaats daarvan wordt een object dan *geïdentificeerd* door de opsomming van een aantal kenmerken, waardoor het eenduidig is bepaald.
Een veel toegepaste identificatie-wijze is de entiteiten in de extensie van een entiteittype van unieke namen te voorzien. Voorbeelden van zulke namen zijn salarisnummers, artikelcodes en ordernummers. De naam van een entiteit is een attribuut van de entiteit.
Indien t een entiteittype is, en n de naam van een entiteit in de extensie van t, dan is de referent van die entiteit dus uniek bepaald door het paar <t, n>.
Een alternatieve notatie van een entiteit van het type t is dan: <t, <t,n>>. Deze notatie wordt in de praktijk afgekort tot <t, n>. Men spreekt dan bijvoorbeeld van "artikel 32761" of "order AB399".
Het gevaar van deze notatie-verkorting is, dat de informatie-analist entiteitnaam en objectnotatie verwisselt. Menig informatie-analist moet dit tot zijn schande ontdekken, wanneer de gebruikers een nieuw coderingsstelsel willen invoeren.

8.3 Het S-schema

De verzameling axioma's van een processor heeft in de praktijk al gauw een zodanige omvang, dat het moeilijk is het overzicht te bewaren. In de informatie-analyse maakt men dan ook vrijwel altijd gebruik van schematechnieken, waardoor sommige soorten axioma's grafisch kunnen worden uitgebeeld.
We doelen hiermee op de schematechnieken, die horen bij de zogeheten semantische datamodellen ([Tsichritzis, Lochovsky 1982]). Enkele voorbeelden daarvan worden in hoofdstuk 10 besproken.

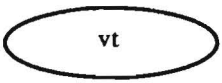
De hieronder gepresenteerde schematechniek is bedoeld om een deel van de axioma's, die de toestandsruimte definiëren, grafisch weer te geven. Zo'n grafische weergave heet daarom een S-schema (State schema).

De verschillende soorten grafische symbolen van een S-schema zijn, tezamen met hun formele en intuïtieve betekenis (waarbij X de verzameling axioma's van de processor is):



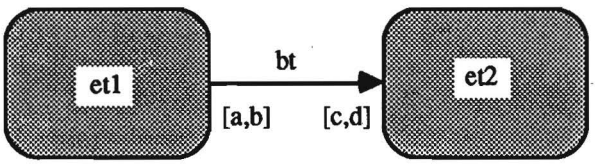
formeel: $et \in PS^1$;
$X \models \forall x: et(x) \rightarrow concreet(x) \wedge$
$(\#x: et(x)) \geq a \wedge$
$(\#x: et(x)) \leq b$;

intuïtief: er is een *entiteittype* et met minc (et) = a en maxc (et) = b.



formeel: $vt \in PS^1$;
$X \models \forall x: vt(x) \rightarrow abstract(x)$;

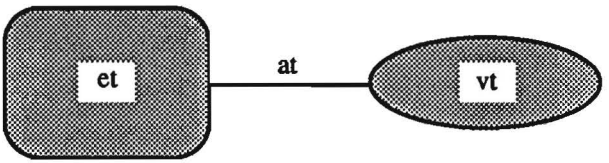
intuïtief: er is een *waardetype* vt;



formeel: $bt \in PS^2$;
$X \models \forall x: \forall y: bt(x, y) \rightarrow et1\ (x) \wedge et2\ (y) \wedge$
$(\forall x: (\#y: bt\ (x, y)) \geq a \wedge (\#y: bt\ (x, y)) \leq b) \wedge$
$(\forall y: (\#x: bt\ (x, y)) \geq c \wedge (\#x: bt\ (x, y)) \leq d)$

intuïtief: er is een *associatietype* bt met dom1 (bt) = et1 en
dom2 (bt) = et2; verder geldt:
$minc_1(bt) = a$, $maxc_1\ (bt) = b$, $minc_2(bt) = c$ en $maxc_2\ (bt) = d$;

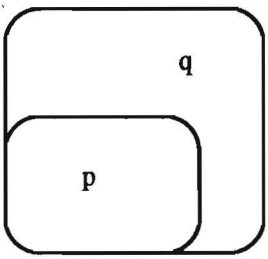
N.B. Door de pijlrichting zijn het eerste en het tweede domein van een associatietype bepaald. De pijl wijst naar het tweede domein.



formeel: $at \in PS^2$;
$X \models \forall x: \forall y: at(x, y) \rightarrow et\ (x) \wedge vt\ (y) \wedge$
$(\forall x: (\#y: at\ (x, y) = 1)$

intuïtief: er is een *attribuuttype* at met dom (at) = et en rng (et) = vt;

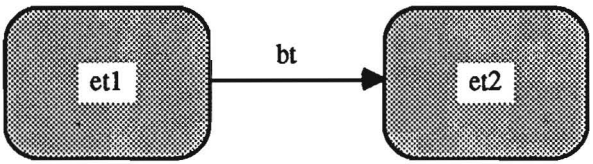
Twee rechthoeken, die niet-overlappend zijn getekend stellen *disjuncte entiteittypen* voor.
Niet-disjuncte entiteittypen waarbij de extensie van de één een deelverzameling is van de extensie van de ander worden als volgt gerepresenteerd:



formeel :X |= ∀x: p(x) → q (x);
intuïtief :p is een *subtype* van q;

De in een S-schema niet-overlappend getekende waardetypen, associatietypen en attribuuttypen hoeven niet disjunct te zijn. Bovendien mag elk entiteittype, waardetype, associatietype en attribuuttype meer dan een keer worden gerepresenteerd (dit kan soms tot duidelijker schema's leiden).

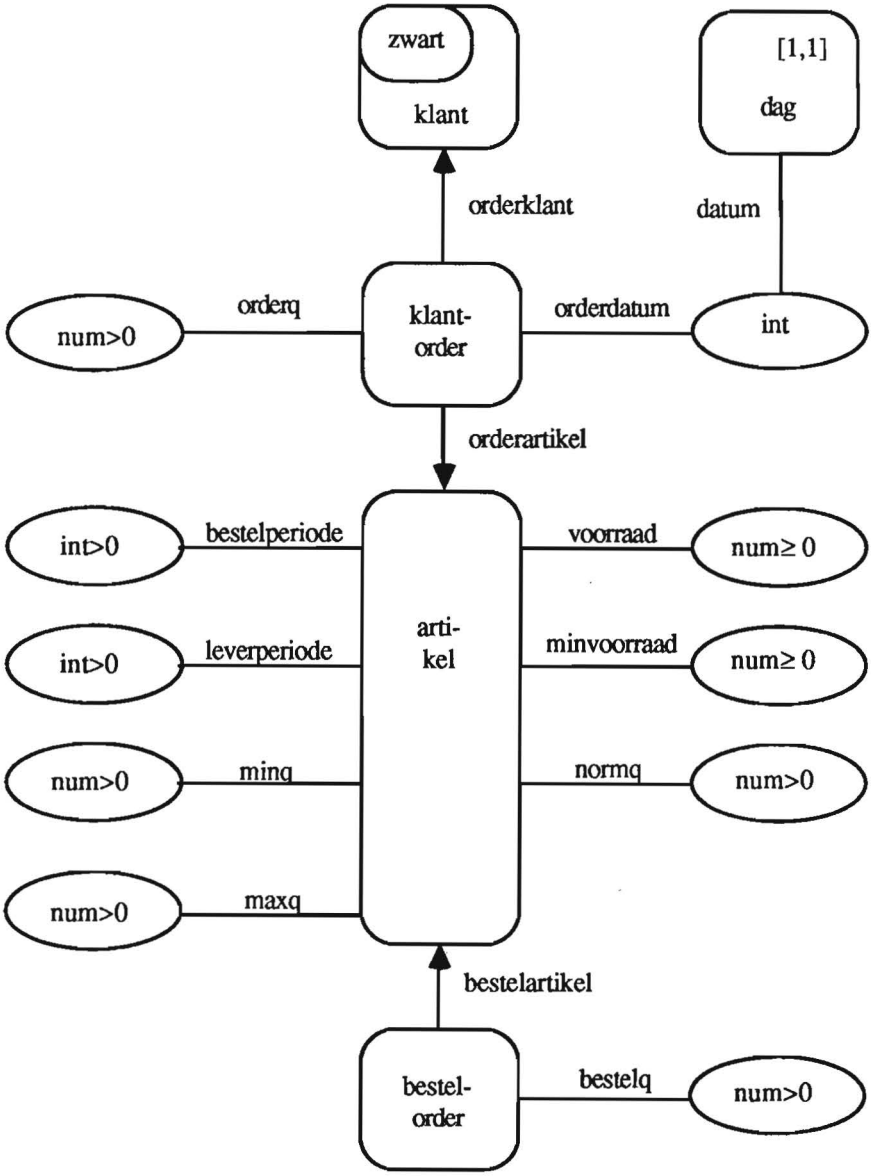
De cardinaliteitswaarde 'onbegrensd' wordt genoteerd als "*".
Als de minimale cardinaliteit van een verzameling o is, en de maximale cardinaliteit "*", wordt de vermelding weggelaten.
Indien voor een associatietype bt met $dom_1(bt) = et_1$ en $dom_2(bt) = et_2$ geldt: $minc1\ (bt) = maxc_1\ (bt) = 1$ en $minc_2\ (bt) = 0$ en $maxc_2\ (bt) = *$ wordt de vermelding van het eerste paar waarden ook weggelaten en wordt het associatie-type-symbool vervangen door het volgende:



bt is nu een functie geworden met domein **et1** en bereik **et2.**

Het is toegestaan deelverzamelingen van de standaardverzamelingen der rationale en der gehele getallen in het symbool van het waardetype aan te geven.

Ter illustratie van de S-schema-techniek is in figuur 8.1 het S-schema van het postorderbedrijf weergegeven.



Figuur 8.1: S-schema van het postorderbedrijf

Onderstaande opmerkingen zijn bedoeld als toelichting bij dit schema.

- Een klantorder is gekarakteriseerd door vier eigenschappen:
  - een associatie van het type orderklant,
  - een associatie van het type orderartikel,
  - een attribuut van het type orderq,
  - en een attribuut van het type orderdatum.
  Wat de associaties betreft houdt dat in dat de geassocieerde entiteiten "bestaan" gedurende het "bestaan" van de klantorder.

- Datums zijn gemodelleerd als waarden van een zogeheten interval-schaal. Dat is een schaal waarvan zowel het nulpunt als de meeteenheid vrij kunnen worden gekozen. (Een ander voorbeeld van een interval-schaal is een temperatuurschaal).
  De afbeelding van de datum-schaal op de gehele getallen betekent dat alle getallen tussen n en n + 1 (waarbij n een geheel getal is) tijdstippen voorstellen die binnen de datum n vallen.

Hoewel een S-schema eruit ziet als een 'entiteitendiagram' is het veel meer dan dat. De symbolen hebben immers een precies gedefinieerde betekenis. Daardoor kan een S-schema een formele vervanging zijn van een groot aantal axioma's. Bij de specificatie van een dds kan men van deze eigenschap dankbaar gebruik maken.

9. Voorbeeld: het postorderbedrijf

In dit hoofdstuk wordt de modellering en de specificatie van een postorderbedrijf als een dds gepresenteerd.
De beschrijving is opgebouwd uit de volgende stappen:

1. Het reëele systeem;
2. Het N-schema;
3. De specificatie van **S**, **A**, **R** en **T**;
4. Het S-schema;
5. De axioma's;
6. De specificatie van **M**.

In stap 1 wordt een informele beschrijving gegeven van het te modelleren systeem. Deze beschrijving bedoelt een algemene toelichting te zijn bij de volgende stappen en is daarom geenszins volledig. De beschrijvingen in de overige stappen zijn formeel.
In stap 4 worden de verzamelingen PS_j voor elke $j \in$ dom (S), en de verzamelingen PA_i, PR_i en PD_i voor elke $i \in$ dom (M) gespecificeerd met behulp van een tabel. Deze tabel bevat ook toelichtingen.

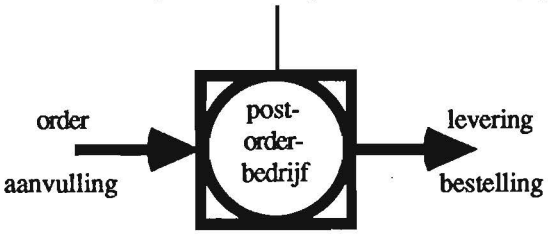
Het beschreven voorbeeld is dat van het postorderbedrijf, waarvan al eerder delen werden besproken (in hoofdstuk 3 en hoofdstuk 4).

9.1 Het reëele systeem

Een postorderbedrijf levert goederen aan klanten en beheert daartoe een magazijn. De goederen zijn geclassificeerd in soorten, artikelen geheten. Klanten kunnen orders op goederen plaatsen. Van elk artikel wordt periodiek bekeken of er orders zijn, die kunnen worden geleverd. Als dat het geval is, worden er binnen een bepaalde leverstrategie, zoveel mogelijk orders geleverd. De magazijnbeheerder plaatst, indien nodig, bestellingen bij leveranciers. Deze soort orders leidt tot de ontvangst van goederen, waarmee de voorraad wordt aangevuld.
De leverbare artikelen (het assortiment) worden niet door de magazijnbeheerder, maar door de directie van de onderneming bepaald. Deze bepaalt ook, via parameters, de 'speelruimte' van de magazijnbeheerder bij het uitvoeren van zijn taak. Tot deze parameters behoren de minimale voorraad, de normatieve bestelhoeveelheid en de bestelperiode per artikel.
Tenslotte stelt de directie vast aan welke klanten mag worden geleverd.

9.2 Het N-schema

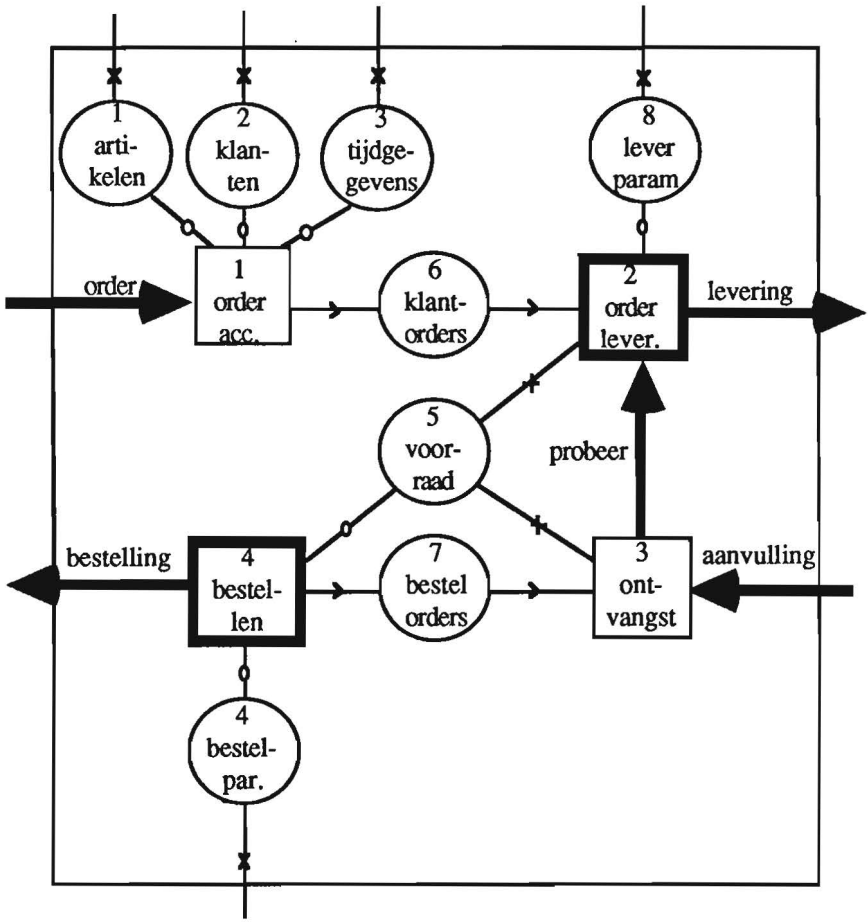
Het basis-dds, dat het postorderbedrijf voorstelt, is weergegeven in figuur 9.1.



Figuur 9.1: N-schema van het postorderbedrijf als basis-dds

Dit is een ioc-dds. Er zijn twee soorten invoeracties: orders en aanvullingen, en twee soorten reacties: leveringen en bestellingen. Daarnaast is er sprake van communicatie met de omgeving.
Figuur 9.2 toont een decompositie van het basisnetwerk.

Deze decompositie komt overeen met de onderscheiden vier deelsystemen in de informele beschrijving (zie hoofdstuk 4).
Processor 1 (order-acceptatie) ontvangt orders van klanten en beslist over acceptatie van de order. Daartoe inspecteert hij geheugen 1 (artikelen) en geheugen 2 (klanten). Indien een order wordt geaccepteerd, wordt dat aangetekend in geheugen 6 (klantorders). Aan die aantekening wordt de orderdatum toegevoegd; deze komt uit geheugen 3 (tijdgegevens).
Processor 2 (order-levering) wordt periodiek geactiveerd; de periode is een parameter in geheugen 5 (lever-parameters). Daarnaast wordt hij geactiveerd door acties afkomstig van processor 3. Volgens een bepaalde, vaste, regel worden bij activering zoveel mogelijk klantorders geleverd. Een levering is een reactie van processor 2.
Processor 3 (ontvangst) wordt geactiveerd door aanvullingen op de voorraad, afkomstig van leveranciers. Als de aanvulling past bij een uitstaande bestelorder in geheugen 7 wordt de voorraad aangevuld. Tevens wordt een "probeer te leveren" reactie voor processor 2 afgegeven.
Processor 4 (bestellen) gaat periodiek na of het nodig is bestellingen te plaatsen. Dit wordt bepaald door de voorraad, door bestelparameters in geheugen 4 en door een vaste bestelregel. Bestellingen zijn reacties van processor 4. De omgeving is gerechtigd mutaties aan te brengen in de geheugens 1, 2, 3, 4 en 8.



Figuur 9.2: N-schema van het postorderbedrijf

Door het N-schema zijn de volgende onderdelen van de formele specificatie vastgelegd:

- dom (M) = {1, 2, 3, 4};
- dom (S) = {1, 2, 3, 4, 5, 6, 7, 8};
- de transactiekanalen T_{22}, T_{32} en T_{44} bestaan;
- I_1 = {1, 2, 3, 6},
- I_2 = {5, 6, 8},

- $I_3 = \{5, 7\}$,
- $I_4 = \{4, 5, 7\}$;

9.3 De specificatie van S, A, R en T

Elke PSj, PAi, PRi en PDi is gespecificeerd in onderstaande tabel. (In kolom "n" is de ariteit vermeld).

pred verz	symbool	n	toelichting
PS 1	artikel	1	artikel (a) betekent dat artikel a tot het leverbare assortiment behoort.
	minq	2	minq (a, q) betekent dat van artikel a de minimale orderhoeveelheid q is;
	maxq	2	maxq (a, q) betekent dat van artikel a de maximale orderhoeveelheid q is;
PS 2	klant	1	klant (k) betekent dat k een klant is;
	zwart	1	zwart (k) betekent dat van klant k geen orders mogen worden geaccepteerd.
PS 3	dag	1	dag (v) betekent dat v een dag is;
	datum	2	datum (v, d) betekent dat d de datum is van dag v;
PS 4	minvoorraad	2	minvoorraad (a, m) betekent dat de minimale voorraad van artikel a de hoeveelheid m is.
	normq	2	normq (a, n) wil zeggen dat steeds een hoeveelheid n van artikel a wordt bijbesteld.
	bestelperiode	2	bestelperiode (a, p) wil zeggen dat elke p dagen gekeken wordt of van artikel a moet worden bijbesteld.
	vrij	1	vrij (x) betekent dat x een concreet object is dat dienst kan doen als bestelling
PS 5	voorraad	2	voorraad (a, v) betekent dat de voorraad van artikel a de hoeveelheid v is.

PS 6	klantorder	1	klantorder (r) stelt een nog niet geleverde order r voor;
	orderklant	2	orderklant (r, k) betekent dat k de klant is van klantorder r;
	orderartikel	2	orderartikel (r, a) betekent dat a het artikel is van klantorder r;
	orderq	2	orderq (r, q) betekent dat q de orderhoeveelheid van klantorder r is;
	orderdatum	2	orderdatum (r, d) betekent dat klantorder r op datum d is ontvangen;
PS 7	bestelorder	1	bestelorder (b) stelt een uitstaande bestelling voor met object b;
	bestelartikel	2	bestelartikel (b, a) betekent dat bestelorder b betrekking heeft op artikel a;
	bestelq	2	bestelq (b, q) betekent dat bestelorder b een bestelhoeveelheid q heeft;
PS 8	uitleverperiode	2	uitleverperiode (a, p) betekent dat elke p dagen gekeken moet worden of er orders op artikel a kunnen worden uitgeleverd.
PA 1	order	4	order (r, k, a, h) is het plaatsen van klantorder r met klant k, artikel a en hoeveelheid h.
PA 2	lever	1	lever (a) is een prikkel tot het uitleveren van orders voor artikel a.
	probeer	1	probeer (a) heeft hetzelfde effect als lever (a);
PR 2	levering	4	levering (r, k, a, h) stelt het uitleveren voor van order r, met klant k, artikel a en hoeveelheid h.
	probeer	1	<zie onder PA 2>.
PA 3	aanvulling	3	aanvulling (b, a, h) stelt de ontvangst voor van de levering van bestelling b, met, artikel a en hoeveelheid h.
PR 3	probeer	1	<zie onder PA 2>.
PA 4	bestel	1	bestel (a) is de prikkel tot het nagaan of er van artikel a moet worden bijbesteld;
PR 4	bestel	1	<zie onder PA 4>.
PD 2	volgende	2	volgende (a, r) betekent dat de volgende te leveren order op artikel a order r is;
PD3	vandaag	1	vandaag (d) betekent dat de datum van vandaag d is;
PD4	volgende-vrij	1	volgende-vrij (b) betekent dat b het object wordt van de volgende bestelling

De component T is als volgt gespecificeerd:

T22 = {< lever (a), lever (a) >| artikel (a) } ∪ {<probeer (a), probeer (a)>| artikel (a) };
T32 = {< probeer (a), probeer (a) >| artikel (a) };
T44 = {< bestel (a), bestel (a) >| artikel (a) }.

9.4 Het S-schema

Het S-schema van het postorderbedrijf is weergegeven in figuur 8.1. Het werd daar gebruikt ter illustratie van de schematechniek.

9.5 De axioma's

De meeste axioma's zijn impliciet bepaald door het S-schema (volgens de regels die in hoofdstuk 8 zijn gegeven). Aan deze axioma's dienen de volgende te worden toegevoegd:

volgende (a, r) ↔ order (r) Λ orderartikel (r, a) Λ orderq (r, q)
Λ orderdatum (r, d) Λ voorraad (a,v) Λ q ≤ v Λ
∀x: (order (x) Λ orderartikel (x, a) Λ orderq (x, y) Λ orderdatum (x,z) Λ y ≤ v) → z > d ∨ (z = d Λ y > q) ∨ (z = d Λ y = q Λ x ≥ r);

De 'oudste' order heeft prioriteit bij levering. Indien er twee of meer 'oudsten' zijn, gaat de order met de kleinste orderhoeveelheid voor. Als er twee of meer 'kleinsten' zijn wordt een willekeurige gekozen (willekeurig betekent in dit geval de order met het 'kleinste' object).

vandaag (d) ↔ ∃v: (dag (v) Λ datum (v, d));

De datum van vandaag is de datum van de (enige) dag.

vrij (b) $\rightarrow$ concreet (b);
volgende-vrij (b) $\leftrightarrow$ vrij (b) Λ ($\forall$x: vrij (x) $\rightarrow$ x $\geq$ b);

Als object van de volgende te plaatsen bestelling wordt een 'willekeurig' vrij object gekozen.

9.6 De specificatie van M

De motorfuncties van de onderscheiden processoren kunnen als volgt worden gespecificeerd.

M1 (acceptatie)

$\models$ order (r, k, a, q) Λ concreet (r) Λ klant (k) Λ $\neg$ zwart (k) Λ artikel (a) Λ num (q) Λ minq (a, x) Λ q $\geq$ x Λ maxq (a, m) Λ q $\leq$ m Λ vandaag (d)

$\overset{+}{\Rightarrow}$ {klantorder (r), orderklant (r, k), orderartikel (r, a), orderq (r, q), orderdatum (r, d)}

Voor elke actie order (r, k, a, q), waarvoor onder andere geldt dat k geen 'zwarte' klant is, en waarvoor de orderhoeveelheid q binnen de normen ligt wordt er een entiteit van het type klantorder gegenereerd.

M2 (levering)

$\models$ (lever (a) $\vee$ probeer (a)) Λ volgende (a, r) Λ orderklant (r, k) Λ orderartikel (r,a) Λ orderq (r, q) Λ orderdatum (r, d) Λ voorraad (a, v) Λ v $\geq$ q

$\overset{-}{\Rightarrow}$ {klantorder (r), orderklant (r, k), orderartikel (r, a), orderq (r, q), orderdatum (r, d), voorraad (a, v)}

+
⇒ {voorraad (a, v - q)}

R
⇒ {<levering (r, k, a, q), * >, <probeer (a), * >}

Bij elke actie lever(a) of probeer (a), wordt er een levering geproduceerd als er een onderhanden klantorder r is die aan de voorwaarden voldoet, zoals bepaald door het predicaat volgende (a, r).

|= lever (a) Λ leverperiode (a, p)

R
⇒ {<lever (a), p >}

Elke actie lever (a) heeft de productie van een reactie lever (a) tot gevolg p dagen later

M3 (ontvangst)

|= aanvulling (b, a, q) Λ bestelorder (b) Λ bestelartikel (b, a) Λ
bestelq (b, q) Λ voorraad (a, v)

-
⇒ {bestelorder (b), bestelartikel (b, a), bestelq (b, q), voorraad (a, v)}

+
⇒ {voorraad (a, v + q)}

R
⇒ {<probeer (a), * >}

Elke ontvangen aanvulling, waarvoor er een corresponderende uitstaande bestelling is, leidt tot aanvulling van de voorraad. Bovendien wordt er een signaal naar processor 2 gestuurd (via kanaal T_{32}), om te kijken of er orders kunnen worden geleverd.

M4 (bestellen)

|= bestel (a) Λ voorraad (a, v) Λ minvoorraad (a, m) Λ (v +
Σx: bestelorder (x) Λ bestelartikel (x, a) Λ bestelq (x, q) : q) < m Λ
normq (a, n) Λ volgende-vrij (b)

-
$\Rightarrow$ {vrij (b)};

+
$\Rightarrow$ {bestelorder (b), bestelartikel (b, a), bestelq (b, n)}

R
$\Rightarrow$ {<bestelling (b, a, n), * >}

Bij elke actie bestel (a) wordt, indien nodig, een bestelling (b, a, n) geproduceerd, waarin n de normatieve bestelhoeveelheid is.

$\models$ bestel (a) Λ bestelperiode (a, p)

R
$\Rightarrow$ {< bestel (a), p >}

Als er een actie bestel (a) is ontvangen, wordt er altijd, dus ook als de actie niet tot de productie van een bestelling heeft geleid, een reactie bestel (a) geproduceerd, p dagen later.

De specificatie van de motorfuncties van deze processoren is tamelijk arbitrair. Kenmerkend voor alle vier de specificaties is dat we een oneindig grote verwerkingscapaciteit hebben aangenomen. Zoals in hoofdstuk 7 is uiteengezet, is dat niet bezwaarlijk indien men alleen de logica van de verwerking wil beschrijven. De vertragingstijden in sommige responsies zijn onbepaald gelaten. De reden hiervoor is dat er geen kennis over bestond.

Er zijn vele andere modelleringen en specificaties van het postorderbedrijf mogelijk. De bedoeling van het voorbeeld was alleen maar om de wijze van modelleren en specificeren van een dds te demonstreren.

In de appendices is nog een drietal voorbeelden van dds-en opgenomen. De wijze waarop de uitwerking van die voorbeelden wordt gepresenteerd is dezelfde als die in dit hoofdstuk werd toegepast.

Deel IV Evaluatie

De bedoeling van dit deel is bij te dragen aan de beoordeling van de bruikbaarheid van de in deel II ontwikkelde techniek voor het modelleren en specificeren van discrete dynamische systemen.
Met opzet spreken we van bijdragen omdat er geen ervaring is met de toepassing van de techniek in praktijkprojecten. De voorbeelden in dit boek zijn in vergelijking met realistische praktijkgevallen niet meer dan vingeroefeningen.
Een beoordeling zal dan ook vooralsnog hoofdzakelijk gebaseerd kunnen zijn op een theoretische evaluatie.
Het metamodel en de specificatietaal uit deel II, en de grafische specificatiewijzen uit deel II, worden in dit deel tezamen het dds-model genoemd.

In de terminologie van de kenniswetenschappen sprekende kan men zeggen dat het conceptuele model dat men heeft van een reëel systeem een hoeveelheid kennis is die men over dat systeem bezit. De rol van een metamodel en een specificatietaal bij het opbouwen van een conceptueel model is ervoor te zorgen dat relevante en coherente kennis over het systeem wordt gevormd, en dat die kennis precies en helder wordt uitgedrukt.
Een 'taal', waarin men deze kennis kan uitdrukken wordt wel een "Conceptual Modeling Language" (CML) genoemd ([Borgida 1985]).

Wat aan CML's, meestal ontbreekt is een expliciete benoeming van de klasse van systemen, waarvoor ze zijn bedoeld, en de mogelijkheid tot compositie/ decompositie van een systeem.
CML's worden dan ook vaak in combinatie met technieken voor het definiëren van systeemstructuren gebruikt. Deze technieken komen in paragraaf 10.1 aan de orde.

Uit het vakgebied "data bases" stammen technieken voor het grafisch presenteren van een belangrijk deel van de generieke kennis over de toestanden van een "Universe of Discourse". Zo'n grafische presentatie wordt een conceptueel schema genoemd. In paragraaf 10.2 worden enkele van deze technieken besproken.

In hoofdstuk 11 wordt het dds-model vergeleken met andere modelleringstechnieken, waaronder CML's.
Zowel in dit hoofdstuk als in hoofdstuk 10 wordt het dds-model gebruikt als referentiekader.

Er is een klasse van technieken, die noch in hoofdstuk 10 noch in hoofdstuk 11 aan de orde komt, en die we toch niet geheel onvermeld willen laten. Dat is de klasse van computerondersteunde technieken of 'talen' voor het documenteren van

gegevensverwerkende systemen. Deze technieken zijn namelijk in zekere zin de voorlopers van de huidige, al of niet computerondersteunde, modelleringstechnieken.
De oudste, en wellicht ook bekendste, techniek in deze klasse is PSL/PSA ([Teichroew, Hershey 1977]). Dit is een produkt van het ISDOS-project aan de Universiteit van Michigan. In nauwe samenwerking hiermee is aan de Universiteit van Namen het product DSL/DSA ontwikkeld ([Bodart, Pigneur 1983]). Het verschil met PSL/PSA is vooral dat men prototypes kan genereren, en dat het operationele gedrag van een systeem kan worden gesimuleerd.
Het verschil tussen dit soort documentatietalen en CML's is vaak maar gering. In het algemeen ontbreekt bij documentatietalen een formele definitie van de semantiek. De 'etikettering' is echter in zekere mate willekeurig.

Als een samenvattende terugblik worden in hoofdstuk 12 enkele conclusies geformuleerd. Tevens zullen we daar richtingen voor het toekomstige onderzoek bespreken.

10. Vergelijking met andere schematechnieken

10.1 Structuurschema's

Een structuurschema van een systeem is de grafische presentatie van de actieve en passieve componenten van het systeem en van de verbindingen tussen deze componenten.
Een N-schema (zie hoofdstuk 7) is een voorbeeld van een structuurschema. In termen van het dds-model uitgedrukt worden in een structuurschema dus de volgende aspecten van een systeem vastgelegd:

- dom (M), ofwel de opsomming van de onderscheiden processoren;
- dom (S), ofwel de opsomming van de onderscheiden geheugens;
- de transactiekanalen en interconnecties, dus de componenten **I** en **T**;
- de globale omschrijving van **A** en **R** in de vorm van commentaar bij de transactiekanalen en de invoer- en uitvoerkanalen.

Om de compositie en decompositie van systemen uit te beelden is een systeemconcept nodig, dat op elke aggregatieniveau van toepassing is. Zoals we in deel II hebben gezien voldoet het dds-concept aan die voorwaarde: de aggregatie van een netwerk is weer een netwerk, dat wil zeggen iets van hetzelfde soort.

Zoals we aan de hand van twee voorbeelden zullen toelichten leveren structuurschema-technieken op dit punt meestal problemen op.
De oorzaak hiervan is dat als systeemconcept het io-dds (zie hoofdstuk 7) wordt genomen, dat is dus een systeem dat alleen interactie kent met zijn omgeving. Elke toestand van het systeem is lokaal, ofwel, het geheugen van een systeem is geheel privé. Zodra nu, in een decompositie, twee processoren een niet-lege doorsnede van hun toestandsruimtes blijken te hebben, ontstaan moeilijkheden. We zullen dit nader toelichten bij de bespreking van de voorbeeld-technieken.

Hoewel er veel structuurschema-technieken bestaan, zijn de verschillen meestal niet essentieel. We volstaan er daarom mee twee technieken nader te bekijken. Deze kunnen als representanten van twee klassen van technieken worden beschouwd:

- het A-schema uit de ISAC-methode; deze techniek is primair bedoeld om (reële) objectsystemen te beschrijven;
- het DFD uit de SA/SD-methode; deze techniek is primair bedoeld om informatiesystemen te beschrijven.

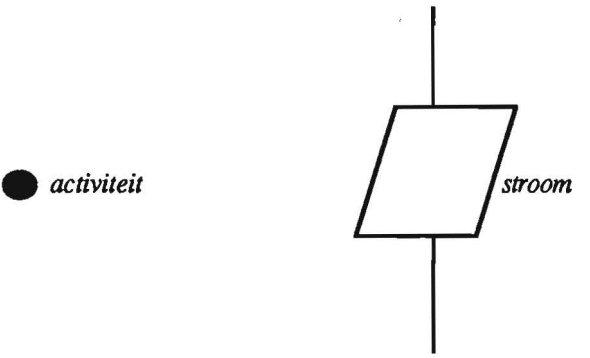
Beide technieken worden op ruime schaal toegepast in de praktijk van de informatie-analyse.

De ISAC-methode en de SA/SD-methode zijn zogeheten *functie-gerichte* of *proces-gerichte* methoden (de twee benamingen worden door elkaar gebruikt). Hiermee wordt uitgedrukt dat de functies van een te ontwikkelen informatiesysteem voorop staan, en dat de benodigde informatiebasis daarvan een afgeleide is.

Het A-schema van ISAC

ISAC ("Information Systems work and Analysis of Change") is een methode, die voortkomt uit onderzoekswerk aan de universiteit van Stockholm en is onder meer beschreven in [Lundeberg, Goldkuhl, Nilsson 1979]. In de ISAC-methode worden verschillende schematechnieken toegepast.
Eén van de schema-soorten is het zogeheten A-schema. De bedoeling van een A-schema is de relevant geachte *activiteiten* (vandaar de "A") in een (deel-) organisatie uit te beelden alsmede de samenhang daartussen.
Activiteiten kunnen met elkaar worden verbonden door stromen. Daarnaast bestaan er stromen van en naar de omgeving.
De activiteiten in een A-schema komen overeen met de processoren in een N-schema. Er bestaat eveneens een overeenkomst tussen de stromen in een A-schema en de transactiekanalen in een N-schema, maar er is ook een belangrijk verschil.
Een stroom bedoelt een werkelijke stroom van dingen in beeld te brengen, bijvoorbeeld ontvangen goederen of ontvangen orders, terwijl via een transactiekanaal het feit van de ontvangst van een partij goederen of van een order wordt doorgegeven.
In een A-schema wordt dan ook onderscheid gemaakt tussen fysieke stromen en gegevensstromen. De als voorbeeld genoemde stroom van ontvangen goederen zou als een fysieke stroom worden weergegeven, terwijl men een stroom orders meestal als een gegevensstroom zal uitbeelden.
Wat de vergelijking tussen stromen en transactiekanalen betreft zou men dus kunnen zeggen dat een transactiekanaal een abstracter concept is dan een stroom.

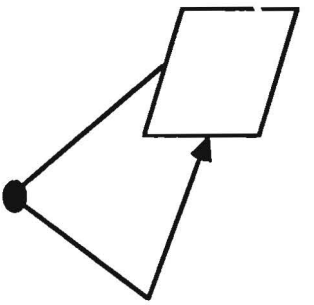
De (voor ons doel benodigde) symbolenverzameling van de A-schematechniek is:



Figuur 10.1: De symbolen in een A-schema

De soort van dingen waaruit een stroom bestaat wordt in het parallellogram genoteerd. Men zegt daarom wel dat een parallellogram een verzameling voorstelt. Als er geen richting (door middel van een pijlpunt) aan een stroom is gegeven, is de afspraak dat de richting van boven naar beneden is. Een activiteit is door middel van ingaande en uitgaande stromen verbonden met andere activiteiten en met de omgeving.

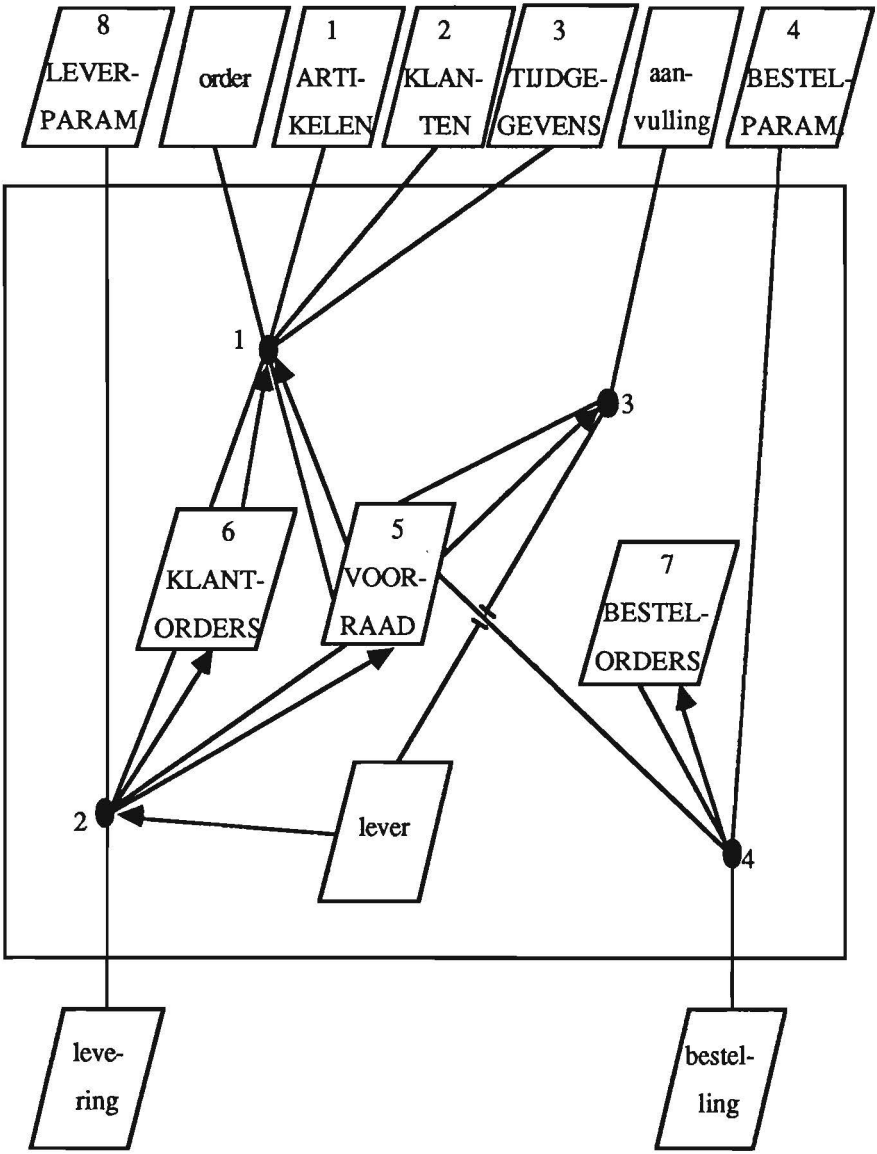
Indien men behoefte heeft aan het modelleren van een geheugen wordt de constructie van figuur 10.2 toegepast.



Figuur 10.2: Constructie voor het modelleren van een toestand.

Eenzelfde stroom is nu dus invoer en uitvoer van een activiteit. De door het parallellogram voorgestelde verzameling wordt een permanente verzameling genoemd. Voorbeelden hiervan zijn: een klantenbestand en een voorraad goederen.

Als voorbeeld van een A-schema is in figuur 10.3 een A-schema, dat correspondeert met het N-schema van het voorbeeld uit hoofdstuk 9 getekend. De permanente verzamelingen zijn daarin voor de duidelijkheid van benamingen in hoofdletters voorzien.



Figuur 10.3: A-schema van het postorderbedrijf

De regel voor de decompositie van een activiteit is een gesloten kromme om de activiteit te tekenen en deze als systeemgrens te beschouwen in het A-schema, dat de 'explosie' van die activiteit voorstelt. Alle ingaande stromen van de activiteit worden dan invoerstromen van het 'nieuwe' systeem en alle uitgaande stromen worden uitvoerstromen. Doet men dit bijvoorbeeld voor activiteit 1, dan ontstaat er een invoerstroom 'KLANTORDERS' en een uitvoerstroom 'KLANTORDERS', maar deze stromen hebben een geheel andere betekenis dan bijvoorbeeld de inkomende orders.

Iets soortgelijks is aan de hand met de 'invoerstromen' 1, 2, 3, 4 en 8. Deze 'stromen' bedoelen eigenlijk gemeenschappelijke geheugens uit te beelden van het systeem en zijn omgeving.

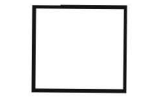
Vanwege de ambigue semantiek van een stroom, blijkt uit een A-schema niet duidelijk door welke stromen een activiteit wordt geactiveerd. Deze onduidelijkheid betreffende de interactiestructuur wordt nog vergroot doordat terugkoppelstromen niet expliciet in het schema worden aangegeven.
Als voorbeeld beschouwen we de activering van activiteit 2. Hoewel de door activiteit 3 verzonden "lever"-signalen activiteit 2 activeren zijn er, zoals in het N-schema is te zien, ook nog "lever"-signalen, die activiteit 2 zelf produceert.

Samenvattend kan men stellen dat, vergeleken met een N-schema, in een A-schema de kennis betreffende dom (**M**), **A** en **R** even goed kan worden uitgedrukt. Omdat er geen expliciet geheugenconcept is, is het niet mogelijk dom(S) te specificeren. Kennis over **T** kan ook worden gerepresenteerd, zij het dat eenzelfde symbool wordt gebruikt voor transactiekanalen en 'interconnecties'.
Het systeemconcept in de ISAC-methodiek is de activiteit, dat wil dus zeggen de transformatie van invoer naar uitvoer. Dit wordt niet expliciet genoemd, maar is bijvoorbeeld af te leiden uit de decompositie-procedure.
Alleen als men strikt vasthoudt aan dit systeemconcept is de compositie/ decompositie van A-schema's consistent. In de praktijk is die voorwaarde echter meestal te beperkend, en neemt men inconsistenties zoals de hierboven genoemde, voor lief.

Het DFD van SA/SD

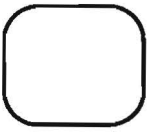
SA/SD staat voor "Structured Analysis/Structured Design". Het is een methode, die vooral in de Verenigde Staten van Amerika populair is. Wat onder "SA/SD" moet worden verstaan is niet op één plaats vastgelegd, vandaar dat we als referentie enkele klassieke publicaties uit deze 'school' noemen: [Gane, Sarson 1977], [Yourdon, Constantine 1979] en [DeMarco 1978].

De grafische techniek, die wordt gebruikt om de structuur van een informatiesysteem uit te beelden is het DFD ("Data Flow Diagram").
De symbolenverzameling van een DFD ziet er als volgt uit:



omgeving, dat wil zeggen de bron of de bestemming van een stroom;

stroom, (van gegevens)



proces , dat wil zeggen een transformatiefunctie van invoerstromen naar uitvoerstromen;

geheugen , opslagplaats voor gegevens.

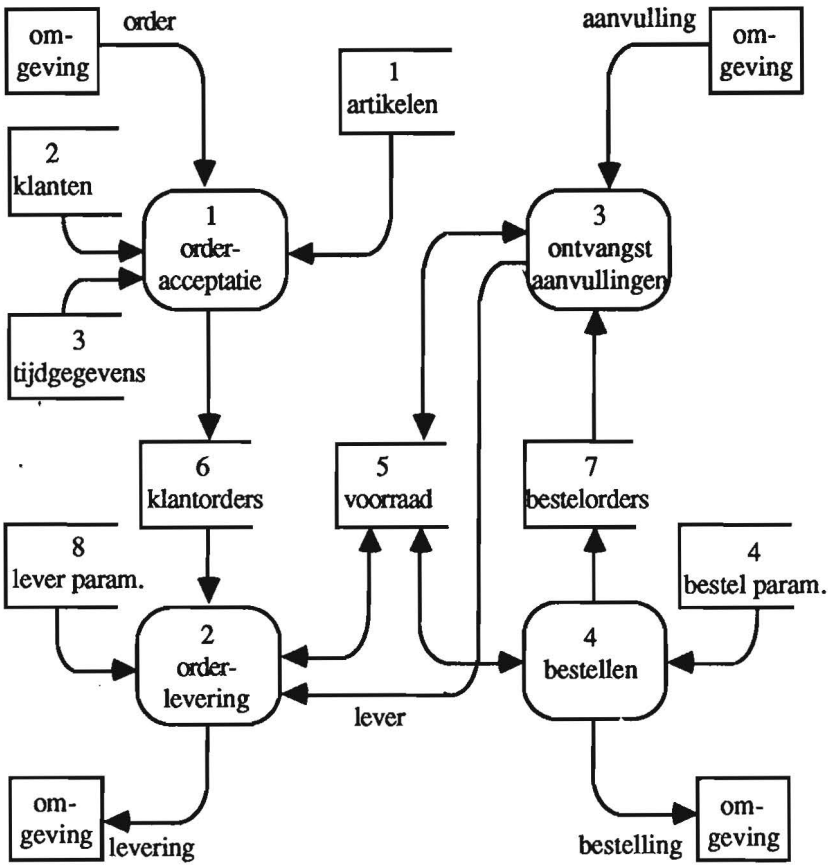
Figuur 10.4: De symbolen van een DFD.

Figuur 10.5 toont het DFD, dat correspondeert met het N-schema van figuur 9.2.

De processoren en geheugens in een DFD komen overeen met respectievelijk de procesoren en geheugens in een N-schema. Het is niet mogelijk transportkanalen en interconnecties te onderscheiden: beide worden als stromen (van gegevens) voorgesteld.
Daardoor wordt ook het inspecteren van de inhoud van een geheugen een stroom, namelijk de stroom gegevens -"copiëen" van het geheugen naar de processor.

Men vergelijke bijvoorbeeld de verbinding tussen geheugen 6 en proces 2 met die tussen geheugen 8 en proces 2. De betekenis van de eerste verbinding is dat proces 2 uit geheugen 6 gegevens verwijdert. De betekenis van de tweede verbinding is dat proces 2 geheugen 8 inspecteert.

Zoals bijvoorbeeld uit de rol van geheugen 5 blijkt, is het modelleren van communicatie tussen processen binnen een systeem eenvoudig.
Bij de communicatie met de omgeving doen zich echter problemen voor. Omdat er alleen invoerstromen van en uitvoerstromen naar de omgeving kunnen bestaan, is het niet mogelijk aan te geven dat er processen in de omgeving van het systeem een 'interconnectie' hebben met de geheugens 1, 2, 3, 4 en 8.



Figuur 10.5: DFD van het postorderbedrijf

De wijze waarop deze geheugens in figuur 10.5 zijn weergegeven is dus eigenlijk niet correct. Het probleem dat zich voordoet komt overeen met de moeilijkheid, die het modelleren van de verzamelingen 1, 2, 3, 4 en 8 in het A-schema opleverde.

Wat de interactiestructuur betreft kan hier dezelfde opmerking worden geplaatst die we maakten bij de bespreking van het A-schema. Er zijn echter varianten op het DFD in ontwikkeling, waarin de activering van processen wel kan worden aangegeven (zie bijvoorbeeld [Ward, Mellor 1985]).

Samenvattend kan men zeggen dat in een DFD de kennis over dom (M), dom (S), **A**, **R**, **T**, en **I** even goed kan worden uitgedrukt als in een N-schema, zij het dat er geen onderscheiden symbolen bestaan voor transactiekanalen en interconnecties.
Ook voor DFD's geldt dat het gehanteerd systeemconcept niet duidelijk wordt aangegeven. Op zijn best is dat het io-dds, dat wil dus zeggen een proces met geheugen, maar zonder 'communicatie' met de omgeving.
Alleen als men strikt vasthoudt aan het io-dds als systeemconcept is de compositie/decompositie van DFD's consistent. Ook voor DFD's geldt dat men in de praktijk het vaak niet zo nauw neemt en een voorstelling zoals in figuur 10.5 acceptabel acht.

10.2 Conceptuele schema's

Naast functie-gerichte methoden onderscheidt men wel *gegevensgerichte* methoden. Bij deze methoden staat het ontwerp van de informatiebasis van een informatiesysteem voorop. De functies zijn secundair, en worden vaak alleen globaal aangegeven. Een informatiebasis bestaat uit een verzameling individuele feiten en een verzameling algemene feiten. De laatste verzameling wordt het *conceptuele schema* van de informatiebasis genoemd. In termen van het dds-model geformuleerd is een conceptueel schema de definitie van een toestandsruimte. Het reële systeem waarop het betrekking heeft wordt in gegevensgerichte methoden meestal de "Universe of Discourse" genoemd.
Een grafische voorstelling van een conceptueel schema wordt ook een conceptueel schema genoemd. Een S-schema (zie hoofdstuk 8) is dus een voorbeeld van een conceptueel schema.

Aan een conceptuele-schema-techniek ligt een datamodel ten grondslag ([Tsichritzis, Lochovsky 1982]).
We zullen in deze paragraaf twee in de praktijk veel toegepaste schematechnieken nader bekijken. Deze zijn:

- het Entiteit-Relatie-Diagram (ERD);
- het "Data Structure Diagram" (DSD).

Het metamodel dat aan het ERD ten grondslag ligt is het Entiteit-Relatie-Model, voor het DSD is dat het netwerkmodel.
Het DSD is ook, en wellicht nog meer, bekend onder de naam Bachman-diagram ([Bachman 1969]).

Het Entiteit-Relatie-Diagram

Het Entiteit-Relatie-Model is een raamwerk voor het beschrijven van een data base op het conceptuele niveau ([ANSI/SPARC 1975]). De Engelse benaming van dit model is "Entity-Relationship-Model". Het Nederlandse woord "relatie" is zowel een vertaling voor "relationship" als voor de (wiskundige) "relation". Om misverstanden te vermijden zullen we het woord "relationship" voortaan onvertaald laten.

De begrippen entiteit en relationship zijn alleen intuïtief gedefinieerd. Een entiteit is iets dat in de "Universe of Discourse" bestaat en waarover men gegevens wil vastleggen. Een relationship is een verband tussen twee of meer entiteiten, dat men van belang acht te kennen.
Vanwege de intuïtieve definitie is de beslissing om iets als een entiteit of als een relationship te modelleren vaak moeilijk te nemen en lijkt de keuze soms arbitrair.
In hoofdlijnen zijn de verschillen tussen entiteiten en relationships de volgende.
Entiteiten zijn dingen, waarvan het bestaan niet afhangt van (het bestaan van) andere dingen. Klanten en artikelen zijn voorbeelden van entiteiten.
Relationships zijn dingen, waarvan het bestaan wel afhangt van andere dingen, namelijk van entiteiten. Orders en bestellingen zijn voorbeelden van relationships.
Naast de begrippen entiteit en relationship wordt meestal een afzonderlijk begrip attribuut onderkend. Een attribuut is een functioneel verband tussen entiteiten en relationships enerzijds en waarden anderzijds.

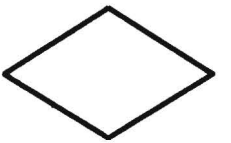
Elk datamodel, dat deze begrippen entiteit en relationship gebruikt heet een Entiteit-Relatie-Model. Het bekendste is ongetwijfeld het voorstel in [Chen 1976].
De hoofdzaken van dit voorstel zijn de volgende.
Entiteiten worden geclassificeerd in entiteitverzamelingen. Relationships worden geclassificeerd in relationshipverzamelingen. Een relationshipverzameling R is als volgt gedefinieerd:

$$R \subset E1 * E2 * ... * En$$

waarbij E1, ... En entiteitverzamelingen zijn.
Tevens worden waardenverzamelingen onderscheiden zoals getallen en namen en wordt het begrip attribuut gedefinieerd. Een attribuut is een functie met een entiteitverzameling of een relationshipverzameling als domein, en een waardeverzameling als bereik.

In een Entiteit-Relatie-Diagram (ERD) komen de volgende symbolen voor:

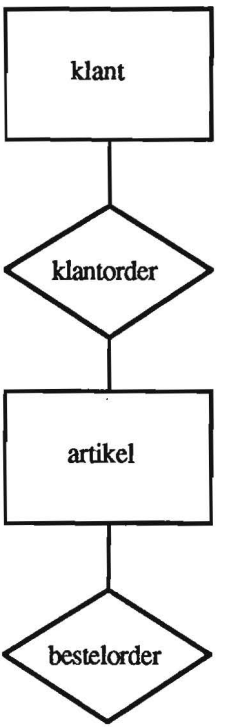
entiteitverzameling



relationshipverzameling

Een relationshipverzameling R wordt verbonden met alle entiteitverzamelingen Ei, die domein zijn van R. Op die wijze wordt de samenhang tussen entiteit-verzamelingen en relationshipverzamelingen uitgebeeld.

Een ERD dat correspondeert met het S-schema in figuur 8.1 is weergegeven in figuur 10.6.



Figuur 10.6: ERD van het postorderbedrijf

Bij de relationshipverzameling bestelorder doet zich een probleem voor: er is maar één domein, namelijk artikel. Dit probleem is te ondervangen door bijvoorbeeld een entiteitverzameling leverancier toe te voegen, en een bestelorder op te vatten als een relationship tussen een artikel en een leverancier.
Het is echter eenvoudiger van de relationshipverzamelingen klantorder en bestelorder entiteitverzamelingen te maken en zo tot een structuur als in figuur 8.1 te geraken.
Men noemt dit het *objectiveren* van relationships. Dit komt erop neer dat elk tupel uit een relationshipverzameling als één ding, namelijk een entiteit, wordt opgevat.

Daarmee is het Entiteit-Relatie-Model eigenlijk vervangen door een functioneel datamodel (zie bijvoorbeeld [Shipman 1981]). Een functioneel datamodel is weer te beschouwen als een speciaal geval van een binair-relatiemodel (zie bijvoorbeeld [Abrial 1974]), namelijk het geval waarbij elke binaire relatie een zuivere functie is.
Ook een binair-relatiemodel kan door middel van objectivering worden omgezet in een functioneel model. Uit deze transformatiemogelijkheden blijkt dus dat het ER-model, het binaire-relatiemodel en het functionele datamodel equivalent zijn.

De objectiveringsstap wordt in de praktijk vaak genomen, ook al omdat daarmee tevens het probleem van de keuze tussen entiteiten en relationships is opgelost. De vraag is echter of men dan nog kan spreken van een Entiteit-Relatie-Model. Op zijn minst is die benaming verwarrend, omdat de "Relatie" nu een functioneel verband is geworden tussen de oorspronkelijke "relationships" en de oorspronkelijke entiteiten.

De relationship lijkt dus niet zo'n bruikbaar concept. Om die reden is de ERD-techniek in zijn oorspronkelijke vorm niet zo'n geschikte techniek voor het presenteren van conceptuele schema's.
Elke variant, die alleen entiteitverzamelingen kent en binaire relaties over deze verzamelingen, is echter wel bruikbaar en kan in principe als vervanger van de S-schema-techniek optreden. Voor de rest is de keuze van een schematechniek een kwestie van smaak.

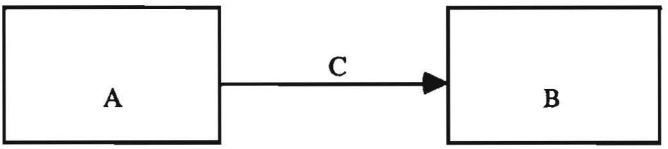
Het DSD (Bachmandiagram)

Het netwerkmodel is een raamwerk voor het beschrijven van een bepaalde samenhang tussen deelverzamelingen van individuele feiten in een informatiebasis. De deelverzamelingen heten recordverzamelingen of bestanden, en de bedoelde samenhang wordt de coset ("Codasyl set") genoemd. Een record representeert in het algemeen een verzameling feiten betreffende eenzelfde entiteit.

Met behulp van het coset-concept worden toegangspaden van records in een verzameling A naar records in een verzameling B weergegeven.
Een coset bestaat uit één record in A en alle bijbehorende records in B.
Het record in A heet het "owner"-record, en de records in B heten "member"-records van de coset.
Het DSD is een grafische presentatietechniek voor coset-structuren. Er zijn twee symbolen:

recordverzameling

coset ("Codasyl set")

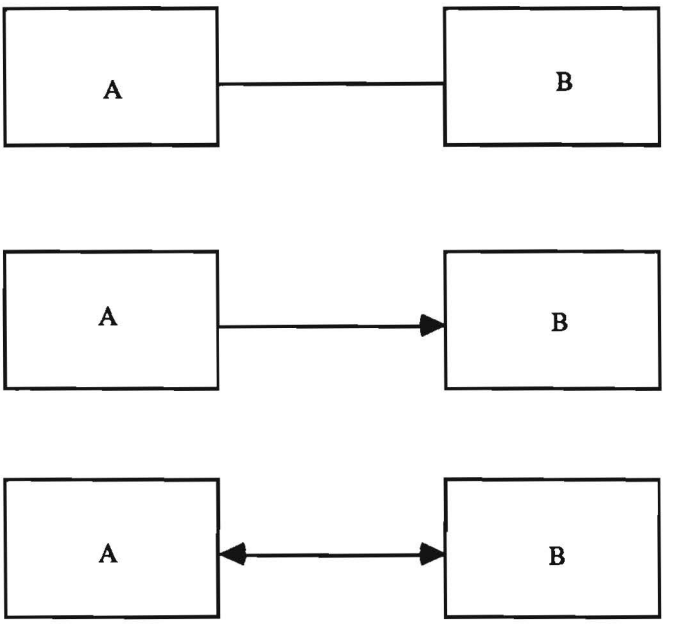


In mathematische termen is een coset-verzameling C met "owner"- verzameling A en "member"- verzameling B als volgt te definiëren:

$$C = \{<a, \{b \mid p(a, b)\} > \} \text{ met } a \in A \text{ en } b \in B,$$

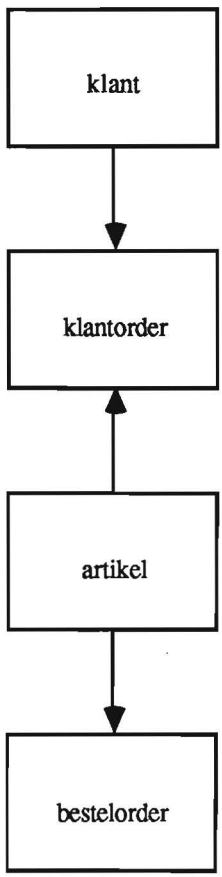
waarin p (a, b) waar is als er een toegangspad is van a naar b. Een coset zonder "member"-records is dus niet leeg, maar bevat het "owner"-record als enig element.

De DSD-schematechniek heeft zich mogen verheugen in een zo overweldigende belangstelling dat hij tegenwoordig ook voor de presentatie van conceptuele schema's wordt gebruikt.
Dat is natuurlijk niet zonder aanpassingen van de semantiek van het DSD gegaan.
De meest gangbare gebruikswijze is die, waarbij de coset feitelijk wordt opgevat als een binaire relatie. Men onderscheidt daarbij drie verschijningsvormen van het coset-symbool (zie bijvoorbeeld [Langerhorst 1981]):



De eerste twee vormen zijn bijzondere gevallen van de derde: de eerste is een bijectie tussen A en B, en de tweede is een functie van B naar A.
Hoewel de verzamelingen A en B nu vaak entiteitverzamelingen worden genoemd blijft het recordconcept toch het belangrijkste onderliggende idee. Attributen bijvoorbeeld worden als 'velden' van de 'entiteit' beschouwd.

Het DSD dat overeenkomt met het S-schema van figuur 8.1 is weergegeven in figuur 10.7.



Figuur 10.7: Het DSD van het postorderbedrijf

Het meest in het oog springend verschil tussen figuur 8.1 en figuur 10.8 is (afgezien van de attributen) de tegenovergestelde pijlrichting van de functies. Dat is vervelend, maar natuurlijk niet essentieel. Wel essentieel is de onduidelijkheid over de betekenis van de symbolen: wanneer stelt een pijl een coset-verzameling voor en wanneer een binaire relatie?
Indien men de verbinding tussen twee recordverzamelingen A en B opvat als een binaire relatie, zeg C', is C' als volgt gedefinieerd:

$$C' = \{<a, b> \mid p\ (a, b)\}$$

waarbij het predicaatsymbool p dezelfde betekenis heeft als in de definitie van de coset.

Als nu voor een bepaalde a en voor alle x geldt dat p (a, x) onwaar is, bevat C' alleen tupels <y, x> waarvoor gedlt dat $y \neq a$.
Indien de verbinding echter wordt opgevat als de coset-verzameling C, zoals boven gedefinieerd, bevat C in dat geval het element <a, ∅>.

De conclusie met betrekking tot de DSD-schematechniek is dat we het geen geschikte techniek achten voor het representeren van conceptuele schema's. De belangrijkste overweging daarbij is dat de concepten van het onderliggende metamodel niet geschikt zijn. Het modelleren van de werkelijkheid als een structuur van records en cosets is niet wat wordt bedoeld met conceptueel modelleren.

11. Vergelijking met andere modelleringstechnieken

In dit hoofdstuk worden modelleringstechnieken besproken, waarin het modelleren van de dynamische aspecten van een systeem een belangrijke plaats inneemt. Allereerst komen de "Conceptual Modeling Languages" (CML's) aan de orde. Vervolgens bespreken we drie modelleringstechnieken, die men normaliter niet tot de CML's rekent, maar die we wel belangwekkend genoeg vinden om in dit verband te bekijken.

11.1 CML's

De ontwikkeling van CML's staat nog in de kinderschoenen. Men kan de meeste dan ook het best typeren als een eclectische mengeling van ideëen en technieken uit andere vakgebieden. De belangrijkste gebieden, waarvan wordt geleend, zijn de kunstmatige intelligentie, de data bases en de programmeertalen.

Kunstmatige intelligentie is de naam van een vakgebied, dat men als toegepaste kenniswetenschappen zou kunnen omschrijven. Het adjectief kunstmatig heeft betrekking op het gebruik van computers om kennisverwervende en kennisgenererende processen in de menselijke geest na te bootsen.
Bekende technieken voor het representeren van kennis zijn semantische netwerken, logica en produktiesystemen ([Brodie, Mylopoulos, Schmidt 1984] en [IEEE 1983]).
Alle drie de technieken hebben hun invloed gehad op de in dit boek gepresenteerde CML, het dds-model dus. Het semantisch netwerk, in het bijzonder de formalisering ervan in conceptuele grafen ([Sowa 1984]) is de grondslag van het intuïtieve toestandsmodel uit hoofdstuk 8, terwijl logica en produktiesystemen de pijlers zijn van de in hoofdstuk 6 gepresenteerde specificatietaal SL.

Data bases is de naam van een vakgebied, waarin men zich toelegt op het ontwerpen en bouwen van systemen voor de opslag van grote hoeveelheden gegevens.
Technieken voor het structureren van gegevens staan bekend onder de naam "data modellen". In [Tsichritzis, Lochovsky 1982] wordt een groot aantal data modellen besproken. Bekende representanten zijn het Relationele Model en het Entiteit-Relatie-Model. De relevantie van data modellen voor CML's is dat gegevensstructuren niet alleen het syntactisch aspect maar ook het semantisch aspect van gegevens betreffen. Een datamodel is een metamodel voor het uitdrukken van generieke kennis over toestanden.

Programmeertalen zijn imperatieve formele talen, die, algemeen gesteld, zijn bedoeld om het manipuleren met dingen te beschrijven. Karakteristiek voor een programmeertaal is de combinatie van acties en objecten.
De relevantie van programmeertalen voor CML's bestaat ons inziens vooral uit de invloed die ze hebben (gehad) op de ontwikkeling van het type-begrip en op het specificeren van toestandsovergangen. De laatste invloed betreft vooral het definiëren van het effect van operaties met behulp van precondities en postcondities.

Zoals al opgemerkt, is het onderzoek naar CML's nog jong. Een van de gevolgen daarvan is dat er geen algemeen aanvaard pakket van eisen is waaraan een CML zou moeten voldoen om die naam waardig te zijn.
We zullen daarom zelf een aantal eisen formuleren, die we noodzakelijk of zeer gewenst achten. Deze eisen zijn de volgende:

1. Het moet mogelijk zijn systemen op een ordelijke manier te decomponeren en te componeren. Daarvoor is, zoals we eerder zagen, een goed aggregatieconcept nodig.
2. Een toestand moet een gestructureerde grootheid zijn, bijvoorbeeld een verzameling. Dit is nodig om een intensionele definitie van een toestandsruimte te kunnen geven (zie punt 5).
3. Er moet een expliciet gedefinieerd mutatiemechanisme zijn. Het is dus niet voldoende te zeggen dat een toestand verandert ten gevolge van een gebeurtenis.
4. Het moet ook mogelijk zijn reacties van het systeem te beschrijven als gevolg van gebeurtenissen. Dat houdt dus de expliciete definitie van een reactiemechanisme is.
5. Men moet statische axioma's kunnen formuleren, dat zijn axioma's, die de toestandsruimte definiëren. Deze axioma's zijn ondersteunend bij de specificatie van het mutatiemechanisme en het reactiemechanisme. Daarnaast vormen ze de basis voor het logisch afleiden van ware beweringen over een toestand.
6. Men moet dynamische axioma's kunnen formuleren, dat zijn axioma's, die de procesruimte definiëren. Deze axioma's zijn ondersteunend bij de specificatie van het mutatiemechanisme en het reactiemechanisme. Daarnaast vormen ze de basis voor het logisch afleiden van ware beweringen over een proces.

Het is niet eenvoudig van CML's aan te geven in welke mate ze aan deze criteria voldoen. Er zijn twee factoren die hierbij een rol spelen. De eerste is dat vaak niet meer documentatie beschikbaar is dan een aantal artikelen. De tweede factor is dat bepaalde aspecten impliciet worden verondersteld of alleen informeel beschreven. Op zo'n basis is het moeilijk een nauwkeurig oordeel te formuleren.

We zullen daarom volstaan met een algemene bespreking van enkele bekende CML's. Deze zijn:

- RML ("Requirements Modeling Language"). Deze CML is één van de resultaten van het Taxisproject aan de universiteit van Toronto, en is onder meer beschreven in [Greenspan, Borgida, Mylopoulos 1986]; research-stadium;
- CIM ("Conceptual Information Model"). Dit model is ontwikkeld in het SYSLAB-instituut van de universiteit van Stockholm (zie bijvoorbeeld: [Gustafsson, Karlsson, Bubenko 1982]); research-stadium;
- ERAE ("Entity-Relationship-Attribute-Event"). Dit is een in het kader van een ESPRIT-project ontwikkelde techniek, onder andere beschreven in [Dubois e.a. 1986]; research-stadium;
- INFOMOD ("Information Modeling"), een hoofdzakelijk binnen Philips ontwikkelde methode onder meer beschreven in: [Griethuysen, Jardine 1984]; operationeel.

Met "research-stadium" bedoelen we dat er geen of weinig ervaring is opgedaan in praktijkprojecten. Als dat wel het geval is, heet de CML "operationeel".

De kern van deze CML's is een semantisch datamodel voor het beschrijven van de statische aspecten van een discreet dynamisch systeem. In het algemeen spreekt men daarbij niet van systeem, maar van "Universe of Discourse", een term die in de database-wereld wordt gebruikt om het subjectdomein van een informatiesysteem aan te duiden.
Om veranderingen in de "Universe of Discourse" te kunnen beschrijven, worden aan het datamodel dynamische concepten toegevoegd, zoals een tijddomein en het gebeurtenis-concept.

Een goed systeemaggregatieconcept, zoals in het dds-model, ontbreekt bij elk van de CML's. Daardoor is het dus niet mogelijk een systeem te decomponeren en te componeren. Er zijn wel enkele andere aggregatieconcepten. INFOMOD bijvoorbeeld biedt de mogelijkheid de toestandsruimte op te delen in submodellen. In RML bestaat het generalisatie/specialisatie-concept voor acties. Men kan daarmee bijzondere acties als specialisaties van algemene acties definiëren.

Aan de tweede eis voldoen alle genoemde CML's. Zoals gezegd is een semantisch datamodel juist de kern van elk.

Geen van de CML's heeft een expliciet gedefinieerd mutatiemechanisme.

In geen van de CML's is een reactiemechanisme expliciet gedefinieerd. Het is wel

bij alle CML's mogelijk informatie-uitvoer naar de omgeving te specificeren.

In alle CML's is het mogelijk statische axioma's te specificeren. De taal waarin axioma's worden uitgedrukt is bij CIM en ERAE eerste orde logica, bij INFOMOD en RML is het een "sugared version" hiervan.

In alle CML's kunnen dynamische axioma's worden geformuleerd. In CIM en ERAE gebeurt dat met behulp van een getypeerde eerste orde logica, waarin het type tijd bestaat. Bij RML en INFOMOD zijn het "sugared versions" van zo'n logica.

Vergelijking met het dds-model toont aan dat geen van de CML's een expliciet gedefinieerde basis heeft zoals het dds-model, en dat het dds-model de andere in principe kan vervangen, behalve waar het de specificatie van dynamische axioma's betreft.

11.2 De FSM

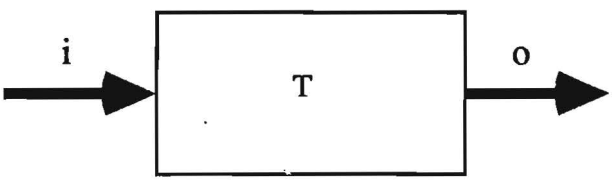
Een FSM (Finite State Machine) is een conceptueel systeem, dat kan worden gedefinieerd als een tuple <S, I, O, T>, waarin:

S, een eindige verzameling, de *toestandsruimte* geheten; een element van S heet een toestand;

I, een eindige verzameling, de *invoerruimte* geheten; een element van I heet een invoer;

O, een eindige verzameling, de *uitvoerruimte* geheten, een element van O heet een uitvoer;

$T \in S * I \rightarrow S * O$, de *transformatiefunctie* geheten.



Figuur 11.1: Grafische voorstelling van een FSM

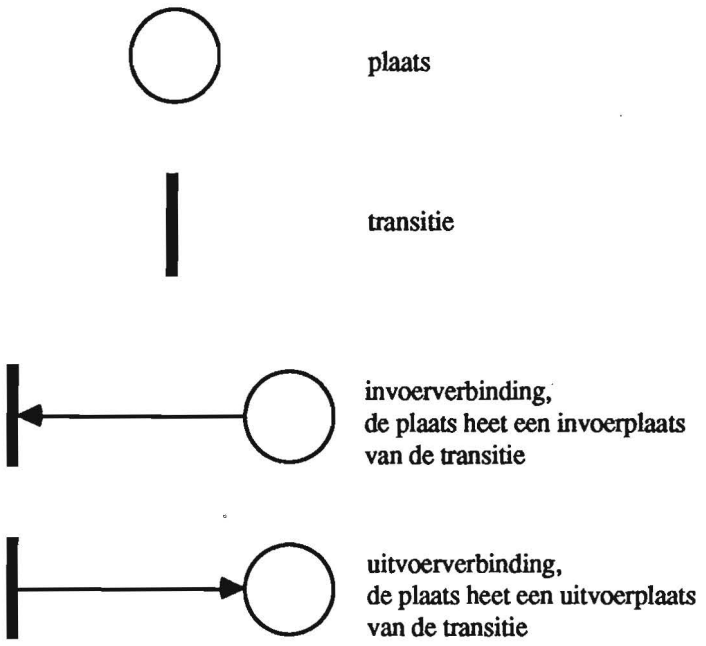
Een FSM kan worden gebruikt als model van een discreet dynamisch systeem. FSM's zijn vooral geschikt in situaties, waarin S, I en O door middel van enumeratie zijn gedefinieerd. Een voorbeeld van zo'n situatie is het ontwerpen van dialogen van interactieve systemen. Omdat FSM's formeel zijn gedefinieerd, is het mogelijk modellen formeel te analyseren. Een bekende analyse is het nagaan of alle gedefineerde toestanden ook bereikbaar zijn.
De belangrijkste verschillen tussen een FSM en een dds zijn de volgende:

- Het aantal toestanden van een FSM is eindig, dat van een dds niet. De toestanden zijn bovendien atomair, dat wil zeggen, men maakt geen gebruik van de eventuele structuur van een toestand.
- De wederzijdse beïnvloeding tussen een FSM en zijn omgeving bestaat alleen uit interactie; er is dus geen communicatie.
- Een FSM kan geen toekomstige eigen invoer genereren. Er is geen terugkoppeling mogelijk. Een FSM heeft geen agenda.
- De koppeling van FSM's in een netwerk is niet gedefinieerd. Dientengevolge is er ook geen aggregatieconcept.

Concluderend kan men zeggen dat een FSM als een bijzonder geval van een io-dds zonder terugkoppelkanaal kan worden gezien.

11.3 Het Petri-net

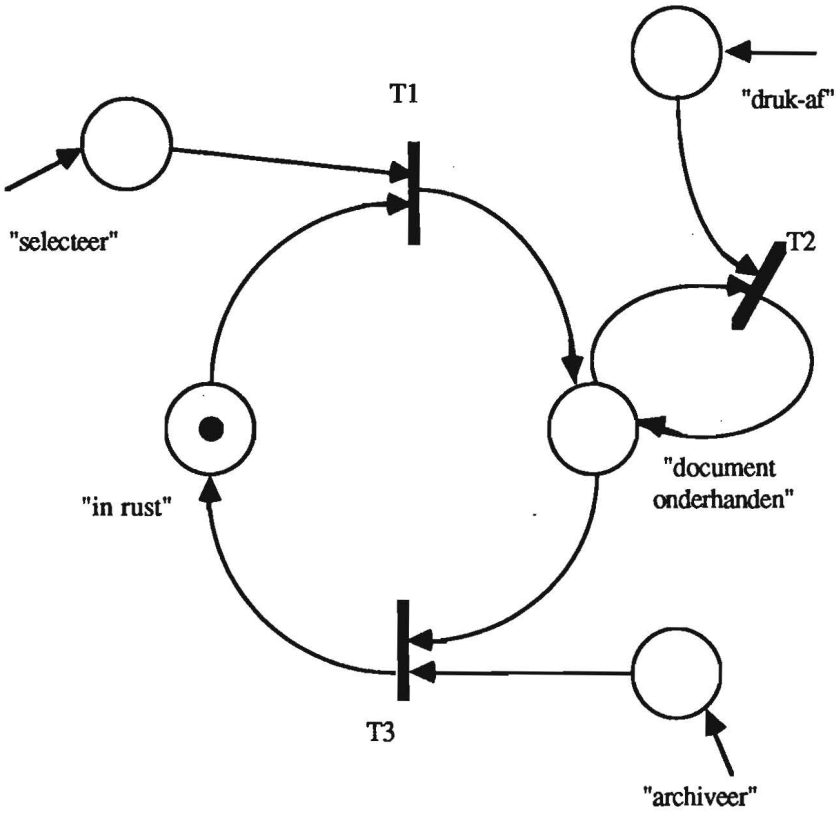
Het Petri-net is een modelleringstechniek, die bedoeld is om de coördinatie of synchronisatie van een verzameling samenwerkende (deel-) systemen te beschrijven.
In [Peterson 1981] wordt een Petri-net gedefineerd als een netwerk bestaande uit vier soorten componenten: plaatsen, transities, invoerverbindingen en uitvoerverbindingen. Een invoerverbinding is een éénrichtingsverbinding tussen een plaats en een transitie.
Een uitvoerverbinding is een éénrichtingsverbinding tussen een transitie en een plaats.
De structuur van een Petrinet wordt meestal uitgebeeld in een diagram, PNG (Petri Net Graph) geheten.
In een PNG worden de volgende symbolen gebruikt:



De werking van een Petrinet is als volgt. Een plaats kan een aantal objecten van eenzelfde soort bevatten, tokens geheten. Indien elk van de invoerplaatsen van een transitie minstens één token bevat, is voldaan aan de voorwaarde voor het actief worden van een transitie. Men zegt dan dat de transitie vuurt. Het effect van het vuren van een transitie is dat van elke invoerplaats een token wordt verwijderd, en dat aan elke uitvoerplaats een token wordt toegevoegd.
Het Petrinet van figuur 11.2 stelt een deel van het proces van een archiveringssysteem voor.

De werking ervan kan als volgt worden toegelicht.

Als het proces in rust is, voorgesteld door een token in de plaats "in rust" (zoals getekend), en er wordt een selecteeropdracht gegeven, dat wil zeggen er komt een token in de plaats "selecteer", dan vuurt transitie T_1. (Selecteren betekent het opzoeken van een bepaald document).
Het resultaat van het vuren van transitie T_1 is dat de tokens uit "selecteer" en "in rust" worden verwijderd, en dat er een token in de plaats "document onderhanden" wordt geplaatst. Op overeenkomstige wijze verloopt het vuren van de andere twee transities.



Figuur 11.2: Voorbeeld van een PNG

Een Petri-net kan eenvoudig worden 'afgebeeld' op een dds. Het hieronder gespecificeerde basis-dds is een algemene 'afbeelding' van een Petri-net.

S = { invoerplaats (. , .), uitvoerplaats (. , .), inhoud (. , .) };
A = { vuur (.) };
X = { vuurconditie (t) $\leftrightarrow$ ($\forall$x: invoerplaats (t, x) $\rightarrow$
$\exists$y: inhoud (x, y) $\wedge$ y > 0)};
M :
$\models$ vuur (t) $\wedge$ vuurconditie (t) $\wedge$ invoerplaats (t, p) $\wedge$
$\neg$ uitvoerplaats (t, p) $\wedge$ inhoud (p, n)
$\Rightarrow$ { inhoud (p, n) }

+
⇒ { inhoud (p, n - 1) }

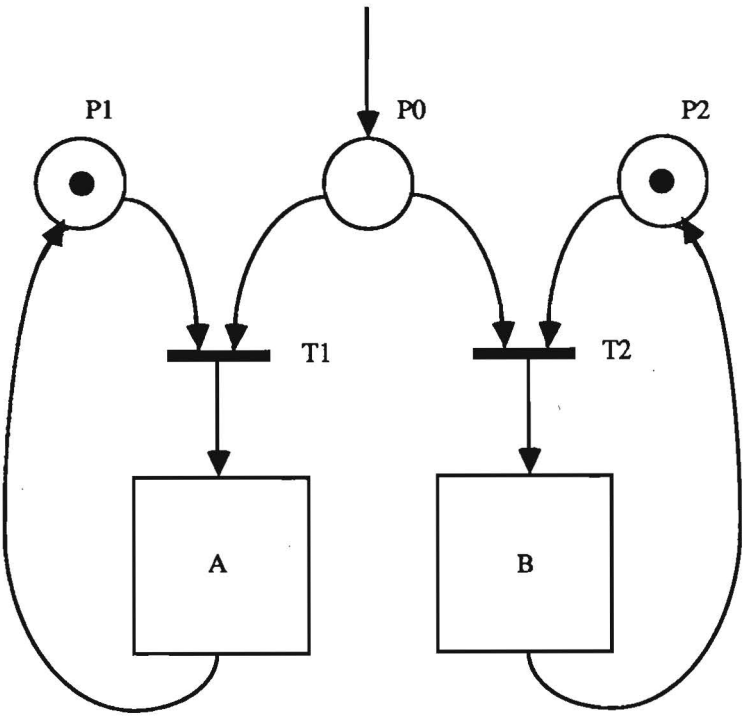
⊨ vuur (t) Λ vuurconditie (t) Λ uitvoerplaats (t, p) Λ
¬ invoerplaats (t, p) Λ inhoud (p, n)

-
⇒ { inhoud (p, n) }
+
⇒ { inhoud (p, n + 1) }

Dit dds is een model voor een Petri-net, dat bestaat uit een willekeurig eindig aantal transities en plaatsen.
De structuur van het netwerk is gedefinieerd door de grondatomen van de typen invoerplaats (t, p) en uitvoerplaats (t, p); invoerplaats (t, p) betekent dat plaats p een invoerplaats is van transitie t; uitvoerplaats (t, p) betekent dat plaats p een uitvoerplaats is van transitie t; inhoud (p, n) betekent dat plaats p n tokens bevat, en vuur (t) is de activering van transitie t.
Indien bij ontvangst van een vuuropdracht aan de vuurconditie is voldaan vuurt de transitie ook werkelijk, anders niet.

Petri-netten zijn niet-deterministische systemen. Dit kan worden gedemonstreerd aan de hand van figuur 11.3.
Dit Petri-net bevat twee parallelle processen, A en B. Deze stellen bijvoorbeeld de verwerking van een order door twee verschillende medewerkers van de afdeling orderacceptatie voor.

De tokens in P1 en P2 betekenen dat A en B allebei niets om handen hebben. Een token in Po betekent dat er een nieuwe te verwerken order is.
Stel dat er een token in Po wordt geplaatst. Dan is aan de conditie om te vuren van beide transities voldaan. Echter, slechts één van beiden kan vuren, omdat bij het vuren van bijvoorbeeld T1 er voor T2 geen token meer is in Po.
Wat dit Petri-net uitbeeldt is dus dat een order óf door A óf door B wordt verwerkt, maar niet welke van de twee het zal zijn.
Indien men er niet in is geïnteresseerd welke het zal zijn, of indien men dit gewoon niet te weten kan komen, is non-determinisme een aantrekkelijke abstractie. Het is evenwel ook niet meer dan dat: in plaats van het volledig beschrijven van een systeem, beschrijft men de bekende, of relevant geachte, eigenschappen ervan.
In termen van het dds-model betekent non-determinisme dat men een klasse van systemen/processen specificeert in plaats van één bepaald systeem/proces. Dit is het gevolg van het niet volledig beschrijven van de interactiestructuur (met name de vertragingstijden van responsies en de externe agenda).



Figuur 11.3: PNG van een niet-deterministisch Petri-net

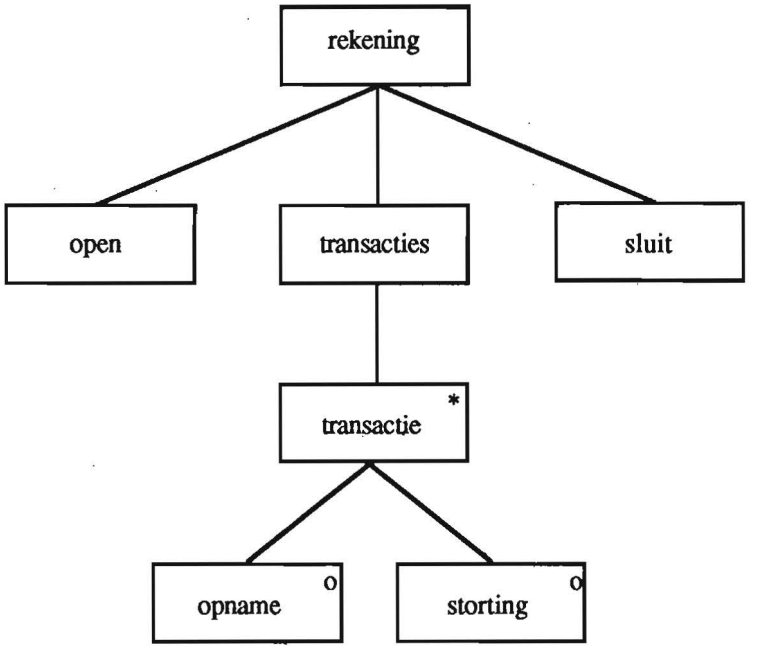
Er zijn in principe twee manieren waarop non-determinisme in een dds kan worden gesimuleerd. De ene is het toepassen van zelfactivering van de processoren, waarbij men de vertragingstijden zo moet kiezen dat twee "concurrerende" processoren nooit tegelijkertijd worden geactiveerd. De andere manier is vuur-acties te laten genereren door de omgeving. Ook daarbij moet uiteraard met bovenstaande voorwaarde rekening worden gehouden.
Door trekkingen uit kansverdelingen te gebruiken als waarden van de vertragingstijden of de activeringstijdstippen bereikt men dat in het netwerk van figuur 11.3, als voorbeeld, soms A en soms B wordt geactiveerd.

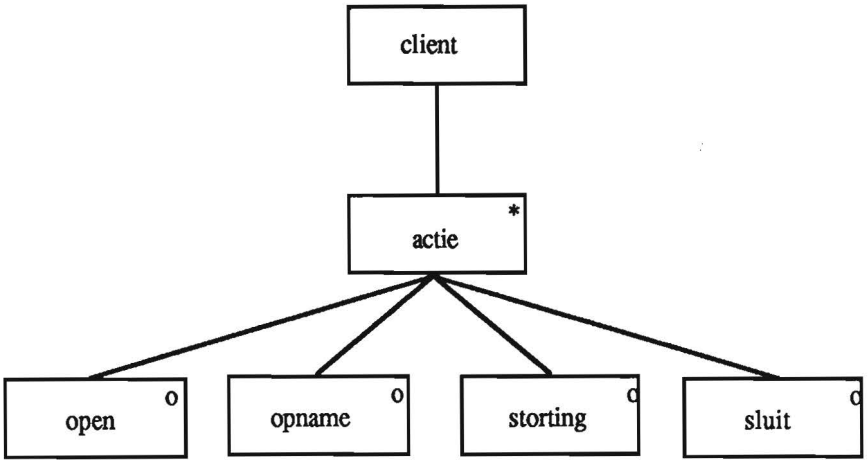
Een variant op het Petri-net is het zogeheten predicaat/transitie-net. In dit net worden toestandspredicaten in plaats van tokens gebruikt om condities voor transities te specificeren [Brauer 1980].
In [Solvberg, Kung 1985] wordt een techniek voor het modelleren en specificeren van systemen beschreven, waarin het predicaat/transitie-net is geïntegreerd met een semantisch datamodel.

11.4 JSD

JSD (Jackson System Development) is een methode voor het ontwikkelen van informatiesystemen ([Jackson 1983]).
JSD is een onorthodoxe methode, dat wil zeggen, het is niet goed mogelijk hem een plaats te geven binnen de indeling functie-gericht versus gegevensgericht. Een operationeel informatiesysteem wordt in JSD primair beschouwd als een werkend dynamisch model van een reëel systeem. Zo'n model bestaat uit een verzameling parallelle sequentiële processen. Om misverstanden te voorkomen zullen we deze processen JSD-processen noemen. Elk JSD-proces stelt een individuele entiteit voor. De samenhang tussen JSD-processen wordt uitgebeeld met behulp van acties. Entiteiten ondernemen en/of ondergaan acties.

Een voorbeeld van een entiteit, die acties ondergaat is een rekening-courant bij een bank (zie appendix B1). Een voorbeeld van een entiteit, die acties onderneemt is een cliënt van een bank.
Figuur 11.4 geeft een schematische voorstelling van een JSD-proces van het type cliënt en een JSD-proces van het type rekening.





Figuur 11.4: JSD-structuurdiagrammen

De semantiek van een structuurdiagram is als volgt gedefinieerd:

- een verbinding tussen twee 'dingen' betekent dat het onderste 'ding' een onderdeel is van het bovenste 'ding';
- een "*" bij een 'ding' betekent dat het 'ding' een aantal keren als onderdeel kan voorkomen (vergelijk de iteratie in algoritmen);
- een "o" bij een aantal 'dingen' op hetzelfde decompositieniveau betekent dat slechts één van deze 'dingen' onderdeel is van het hogere 'ding' (vergelijk de selectie in algoritmen).

De twee diagrammen moet men als volgt lezen:

> Een cliënt onderneemt gedurende zijn bestaan (waarvan het begin en het einde overigens niet door deze diagrammen is gedefinieerd) acties op rekeningen. Een actie is het openen van een rekening of het opnemen vanaf een rekening of het storten op een rekening of het sluiten van een rekening.
>
> Het bestaan van een rekening begint bij het openen ervan en eindigt bij het sluiten ervan. In de tijd daartussen worden transacties voor de rekening ontvangen. Een transactie is het opnemen van een bedrag en het storten van een bedrag.

Het verband tussen structuur-diagrammen wordt gelegd door de overeenkomst in naamgeving van acties. In figuur 11.4 gebeurt dat door de acties "open", "opname", "storting" en "sluit". In het tweede diagram moet men daar een rekening als parameter aan toegevoegd denken.
Aan het aldus verkregen dynamisch model worden informatiefuncties in de vorm van nieuwe JSD-processen toegevoegd. Voorbeelden van informatiefuncties zijn:

- het sturen van een bericht van weigering van een opname, omdat het saldo lager dan het toegestane minimum zou worden;
- het periodiek produceren van een saldo-overzicht van alle rekeningen.

De definitie van de informatiefuncties zou men de functionele specificaties van het informatiesysteem kunnen noemen.

Het is mogelijk voor de JSD-processen, die informatiefuncties voorstellen, prestatie-eisen ten aanzien van tijdaspecten te formuleren.
Voorbeelden van zulke eisen zijn:

- een bericht van weigering moet binnen 24 uur na de feitelijke weigering worden verstuurd;
- de saldi op een salo-overzicht, dat op tijdstip t is geproduceerd, moeten bijgewerkt zijn tot tijdstip t-d, waarbij d een nader te bepalen aantal dagen is.

JSD-processen kunnen op twee manieren met elkaar worden verbonden, door gegevensstromen en door toestandsconnecties.
Als er een *gegevensstroom*-verbinding is van proces A naar proces B, produceert A gegevens, die door B worden ontvangen en verwerkt.
Als er een *toestandsconnectie* is van proces A naar proces B, heeft B 'inzagerecht' in de toestand van A.

Een JSD-proces wordt gespecificeerd door middel van een algoritme in pseudocode. Dat gebeurt per procestype, dus bijvoorbeeld voor alle rekeningen, in de vorm van een geparametriseerde subroutine.
De toestand van een JSD-proces is een lokale variabele in de subroutine.

Vergelijking van de JSD-methode met het dds-model laat zien dat er een grote mate van overeenkomst is tussen de basisconcepten, die aan beide ten grondslag liggen. Met name bedoelen we hiermee de voorstelling van een systeem, in de JSD-methode, als een netwerk van parallelle JSD-processen. Hoewel we in de toepassingsvoorbeelden van het dds-model niet zover zijn gegaan, is het uiteraard mogelijk elke JSD-proces als een (basis-) dds te modelleren. Men kan immers een dds altijd beschouwen als de aggregatie van een verzameling gelijksoortige dds-en:

Bijvoorbeeld, processor 1 in appendix B is op te vatten als de aggregatie van alle "rekening"-processoren. De motorspecificaties van deze processoren verschillen alleen in de waarde van de variabele r.
Een mogelijke implementatie van een "rekening"-processor is een chip, die een microprocessor met geheugen bevat. De microprocessoren zijn allemaal verbonden met geheugen 2, en de geheugens zijn allemaal verbonden met processor 2.

Een nauwkeurige analyse van de overeenkomsten en de verschillen tussen het dds-model en JSD is niet mogelijk omdat JSD niet formeel is gedefinieerd.
De belangrijkste verschillen die er lijken te bestaan zijn de volgende:

- JSD kent geen aggregatieconcept. Het is dus niet mogelijk hiërarchische compositie/decompositie van JSD-processen toe te passen.
- Het proces van een systeem, in de zin van het dds-model, is niet gedefinieerd. Daardoor kunnen tijdaspecten (die in wezen vertragingstijden van responsies zijn) niet in de specificatie van een JSD-proces worden opgenomen. Evenmin is het mogelijk bij een toestandsconnectie tussen een proces A en een proces B aan te geven hoe vaak B de toestand van A inspecteert.
  Het enige, dat men met betrekking tot tijdaspecten kan specificeren, is een (informele) opsomming van prestatie-eisen voor informatiefuncties, zoals eerder vermeld.
- In JSD is slechts een beperkte vorm van communicatie mogelijk, namelijk alleen het 'inzagerecht', maar niet bijvoorbeeld een buffer tussen twee JSD-processen.
- De structuurdiagrammen tonen niet alleen de actiebasis of reactiebasis van een processor, maar ook dynamische axioma's. Uit het tweede diagram van figuur 11.6 bijvoorbeeld blijkt dat het openen van een rekening voorafgaat aan opnames van of stortingen op die rekening, en aan het sluiten ervan. Overigens drukt dit diagram ook de, ons inziens niet noodzakelijke, eis uit dat er minstens één opname of storting moet zijn geweest alvorens een rekening kan worden opgeheven.
- Het door JSD beschouwde reële systeem is in feite een gesloten systeem. De 'producenten' van externe acties en externe mutaties zijn namelijk JSD-processen in het dynamisch model. Een voorbeeld van zo'n 'producent' is de cliënt in figuur 11.4. In een implementatie van een dynamisch model worden deze processen 'gevoed' door hun tegenhangers in het reële systeem.

Alle in [Jackson 1983] opgenomen voorbeelden kunnen eenvoudig, en preciezer, als een dds worden beschreven. Het voorbeeld in appendix B is een demonstratie hiervan.

Met deze korte uiteenzetting over JSD sluiten we de vergelijking tussen het dds-model en andere modelleringstechnieken af.

12 Nabeschouwing

In de voorgaande hoofdstukken is een metamodel gepresenteerd en geëvalueerd voor het maken van conceptuele modellen van discrete dynamische systemen. Dit metamodel is het dds-model genoemd. Een alternatieve definitie van het dds-model, die we ook hebben gebruikt, is: een techniek voor het modelleren en specificeren van discrete dynamische systemen.

Met behulp van deze techniek kan het gedrag van een discreet dynamisch systeem, zoals gedefinieerd in hoofdstuk 3, op intensionele wijze worden gedefinieerd. Hoewel we dat niet formeel hebben bewezen, lijkt het aannemelijk dat elk discreet dynamisch systeem als een dds kan worden gemodelleerd.
Methodische aspecten zijn buiten beschouwing gebleven, hoewel die in de praktijk zeker niet onbelangrijk zijn. Zoals men uit de gepresenteerde voorbeelden kan afleiden, is het vinden van een dds, dat een geschikt model is van een reëel systeem, geen triviale aangelegenheid. Onderzoek naar de methodische kant van het modelleren lijkt dan ook zinvol.
Men zou daarbij onder andere kunnen denken aan een manier om systemen te decomponeren in een stadium dat nog niet meer bekend is dan de invoerbasis, de uitvoerbasis en de externe-mutatiebasis (zie hoofdstuk 3), een soort "black-box"-aggregatieconcept dus.

Wat discrete dynamische systemen met informatiesystemen te maken hebben is in deel I al gedeeltelijk besproken. Thans kunnen we daar preciezer over zijn. Een dds is een conceptueel model van een reëel systeem. De formalisatie van een dds is een symbolisch systeem. Met betrekking tot symbolische systemen kan men van een materialisering spreken. Daarmee bedoelen we dat symbolen een fysische drager krijgen en dat voor transformatiemechanismen (zoals mutatiemechanismen en reactiemechanismen) ook geschikte materialiseringen worden gekozen.
In automatiseringstermen uitgedrukt is een materialisering een implementatie van een gegevensverwerkend systeem.
Indien nu een materialisering van een dds wordt 'gevoed' met een E-component, noemen we het dds *operationeel*. Er voltrekt zich dan een proces van het dds.
Een operationeel dds, dat reacties afgeeft aan zijn omgeving is een informatiesysteem. De afgegeven reacties zijn informatie over het gemodelleerde reële systeem.

Het onderscheiden van communicatie naast interactie als een vorm van wederzijdse beïnvloeding tussen systemen, of tussen een systeem en zijn omgeving, is niet alleen zinvol maar ook nodig, zoals bij de bespreking van de schematechnieken in paragraaf 10.1 is gebleken.

Het is ook niet moeilijk in het leven van alledag voorbeelden van de twee vormen te vinden. Men vergelijke bijvoorbeeld het ontvangen van post via een brievenbus (=communicatie) met opgebeld worden (=interactie). Het essentiële verschil tussen de twee vormen is dat bij interactie de zender van de boodschap het tijdstip van ontvangst door de ontvanger bepaalt, terwijl dat bij communicatie de ontvanger zelf is.
Het is waarschijnlijk zo dat communicatie veel vaker voorkomt dan interactie. De interruptie van de CPU van een computer bijvoorbeeld lijkt in eerste instantie een voorbeeld van interactie, maar is een voorbeeld van communicatie. Er is namelijk een geheugenplaats, waarin de 'interruptiemelding' wordt gezet. De CPU inspecteert die geheugenplaats zeer frequent.

Het dds-model is een bruikbaar lijkende, en deels al bruikbaar gebleken, combinatie van bestaande ideëen en technieken. Voor het conceptueel modelleren van systemen biedt het een aantal nieuwe gezichtspunten.
Een van deze gezichtspunten is dat een processor via zijn terugkoppelkanaal zijn eigen agenda kan aanvullen. De agenda van een processor vormen ook een soort toestand, maar dan betreffende het proces van een systeem.
Toestand en proces zijn analoge begrippen. Het verschil tussen beide heeft te maken met het standpunt dat men ten aanzien van de tijd inneemt.
Met behulp van de metafoor van de film kan dit worden toegelicht.
Indien men een film, zoals in de bioscoop, beeldje na beeldje voor het oog laat afrollen wordt men een reeks van gebeurtenissen gewaar, die toestands-veranderingen veroorzaken. Een gebeurtenis is het verschijnen van het volgende beeldje. De toestandsverandering is het verschil tussen het nieuwe beeldje en het vorige. Men volgt dus een proces van gebeurtenis tot gebeurtenis. We zullen dit standpunt ten aanzien van de tijd het standpunt in de tijd noemen.
Het alternatief van dit standpunt is het standpunt buiten de tijd. Hiermee bedoelen we dat men in één keer de gehele film waarneemt. Het proces is nu een, onveranderlijke, toestand geworden.

Een ander nieuw gezichtspunt is het 'uit elkaar halen' van processoren en geheugens. Hiermee hangt het onderscheiden van communicatie naast interactie nauw samen. Men kan het echter ook gebruiken om bijvoorbeeld het toekennen van mutatierecht en inspectierecht op informatieverzamelingen in een organisatie te onderbouwen.

De definitie van het begrip entiteit, zoals gegeven in hoofdstuk 8, lijkt beter te passen bij de wijze waarop mensen de wereld observeren en erover redeneren dan de omschrijvingen van het begrip die men doorgaans aantreft in de database-literatuur. Een van de zaken, die door de gegeven definitie helder worden verklaard, is dat een entiteit een tijdgebonden concept is: het is een periode in het eeuwige bestaan van een object.

Een andere zaak, die duidelijk wordt gemaakt, is dat een entiteit wordt 'gecreëerd' door de definitie van een entiteittype en dus bepaald is door een tweetal <object, type>.
Omdat een object de referent van een aantal entiteiten kan zijn, zijn ook de begrippen subtype, supertype, specialisatie en generalisatie helder te definiëren.

Een hulpmiddel bij het modelleren is het in deel II geïntroduceerde aggregatie-concept, waardoor compositie en decompositie van systemen mogelijk is.
In de praktijk van de informatie-analyse heeft men echter ook behoefte aan aggregatieconcepten, met behulp waarvan acties en toestandselementen op een hoog 'aggregatie'-niveau kunnen worden gedefinieerd. Men denke bijvoorbeeld aan het beschouwen van het opnemen van een patiënt als één actie, die op alle patiënten van toepassing is, en waarbinnen men naar diverse gezichtspunten kan specialiseren (bijvoorbeeld naar leeftijd, naar medische indicatie of naar urgentie). Het generalisatie/specialisatie-concept in RML ([Greenspan, Borgida, Mylopoulos 1986]) is een voorbeeld van zo'n aggregatie-concept.
Ook zou men een aggregatievorm willen hebben, die het mogelijk maakt deelsystemen van eenzelfde soort te genereren en te vernietigen. Die behoefte ontstaat wanneer men het proceskarakter van entiteiten wil benadrukken door ze als systemen te beschouwen. Bijvoorbeeld zou men zinvol een rekening-courant of een patiënt als systeem kunnen opvatten. We bespraken dit aggregatie-concept al in paragraaf 11.4.

Eerste orde talen zijn goed gedefinieerd specificatietalen, maar zullen vooralsnog niet de prijs voor gebruiksvriendelijkheid in de wacht slepen. Het probleem met 'vriendelijke' talen echter is dat de semantiek ervan vaak zo ondoorzichtig is. Een wel goed gedefinieerde 'vriendelijke' kandidaat vormt wellicht de conceptuele graaf ([Sowa, Foo 1987]). Ter illustratie geven we hieronder de formule in eerste orde logica van de bewering dat een hond vier poten heeft:

$$\forall x: \text{hond}(x) \rightarrow$$
$$(\exists y: \exists z: \exists u: \exists v: \text{poot}(y) \wedge \text{poot}(z) \wedge \text{poot}(u) \wedge \text{poot}(v)$$
$$\wedge \text{deel}(y, x) \wedge \text{deel}(z, x) \wedge \text{deel}(u, x) \wedge \text{deel}(v, x)$$
$$\wedge y \neq z \wedge y \neq u \wedge y \neq v \wedge z \neq u \wedge z \neq v \wedge u \neq v)$$

De conceptuele graaf voor diezelfde bewering luidt:

[HOND: $\forall$] $\rightarrow$ (DEEL) $\rightarrow$ [POOT: {*} @4]

De betekenis van deze conceptuele graaf is de volgende. Voor elk individu van het type hond geldt dat hij een verzameling dingen als deel heeft. Die verzameling be-

staat uit vier, niet nader bepaalde, individuen van het type poot.

Een ander probleem bij het specificeren is de specificatie van de motor. Deze is, praktisch gesproken, onuitvoerbaar als men niet kan beschikken over hulpmiddelen waarmee de geldigheid van de integriteitsregels kan worden geverifieerd. Tevens is het nodig, dat naast statische axioma's ook dynamische axioma's kunnen worden geformuleerd en geverifieerd.

Het gezegde "Er is niets praktischer dan een goede theorie" betekent voor een theoreticus dat hij met de presentatie van een theorie nuttig werk heeft voltooid, en dus met alle recht aan de ontwikkeling van een nog betere kan beginnen.
Een practicus echter kent het verschil tussen praktisch zijn en in de praktijk gebruikt worden. Hij is eerder geneigd het gezegde te vervangen door "Er is niets onpraktischer dan een ongebruikte goede theorie".
Indien het dds-model een goede theorie is (en daar heeft het de schijn van), is het dus van belang in de nabije toekomst aandacht te schenken aan het praktisch toepasbaar maken ervan.

Als afsluiting van dit hoofdstuk, en daarmee van het boek, volgen hieronder drie onderwerpen, die ons inziens prioriteit verdienen in het voortgezet onderzoek naar het modelleren en specificeren van systemen:

- De ontwikkeling van een modelleermethode. Daarvoor zijn, zoals we hebben gezien, onder meer nieuwe aggregatieconcepten nodig.
- Voortzetting van de theoretische maar vooral ook van de praktische evaluatie van het dds-model, onder andere door toepassing in projecten met een 'realistische' omvang.
- De ontwikkeling van geautomatiseerde hulpmiddelen, die de constructie en de analyse van modellen ondersteunen.

In deel I is gesuggereerd dat formalisering van de informatie-analyse het kwaliteitsniveau, waarop informatiesystemen worden ontwikkeld, verhoogt.
De delen II en III tonen aan dat formalisering mogelijk is, terwijl deel IV de wenselijkheid ervan onderstreept. Het is te hopen dat informatie-analisten de noodzaak ervan zullen gaan inzien.

Literatuurverwijzingen

Abrial 1974	Abrial, J.R., Data Semantics, in: Klimbie J.W., Koffeman K.L., (eds), Data Base Management, North Holland Publishing Co., 1974
ANSI/SPARC 1975	Study Group on Data Base Management Systems, Interim report, in: ACM SIGMOD Newsletter, FDT, vol. 7, no. 2, 1975
Apostel 1960	Apostel, L., Towards the formal study of models in the non-formal sciences, Synthese, no. 12, 1960
Bachman 1969	Bachman, C.W., Data Structure Diagrams, ACM SIGBDP, Data Base 1, no. 2, 1969
Bemelmans 1986	Bemelmans, T.M.A., Bedrijfskundig ontwerpen van bestuurlijke informatiesystemen, in: Cornelis, P., Oorschot, J. van (eds), Automatisering met een menselijk gezicht, Kluwer, Deventer, 1986
Bemelmans 1987	Bemelmans, T.M.A., Bestuurlijke informatiesystemen en automatisering, Stenfert Kroese B.V., Leiden, 1987
Bertels, Nauta 1974	Bertels, K., Nauta, D., Inleiding tot het modelbegrip, Wetenschappelijke Uitgeverij BV, Amsterdam, 1974
Bodart, Pigneur 1983	Bodart, F., Pigneur, Y., Conception assistée des applications informatiques, Masson, Presses Universitaires de Namur, 1983
Borgida 1985	Borgida, A., Features of Languages for the Development of Information Systems at the Conceptual Level, IEEE Software, januari 1985
Brauer 1980	Brauer, W., (ed), Net Theory and Applications, Lecture Notes in Computer Science 84, Springer, Berlin, 1980
Brodie, Mylopoulos, Schmidt 1984	Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds), On Conceptual Modelling: Perspectives from Artificial

Intelligence, Databases, and Programming Languages, Springer-Verlag, New York, 1984

Checkland 1981 — Checkland, P., Systems Thinking, Systems Practice, John Wiley & Sons Ltd., 1981

Chen 1976 — Chen, P.P., The Entity-Relationship-Model: Toward a Unified View of Data, in: ACM TODS, vol. 1, no. 1, 1976

DeMarco 1978 — DeMarco, T., Structured Analysis and Systems Specification, Yourdon Press, N.Y., 1978

Dietz, Hee 1987 — Dietz, J.L.G., Hee, K.M. van, A framework for the conceptual modeling of discrete dynamic systems, Proceedings TAIS Conference, Sophia Antipolis, Frankrijk, 1987

Dubois e.a. 1986 — Dubois, E., Hagelstein, J., Lahou, E., Ponsaert, F., Rifaut, A., A Knowledge Representation Language for Requirements Engineering, Proc. IEEE, vol. 74, no. 10, october 1986

Elgerd 1967 — Elgerd, O.I., Control Systems Theory, Mc Graw-Hill, Inc., 1967

Emery 1969 — Emery, F.E., (ed), Systems Thinking, Penguin books Ltd., 1969

Floyd 1984 — Floyd, C., A systematic look at prototyping, in: Approaches to prototyping, Springer-Verlag, 1984

Gallaire 1986 — Gallaire, H., Bridging the gap between AI and Data bases: Logic Approach, Proc. IFIP TC2 Working Conference on Knowledge and Data (DS2), North-Holland Publ., 1986

Gane, Sarson 1977 — Gane, Ch., Sarson, T., Structured Systems Analysis: Tools & Techniques, Improved System Technologies, Inc., 1979

Greenspan, Borgida, Mylopolous 1986 — Greenspan, S.J., Borgida, A., Mylopoulos, J., A Requirements Modeling Language and its Logic,

Information Systems, vol. 11, no. 1, 1986

Griethuysen, Jardine 1984 — Griethuysen, J.J. van, Jardine, D.A., De INFOMOD-benadering van informatiemodellering, Informatie, jaargang 26, nr. 6, mei 1984

Gustafsson, Karlsson, Bubenko 1982 — Gustafsson, M.R., Karlsson, T., Bubenko jr., J.A., A Declarative Approach to Conceptual Information Modeling, Proc. IFIP TC8 Working Conference on Comparative Review of Information Systems Design Methodologies (CRIS-1), North-Holland Publishing Co., 1982

Hall 1962 — Hall, A.D., A Methodology for Systems Engineering, Van Nostrand, Princeton N.J., 1962

Hee, Houben, Dietz, 1987 — Hee, K.M. van, Houben, G.J., Dietz, J.L.G., Modelling of Discrete Dynamic Systems; framework and examples, Computing Science Notes, Technische Universiteit Eindhoven

Hunt 1954 — Hunt, M.M., Bell Labs 230 Long Range Planners, Fortune, mei 1954

IEEE 1983 — Special Issue on Knowledge Representation, IEEE Computer, vol. 16, no. 10, october 1983

ISO 1982 — Griethuysen, J.J. van (ed), Concepts and Terminology for the Conceptual Schema and the Information Base, ISO/TC97/SC5-N695

Jackson 1983 — Jackson, M., System Development, Prentice-Hall International, 1983

Langefors 1980 — Langefors, B., Infological models and information user views, Information Systems, vol. 5, 1980

Langerhorst 1981 — Langerhorst, R.P., Gegevensanalyse, Academic Service, Den Haag, 1981

Lloyd 1984 — Lloyd, J.W., Foundations of Logic Programming, Springer-Verlag, 1984

Lundeberg, Goldkuhl, Nilsson 1979	Lundeberg, M., Goldkuhl, G., Nilsson, A., A Systematic Approach to Information Systems Development, Information Systems, vol. 4, 1979
Meyer 1985	Meyer, B., On Formalism in Specifications, IEEE Software, januari 1985
Morris 1955	Morris, C.W., Signs, Language and Behavior, New York, 1955
Nauta 1972	Nauta, D., Jr., The meaning of information, Mouton, The Hague - Paris, 1972
Nielen 1976	Nielen, G.C., De bedoeling van informatie voor mens en organisatie, Samsom Uitgeverij, 1976
Ogden, Richards 1923	Ogden, C.K., Richards, I.A., The meaning of meaning, Harcourt, Brace & World, NY, 1923
Peirce 1960	Hartshorne, C., Weiss, P., (eds), Collected papers of C.S. Peirce, vol. I t/m IV, Cambridge,Massachusetts, 1960
Peursen 1968	Peursen, C.A. van, Bertels, C.P., Nauta, D., Informatie: een interdisciplinaire studie, Aula-boeken, Utrecht, Het Spectrum, 1968
Reiter 1978	Reiter, R., On Closed World Data Bases, in: Gallaire, H., Minker, J. (eds), Logic and Data Bases, Plenum Press, N.Y., 1978
Reiter 1984	Reiter, R., Towards a Logical Reconstruction of Relational Database Theory, in: [Brodie, Mylopoulos, Schmidt 1984]
Roman 1985	Roman, G-C., A Taxonomy of Current Issues in Requirements Engineering, IEEE Computer, april 1985
Ross 1975	Ross, D.T., Plex 1: Sameness and the need for rigor, and Plex 2: Sameness and Type, Softech, Inc., Waltham, MA, 1975

Shipman 1981 — Shipman, D.W., The Functional Data Model and the Data Language DAPLEX, ACM TODS, vol. 6, no. 1, 1981

Solvberg, Kung 1985 — Solvberg, A., Kung C.H., On Structural and Behavioral Modelling of Reality, Proc. IFIP WG 2.6 Working Conference on Database Semantics (DS-1), North-Holland Publishing Co., 1985

Sowa 1984 — Sowa, J.F., Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley P.C., 1984

Sowa, Foo 1987 — Sowa, J.F., Foo, N.Y., Conceptual Graphs for Knowledge Systems (in voorbereiding)

Sridhar, Hoare 1985 — Sridhar, K.T., Hoare, C.A.R., JSD expressed in CSP, Technical Monograph PRG-51, Oxford University Computing Laboratory, England

Stamper 1973 — Stamper, R., Information in Business and Administrative Systems, B.T. Batsford, London, 1973

Sundgren 1978 — Sundgren, B., Een raamwerk voor het ontwerpen van informatiesystemen, Informatie, jaargang 20, no. 1, januari 1978

Teichroew, Hershey 1977 — Teichroew, D., Hershey III, E.A., PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems, IEEE Transactions on Software Engineering, vol. 3, no. 1, januari 1977

Tsichritzis, Lochovsky 1982 — Tsichritzis, D.C., Lochovsky, F.H., Data Models, Prentice-Hall Inc., Englewood Cliffs, N.J., 1982

Ward, Mellor 1985 — Ward, P.T., Mellor, S.J., Structured Development for Real-Time Systems, Yourdon Press, 1985

Welke 1975 — Welke, R.J., A Conceptual Framework and Methodology for Describing and Evaluating Alternative Information Systems from a User or Organizational Perspective, dissertatie University of

Michigan, Ann Arbor, 1975

Yourdon, Constantine 1979 — Yourdon, E., Constantine, L.L., Structured Design, Prentice-Hall, Englewood Cliffs, N.J. 1979

Appendices

Dit deel bestaat uit een lijst van gebruikte symbolen (appendix A) en een drietal toepassingen van het dds-model (appendices B, C en D). De vorm, waarin de toepassingen worden gepresenteerd is dezelfde als die we gebruikten bij de behandeling van het postorderbedrjif (hoofdstuk 9).

A. Lijst van gebruikte symbolen

In deze appendix worden de in dit boek gebruikte wiskundige symbolen, waarvan de notatie of de betekenis afwijkt van de gangbare notatie of betekenis nader verklaard. Tevens wordt onder meer het niet zo gebruikelijke symmetrisch verschil van verzamelingen gedefinieerd.

- De notatie van de verzameling der reële getallen is R;

- De notatie van de verzameling der natuurlijke getallen is N;

- De *machtsverzameling* van een verzameling A wordt genoteerd als $P(A)$. De betekenis is: de verzameling van alle *eindige* deelverzamelingen van A;

- Het symmetrisch verschil van twee verzamelingen A en B wordt genoteerd als $A \Delta B$ en is als volgt gedefineerd:

  $$A \Delta B = (A \cup B) \setminus (A \cap B);$$

  Deze operator is *commutatief* en *associatief*.
  Zoals $A \cup B$ wordt gelezen als "A verenigd met B", zo wordt $A \Delta B$ gelezen als "A verschild met B".
  Zoals $\cup i \in D: A_i$ wordt gelezen als "De vereniging over D van alle A_i"; zo wordt $\Delta i \in D: A_i$ gelezen als "De verschilling over D van alle A_i".

- De cardinaliteit van een verzameling A wordt genoteerd als card (A).

- De betekenis van de symbolen "$\models$", $\Rightarrow$ $\Rightarrow$ $\Rightarrow$ en $\Rightarrow$ wordt verklaard in hoofdstuk 6. "M", "+", "-" "R"

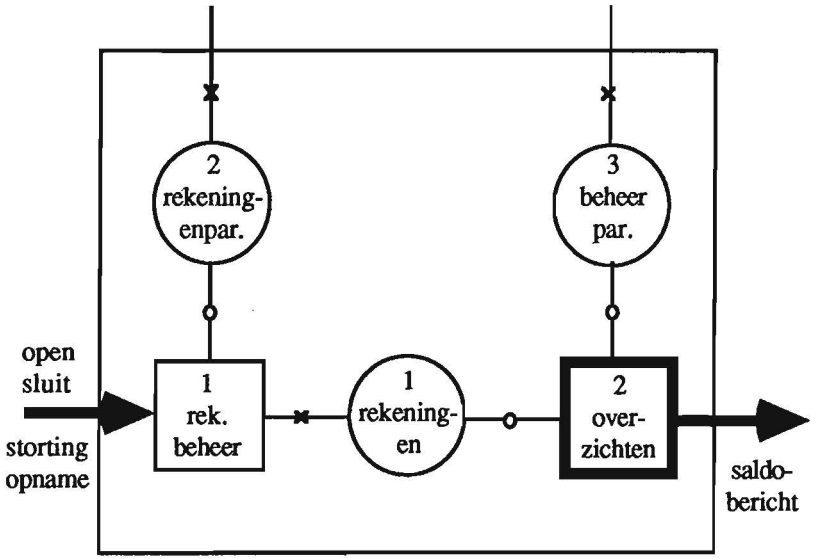
B. Voorbeeld: beheer bankrekeningen

Dit voorbeeld is een variant van een voorbeeld in [Jackson 1983], dat eerder werd behandeld in [Hee, Houben, Dietz 1987]. Het voorbeeld in de oorspronkelijke vorm is tevens gebruikt ter illustratie van de specificataal CSP in [Sridhar, Hoare 1985].

B1 Het reële systeem

Een bank beheert rekeningen-courant van cliënten. Cliënten kunnen rekeningen openen en opheffen. Op een geopende rekening zijn twee soorten transacties mogelijk: stortingen en opnames.
De enige voorwaarde voor het uitvoeren van een transactie is dat het rekeningsaldo niet lager wordt dan een bepaald minimum. Dit minimum is in principe voor elke rekening anders. Het wordt vastgesteld door de directie, en kan te allen tijde worden gewijzigd.
Periodiek wordt er een overzicht geproduceerd van de saldi van alle bestaande rekeningen. De periode wordt door de directie bepaald.

B2 Het N-schema



Figuur B.1: N-schema van beheer bankrekeningen

Processor 1 verwerkt open- en sluit-opdrachten van rekeningen, alsmede stortingen en opnames. Daartoe heeft de processor toegang, met mutatierecht, tot geheugen 1. Daarnaast mag processor 1 geheugen 2 inspecteren.
Processor 2 mag de inhoud van geheugen 1 alleen inspecteren. Hetzelfde geldt met betrekking tot geheugen 3. Via het terugkoppelkanaal activeert processor 2 zichzelf periodiek.

B3 De specificatie van S, A, R en T

Elke PS_j, PA_i, PR_i en PD_i is gespecificeerd in onderstaande predicatentabel (in kolom "n" is de ariteit vermeld).

pred verz	symbool	n	toelichting
PS1	rekening	1	rekening (r) betekent dat r een rekening-courant is.
	saldo	2	saldo (r, s) betekent dat s het saldo is van rekening r.
PS2	minimum	2	minimum (r, m) betekent dat m het minimum saldo is voor rekening r.
PS3	rapportperiode	1	rapportperiode (p) betekent dat de periode voor rapportage p dagen is.
PA1	open	2	open (r, s) stelt het openen voor van rekening r met beginsaldo s.
	sluit	1	sluit (r) betekent het opheffen van rekening r.
	storting	2	storting (r, b) betekent het storten van bedrag b op rekening r.
	opname	2	opname (r, b) betekent het opnemen van bedrag b van rekening r.
PD1	nieuwsaldo	2	<zie axioma's>
PA2	maakrapport	0	maakrapport is het signaal voor het maken van een rapport.
PR2	maakrapport	0	<zie PA2>

De component T is als volgt gespecificeerd:

T_{22} = {<maakrapport, maakrapport> | Θ};

B4 Het S-schema

Figuur B2: S-schema van beheer bankrekeningen

B5 De axioma's

Naast de axioma's, die impliciet zijn bepaald door het S-schema, zijn er nog de volgende:

nieuwsaldo (r, x) ↔ saldo (r, s) Λ x = s +
(Σy: storting (r, y): y) - (Σy: opname (r, y): y);

Het nieuwe saldo van een rekening is de algebraïsche som van de gestorte en de opgenomen bedragen.

rapportperiode (p) → int (p) Λ p > 0;

De periode van rapportage is een geheel aantal dagen groter dan nul.

B6 De specificatie van M

De motorfuncties van de onderscheiden processoren kunnen als volgt worden gespecificeerd.

M1 (rekeningenbeheer)

$\models$ open (r, s) $\wedge \neg$ rekening (r) $\wedge$ num (s) $\wedge$ s > 0

+
$\Rightarrow$ {rekening (r), saldo (r, s)}

$\models$ sluit (r) $\wedge$ rekening (r) $\wedge$ saldo (r, s) $\wedge$ s = 0

-
$\Rightarrow$ {rekening (r), saldo (r, s)}

$\models$ rekening (r) $\wedge$ saldo (r, s) $\wedge \neg$ open (r, s) $\wedge \neg$ sluit (r) $\wedge$
nieuw saldo (r, x) $\wedge$ minimum (r, m) $\wedge$ x $\geq$ m

-
$\Rightarrow$ {saldo (r, s)}

+
$\Rightarrow$ {saldo (r, x)}

Als voor een bestaande rekening een verzameling transacties moet worden verwerkt, waaronder zich niet het openen of het sluiten van de rekening bevindt, dan is het effect een saldowijziging, die gelijk is aan de algebraïsche som van de gestorte en de opgenomen bedragen, mits het nieuwe sldo niet lager is dan het toegestane minimum.

M2 (overzichten)

$\models$ maakrapport $\wedge$ saldo (r, s)

R
$\Rightarrow$ {<saldo-bericht (r, s), * > }

Van elke bestaande rekening wordt bij ontvangst van de actie "maakrapport" een saldo-bericht geproduceerd.

$\models$ maakrapport $\wedge$ rapportperiode (p)

R
$\Rightarrow$ {<maakrapport, p>}

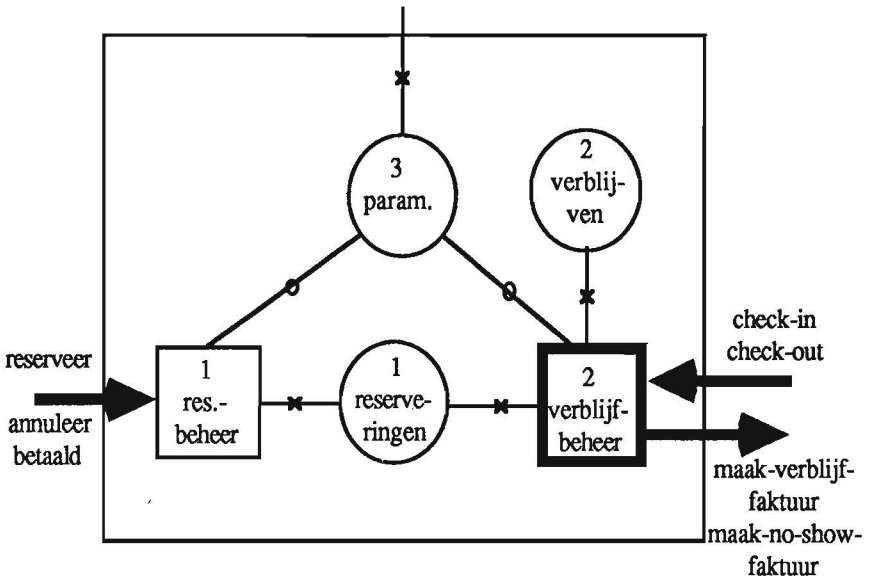
Elke p dagen wordt er een actie "maakrapport" gegenereerd.

C. Voorbeeld: hotelreserveringen

C1 Het reële systeem

In een hotel vindt het verhuren van kamers op de volgende wijze plaats.
Klanten van het hotel (dat zijn bijvoorbeeld bedrijven of reisagenten) kunnen reserveringen plaatsen. Een reservering betreft de toekomstige verhuur van een kamer van een bepaald type. Reserveringen kunnen worden geannuleerd, mits dat tijdig gebeurt.
Een reservering wordt als afgedaan beschouwd als hij is geannuleerd of als de bijbehorende factuur is voldaan. Er zijn twee soorten facturen: verblijfsfacturen en "no-show"-facturen. Een "no-show" betekent dat er op de dag van aankomst geen "check-in" voor een reservering heeft plaatsgevonden.
Met "check-in" wordt de gebeurtenis bedoeld dat een gast zich meldt als degene die van een reservering daadwerkelijk gebruik gaat maken. Een "check-in"-gebeurtenis is het begin van een verblijf.
Een verblijf eindigt op de vertrekdatum van de reservering. Er wordt dan een verblijfsfactuur opgemaakt. Het is ook mogelijk een verblijf te beëindigen vóór de afgesproken vertrekdatum. De gebeurtenis, waardoor een verblijf wordt beëindigd, heet een "check-out".

C2 Het N-schema



Figuur C1: N-schema van hotelreserveringen

C3 De specificatie van S, A, R en T

Elke PSj, PAi, PRi en PDi is gespecificeerd in onderstaande predicatentabel (in kolom "n" is de ariteit vermeld).

pred verz	symbool	n	toelichting
PS1	reservering	1	reservering (r) betekent dat r een reservering is
	res-klant	2	res-klant (r, l) betekent dat l de klant is van reservering r
	res-type	2	res-type (r, t) betekent dat t het kamertype is, behorend bij reservering r
	res-aankomst	2	res-aankomst (r, a) betekent dat a de aankomstdatum van reservering r is
	res-vertrek	2	res-vertrek (r, v) betekent dat v de vertrekdatum van reservering r is
PA1	reserveer	5	reserveer (r, l, t, a, v) is de opdracht tot het plaatsen van een reservering r met klant k, kamer-type t, aankomstdatum a en vertrekdatum d
	annuleer	1	annuleer (r) is de opdracht tot het annuleren van reservering r
	betaald	1	betaald (r) betekent dat de factuur voor reservering r is betaald
PD1	kamertype-vrij	2	kamertype-vrij (t, d) betekent dat er op datum d een kamer van type t vrij is
PS2	verblijf	1	verblijf (r) betekent dat r een verblijf is
	gast	2	gast (r, g) betekent dat g de naam van de gast is van verblijf r
	ver-kamer	2	ver-kamer (r, k) betekent dat aan verblijf r kamer k is toegewezen
PA2	check-in	2	check-in (r, g) betekent dat een gast met naam g zich meldt voor reservering r
	check-out	1	check-out (k) is de opdracht tot beëindiging van het verblijf met kamer k

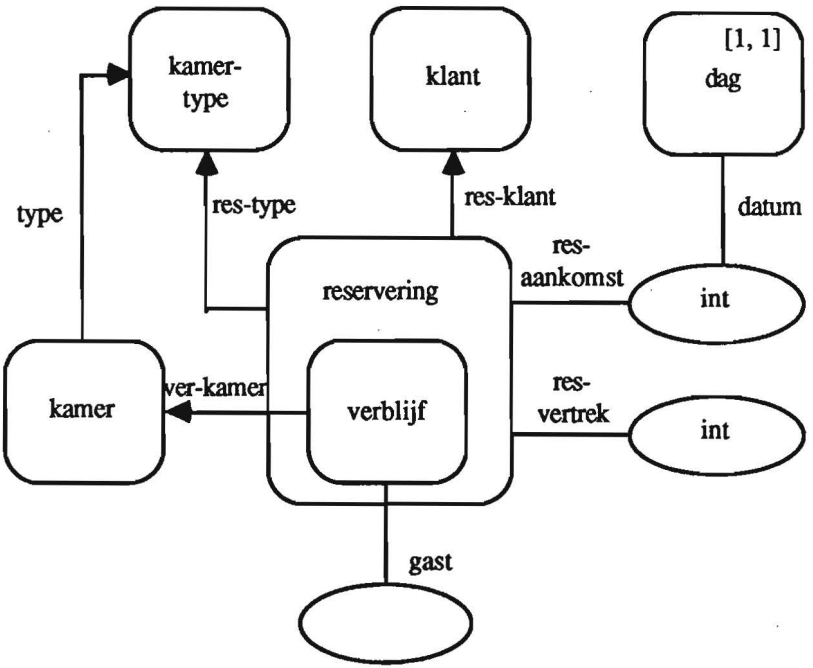
	check-in-controle	0	signaal om te controleren of er "no-shows" zijn
	check-out-controle	0	signaal om check-outs voor te bereiden
PR2	check-in-controle	0	<zie PA2>
	check-out-controle	0	<zie PA2>
	maak-verblijf-faktuur	1	maak-verblijf-faktuur (r) is een opdracht tot het maken van een verblijf-faktuur voor reservering r
	maak-no-show-faktuur	1	maak-no-show-faktuur (r) is een opdracht tot het maken van een no-show-faktuur voor reservering r
PD2	kamer-vrij	1	kamer-vrij (k) betekent dat k een vrije kamer is
	volgende-kamer	2	volgende-kamer (t, k) betekent dat kamer k de eerstvolgend toe te wijzen kamer van type t is
PS3	dag	1	dag (x) betekent dat x een dag voorstelt
	datum	2	datum (x, d) betekent dat d de datum van dag x is
	kamer	1	kamer (k) betekent dat k een kamer is
	type	2	type (k, t) betekent dat t het type van k is
PD3	vandaag	1	vandaag (d) betekent dat de datum van vandaag d is

---

De component **T** is als volgt gespecificeerd:

$$T_{22} = \{<\text{check-in-controle}, \text{check-in-controle}> \mid \Theta \} \cup \{<\text{check-out-controle}, \text{check-out-controle}> \mid \Theta\};$$

C4 Het S-schema



Figuur C2: S-schema van hotelreserveringen

C5 De axioma's

Aan de door het S-schema reeds impliciet bepaalde axioma's dienen de volgende te worden toegevoegd:

vandaag (d) $\leftrightarrow \exists$x: (dag (x) Λ datum (x, d));

De datum van vandaag is de datum van de (enige) dag

kamertype-vrij (t, d) $\leftrightarrow$ (# r: reservering (r) Λ res-type (r, t) Λ
res-aankomst (r, a) Λ res-vertrek (r, v) Λ d $\geq$ a Λ d < v)
< (# k: kamer (k) Λ type (k, t));

Er is op datum d een kamer van type t vrij als het aantal reserveringen op een kamer van type t, waarvoor d in de reserveringsperiode valt, kleiner is dat het totaal aantal kamers van type t

kamer-vrij (k) $\leftrightarrow \neg \exists$x: (verblijf (x) Λ ver-kamer (x, k));

volgende-kamer (t, k) $\leftrightarrow$ kamer (k) Λ type (k, t) Λ kamer-vrij (k) Λ
$\forall$x: (kamer (x) Λ type (x, t) Λ kamervrij (x) $\rightarrow$ x $\leq$ k);

De eerstvolgend toe te wijzen kamer van type t aan een verblijf is een 'willekeurige' vrije kamer; 'willekeurig' betekent dat er geen vrije kamer is met een 'groter' object

C6 De specificatie van M

M1 (reserveringenbeheer)

$\models$ reserveer (r, l, t, a, v) Λ concreet (r) Λ klant (l) Λ kamertype (t) Λ int(a) Λ int(v) Λ a < v Λ vandaag (d) Λ a $\geq$ d Λ
($\forall$x: x $\geq$ a Λ x < v: kamertype-vrij (t, x))

+

$\Rightarrow$ { reservering (r), res-klant (r, l), res-type (r, t), res-aankomst (r, a), res-vertrek (r, v) }

elke nieuwe reserveeropdracht waarvan de klant bekend is en waarvoor gedurende de reserveringsperiode een kamer van het gewenste type vrij is, wordt geaccepteerd

$\models$ annuleer (r) Λ reservering (r) Λ res-klant (r, l) Λ res-type (r, t) Λ res-aankomst (r, a) Λ res-vertrek (r, v) Λ vandaag (d) Λ d < a

-

$\Rightarrow$ { reservering (r), res-klant (r, l), res-type (r, t), res-aankomst (r, a), res-vertrek (r, v) }

een annulering minstens één dag voor de aankomstdatum wordt geaccepteerd

⊨ betaald (r) Λ reservering (r) Λ res-klant (r, l) Λ res-type (r, t) Λ res-aankomst (r, a) Λ res-vertrek (r, v)

-

⇒ { reservering (r), res-klant (r, l), res-type (r, t), res-aankomst (r, a), res-vertrek (r, v) }

Pas als er een bevestiging van betaling is ontvangen worden de gegevens van een reservering verwijderd

M2 (verblijfbeheer)

⊨ check-in (r, g) Λ abstract (g) Λ reservering (r) Λ res-aankomst (r, a) Λ vandaag (a) Λ res-type (r, t) Λ volgende kamer (t, k)

+

⇒ { verblijf (r), gast (r, g), ver-kamer (r, k) }

Van een gast die zich meldt aan de balie voor de check-in van reservering r, wordt de naam genoteerd bij het dan aanvangende verblijf. Een naam is een abstract concept.

⊨ check-in-controle Λ reservering (r) Λ ¬ verblijf (r) Λ res-aankomst (r, a) Λ vandaag (d) Λ d > a Λ res-klant (r, l) Λ res-type (r, t) Λ res-vertrek (r, v)

-

⇒ { reservering (r), res-klant (r, l), res-type (r, t), res-aankomst (r, a), res-vertrek (r, v) }

R

⇒ { < maak-no-show-faktuur (r), * >}

⊨ check-in-controle

R

⇒ {<check-in-controle, 1>}

Dagelijks gaat men na of er "no-show's" zijn.

Van een "no-show" is sprake als er geen check-in is geweest op de aankomstdag van een reservering.
In dat geval wordt een boetefaktuur opgemaakt.
Als een gast zich te laat meldt, is de reservering dus opgeheven.

⊨ check-out-controle Λ verblijf (r) Λ res-vertrek (r, v) Λ vandaag (d) Λ v = d Λ gast (r, g) Λ ver-kamer (r, k)

-

⇒ { verblijf (r), gast (r, g), ver-kamer (r, k) }

R

⇒ { <maak-verblijf-faktuur (r), * >}

⊨ check-out-controle

R

⇒ {<check-out-controle, 1>}

Dagelijks wordt van elke reservering, waarbij een actueel verblijf hoort, en waarvan de vertrekdatum de datum van vandaag is, het verblijf opgeheven. Tevens wordt een opdracht tot fakturering geproduceerd.

⊨ check-out-(k) Λ verblijf (r) Λ gast (r, g) Λ ver-kamer (r, k)

-

⇒ { verblijf (r), gast (r, g), ver-kamer (r, k) }

R

⇒ { <maak-verblijf-faktuur (r), * > }

Een check-out van een verblijf wordt ook geaccepteerd, als die vroeger is dan de vertrekdatum van de reservering.

Opmerking. Bij bovenstaande specificatie van de motor wordt verondersteld dat de acties check-out-controle en check-out (k) voor een kamer k niet tegelijkertijd optreden.

D. Voorbeeld: besturing liften

D1 Het reële systeem

Het probleem in dit voorbeeld is het ontwerp van de besturing van een liftsysteem, bestaande uit n liften, die dienen voor het vervoer van personen in een gebouw met m verdiepingen.
Het bestuurd systeem is als volgt nader te omschrijven. Er zijn n liftschachten. Op elke verdieping is er voor elke schacht een deur. In elke schacht bevindt zich een kooi, die door een krachtinstallatie omhoog en omlaag kan worden bewogen.
Dit systeem kan worden opgevat als een dds, dat informeel en summier als volgt kan worden beschreven.

Er zijn twee soorten acties:

ga (l, r): een actie waarvan het effect is dat de kooi van de lift l in de richting r wordt bewogen; de voorwaarde daarbij is dat de kooi stil staat op een verdieping. De deur wordt daarbij automatisch gesloten.

bezoek (l, v, r):
een actie waarvan het effect is dat de kooi van lift l in de wachttoestand wordt gebracht op verdieping v; de voorwaarde daarbij is, dat juist voordien de reactie "nadert (l, v, r)" is gegeven.

Er zijn twee soorten reacties:

nadert (l, v, r):
een reactie, die wordt afgegeven als de kooi van lift l verdieping v in de richting r op een zodanig korte afstand is genaderd, dat v de eerstvolgend te bereiken verdieping is. Anderzijds is er nog voldoende tijd na afgifte van deze reactie om de kooi op verdieping v te laten stoppen.

bezocht (l, v, r):
een reactie, die een bepaalde tijd na het in de wachttoestand komen van lift l op verdieping v wordt afgegeven. In die tijd hebben passagiers de gelegenheid gehad in en uit te stappen.
r is de richting die de kooi vóór het bezoek had.

Passagiers kunnen op de volgende manier transportopdrachten geven aan het liftsysteem.

1. Elke kooi heeft een paneel met knoppen, voor elke verdieping één. Door het indrukken van een knop geeft men een uitstapverzoek. De knop licht daarbij op.
   De verlichting wordt gedoofd zodra de corresponderende verdieping wordt bezocht.
2. Elke verdieping heeft twee knoppen waarmee instapverzoeken kunnen worden gegeven, één om te kennen te geven, dat men omhoog wil, en één om te kennen te geven, dat men omlaag wil.
   Op de hoogste verdieping ontbreekt de knop 'omhoog' en op de laagste verdieping ontbreekt de knop 'omlaag'.
   De knoppen lichten op indien ze worden ingedrukt en worden gedoofd zodra een van de liften, waarvan de richting overeenkomt met de gewenste richting, stopt op de betreffende verdieping.

Een lift die nog transportverzoeken heeft af te werken, heeft een bepaalde bewegingsrichting (omhoog of omlaag). Dat geldt ook als de kooi op een verdieping stil staat om passagiers in en uit te laten (de wachttoestand).

Bij het ontwerpen van het besturend systeem moeten de volgende randvoorwaarden in acht worden genomen.

1. Alle uitstapverzoeken die in een kooi worden gegeven dienen te worden ingewilligd, en wel in de volgorde waarin de kooi de betreffende verdiepingen passeert.
2. Alle instapverzoeken die vanaf een verdieping worden gegeven dienen te worden ingewilligd. Daarbij mag geen enkele verdieping worden bevoordeeld.
3. Als er voor een lift geen opdrachten meer zijn, blijft de kooi stil staan met geopende deur op de laatste bezochte verdieping (we zullen deze situatie de rusttoestand van een lift noemen).
4. Elke kooi heeft een alarmknop. Het aanzetten hiervan heeft tot gevolg dat alle uitstapverzoeken die in de kooi waren gegeven, vervallen. De lift is nu 'buiten dienst'. Een corresponderend signaal gaat aan op elke verdieping bij de deur van de betreffende liftschacht. Er is ook in elke kooi een mogelijkheid om de 'buiten dienst'-toestand op te heffen.

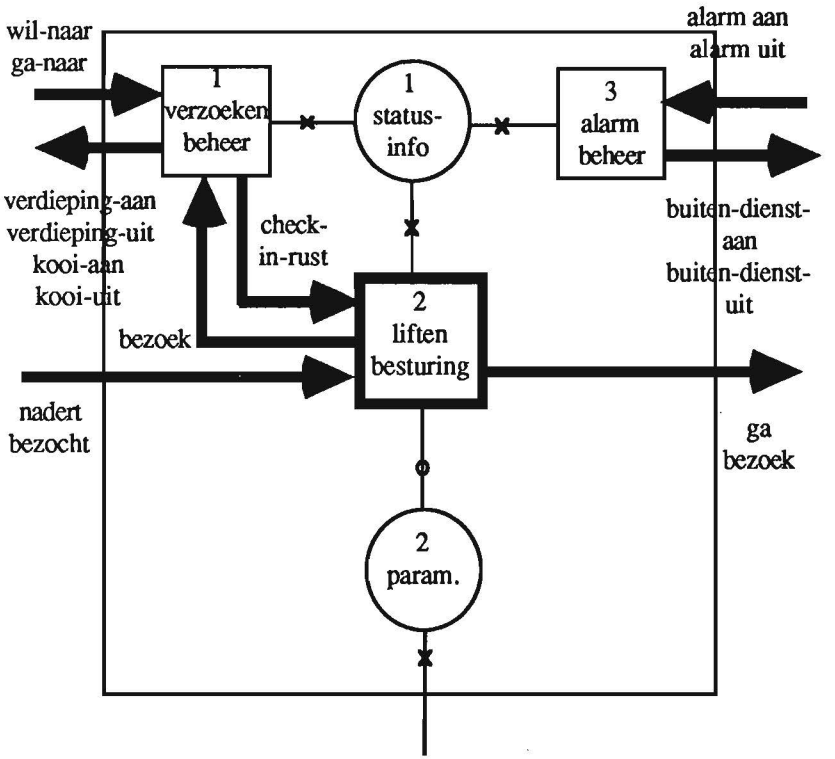
Na analyse van bovenstaande wensen eisen en randvoorwaarden, stellen we de volgende afspraken voor het noteren van liftsituaties voor.
De liften worden genummerd van 1 tot en met n. De verdiepingen worden opvolgend genummerd van 1 tot en met m, te beginnen bij de laagste verdieping.

De bewegingsrichting van een lift wordt genoteerd door +1 (omhoog) of -1 (omlaag). De regels voor het accepteren van uitstapverzoeken in een lift kan dan als volgt worden geformuleerd: als v de verdieping is waar de kooi zich bevindt (of die het laatst is gepasseerd), w de verdieping waar iemand wil uitstappen, en r de bewegingsrichting, dan wordt de opdracht geaccepteerd als (w - v) * r > 0.

Het hierna gespecificeerde ontwerp van het besturend systeem berust op de volgende uitgangspunten.
Het eerste is dat een lift zich óf in de werktoestand óf in de rusttoestand bevindt. Een lift die nog verzoeken heeft af te werken bevindt zich in de werktoestand.
Het tweede is dat een lift, die in een bepaalde richting beweegt, onderweg niet alleen uitstapverzoeken inwilligt maar ook instapverzoeken in dezelfde richting.
Het derde principe luidt dat een lift door blijft gaan in een bepaalde richting, totdat er helemaal geen opdrachten meer voor of vanaf verder gelegen verdiepingen zijn (dus ook geen instappers voor de tegenovergestelde richting). Als een lift op zo'n punt is aanbeland, keert hij zijn richting om en bezoekt de verdieping, waarop hij zich bevindt (zodat eventuele passagiers voor de nieuwe richting kunnen instappen). Alleen als er ook voor de nieuwe richting helemaal geen opdrachten zijn, die de lift zou kunnen inwilligen, gaat hij over van werktoestand naar rusttoestand.
Het vierde principe is, dat periodiek wordt nagegaan of het nodig is een lift in de rusttoestand weer in te zetten om de totale werklast sneller af te werken.

D2 Het N-schema

Het besturingssysteem is gemodelleerd als een netwerk, bestaande uit drie processoren (met indices 1, 2 en 3) en 2 opbergplaatsen (met indices 1 en 2). De interconnecties blijken uit het diagram, evenals de transactiekanalen.



Figuur D1: N-schema van besturing liften

D3 De specificatie van S, A, R en T

Elke PS_j, PA_i, PR_i en PD_i is gespecificeerd in onderstaande tabel (in kolom "n" is de ariteit vermeld).

pred verz	predikaatsymbool	n	toelichting
PS1	instapverzoek	2	instapverzoek (v, r) betekent dat er iemand op verdieping v staat te wachten op

			vervoer in de richting r.
	uitstapverzoek	2	uitstapverzoek (l, v) betekent dat er iemand in lift l naar verdieping v wil gaan.
	in-rust	2	in-rust (l, v) betekent dat lift l in de rusttoestand staat op verdieping v.
	buiten-dienst	1	buiten-dienst (l) betekent dat lift l niet meedoet aan het vervoer.
PA1	wil-naar	2	wil-naar (v, r) betekent dat er iemand op verdieping v te kennen geeft dat hij in de richting r vervoerd wil worden.
	ga-naar	2	ga-naar (l, v) betekent dat er iemand in lift l de paneelknop van verdieping v indrukt.
	bezoek	3	betekent dat lift l verdieping v bezoekt in de richting van r
PR1	check-in-rust	1	<zie PA2>
	verdieping-aan	2	verdieping-aan (v, r) heeft tot gevolg dat de met richting r corresponderende knop op verdieping v oplicht.
	verdieping-uit	2	verdieping-uit (v, r) heeft tot gevolg dat de met richting r corresponderende knop op verdieping v wordt gedoofd.
	kooi-aan	2	kooi-aan (l, v) heeft tot gevolg dat de paneelknop van verdieping v in de kooi van lift l oplicht.
	kooi-uit	2	kooi-uit (l, v) heeft tot gevolg dat de paneelknop van verdieping v in de kooi van lift l wordt gedoofd.
PD1	werk	3	<zie de axioma's>
	uitagenda	3	<zie de axioma's>
	inagenda	3	<zie de axioma's>
	voorkeursrichting	3	<zie de axioma's>
PS2	max	1	max (m) wil zeggen dat als het aantal "hangende" instapverzoeken groter is dan m, er zo mogelijk liften die in rust zijn worden ingeschakeld voor vervoer.
	periode	1	periode (p) wil zeggen dat de periode voor het herhaald testen van de conditie voor het inzetten van liften die in rust zijn, gelijk is aan p.
	aantal-verdiepingen	1	aantal-verdiepingen (m) wil zeggen dat er m verdiepingen zijn.

	aantal-liften	1	aantal-liften (n) wil zeggen dat er n liften zijn
PA2	nadert	3	nadert (l, v, r) betekent dat lift l verdieping v nadert in de richting r.
	bezocht	3	bezocht (l, v, r) betekent dat het bezoek van lift l aan verdieping v vanuit de richting r zojuist is beëindigd.
	check-in-rust	2	check-in-rust (l, v) is een actie om te controleren of lift l, als die in rust verkeert, niet geactiveerd moet worden.
	check-liftinzet	0	deze actie is bedoeld om te kijken of er niet meer liften voor vervoer moeten worden ingezet.
PR2	ga	2	ga (l, r) is een reactie, die wordt doorgegeven aan het bestuurd systeem en die beoogt lift l in de richting r te laten bewegen.
	bezoek	3	<zie PA1>
	check-liftinzet	0	<zie PA2>
PA3	alarm-aan	1	alarm-aan (l) betekent dat de alarmknop in lift l is ingedrukt.
	alarm-uit	1	alarm-uit (l, v) betekent dat de alarmtoestand van lift l wordt opgeheven, en dat lift l zich op verdieping v bevindt.
PR3	buiten-dienst-aan	1	buiten-dienst-aan (l) heeft tot gevolg dat op alle verdiepingen het signaal "buiten dienst" bij de deur van lift l wordt gegeven.
	buiten-dienst-uit	1	buiten-dienst-uit (l) heeft tot gevolg dat het signaal "buiten dienst" voor lift l wordt uitgezet.

De component **T** is als volgt gespecificeerd (waarin n het aantal liften en m het aantal verdiepingen voorstelt):

T_{12} = {<check-in-rust (l), check-in-rust (l) > | int (l) $\wedge$ l > 0 $\wedge$ l $\leq$ n};

T_{21} = {<bezoek (l, v, r), bezoek (l, v, r) > | int (l) $\wedge$ l > 0 $\wedge$ l $\leq$ n
$\wedge$ int(v) $\wedge$ v > 0 $\wedge$ v $\leq$m $\wedge$ (r = 1 $\vee$ r = -1) };

T_{22} = {<check-liftinzet, check-liftinzet> | Θ };

D4 Het S-schema

In de verzamelingen PSj zijn alleen de voor de besturing benodigde predicaatsymbolen opgenomen.
Om een zinvol S-schema te kunnen geven, zouden in de toestandsbasis ook grondatomen van bijvoorbeeld het type lift (l) en verdieping (v) opgenomen moeten worden. Deze zijn echter voor de beschrijving van het besturend systeem weinig interessant.
Het S-schema laten we daarom achterwege.

D5 De axioma's

We volstaan met de specificatie van de volgende definities:

werk (l, v, r) $\leftrightarrow$ uitstapverzoek (l, v) $\vee$ instapverzoek (v, r);

> Er is werk voor lift l, bewegend in de richting r, op verdieping v als er iemand daar wil uitstappen of als er iemand op verdieping v in de richting r wil instappen.

uitagenda (l, v, r) $\leftrightarrow \exists$w: (uitstapverzoek (l, w) $\wedge$ (w - v) * r > 0)

> De uitagenda van lift l op verdieping v voor de richting r is niet leeg als er minstens één passagier na verdieping v wil uitstappen.

inagenda (v, r) $\leftrightarrow \exists$w: ((w - v) * r > 0 $\wedge$ (instapverzoek (w, r) $\vee$ instapverzoek (w, -r))

> De inagenda, gezien vanaf verdieping v in de richting r is niet leeg als er op een van de verdiepingen na v iemand wil instappen. Het doet er daarbij niet toe of hij in dezelfde richting of in tegenovergestelde richting vervoerd wil worden.

voorkeursrichting (l, v, r) $\leftrightarrow$ (uitagenda (l, v, r) $\wedge$ r = 1) $\vee$
(uitagenda (l, v, r) $\wedge$ r = -1 $\wedge \neg$ uitagenda (l, v, -r))

De richting na de rusttoestand is de richting van een uitstapverzoek. Als er in beide richtingen uitstapverzoeken zijn, wordt de richting omhoog gekozen.

periode (p) → num (p) Λ p > 0;

D6 De specificatie van M

M1 (verzoekenbeheer)

|= wil-naar (v, r) Λ ¬ instapverzoek (v, r)

+
⇒ {instapverzoek (v, r)}

R
⇒ {<verdieping-aan (v, r), *>}

Voor alle nieuwe opdrachten "wil-naar (v, r)" wordt een instapverzoek (v, r)" bewaard. Tevens wordt het bijbehorende lampje aangezet.

|= ga-naar (l, v) Λ ¬ uitstapverzoek (l, v) Λ ¬ buiten-dienst (l)

+
⇒ {uitstapverzoek (l, v)}

R
⇒ {<kooi-aan (l, v), *>, <check-in-rust (l), *>}

Voor alle nieuwe opdrachten "ga-naar (l, v)" wordt een "uitstapverzoek (l, v)" bewaard. Tevens wordt het bijbehorende lampje aangezet. Bovendien wordt er een actie voor de liftenbesturder voorbereid om, in het geval lift l in rust mocht zijn, deze weer te laten werken.

|= bezoek (l, v, r) Λ instapverzoek (v, r)

-
⇒ {instapverzoek (v, r)}

R
$\Rightarrow$ {<verdieping-uit (v, r), *>}

Bij bezoek van een lift l aan een verdieping v in de richting r wordt een eventueel nog "hangend" instapverzoek ingewilligd. Tevens wordt het bijbehorende lampje gedoofd.

$\models$ bezoek (l, v, r) $\wedge$ uitstapverzoek (l, v)

$\Rightarrow$ {uitstapverzoek (l, v)}

R
$\Rightarrow$ {<kooi-uit (l, v), *>}

Bij bezoek van een lift l aan een verdieping v, wordt een eventueel "hangend" uitstapverzoek ingewilligd.

M2 (liftenbesturing)

$\models$ nadert (l, v, r) $\wedge$ werk (l, v, r)

R
$\Rightarrow$ {<bezoek (l, v, r), *>}

Als lift l verdieping v nadert in de richting r, en er zijn uitstappers of er zijn instappers in die richting, dan wordt verdieping v bezocht.

$\models$ nadert (l, v, r) $\wedge \neg$ werk (l, v, r) $\wedge \neg$ (uitagenda (l, v, r) $\vee$ inagenda (v, r))

R
$\Rightarrow$ {<bezoek (l, v, -r), *>}

Als lift l verdieping v nadert in de richting r, en er zijn geen uitstappers en ook geen instappers in die richting, en er zijn ook geen uitstappers of instappers (in beide richtingen) voor verder gelegen verdiepingen, dan wordt verdieping v toch bezocht, maar alsof de kooi uit de tegenovergestelde richting kwam. Op deze manier wordt een eventueel instapverzoek in die richting ingewilligd.

|= bezocht (l, v, r) Λ (uitagenda (l, v, r) ∨ inagenda (v, r))

R
⇒ {<ga (l, r), *>}

Als een bezoek van lift l aan verdieping v in de richting r is afgelopen, en er zijn nog uit- of instappers verderop, dan gaat de kooi verder in de richting r.

|= bezocht (l, v, r) Λ ¬ (uitagenda (l, v, r) ∨ inagenda (v, r))
Λ (uitagenda (l, v, -r) ∨ inagenda (v, -r))

R
⇒ {<ga (l, -r), *>}

Als er na een bezoek van lift l aan verdieping v in de richting r in diezelfde richting niets meer te doen is, maar wel in de tegenovergestelde richting, dan gaat de kooi in die tegenovergestelde richting verder.

|= bezocht (l, v, r) Λ ¬ (uitagenda (l, v, r) ∨ inagenda (v, r))
Λ ¬ (uitagenda (l, v, -r) ∨ inagenda (v, -r))

+
⇒ {in-rust (l, v)}

Als er na een bezoek van een lift aan een verdieping in beide richtingen geen opdrachten zijn, treedt de rusttoestand voor die lift in.

|= check-in-rust (l) Λ in-rust (l, v) Λ voorkeursrichting (l, v, r)

-
⇒ {in-rust (l, v)}

R
⇒ {<ga (l, r), *>}

Als de actie "check-in-rust (l)" wordt ontvangen en er is een passagier, dan gaat de lift l weer aan het werk.

|= check-liftinzet Λ in-rust (l, v) Λ max (m) Λ periode (p) Λ
(#x: instapverzoek (x, r)) > m

-
⇒ {in-rust (l, v)}

R
⇒ {<ga (l, -1)>}

Bij ontvangst van de (periodieke) actie "check-liftinzet" worden, als het aantal "hangende" instapverzoeken groter is dan een toegestane waarde, de liften "in-rust" weer actief, met beginrichting omlaag.

|= check-liftinzet Λ aantal-liften (n) Λ (#x: in-rust (x, v)) = n Λ
(#x: instapverzoek (x, r)) > 0 Λ in-rust (n, y)

-
⇒ {in-rust (n, y) }

R
⇒ {<ga (n, -1), * > }

Deze regel garandeert, dat in de situatie, dat alle liften in rust zijn, en er vanaf een verdieping waar geen lift staat te wachten een instapopdracht wordt gegeven, deze opdracht wordt uitgevoerd. Daartoe wordt lift n actief, met beginrichting omlaag.

|= check-liftinzet Λ periode (p)

R
⇒ {<check-liftinzet, p >}

M3 (alarmbeheer)

|= alarm-aan (l) Λ uitstapverzoek (l, v)

-
⇒ {uitstapverzoek (l, v)}

+
⇒ {buiten-dienst (l)}

R
⇒ {<buiten-dienst-aan (l), *>}

Als de alarmknop in lift l wordt ingedrukt vervallen alle "hangende" uitstapverzoeken, en licht op elke verdieping bij de betreffende schacht het signaal "buiten dienst" op.

⊨ alarm-uit (l, v) Λ buiten-dienst (l)

-
⇒ {buiten-dienst (l)}

+
⇒ {in-rust (l, v)}

R
⇒ {<buiten-dienst-uit (l), *>}

Als op verdieping v een lift die buiten dienst was weer in dienst wordt gesteld, begint hij in de rusttoestand.

Opmerking. In bovenstaande specificatie van de motorfuncties is verondersteld dat de actie "alarm-aan(l)" niet samenvalt met andere acties voor lift l.

Summary

In this book we present a technique for modeling and specifying discrete dynamic systems. The underlying framework or metamodel, called the dds-model, provides for a full and well-defined integration of the static and dynamic aspects of systems. A model, constructed within this framework, is a mathematical model. Due to the rigorous basis, the process of a system can also be described formally. Although the technique appears to be applicable to the analysis and synthesis of a broad class of systems, we have written the book with a particular application area in mind, viz. management information systems.
The book is divided in four parts.

Part I starts with an introduction into the area of management information systems and the problems related to their development. The attention is focused on the *requirements engineering* phase. We suggest that a formally defined conceptual model of the "business system" or "Universe of Discourse" is the necessary basis for discussing information requirements.
In chapter 2 we give an account of the meaning of models, particularly of conceptual models. A classification of models is presented, by means of which the meaning of the terms *conceptual model* and *information system* is clarified.
In chapter 3 we define the class of *discrete dynamic systems*. The principal difference with the conventional notion of a system is that *communication* is discerned as a separate way of mutual influencing between systems, next to *interaction*.

Part II sets the theoretical basis. It starts with an informal description of the conceptual framework for modeling systems. Basically, a conceptual model of a discrete dynamic system is a network consisting of four types of components: processors, stores, transaction channels and interconnections. The state of a system is distributed over the *stores* of the network. *Processors* are connected to stores via *interconnections*. The contents of all stores to which a processor is connected is collectively called the *state* of the processor.
Processors are connected to each other by means of *transaction channels*. Through these channels actions are transferred between processors. Upon reception of a set of actions, a processor produces a set of *mutations* and a set of (delayed) *reactions*. The mutations lead to a state transition. The reactions are transferred as future actions to other processors, or to itself.
In chapter 5 we formalize the metamodel. A conceptual model is defined as a tuple <S, M, A, R, T, I>, of which the components are sets or functions. By adding a component E, representing the *environmental influences*, a particular *process* is determined.
A process is formally described as a set of time-varying functions. In order to

compose and decompose systems, the concept of *aggregation* of a system is formally defined.
A specification of the components of the above mentioned tuple defines a particular system. The specification language, proposed in chapter 6, is a *first order language*, to which a facility is added for the specification of mutations and reactions.

Part III discusses several practical issues.
Two diagramming techniques are presented, called the N-schema and the S-schema. The *N-schema* is a technique for describing the *network structure* of a system. An *S-schema* describes a large part of the *state space* of a processor.
Both schemas are formally defined, and can therefore be used as part of the specification of a model. This is demonstrated in chapter 9 from the example of a mail-order business.

In part IV we evaluate the proposed technique. This is done by using the dds-model as a reference framework in discussing other techniques.
In chapter 10 four diagramming techniques are compared, viz. the ISAC A-schema, the SA/SD Data Flow Diagram, the Entity-Relationship-Diagram and the Data Structure Diagram.
Other, more complete and more formally based, techniques are discussed in chapter 11. They comprise so-called Conceptual Modeling Languages, the Finite State Machine, the Petri-net and Jackson System Development.
Chapter 12 presents concluding remarks and suggestions for future research.

The book is completed by the detailed description of the conceptual models of three examples: a banking system, a hotel reservation and the control of a lift system.

Curriculum Vitae

De auteur van dit proefschrift werd op 20 juni 1945 geboren te Brunssum.
In 1963 behaalde hij het diploma gymnasium-β aan het St.-Thomascollege te Venlo, en begon hij zijn studie aan de Technische Universiteit Eindhoven, in de faculteit der Electrotechniek. Het ingenieursdiploma werd in 1970 behaald. Het afstudeeronderzoek lag op het gebied van de meet- en regeltechniek.
Van 1970 tot 1973 werkte hij als systeemanalist/ontwerper en projectleider bij de machinefabrieken van de N.V. Philips' gloeilampenfabrieken.
Van 1973 tot 1981 was hij verbonden aan het Rekencentrum van de Technische Universiteit Eindhoven, voor een deel van de tijd als financieel-economisch adjunct-directeur en voor een ander deel als onderzoeker op het gebied van de administratieve gegevensverwerking.
Sinds 1981 is hij als wetenschappelijk hoofdmedewerker in dienst bij de faculteit Bedrijfskunde van dezelfde Universiteit, in de vakgroep Bestuurlijke Informatiesystemen en Automatisering.
Hij is lid van IFIP werkgroep 8.1 (Information Systems).

STELLINGEN

behorend bij het proefschrift

Modelleren

en

Specificeren

van

Informatiesystemen

van

Jan Dietz

I

Het dds-model is een adequaat formalisme voor het conceptueel modelleren van discrete dynamische systemen. De in hoofdstuk 6 gepresenteerde specificatietaal moet echter nog nader worden onderzocht op praktische bruikbaarheid.

Lit.: dit proefschrift, deel II.

II

De in hoofdstuk 3 onderscheiden vormen van wederzijdse beïnvloeding tussen systemen, namelijk interactie en communicatie, zijn wezenlijk verschillend.

Lit.: dit proefschrift, hoofdstuk 3.

III

De aggregatieconcepten, die ten grondslag liggen aan het ISAC A-schema en het SA/SD DFD, deugen niet.

Lit.: dit proefschrift, hoofdstuk 10.

IV

Bij het conceptueel modelleren is het van belang concepten en relaties tussen concepten te onderscheiden van de representaties ervan. In veel methoden voor gegevensmodellering wordt dit onderscheid niet of onvoldoende gemaakt.

V

Het symmetrisch verschil van verzamelingen is een adequate operatie om op conceptueel niveau toestandsveranderingen te specificeren.

Lit.: dit proefschrift, deel II.

VI

Het gebruik van schematechnieken met een slecht gedefinieerde semantiek leidt gemakkelijk tot schijnbegrip. Computerondersteuning van die technieken lijkt dit alleen maar te kunnen verergeren.

VII

Tussen de in [1] voorgestelde basisbegrippen entiteit en propositie is geen consistent onderscheid te maken. Enerzijds wordt een propositie gedefinieerd als een uitspraak betreffende entiteiten. Anderzijds blijkt een entiteit equivalent te zijn aan een (unaire) propositie.
De begrippen entiteit en propositie, zoals in [1] gedefinieerd zijn daarom ongeschikt als primitiva in conceptuele modellen.

[1]: Griethuysen, J.J. van (ed), Concepts and Terminology for the Conceptual Schema and the Information Base, ISO/TC97/SC5-N695, hoofdstuk 2.

VIII

Het ontwerpen van informatiesystemen, die een effectieve bijdrage leveren aan het bereiken van de doelstellingen van een organisatie, vereist bedrijfskundig inzicht. In slechts weinig ontwikkelingsmethoden wordt hiermee expliciet rekening gehouden.

Lit.: Bemelmans, T.M.A., Bestuurlijke Informatiesystemen en automatisering, Stenfert Kroese, 1987, i.h.b. hoofdstuk 2.

IX

De bestuurlijke informatiekunde vervult een brugfunctie tussen bedrijfskunde en informatica. De bedrijfskundige ontwerpt organisaties en de besturing daarvan, de (bestuurlijk) informatiekundige ontwerpt de specificaties van de voor die besturing benodigde informatiesystemen, en de informaticus ontwerpt efficiënte implementaties van die systemen.

Lit.: Boer, J.G. de, Dietz, J.L.G., Bestuurlijke informatiekunde: een vak apart? Informatie jaargang 25 nr. 3, pag. 6 - 8, maart 1983.

X

Wie het inbreken in computers een sport noemt, begaat een ernstige rubriceringsfout: sport ressorteert onder het ministerie van WVC, misdaden vallen onder justitie.

XI

Antropomorfiseringen van computers, zoals in "de computer heeft beslist, dat ..." of "de computer had toch gelijk" verschaffen computers een schijn van eigenmachtigheid, waardoor menselijke verantwoordelijkheden gemakkelijk kunnen worden verhuld. Het gebruik ervan dient daarom krachtig te worden bestreden.

XII

De maatschappelijke overwaardering van handel en industrie ten opzichte van wetenschap en kunst is treffend vervat in de gangbare betekenis van het gezegde "ergens wijzer van worden".

Eindhoven, 20 september 1987
J.L.G. Dietz