De vaste waarden van de systeemontwikkeling

Maurice Verhelst
K.U.Leuven, Departement Toegepaste Economische Wetenschappen
Maurice.Verhelst@econ.kuleuven.ac.be

Abstract

De informatica evolueert zeer snel. Nieuwe programmeertalen, nieuwe ontwerpmethoden, nieuwe technieken allerhande duiken op. De vraag hierbij is: wat is "hype" en wat is blijvend ? Zeer dikwijls worden bij toepassing van de nieuwigheden beproefde werkwijzen uit het verleden overboord gegooid met alle gevolgen van dien. In dit artikel wordt betoogd dat men er best aan doet doorheen de snelle evolutie een aantal onwijzigbare principes te blijven toepassen. Eerst worden de doelstellingen van systeemontwikkeling onder de aandacht gebracht. Daarna introduceren wij de genoemde principes en tonen we hoe deze zich verhouden tot de besproken doelstellingen.

Inhoudstafel

Abstract 1

1 Inleiding 3

2 De doelstellingen 4
2.1 Betrouwbaarheid 4
2.2 Verbeterbaarheid 4
2.3 Flexibiliteit 4
2.4 Doorzichtigheid 5
2.5 Efficiëntie 5
2.6 Gebruiksvriendelijkheid 5

3 De remedies 5
3.1 Ruimtelijk denken 6
3.2 Programmastructuur = Probleemstructuur 7
3.3 De objectoriëntatie 8
3.4 Modelmatige ontwikkeling 9
3.4.1 Een architectuur die leidt tot flexibele systemen 10
3.4.2 Een voorbeeld: de vereenvoudigde bank 13
3.5 Specificatie en implementatie 21
3.6 Beslissingstabellen 27
3.6.1 Definitie 27
3.6.2 Het nut van beslissingstabellen 31
3.6.2.1 Toepassing van procedures 32
3.6.2.2 Controle op volledigheid, contradictie en juistheid 34
3.6.2.3 Opstelling van voorwaardelijke uitspraken 37
3.6.3 Relevantie voor de informatica 37
3.7 Soberheid 38

4 Besluit 39

Referenties 39

1 Inleiding

Na het initieel enthousiasme over de automatisering van de bestuurlijke informatieverwerking, begon men zich, aan het einde van de jaren 60, te realiseren dat het allemaal niet zo goed verliep. Er was sprake van 'software crisis', een begrip voor het eerst gebruikt op een door de NATO gesponsorde conferentie in 1968 [6].

'Software crisis' sloeg op het feit dat informatiesystemen uitermate star waren en moeilijk te veranderen. Dit was zeer ernstig, want aangezien wetgeving, marktomstandigheden en technologie steeds sneller veranderen, moeten informatiesystemen ook steeds sneller kunnen inspelen op hieruit voortvloeiende wijzigingen in de informatiebehoeften. Binnen de informatiemaatschappij is systeemflexibiliteit een strategische succesfactor. Niettemin klaagden managers erover dat zelfs kleine wijzigingen aan het systeem onmogelijk te realiseren waren. Volgens hun informatici zou een dergelijke kleine verandering van de vereisten een zodanig grote reorganisatie van het systeem veroorzaken dat de wijziging economisch niet haalbaar was. Ook strikt noodzakelijke wijzigingen vroegen onvoorstelbaar veel tijd.

Dit alles leidde tot de situatie dat de backlog aan gevraagde wijzigingen in oude programma's en voor de creatie van nieuwe steeds aangroeide. Studies wezen uit dat in veel bedrijven 80 % van de tijd besteed werd aan het veranderen van bestaande programma's en slechts 20 % aan het opstellen van nieuwe.

Dit was het einde van de jaren zestig. 35 jaar later moeten we vaststellen, dat ondanks nieuwe programmeertalen, nieuwe methoden en hypes allerhande, de toestand niet veel is verbeterd. Een recente studie in de USA bvb. heeft uitgewezen dat op dit ogenblik 30 % van de grote informatica-projecten afgeblazen worden vooraleer ze beëindigd zijn, 50 % van de projecten het budget met meer dan 200 % overschrijden en dat de meerderheid van de afgewerkte projecten slechts 60 % of minder van hun vooropgestelde functionaliteit verschaffen [2].

Hier is duidelijk iets eigenaardigs aan de gang. Het vermoeden is groot, dat de oorzaak van dit alles te zoeken is in de obsessie om steeds het laatste snufje van de ontwikkeling te gebruiken, zonder zich te realiseren dat er voor het ontwerp van informatiesystemen basisprincipes bestaan die niet straffeloos mogen verwaarloosd worden. Men is blijkbaar verblind door het nieuwe en gooit al het vroeger aangeleerde overboord als oude koek. Iedere nieuwe "hype" belooft de hemel op aarde, maar leidt uiteindelijk vaak tot chaos. Soms speelt zich hierbij een processie van Echternach af: twee stappen vooruit en één achteruit (of in sommige gevallen één stap vooruit en twee achteruit).

Hierop aansluitend en deze tendens versterkend, wordt soms het verkeerde personeel ingezet: ontwerpers en programmeurs met als enige informaticakennis wat zij geleerd hebben in een snelcursus over de laatste nieuwigheid, maar die niets afweten van abstracte ontwerpprincipes, die onafhankelijk van de gehanteerde techniek steeds geldig blijven.

Deze bijdrage heeft tot doel de aandacht te vestigen op de voornaamste onveranderlijke basisprincipes, die telkens opnieuw moeten vertaald worden naar de nieuwigheid die zich aandient.

Hiertoe presenteren we eerst in paragraaf 2 de doelstellingen die aan de grondslag zouden moeten liggen van elk softwareproject. In paragraaf 3 worden dan de genoemde principes onder de aandacht gebracht.

2 De doelstellingen

Bij het ontwerpen van een systeem of van een programma dient men zes doelstellingen na te streven: betrouwbaarheid, verbeterbaarheid, flexibiliteit, doorzichtigheid, gebruiksvriendelijkheid en efficiëntie.

2.1 Betrouwbaarheid

Een systeem is betrouwbaar in de mate dat er weinig fouten in optreden. Het moet dus juist werken. In een groeiend percentage gevallen wordt betrouwbaarheid aangezien als de voornaamste doelstelling bij de systeemontwikkeling. Deze tendens is een natuurlijk uitvloeisel van twee verschijnselen: (1) een steeds groter gedeelte van de administratieve en technische processen in het bedrijf en in de maatschappij wordt uitgevoerd of bestuurd door computerprogramma's; (2) de te implementeren systemen worden steeds ingewikkelder, en complexiteit is op zichzelf al een foutenbron. Omdat in vergelijking met vroeger een groter gedeelte van de maatschappij computerafhankelijk is, hebben software-fouten grotere gevolgen dan voorheen. Het wordt dus steeds dwingender fouten te voorkomen.

2.2 Verbeterbaarheid

Een systeem is verbeterbaar in de mate dat het weinig tijd en inspanning vergt om optredende fouten te localiseren en te herstellen, *zonder dat hierbij op andere plaatsen nieuwe fouten worden gecreëerd*. De laatste toevoeging is van belang. Immers, soms is het te verkiezen niet-cruciale fouten onverbeterd te laten, namelijk indien bij verbetering een grote kans bestaat nieuwe fouten binnen te brengen.

2.3 Flexibiliteit

Systemen zijn aan een steeds snellere evolutie onderhevig; onze samenleving wordt immers met de jaren dynamischer. Flexibiliteit wordt een norm voor kwaliteit van bedrijfsvoering. Het gemak waarmee een systeem kan gewijzigd worden is daarom een belangrijke dimensie van de kwaliteit van het systeem.

2.4 Doorzichtigheid

Als een lezer van de systeemdocumentatie de juiste werking van het systeem snel begrijpt, dan is het doorzichtig. Men ziet gemakkelijk in dat er een positief verband bestaat tussen de doorzichtigheid en de hierboven reeds besproken doelstellingen: naargelang een systeem doorzichtiger wordt, zal het ook betrouwbaarder, verbeterbaarder en flexibeler (aanpasbaarder) zijn.

2.5 Efficiëntie

Efficiëntie kan bepaald worden als de mate waarin de programma's waaruit het systeem bestaat weinig geheugen nodig hebben en/of een snelle uitvoering te zien geven. Hoe minder geheugenruimte en/of hoe sneller de uitvoering, hoe efficiënter deze programma's zijn.

In uiterst zeldzame gevallen bestaat er een conflict tussen de vier eerste doelstellingen enerzijds en efficiëntie anderzijds, m.a.w. de systeem-structuur die leidt tot een systeem met een goede graad van betrouwbaarheid, verbeterbaarheid, aanpasbaarheid en doorzichtigheid is in uiterst zeldzame gevallen onvoldoende efficiënt. De beste strategie bestaat er dan in zich bij het ontwerpen van de systeemstructuur toch niet te bekommeren om de efficiëntie, maar enkel om het bereiken van de overige vier doelstellingen. Werd op deze wijze een systeemstructuur ontworpen, dan pas gaat men na of het aangewezen is de structuur te veranderen met de bedoeling de efficiëntie te verhogen. Deze strategie maakt het mogelijk beter de consequenties van een verhoogde efficiëntie op het bereiken van de andere doelstellingen te onderkennen en een bewuster keuze te doen.

Daarenboven blijkt het nastreven van de eerste vier doelstellingen in de meerderheid van de gevallen de efficiëntie te bevorderen. Een doorzichtig systeem bvb. maakt het mogelijk gemakkelijk de plaatsen aan te duiden waar lokaal de efficiëntie kan verbeterd worden, zonder de efficiëntie op andere plaatsen hierdoor te verminderen. Deze faciliteit heeft men niet bij een ondoorzichtig systeem.

2.6 Gebruiksvriendelijkheid

Een systeem is gebruiksvriendelijk in de mate dat het de informatie op het scherm op een gemakkelijk leesbare wijze voorstelt en in de mate dat het de gebruiker op een klare wijze aanduidt wat hij/zij moet doen.

Hoe gebruiksvriendelijker een systeem, hoe minder kans er bestaat dat de gebruiker inputfouten zal maken en hoe nuttiger het systeem wordt.

3 De remedies

Reeds in 1968 merkte Edsger Dijkstra op, dat veel programmeurs "spaghetti"-programma's schreven, hierbij op een gevatte manier aanduidend dat hun programma's geen enkele zinvolle structuur vertoonden. [3] Als gevolg hiervan

kon de correctheid van de programma's niet geverifieerd worden en waren ze verschrikkelijk moeilijk te wijzigen. Daarna heeft het vijf tot zeven jaar geduurd vooraleer dit idee van gestructureerd programmeren doordrong tot het data processing publiek. En zelfs op dit ogenblik wordt, in de meerderheid der gevallen, gestructureerd programmeren niet correct aangeleerd. Veelal worden immers enkel de technieken van gestructureerd programmeren aangebracht, maar de studenten worden niet getraind in het gebruik van de fundamentele denkwijze die achter de technieken schuilgaat. En juist deze denkwijze is het essentiële van gestructureerd programmeren.

De twee belangrijkste elementen waaruit de bewuste denkwijze bestaat zijn "ruimtelijk denken in plaats van sequentieel denken" en "afleiding van de programmastructuur uit de probleemstructuur", voor het eerst op een concrete wijze geïntroduceerd door Michael Jackson [4]. Deze twee denkpatronen zijn zeer krachtig, maar zij vereisen een zekere vaardigheid tot abstract denken en dus voldoende training. Men kan ze niet leren door ze louter theoretisch te verstaan. Als men daarenboven gedurende een zekere tijd op een andere wijze software heeft ontworpen, moet men eerst veel verkeerde denkpatronen, die ondertussen diepe wortels hebben geschoten, afleren. En diep gewortelde denkpatronen afleren is veel moeilijker dan nieuwe patronen aanleren. Dit verklaart waarom de meerderheid der gevorderde programmeurs niet bekwaam is gestructureerd te programmeren op de wijze zoals voorzien. Zij gebruiken eventueel wel de technieken, maar niet het denkpatroon. Zij passen dus de essentieelste en krachtigste elementen van gestructureerd programmeren niet toe.

3.1 Ruimtelijk denken

Zoals gezegd, is het eerste element van het denkpatroon "ruimtelijk denken in plaats van sequentieel". Traditioneel hebben programmeurs geleerd computertaal-instructies te schrijven in de volgorde waarin de computer de instructies moet uitvoeren. Het devies is: denk eerst aan wat de computer eerst moet doen, en dan aan wat hij vervolgens moet doen. Dit is een denkpatroon dat diepe wortels heeft, maar volledig verkeerd is. Het kan vergeleken worden met de werkwijze waarbij een architect het plan van een gebouw zou tekenen in de volgorde waarin het zal gebouwd worden: eerst de fundamenten, dan de eerste verdieping, dan de tweede verdieping, enz. Dit zou als volkomen onaanvaardbaar bestempeld worden. Spijtig genoeg is het echter nog steeds de wijze waarop veel programma's heden worden ontwikkeld. Architecten maken een duidelijk onderscheid tussen ontwerp en uitvoering. Zij weten dat zij de fundamenten van het gebouw niet kunnen ontwerpen zonder meer te weten over het gebouw dat er op moet rusten; zij maken dus hun ontwerp in een volgorde die totaal verschillend is van deze waarin het gebouw zal worden opgetrokken.

Zoals een architect, zou een programmeur een stuk papier moeten nemen en ergens in het midden beginnen of soms zelfs onderaan, instructies toevoegend boven, onder en tussen wat hij op dit moment heeft. Men kan bijvoorbeeld niet

correct beslissen wat de initialisatie van een programma dient te zijn, vooraleer men de centrale functie van het programma ontworpen heeft. In dit verband moet worden opgemerkt dat het gebruik van flowcharts bij het programmeren een verderfelijke praktijk is; flowcharts behoren tot de slechtste technieken die men bij het programmeren ooit heeft gebruikt. Ze dwingen de programmeur sequentieel te denken en verhinderen juist daarom dat hij ruimtelijk denkt.

Een programmeur die ruimtelijk denkt zal veel sneller programmeren en daarenboven veel correcter programma's ontwerpen.

3.2 Programmastructuur = Probleemstructuur

Het tweede element van het denkpatroon bestaat uit het ontwikkelen van de programmastructuur uit de probleemstructuur. Dit betekent dat men bij het bekijken van het programma duidelijk de onderscheiden elementen moet zien van het probleem dat door het programma wordt behandeld. De probleemstructuur moet weerspiegeld zijn in de programmastructuur. Daar is een zeer goede reden voor: een programma dat de probleemstructuur weerspiegelt kan gemakkelijk gewijzigd worden. Indien we inderdaad spreken over een vereiste wijziging aan het programma, specificeren we in wezen een probleemwijziging. Als het programma precies dezelfde structuur heeft als het probleem, kunnen we meteen het gedeelte van het programma aanwijzen dat moet worden aangepast. De kans is dan groot dat een dergelijke wijziging snel en accuraat door te voeren is. Indien daarentegen het programma een structuur vertoont die haaks staat op de probleemstructuur, zal het zeer vaak voorkomen dat de wijziging slechts kan doorgevoerd worden door een groot deel van het programma te herschrijven, of door een gans nieuw programma te maken.

'Echt' gestructureerd programmeren kan een belangrijke bijdrage leveren tot het verminderen van de softwarecrisis. Het is echter tragisch vast te stellen, dat veel programmeurs zichzelf een rad voor de ogen draaien als ze denken gestructureerd te programmeren. Dit verklaart in grote mate waarom gestructureerd programmeren de beloften niet heeft ingelost.

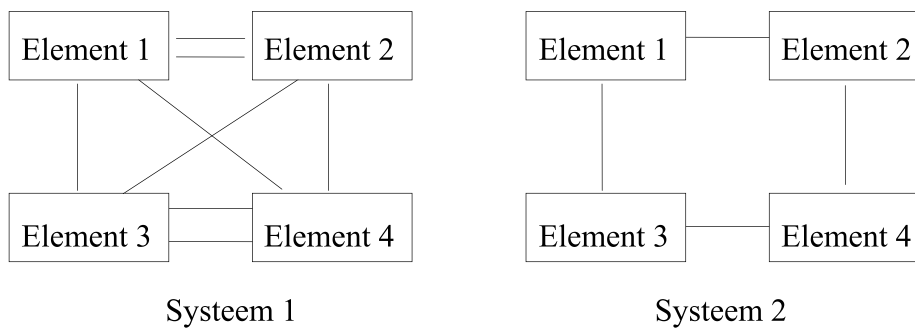
Als de programmastructuur een weerspiegeling moet zijn van de probleemstructuur, stelt zich de vraag waar men deze probleemstructuur moet gaan zoeken. In dit verband zijn er twee tegengestelde scholen: de functionele school (o.a. Myers, Yourdon, Constantine, Gane & Sarson) en de data-school (Warnier, Jackson). Het onderscheid tussen beide vloeit voort uit de wijze waarop zij tegen een programma aankijken. De functionalisten zien een programma als een algoritme om een functie uit te voeren, terwijl de aanhangers van de dataschool een programma zien als een algoritme om inputdata om te zetten tot outputdata.

Na verloop van tijd is gebleken dat de tweede zienswijze, waarbij de programmastructuur wordt afgeleid uit de datastructuur, leidt tot robuuster en gemakkelijker wijzigbare programma's.

3.3 De objectoriëntatie

In sectie 2 werden de na te streven doelstellingen bij de systeemontwikkeling en het programmeren voorgesteld. Eén hiervan is het verwezenlijken van flexibele systemen, d.w.z. systemen die vlot en gemakkelijk aanpasbaar zijn aan gewijzigde omstandigheden. In het vervolg zullen wij ons vooral op deze doelstelling concentreren. Er kan inderdaad gesteld worden dat een systeem dat flexibel is ook betrouwbaar, verbeterbaar en doorzichtig zal zijn. De flexibiliteit van een systeem is in grote mate afhankelijk van zijn structuur. Per definitie bestaat een systeem uit elementen die met elkaar verbonden zijn. Om een flexibel systeem te verkrijgen is het van belang de juiste elementen en de juiste verbindingen ertussen te kiezen.

Tot op zekere hoogte bepaalt de aard van de elementen ook de aard van de verbindingen. Verder moet men ernaar streven het aantal verbindingen te minimaliseren en de verbindingen zo 'zwak' mogelijk te maken. Stellen wij een verbinding tussen twee elementen voor door een lijn en de sterkte van een verbinding door de dikte van de lijn, dan is Systeem 2 van Figuur 1 zonder twijfel te verkiezen boven Systeem 1.



Systeem 1 Systeem 2

Figuur 1: Twee systemen van verschillende flexibiliteit

Systeem 2 heeft een structuur die gemakkelijker zal kunnen gewijzigd worden dan deze van Systeem 1, omdat ze niet alleen minder verbindingen bevat, maar tevens omdat de verbindingen zwakker zijn dan deze van systeem 1.

Dat het aantal verbindingen en de sterkte van de verbindingen de flexibiliteit van een systeem bepalen is niet moeilijk te begrijpen: een vereiste wijziging aan een bepaald systeemelement kan zich via de verbindingen eventueel voortplanten naar andere elementen.

In principe bevordert het objectgericht programmeren en ontwikkelen van systemen de flexibiliteit.

De elementen van objectgerichte systemen worden 'objecten' genoemd. Een *object* is een systeemelement dat (1) bij het ontvangen van een *bericht* verstuurd door een ander element of door de omgeving, een *dienst* verleent aan de afzender;

(2) een *verzameling attributen* (soms ook 'statusvector' genoemd) en een *verzameling 'methoden'* (soms ook 'routines', 'functies'... genoemd) bezit, waarbij iedere 'methode' een bepaald soort dienst verricht.

Dat objectgerichte systemen de flexibiliteit bevorderen vloeit voort uit twee kenmerken die zij bezitten:

1. De verbindingen tussen de systeemelementen zijn uitermate zwak. Men kan inderdaad geen zwakkere verbinding bedenken dan het zenden van een bericht of het ontvangen van een bericht.
2. Het systeem bestaat volledig of in overwegende mate uit elementen die overeenkomen met elementen uit de werkelijkheid, zoals een klant, een order, een rekening. De werkelijkheid is dus weerspiegeld in het systeem. Als zich de werkelijkheid wijzigt kan men relatief snel de overeenkomstige plaatsen vinden waar de systeemwijziging moet aangebracht worden. De structuur van het systeem is immers identiek aan de structuur van de werkelijkheid.

Daarenboven is er, gedragsmatig, geen enkele notie van hiërarchie aanwezig: alle objecten zijn gelijkwaardig; objecten kunnen geen deel uitmaken van andere objecten; objecten kunnen andere objecten niet bevelen; de enige communicatie is door middel van berichten van gelijke tot gelijke.

Het opkomen van objectgericht programmeren werd aangekondigd als een revolutie. Een gevolg hiervan was, dat programmeurs het veelal aanzagen als vervanging van gestructureerd programmeren. De principes van gestructureerd programmeren werden overbodig en verouderd geacht. Dit was een mooi voorbeeld van de processie van Echternach. Ook bij het objectgericht programmeren zijn de zes doelstellingen die we hoger hebben voorgesteld geldig en niet automatisch vervuld; en ook bij het objectgericht programmeren moeten de “methoden”, die niets anders zijn dan programma’s, volgens de principes van het gestructureerd programmeren en rekening houdend met de gepaste denkpatronen, ontworpen zijn.

3.4 Modelmatige ontwikkeling

Een objectgericht systeem bestaat dus uit objecten die met elkaar communiceren door middel van berichten. Als alle omstandigheden gelijk blijven, zal een systeem dat met een objectvisie is ontworpen flexibeler zijn dan een systeem waarvoor de objectvisie niet werd gebruikt.

Dit neemt niet weg dat twee personen die eenzelfde systeem objectgericht ontwerpen, tot systemen met een verschillende structuur kunnen komen, waarbij elk systeem een eigen graad van flexibiliteit vertoont. Er is dus ***meer nodig dan alleen maar objectoriëntatie***. Even belangrijk is de wijze waarop men het systeem opdeelt, m.a.w. welke systeemarchitectuur men kiest.

Ter illustratie van een goede opdeling, presenteren we hierna de decompositie van MERODE (**M**odel-based, **E**xistence-dependency **R**elationship **O**bject-oriented

Development). [8], een verwezenlijking van de vakgroep Beleidsinformatica van het Departement Toegepaste Economische Wetenschappen.

3.4.1 Een architectuur die leidt tot flexibele systemen

Ter bevordering van de flexibiliteit delen we het systeem op in drie delen: het *bedrijfssubsysteem*, *het inputsubsysteem* en *het outputsubsysteem*. Ieder subsysteem bevat een bepaalde soort objecten: bedrijfsobjecten in het bedrijfssubsysteem, inputobjecten in het inputsubsysteem, outputobjecten en informatie-objecten in het outputsubsysteem. Objecten communiceren nooit met objecten uit hetzelfde subsysteem; zij communiceren enkel met objecten uit andere subsystemen.

Bedrijfsobjecten komen overeen met objecten die in de werkelijkheid bestaan. Voorbeelden: klanten, orders, rekeningen, boeken, contracten, patiënten. Met ieder relevant object uit de werkelijkheid komt een object van het bedrijfssubsysteem overeen. Deze bedrijfsobjecten hebben methoden die geactiveerd worden door gebeurtenisberichten en die de toestand van het object, waartoe zij behoren, bijwerken. Sommige methoden worden verondersteld pre-condities te hebben die bedrijfsregels afdwingen. Men kan dus zeggen dat het bedrijfssubsysteem het bedrijf simuleert en garandeert dat de bedrijfsregels gevolgd worden, noch min noch meer. Als de bedrijfsregels wijzigen moet enkel het bedrijfssubsysteem veranderd worden. Het bedrijfssubsysteem kan in principe nooit beïnvloed worden door veranderingen in de informatiebehoeften (voorwerp van het outputsubsysteem) of door veranderingen in de werkorganisatie (voorwerp van het inputsubsysteem).

Benevens bedrijfsobjecten zijn er drie soorten ***functie-objecten***: inputobjecten, outputobjecten en informatie-objecten.

Inputobjecten verzamelen informatie over gebeurtenissen die zich in de werkelijkheid voordoen en zenden gebeurtenisberichten naar bedrijfsobjecten. Voorbeeld: een object dat, wanneer in een ziekenhuis een patiënt binnenkomt (een transactie), alle noodzakelijke informatie hierover verzamelt in een gebeurtenisbericht en dit bericht stuurt naar een bedrijfsobject van het type "patiënt". In veel gevallen stuurt een inputobject in één klap hetzelfde bericht naar meerdere bedrijfsobjecten, elk van een verschillend type, en eventueel naar informatie-objecten en outputobjecten. Indien gewenst, kunnen inputobjecten statusvectoren (synoniem van 'datastructuren') van bedrijfsobjecten inspecteren om informatie, noodzakelijk voor het produceren van gebeurtenisberichten, te verkrijgen.

Als de werkorganisatie verandert, hetgeen betekent dat de bedrijfsgebeurtenissen op een andere wijze gegroepeerd worden, moet enkel het inputsysteem gewijzigd worden. De andere twee subsystemen worden niet verstoord door een dergelijke verandering.

Outputobjecten worden geactiveerd door berichten van het systeem dat het bedrijf stuurt (bvb. een manager), of door berichten ontvangen van inputobjecten. Outputobjecten kunnen statusvectoren van bedrijfs- en informatie-objecten inspecteren en, na verwerking van de informatie, berichten sturen naar het stuursysteem. Ze kunnen ook nieuwe gebeurtenissen creëren en de overeenkomstige gebeurtenisberichten naar bedrijfsobjecten sturen (in dat geval vervullen zij dus de rol van zowel output- als van inputobjecten). Voorbeeld van een outputobject: een object dat facturen maakt.

Informatie-objecten zijn noodzakelijk als outputobjecten informatie nodig hebben over gebeurtenissen die zich in het verleden hebben voorgedaan. Een voorbeeld hiervan: een verrichting (storting of afhaling) op een zichtrekening. Deze informatie is nodig voor outputobjecten die periodisch rekeninguittreksels moeten produceren.

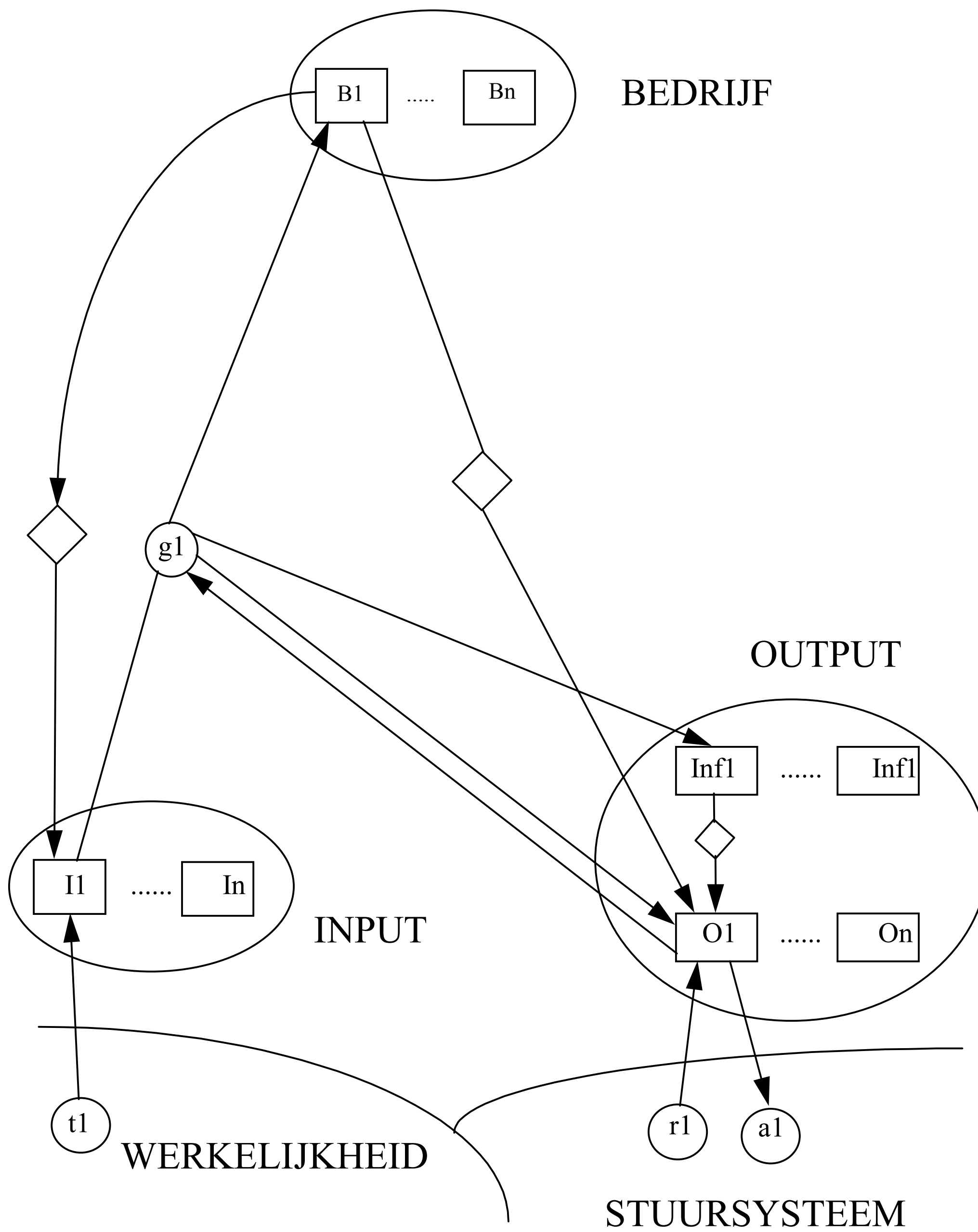
Samen met de outputobjecten verzorgen de informatie-objecten de informatiebehoeften van het bedrijf. Veranderen deze informatiebehoeften, dan moet enkel het outputsysteem wijzigingen ondergaan: het bedrijfssubsysteem en het inputsubsysteem blijven onberoerd.

In het algemeen is er een verschil tussen bedrijfsobjecten en informatie-objecten enerzijds, en input- en outputobjecten anderzijds: meestal bevat de statusvector van input- en outputobjecten slechts één enkel attribuut, namelijk een identificator van de objectklasse. De reden hiervoor is dat er slechts één object bestaat in die klasse en dat dit object zich daarenboven niets moet herinneren uit het verleden om zijn diensten te kunnen leveren. Maar hierop bestaan uitzonderingen, zoals later zal blijken.

In figuur 2 wordt de besproken architectuur grafisch voorgesteld. In deze figuur stellen wij objectklassen voor en geen individuele objecten. Objectklassen zijn weergegeven door rechthoeken, gebeurtenistypes door cirkels en toestandsvectorinspecties door ruiten. Hierbij zijn de volgende afkortingen gebruikt:

- Bx: bedrijfsobjectklasse x;
- Ix: inputobjectklasse x;
- Ox: outputobjectklasse x;
- INFx: informatieobjectklasse x;
- tx: transactieberichttype x;
- gx: gebeurtenisberichttype x;
- rx: rekwesttype x;
- ax: antwoordtype x.

Om de figuur niet te overladen werden de mogelijke verbindingen tussen de onderscheiden elementen van de tekening aangeduid, gebruikmakend van slechts één klasse van elk soort element, namelijk: I1, B1, INF1, O1 en g1.



Figuur 2: Systeemarchitectuur

De motivatie voor deze architectuur is bevordering van flexibiliteit. Wij gaan uit van het principe dat een systeem dat is onderverdeeld in zwak-gekoppelde (dus relatief onafhankelijke) subsystemen, die elk overeenkomen met één enkel aspect van het probleem, dat onafhankelijk evolueert van alle andere aspecten, een hoge

graad van flexibiliteit moet bezitten. In het hier beschouwde geval zijn deze drie onafhankelijke aspecten:

- de bedrijfsregels (bedrijfssubsysteem);
- de informatiebehoeften (outputsubsysteem);
- de werkorganisatie (inputsubsysteem).

De flexibiliteit wordt verder bevorderd door de eis dat objecten behorend tot hetzelfde subsysteem niet met elkaar mogen communiceren. Daardoor wordt het mogelijk zonder moeite een bepaald object of een bepaalde objectklasse te verwijderen of een object of objectklasse toe te voegen.

3.4.2 Een voorbeeld: de vereenvoudigde bank

Het is nuttig het bovenstaande met een voorbeeld te illustreren. Beschouwen we het overgesimplifieerd geval van een bank waarvan de klanten een of meerdere zichtrekeningen kunnen hebben. Nadat een rekening geopend werd, kan men er geld op storten en er geld van afhalen. Na een zekere tijd wordt de rekening afgesloten. Klantobjecten worden gecreëerd en komen op een zeker ogenblik aan hun einde.

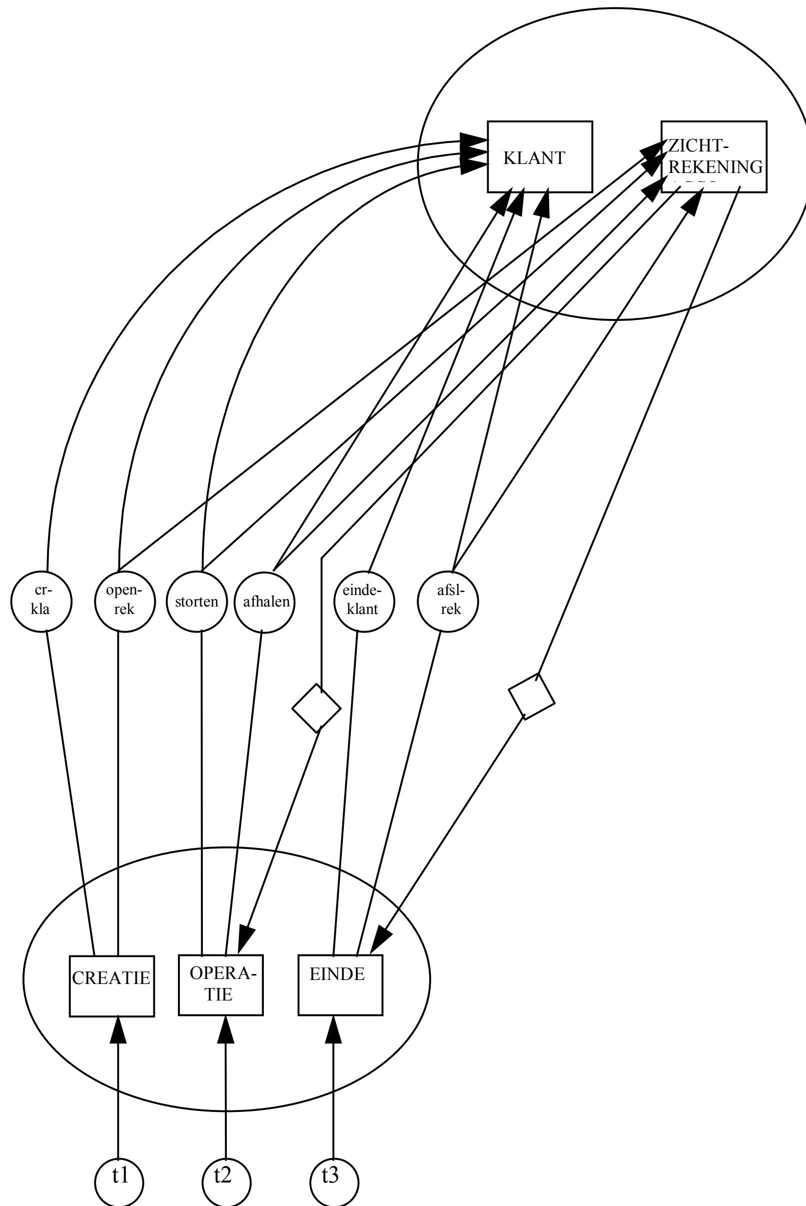
Figuur 3 toont het inputsubsysteem en het bedrijfssubsysteem van dit vereenvoudigd geval.

Kijken we eerst naar de gebeurtenisberichttypes. Alle berichten van de types 'creëren klant' en 'einde klant' worden enkel gestuurd naar de klantklasse. Berichten van de andere types echter (openen rekening, storten, afhalen en afsluiten rekening) worden zowel naar de klantklasse als naar de zichtrekeningklasse gezonden

Om organisatorische redenen zijn de gebeurtenissen gegroepeerd tot drie soorten transacties, t_1, t_2 en t_3. Deze transacties vormen de input voor drie verschillende inputobjectklassen: creatie, verrichting, einde. Ieder van deze klassen bezit slechts één object, nooit meer dan één.

De objecten en de gebeurtenisberichten hebben de volgende attributen:

- klant: klantnr, aantal-rekeningen;
- zichtrekening: rekeningnr, klantnr, saldo, afgesloten-of-niet;
- creëren-klant: gebeurtenistype, datum, klantnr;
- openen-rekening: gebeurtenistype, datum, rekeningnr, klantnr;
- storten: gebeurtenistype, datum, rekeningnr, klantnr, bedrag;
- afhalen: gebeurtenistype, datum, rekeningnr, klantnr, bedrag;
- afsluiten-rekening: gebeurtenistype, datum, rekeningnr, klantnr;
- einde-klant: gebeurtenistype, datum, klantnr.



Figuur 3: Input- en bedrijfssubsystemen voor een vereenvoudigd bankvoorbeeld

Het inputobject 'Creatie' werkt als volgt. Telkens als dat object een transactie ontvangt, produceert het een bericht van het type 'openen-rekening' en, als dan de klant nog niet bestaat, ook een bericht van het type 'creëren-klant'.

Na ontvangst van een transactie van type 2, produceert het inputobject 'Verrichting' ofwel een gebeurtenisbericht van het type 'storten' ofwel een gebeurtenisbericht van het type 'afhalen'. Aangezien de transactie enkel het rekeningnummer, maar niet het klantnummer, vermeldt, en aangezien het

inputobject tevens een bericht moet sturen naar het klantobject, dient het de statusvector van de zichtrekening te consulteren om het klantnummer in het gebeurtenisbericht te kunnen stoppen.

Het inputobject ‘Einde’ ontvangt transacties van het type t_3. Met de informatie uit de transactie produceert het ofwel een gebeurtenis van het type ‘einde-klant’ of van het type ‘afsluiten-rekening’. Aangezien de transactie die meldt dat een rekening moet afgesloten worden, niet het klantnummer maar enkel het rekeningnummer aangeeft, en aangezien het inputobject tevens een bericht van het type ‘afsluiten-rekening’ moet sturen naar het klantobject, inspecteert het de statusvector van de zichtrekening teneinde het klantnummer te kennen om het in het gebeurtenisbericht te plaatsen.

Teneinde alle mogelijke vormen van connectie tussen de input- en bedrijfssubsystemen enerzijds en het outputsubsysteem anderzijds te illustreren, zullen we nu 4 verschillende outputfuncties tonen voor het geval van de vereenvoudigde bank.

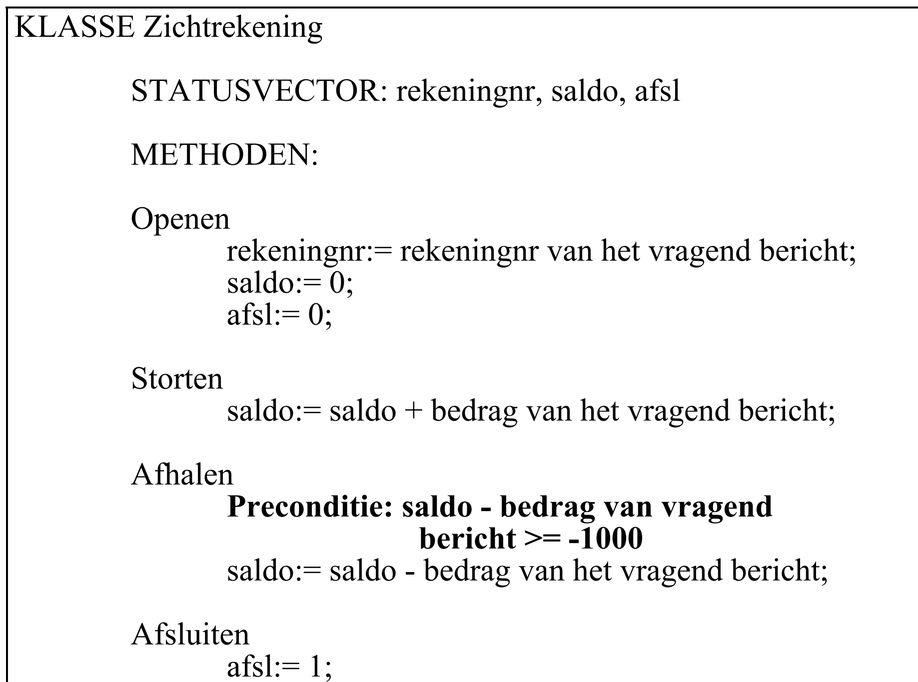
Aangezien deze functies informatie moeten halen uit bedrijfsobjecten van het type ‘zichtrekening’, geven we hieronder, voor een goed begrip van het vervolg, het abstract datatype van dit objecttype.

Iedere zichtrekening heeft een rekeningnummer. Verder willen we omtrent de rekening het saldo bijhouden, en of de rekening al dan niet afgesloten is. De diensten die van objecten van het type ‘zichtrekening’ gevraagd worden zijn:

- openen van de rekening;
- storten van een bedrag op de rekening;
- afhalen van een bedrag van de rekening;
- afsluiten van de rekening.

Dit geeft aanleiding tot een statusvector met 3 elementen: rekeningnr, saldo, afsl. Tevens moeten we 4 verschillende soorten diensten (methoden) voorzien. Iedere dienst kan afzonderlijk gevraagd worden door middel van een apart soort bericht, dat de noodzakelijke gegevens bevat om de dienst uit te voeren. Zo is er een soort bericht dat vraagt de rekening te openen, een soort bericht dat vraagt een storting te boeken, enz. Een bericht dat vraagt een rekening te openen moet het rekeningnr bevatten, een bericht dat vraagt een storting te boeken moet het rekeningnr en het gestorte bedrag bevatten, enz. Figuur 4 stelt de objectklasse ‘zichtrekening’ voor onder de vorm van een abstract datatype.

Bemerk dat er in de methode ‘Afhalen’ een preconditie staat: deze bank laat niet toe dat het saldo van een rekening 1000 € of meer in het rood zou gaan. Als bij een aangevraagde afhaling deze preconditie niet vervuld is, dan gaat de afhaling niet door.



KLASSE Zichtrekening

STATUSVECTOR: rekeningnr, saldo, afsl

METHODEN:

Openen
rekeningnr:= rekeningnr van het vragend bericht;
saldo:= 0;
afsl:= 0;

Storten
saldo:= saldo + bedrag van het vragend bericht;

Afhalen
Preconditie: saldo - bedrag van vragend bericht >= -1000
saldo:= saldo - bedrag van het vragend bericht;

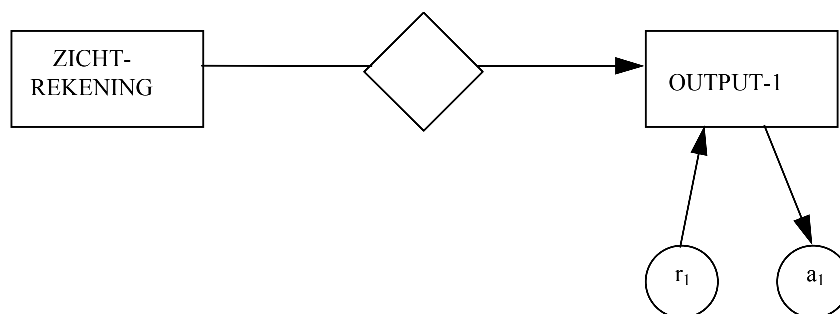
Afsluiten
afsl:= 1;

Figuur 4: De objectklasse 'zichtrekening' voorgesteld als abstract datatype

Functie 1: Na opgave van het rekeningnummer, toon het saldo van de rekening

Na verkrijging van een inputrekwest van type r_1, waarin het rekeningnr staat, moet het object van functie 1 de statusvector van de overeenkomstige rekening inspecteren en antwoord a_1 uitsturen, waarin het bewuste rekeningsaldo staat. Dit is voorgesteld in figuur 5.

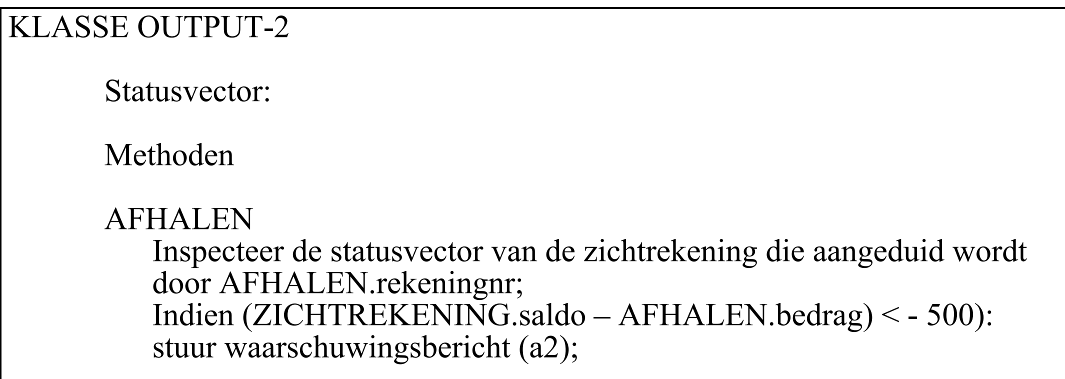
Aangezien hetzelfde outputobject toegang heeft tot alle statusvectoren van de rekeningen, bestaat er slechts één enkel outputobject van klasse 'output 1'.



Figuur 5: Specificatie van Functie 1

Functie 2: Telkens een afhaling het saldo van een zichtrekening beneden - 500 € doet dalen, zend een waarschuwingsbericht naar de directie

In figuur 6 is deze functie gespecificeerd in relatie tot haar omgeving. Er is een outputobjectklasse met één enkel object vereist. Het abstract datatype van de klasse 'Output-2' heeft 1 methode, 'Afhalen' genoemd. Die methode wordt geactiveerd door het 'Afhalen'-bericht dat uitgezonden wordt door het inputobject 'Verrichting'. Merk op (zoals kan gezien worden in figuur 6) dat het bericht niet geactiveerd wordt door informatierekwesten. Het is niet moeilijk in te zien dat het abstract datatype voor de 'Output-2'-klasse er als volgt uitziet.


KLASSE OUTPUT-2

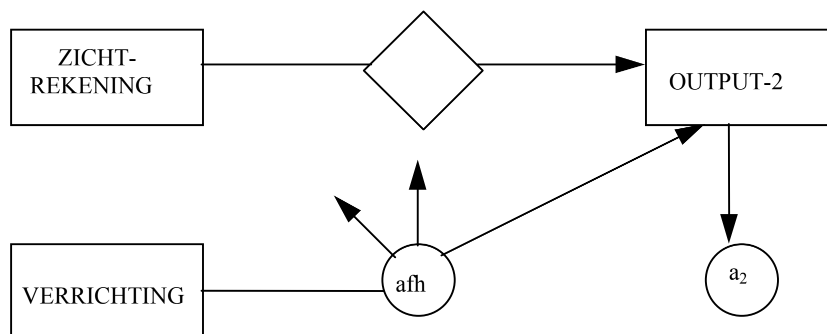
Statusvector:

Methoden

AFHALEN
Inspecteer de statusvector van de zichtrekening die aangeduid wordt door AFHALEN.rekeningnr;
Indien (ZICHTREKENING.saldo – AFHALEN.bedrag) < - 500):
stuur waarschuwingsbericht (a2);

Noteer dat de statusvector van de klasse OUTPUT-2 ledig is. Aangezien er, conceptueel, slechts één enkel object tot deze klasse behoort, vallen dat object en de bewuste klasse samen. Daarom moet het object niet geïdentificeerd worden. Het is voldoende de klasse met haar naam 'Output-2' aan te duiden. Bovendien is er tussen twee opeenvolgende activeringen van de methoden geen geheugen nodig.

Merk ook op dat we bij de implementatie niet mogen vergeten dat deze methode moet uitgevoerd worden vóór de methode 'afhalen' van de klasse 'zichtrekening'.



Figuur 6: Specificatie van Functie 2

Functie 3: Op aanvraag van een klant, maak een rekeninguittreksel voor de rekening waarvan het nummer in de aanvraag opgegeven wordt.

Een rekeninguittreksel dient de volgend elementen te bevatten: het vorig saldo dat als nieuw saldo op het vorig rekeninguittreksel stond, alle verrichtingen sedert het laatste uittreksel en het nieuw saldo op heden. Men ziet dus gemakkelijk in dat deze functie informatie-objecten moet inspecteren, die wij 'rekening-verrichtingen' noemen in figuur 7.

Het inputobject 'Verrichting' moet nu niet enkel zijn 'afhalen'- en 'storten'-berichten sturen naar de 'Zichtrekening'- en de 'Klant'-klassen, maar ook naar de 'Rekening-verrichting'-klasse. De aankomst van ieder bericht van het type 'afhalen' of 'storten' bij de 'Rekening-verrichting'-klasse veroorzaakt de creatie van een object van die klasse door middel van respectievelijk een methode 'afhalen' of 'storten'. Het abstract datatype voor de klasse 'Rekening-verrichting' is dus als volgt.

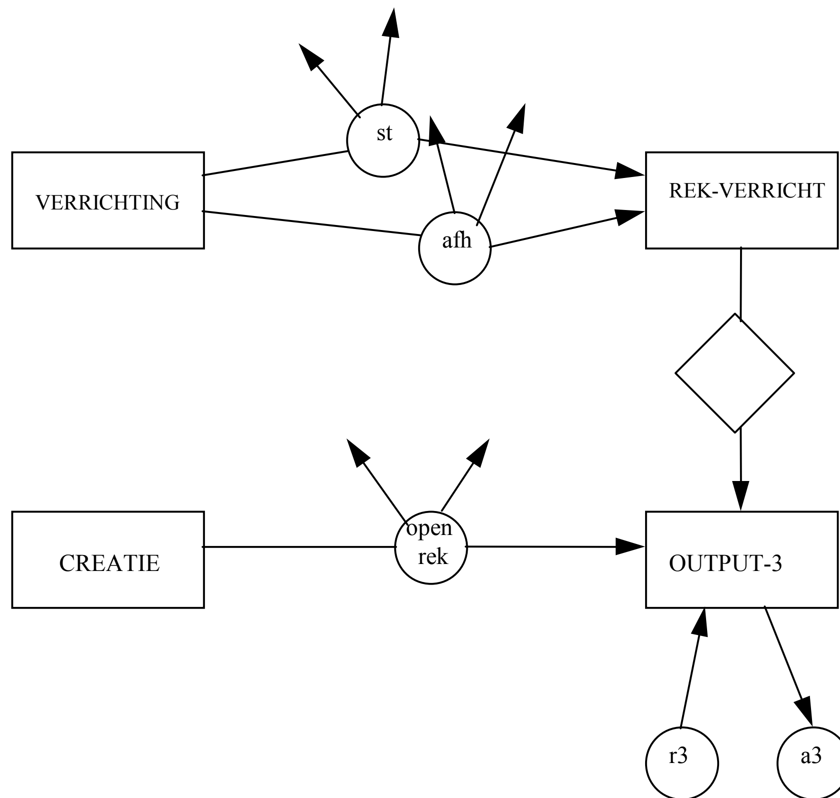
```
KLASSE REKENING-VERRICHTING

Statusvector: rekeningverrichtingnr, rekeningnr, soort,
              bedrag, datum

Methoden

  STORTEN
      rekeningverrichtingnr:= STORTEN.rekening-
      verrichting nr;
      rekeningnr:= STORTEN.rekeningnr;
      soort:= '+';
      bedrag:= STORTEN.bedrag;
      datum:= STORTEN.datum;

  AFHALEN
      rekeningverrichtingnr:= AFHALEN.rekening-
      verrichting nr;
      rekeningnr:= AFHALEN.rekeningnr;
      soort:= '-';
      bedrag:= AFHALEN.bedrag;
      datum:= AFHALEN.datum;
```

Noteer dat de STORTEN-gebeurtenis en de AFHALEN-gebeurtenis moeten voorzien worden van een bijkomend veld: 'rekeningverrichtingnr'.

Na ontvangst van een r_3-rekwest, moet het aangewezen ‘Output-3’-object de statusvectoren inspecteren van alle rekeningverrichtingen die hetzelfde rekeningnr hebben en met een tijdsaanduiding die gelijk is aan of later dan de datum van het vorig rekeninguittreksel maar vroeger dan vandaag. Om dit te kunnen doen moet het outputobject in zijn statusvector beschikken over 3 elementen: het rekeningnr, het vorig saldo en de datum van het vorig rekeninguittreksel. Aangezien er daarenboven evenveel objecten van het type ‘Output-3’ voorkomen als er objecten zijn van het type ‘zichtrekening’, moet het inputobject ‘Creatie’ zijn ‘open-rekening’ berichten tevens sturen naar de klasse ‘Output-3’ teneinde telkens een statusvector van het individueel object te kunnen creëren.

Als gevolg van dit alles zal het abstract datatype voor de klasse ‘Output-3’ er als volgt uitzien:

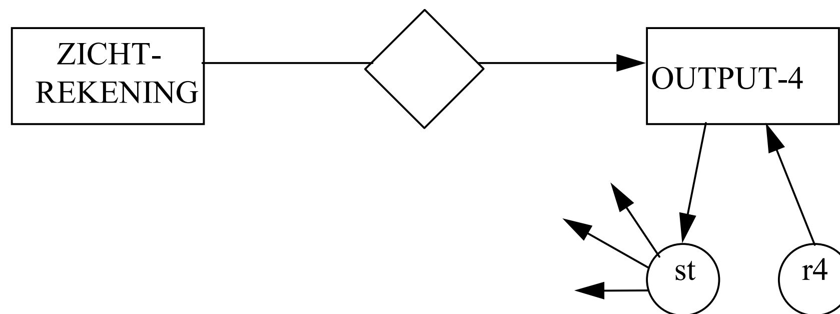
```
KLASSE OUTPUT-3

Statusvector:output-3-id, rekeningnr, vorig-saldo, datum-vorig-rekuittr

Methoden

  OPEN-REKENING
    output-3-id:= OPENEN-REKENING.output-3-id;
    rekeningnr:= OPENEN-REKENING.rekeningnr;
    vorig-saldo:= 0;
    datum-vorig-rekuittr:= vandaag;

  R3
    Dit is een methode die de rekeninguittreksels maakt, ze afdrukt
    (a3) en de attributen ‘vorig-saldo’ en ‘datum-vorig-rekuittr’
    bijwerkt;
```



Figuur 7: Specificatie van Functie 3

Functie 4: Op aanvraag van de directie, onderzoek alle zichtrekeningen en stort een premie van x op alle rekeningen die, op dat ogenblik, een saldo vertonen van tenminste y, waarbij x en y gegeven zijn in het rekwestbericht.

Deze functie moet verwezenlijkt worden door outputklasse 4 (O-4), waarvan slechts 1 object voorkomt. Eigenlijk is het zowel een input- als een outputobject, want het produceert gebeurtenissen van het type 'storten'. Telkens het object geactiveerd wordt door een r4-rekwest, inspecteert het de statusvectoren van alle 'zichtrekening'-objecten en produceert het gebeurtenisberichten van het type 'storten'. Het diagramma is weergegeven in figuur 8.



Figuur.8: Specificatie van Functie 4

3.5 Specificatie en implementatie

In moderne methodologieën voor systeemontwikkeling maakt men een onderscheid tussen 'wat' een systeem moet doen enerzijds en 'hoe' dit moet geschieden anderzijds, met andere woorden tussen 'specificatie' en 'implementatie'. De kunst bestaat erin een specificatie te verwezenlijken die zo vrij mogelijk is van implementatie-facetten. Het voordeel daarvan is tweevoudig: (1) het bevordert de flexibiliteit van het systeem en (2) het maakt een vlotte en klare communicatie met de gebruiker mogelijk.

Een specificatie moet vrij zijn van implementatie-aspecten, zoals de hardware waarop het systeem zal draaien, het operating-systeem dat daarbij zal gebruikt worden, de taal waarin de programma's zullen geschreven worden, de database-organisatie die men zal gebruiken, of het systeem on-line of in batch zal draaien, enz. Dit zijn allemaal concrete verschijningvormen van het systeem, die als gevolg van de evolutie van de techniek en andere oorzaken, soepel moeten kunnen wijzigen, zonder dat daarom de specificatie moet veranderd worden. Een specificatie wordt slechts gewijzigd als de informatiebehoeften of het bedrijfsmodel veranderen.

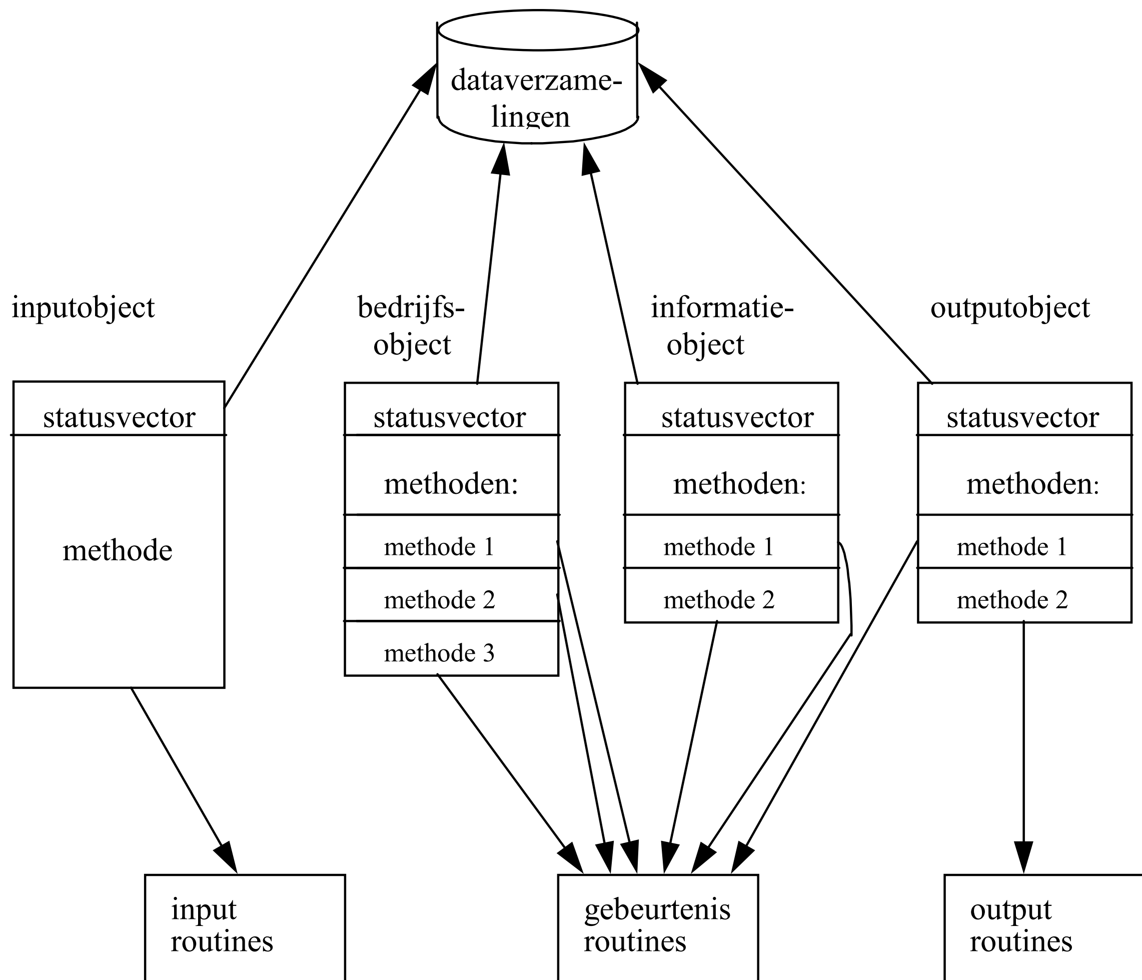
Het is niet altijd gemakkelijk de specificatie onafhankelijk te houden van de implementatie. Vooral wanneer zich een nieuwigheid aandient, zoals een nieuw paradigma (bvb. de objectoriëntatie), een nieuwe programmeermethode (bvb. aspectgericht programmeren), een nieuwe programmeertaal (bvb. Java) enz., zijn enthousiaste informatici geneigd hun specificatie te bezoedelen met concrete implementatieaspecten. Dit leidt zeer snel tot scheve systemen, die bij later gebruik van nieuwe ontwikkelingen, volledig opnieuw moeten ontworpen worden, alhoewel de informatiebehoeften ongewijzigd zijn.

De specificatie wordt gebruikt om met de gebruiker te communiceren, en de implementatie is een zaak van technisch gerichte informatici. Vaak worden gebruikers lastig gevallen met voor hen onverstaanbare implementatie-aspecten. Dit leidt tot systemen die hun doel voorbijschieten: aangezien de gebruiker de

voorgelegde documenten niet verstaat, kan hij ook niet zinvol zien of zijn werkelijkheid en zijn informatiebehoeften er adequaat in weerspiegeld zijn. Meer informatie over het onderscheid tussen specificatie en implementatie kan men vinden in het boek "Software Requirements & Specifications" van M. Jackson [5].

Verder is het aan te raden voor de implementatie een methode te gebruiken, waarbij de specificatie via vastgelegde (en eventueel automatiseerbare) regels *getransformeerd* wordt tot een implementatie. In dit geval is een wijziging in de specificatie zeer snel en foutloos te implementeren. Hieronder geven wij een voorbeeld van 'implementatie via transformatie', waarbij de implementatie-omgeving eerder traditioneel is (geen gebruik van objecttechnologie noch bij het programmeren, noch voor de databases) en waarbij de specificatie via Merode gedaan werd. Vanzelfsprekend kunnen ook voor objectgerichte implementatie transformatieregels opgesteld worden. Gebruikt men voor de specificatie Merode, dan zijn deze regels zelfs zeer eenvoudig; immers een Merode-specificatie is zelf reeds objectgericht.

De eerste transformatiestap is de ontmanteling van de objecten door de statusvectoren te scheiden van de methoden en door iedere methode zodanig uit te knippen dat het een geïsoleerde programma-routine wordt. De statusvectoren worden dan in een dataverzameling geplaatst, die naargelang het geval een database of een geheel van klassieke bestanden kan zijn, en de routines worden in routineverzamelingen gezet. We onderscheiden drie soorten routineverzamelingen: voor de inputroutines (d.w.z. voor de routines van de inputobjecttypen), voor de gebeurtenisroutines (d.w.z. voor de routines van de bedrijfsobjecttypen) en voor de outputroutines (d.w.z. voor de routines van de outputobjecttypen). Deze situatie is voorgesteld in figuur 9.



Figuur 9: Objectontmanteling

Na het uitsnijden van de gebeurtenismethoden en hun transformatie naar routines, plakken we alle routines samen die hetzelfde gebeurtenisbericht moeten verwerken. Voor het vroeger behandelde bankgeval bijvoorbeeld, moeten we de routine AFHALEN van de klantobjectklasse combineren met de routine AFHALEN van de zichtrekeningklasse, met de routine AFHALEN van de rekening-verrichtingklasse en met de routine AFHALEN van de Output-2 klasse. Dergelijk plakwerk doen we op de volgende wijze: eerst snijden we de precondities weg en we plakken ze samen aan het begin van de gecombineerde routine; we plakken dan de rest van de routines samen en we plaatsen ze na de gecombineerde precondities. Dit is geïllustreerd in figuur 10 met de routines van het type ‘afhalen’. Bemerk dat de implementatieroutines input- en outputoperaties behoeven (‘haalsv’ betekent ‘haal de statusvector’ en ‘stockeersv’ betekent ‘stockeer de statusvector).

AFHALEN (Klant)

PRECON:
    geblokkeerd = 0;
BODY:
    aantal-bewerkingen:=
    aantal-bewerkingen + 1;

AFHALEN (zichtrekening)

PRECON:
    saldo - AFHALEN.bedrag) >-1000;
BODY:
    saldo:=
    saldo - AFHALEN.bedrag;

AFHALEN (Rekeningverrichting)
    rekening-verrichtingnr:= AFHALEN.rekening.verrichtingnr;
    rekeningnr:= AFHALEN.rekeningnr;
    soort:= '-';
    bedrag:= AFHALEN.bedrag;
    datum:= AFHALEN.datum;

AFHALEN (Output-2)
    Inspecteer de statusvector van de zichtrekening die aangeduid wordt door
    AFHALEN.rekeningnr;
    If ((ZICHTREKENING.saldo - AFHALEN.bedrag) < -500)
    stuur waarschuwingsbericht (a_2);

AFHALEN (Implementatie)

haalsv Klant (AFHALEN.klantnr);
haalsv Zichtrekening (AFHALEN.rekeningnr);
    If (KLANT.geblokkeerd = 0) en
      ((ZICHTREKENING.saldo - AFHALEN.bedrag) > -1000)
    do
        If ((ZICHTREKENING.saldo - AFHALEN.bedrag)< -500) stuur
        waarschuwingsbericht a_2
        KLANT.aantal-bewerkingen:= KLANT.aantal-bewerkingen + 1;
        ZICHTREKENING.saldo:= ZICHTREKENING.saldo - AFHALEN.bedrag;
        REKENINGVERRICHTING.rekeningverrichtingnr:=
                                AFHALEN.rekeningverrichtingnr;
        REKENINGVERRICHTING.zichtrekeningnr:= AFHALEN.rekeningnr;
        REKENINGVERRICHTING.soort:= '-';
        REKENINGVERRICHTING.bedrag:= AFHALEN.bedrag;
        REKENINGVERRICHTING.datum:= AFHALEN.datum;
        stockeersv Klant;
        stockeersv Zichtrekening;
        stockeersv Rekeningverrichting;
    Else
        zend verwerpingsbericht;

Figuur 10: Samenvoeging van vier gebeurtenisroutines tot één implementatieroutine

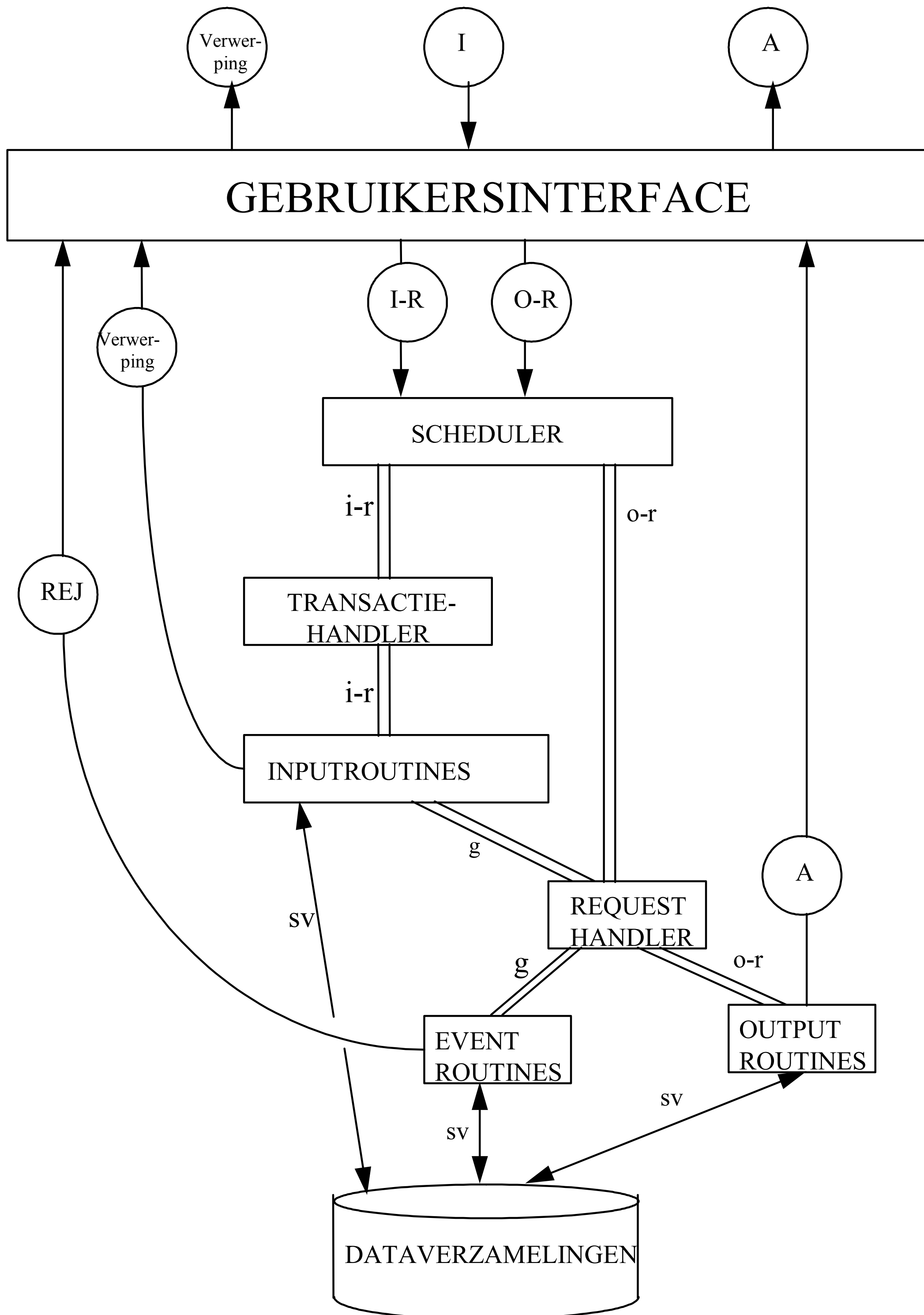
Eenmaal de objecten gesneden en geplakt, moeten we nog (in alle andere systemen herbruikbare) programma's toevoegen om het systeem te laten functionneren: een scheduler, een transactiehandler en een rekwesthandler. Deze elementen zijn weergegeven in figuur 11. In die figuur stelt een dubbele lijn een Call voor die gaat van het bovenste blok naar het onderste; de ruimte tussen de lijnen suggereert het doorgeven van de informatie, die aangeduid is naast de dubbele lijn.

Een enkelvoudige lijn met een pijl en een cirkel stelt outputberichten voor. Staat er geen cirkel, dan betekent dit dat de gegevens verkregen werden uit de dataverzamelingen of (her)opgeslagen worden in de dataverzamelingen, afhankelijk van de richting waarin de pijl wijst.

De ***scheduler*** leest alle input die het systeem bereikt via de gebruikers-interface, roept de transactiehandler op (ingeval van een rekwest voor een inputobject) of de rekwesthandler (ingeval van een rekwest voor een outputobject), en geeft hen de bedoelde rekwesten door. De scheduler beslist op welk moment de input zal verwerkt worden (interactief of in batch). Indien de input (of een gedeelte van de input) in batch moet verwerkt worden, houdt hij die input gereed in buffers tot de batch doorgegeven wordt aan de transactiehandler of aan de rekwesthandler. Hij kan ook de input sorteren. Indien we van on-line willen overstappen naar batch of vice-versa, moeten we enkel de scheduler wijzigen. Geen enkele andere systeemwijziging is vereist.

Zoals reeds gezegd, ontvangt de ***transactiehandler*** rekwesten bestemd voor inputobjecten en, voor ieder dergelijk rekwest, roept hij de geschikte inputroutine aan (d.w.z. de routine van een inputobject). De inputroutine produceert de gespecificeerde gebeurtenisberichten en roept de rekwesthandler op, waaraan zij tevens de geproduceerde gebeurtenisberichten doorgeeft. In figuur 11 ziet men verder dat de inputroutines soms, om hun werk te kunnen doen, statusvectoren van bedrijfsobjecten moeten ophalen uit de dataverzamelingen. Zij kunnen ook binnenkomende inputrekwestberichten als geheel verwerpen indien één of meerdere precondities uit de inputroutines niet vervuld zijn.

De ***rekwesthandler*** wordt geactiveerd door de inputroutines die hem gebeurtenisberichten doorgeven, en door de scheduler die hem outputrekwesten doorstuurt. Indien de rekwesthandler een gebeurtenisbericht ontvangt, activeert hij de gepaste gebeurtenisroutine en geeft aan deze routine het gebeurtenisbericht door. Indien de rekwesthandler geactiveerd wordt door de scheduler, activeert hij op zijn beurt de geschikte outputroutine. Noteer dat outputroutines niet alleen hun eigen statusvectoren kunnen ophalen, maar ook statusvectoren van bedrijfsobjecten en van informatie-objecten, en dat zij antwoordberichten kunnen sturen naar de omgeving. Natuurlijk, zoals reeds vroeger opgemerkt, kunnen ook gebeurtenisroutines statusvectoren ophalen en heropslaan.



Figuur 11: Architectuur voor een traditionele implementatie

Aan het Departement Toegepaste Economische Wetenschappen werd een tool ontwikkeld voor specificatie en implementatie van Merode-modellen. De naam van het tool is *MERMAID (Merode Modeling Aid)*. Lezers die hierin verder geïnteresseerd zijn kunnen contact opnemen met Prof. Monique Snoeck [monique.snoeck@econ.kuleuven.ac.be].

3.6 Beslissingstabellen

De techniek van de beslissingstabellen is een der oudste en helaas, veel te weining gebruikte technieken in de informatica. De techniek werd oorspronkelijk uitgevonden in 1959 [7], heeft opgang gemaakt in de zestiger en de zeventiger jaren van de vorige eeuw, is ondertussen al verschillende keren “heruitgevonden” en is in de loop der tijden in onbruik geraakt, althans bij het ontwerp van informatiesystemen. Ten onrechte.

Gedurende de laatste jaar komen beslissingstabellen terug op de voorgrond (en worden ze ook soms nog ‘heruitgevonden’) bij het ontwerp van KBS (kennis-gebaseerde systemen).

Wij zijn van mening dat de techniek der beslissingtabellen behoort bij de fundamentele onveranderlijke werktuigen doorheen alle veranderingen in de informatica. Een omvangrijker gebruik van beslissingstabellen zou veel ellende kunnen vermijden.

3.6.1 Definitie

De belangrijkste reden waarom de beslissingstabel niet meer zo vaak gebruikt wordt, is dat men ze niet juist gebruikt, m.a.w. dat men geen juiste beslissingtabellen opstelt. Daarom staan we even stil bij de eigenschappen die een beslissingstabel moet hebben om als zodanig bruikbaar te zijn. Deze eigenschappen vindt men terug in de volgende definitie.

> *Een beslissingstabel is een tabel die voorstelt het **exhaustief** geheel van **elkaar uitsluitende*** uitspraken binnen een vooraf bepaald probleemgebied.

In bovenstaande definitie zijn twee kenmerken in reliëf gebracht: (1) de exhaustiviteit of volledigheid en (2) de exclusiviteit of het elkaar niet overlappen van de uitspraken. De reden waarom tabellen in de praktijk niet gebruikt worden is precies omdat ze niet aan beide kenmerken voldoen. Tabellen die een van deze twee kenmerken niet bezitten zijn inderdaad waardeloos. In figuur 12 wordt een voorbeeld gegeven van een beslissingstabel die wel aan beide voorwaarden voldoet.

Orderbehandeling	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**
C1. Klant = groothandelaar C2. Bestelde hoeveelheid (**H**) C3. Afstand tussen fabriek en woning klant (in km) (**A**)	Ja H<10 -	Ja 10≤H<15 A<50	Ja 10≤H<15 50≤A<100	Ja 10≤H<15 A≥100	Ja H≥15 -	Neen - -
A1. Korting in % A2. Spoorvervoer A3. Wegvervoer A4. Type factuur	0 - X A	10 - X A	5 - X A	2 - X A	10 X - B	0 X - A

Figuur 12: Een beslissingstabel

Deze tabel heeft betrekking op het probleemgebied ‘Orderbehandeling’ en doet hierover zes uitspraken (één uitspraak per kolom). In vaktermen wordt een voorwaardelijke uitspraak *beslissingsregel* genoemd. Daarom zijn de hoofdingen van de kolommen aangeduid door R1, R2, R3, R4, R5, R6. R1 is dus de afkorting van Regel 1. In het vervolg van dit artikel zijn de termen ‘voorwaardelijke uitspraak’ en ‘beslissingsregel’ synoniemen.

De beslissingsregels hebben de volgende betekenis.

R1: Als de klant groothandelaar is ***en*** als de bestelde hoeveelheid minder is dan 10 stuks,
dàn korting 0 % ***en*** factuur type A.

R2: Als de klant groothandelaar is ***en*** als de bestelde hoeveelheid tenminste 10, maar minder dan 15 stuks is ***en*** als de afstand minder is dan 50 km,
dàn korting 10 % ***en*** wegvervoer ***en*** factuur type A.

Beslissingstabellen vertonen een geijkte structuur. Bovenaan staat een titel die het probleemgebied omschrijft (hier: *Orderbehandeling*). Het tabelgedeelte links van de dubbele verticale lijn noemen wij de *strook*. Hierin staan de elementen die de aard van de voorwaardelijke uitspraken bepalen. Deze elementen zijn van tweeërlei aard. Vooreerst vinden we er de *condities* (hier C1, C2, C3). Naargelang de omstandigheden zou men deze condities ook kunnen beschouwen als *inputs* of *oorzaken*. De condities bevinden zich altijd in de strook boven de dubbele horizontale lijn. De tweede categorie elementen zijn de *acties* (hier A1, A2, A3, A4). Hun plaats in de strook is onder de dubbele horizontale lijn. Een algemener benaming voor acties is *outputs* of *gevolgen*. Algemeen kan worden gezegd, dat een beslissingstabel het verband weergeeft tussen:

- condities en acties, of
- inputs en outputs, of
- oorzaken en gevolgen.

Rechts van de strook, en van deze strook gescheiden door een dubbele verticale lijn, bevinden zich de voorwaardelijke uitspraken of beslissingsregels, één uitspraak per kolom. Boven de dubbele horizontale lijn vertoont iedere kolom voor iedere conditie een *conditietoestand*. Dit is ofwel een elementaire toestand of de samenvatting van diverse elkaar uitsluitende elementaire toestanden. Soms wordt de conditietoestand aangeduid door een streepje, hetgeen betekent dat voor de desbetreffende conditie de toestand irrelevant is in het kader van de uitspraak in kwestie. In figuur 12 vinden we een streepje voor - onder andere - conditie 3 van R1. De afstand tussen de fabriek en de woning is dus irrelevant voor het doen van de uitspraak vervat in R1.

In de hoger gegeven definitie van een beslissingstabel wordt gezegd dat de uitspraken elkaar moeten uitsluiten en dat het geheel der uitspraken exhaustief moet zijn. Beide kenmerken (de exclusiviteit en de exhaustiviteit) liggen inderdaad aan de grondslag van het nut van de beslissingstabel. Dit is de *kern* van de zaak. Is tenminste één van deze kenmerken afwezig, dan verliest de beslissingstabel volledig haar reden van bestaan.

Het kenmerk *exhaustiviteit* slaat op het feit dat in de tabel alle mogelijke gevallen voorzien zijn. Dit betekent dat zich binnen het beschouwde probleemgebied geen enkel geval voordoet, dat niet vervat is in het conditiegedeelte van de beslissingsregels. Met *exclusiviteit* wordt bedoeld dat zich binnen het beschouwde probleemgebied geen enkel geval voordoet, dat vervat is in meer dan één beslissingsregel. Een tabel is dus exhaustief en exclusief als ieder mogelijk geval van het probleemgebied past in één en slechts één beslissingsregel.

Er bestaan methoden die bij het opstellen van de tabel feilloos garanderen dat de twee cruciale kenmerken (exhaustiviteit en exclusiviteit) vervuld zijn. Zie bvb. [9]. Wordt men geconfronteerd met tabellen door iemand anders opgesteld, en waarvan men niet weet of één van dergelijke methoden gebruikt werd, dan doet men er dus goed aan, op straffe van zinloosheid, eerst het exhaustieve en exclusieve karakter van de uitspraken te controleren. Dit kan op de volgende wijze.

Vooreerst gaat men na of in het conditiegedeelte, voor iedere conditie de aangegeven toestanden exhaustief en exclusief zijn. Men ziet gemakkelijk in dat dit het geval is voor de condities van figuur 12:
C1: Klant = groothandelaar: Ja of Neen;
C2: Bestelde hoeveelheid: H<10, 10≤H<15, H≥15;
C3: Afstand: A<50, 50≤A<100, H≥100.

Bijvoorbeeld de conditie ‘Afstand’: er bestaat geen enkele afstand die niet: ofwel minder is dan 50, ofwel gelegen is tussen 50 inclusief en minder dan 100, ofwel tenminste gelijk is aan 100. Deze conditie is dus exhaustief. Daarenboven kan geen enkele afstand gelijkertijd tot meer dan één van de drie genoemde categorieën behoren. Dus de conditie is tevens exclusief.

Na deze controle op het exhaustieve en exclusieve karakter van iedere conditie afzonderlijk, berekent men het aantal mogelijke enkelvoudige beslissingsregels die uit de elementaire conditietoestanden voortvloeien. Een *enkelvoudige beslissingsregel* bevat voor iedere conditie een elementaire toestand. Het aantal mogelijke enkelvoudige beslissingsregels is het product van het aantal toestanden voor iedere conditie. Voor de tabel van figuur 12 is dit: 2 x 3 x 3 = 18.

Immers, de eerste conditie heeft twee mogelijke toestanden en ieder van deze toestanden kan worden gecombineerd met ieder van de drie toestanden van conditie 2, hetgeen aanleiding geeft tot 2 x 3 of 6 gecombineerde toestanden van de eerste twee condities. Beschouwen we nu ook de derde conditie: ieder van de zes gecombineerde toestanden van de eerste twee condities kan worden gecombineerd met ieder van de drie toestanden van conditie 3. Met de drie condities samen verkrijgen we dus 6 x 3 = 18 gecombineerde toestanden of enkelvoudige beslissingsregels.

Heeft men het aantal mogelijke beslissingsregels berekend, dan telt men hoeveel enkelvoudige beslissingsregels de tabel werkelijk bevat. Een kolom die voor tenminste één conditie een toestand aangeeft die de samenvatting is van meer dan één elementaire toestand, bevat meer dan één enkelvoudige beslissingsregel. Men noemt daarom de uitspraak van deze kolom een *samengestelde beslissingsregel.* Zo zien wij dat de tabel van figuur 12 drie samengestelde beslissingsregels vertoont, t.w. R1, R5 en R6. De telling van het totaal aantal enkelvoudige beslissingsregels in figuur 12 geschiedt als volgt:

R1: 3, namelijk 3 mogelijke toestanden voor C3;
R2: 1;
R3: 1;
R4: 1;
R5: 3, namelijk 3 mogelijke toestanden voor C3;
R6: 9, namelijk 3 mogelijke toestanden voor C2 vermenigvuldigd met 3 mogelijke toestanden voor C3
__
18 totaal.

Tenslotte moet worden nagegaan of alle beslissingsregels, twee aan twee genomen, elkaar uitsluiten. Twee beslissingsregels sluiten elkaar uit, als ze voor tenminste één conditie twee elkaar uitsluitende toestanden vertonen. Men kan gemakkelijk nagaan dat dit voor figuur 12 het geval is. In totaal moet men hiervoor 15 regelparen onderzoeken: R1R2, R1R3, R1R4, R1R5, R1R6, R2R3,

R2R4, R2R5, R2R6, R3R4, R3R5, R3R6, R4R5, R4R6, R5R6. Men ziet bij voorbeeld dat R1 en R2 elkaar uitsluiten, omwille van het feit dat voor dit regelpaar de toestanden van C2 elkaar uitsluiten. Bij de andere 14 regelparen vindt men ook telkens een conditie met twee elkaar uitsluitende toestanden. Figuur 12 stelt dus wel degelijk een beslissingstabel voor die aan onze voorwaarden voldoet.

Bemerk dat een streepje in het conditiegedeelte een andere betekenis heeft dan een streepje in het actiegedeelte. In het conditiegedeelte betekent het dat de toestand van de bewuste conditie irrelevant is voor die beslissingsregel; in het actiegedeelte echter betekent het dat de in de strook aangeduide actie in dat geval *niet mag* uitgevoerd worden.

Tenslotte is het niet overbodig te verduidelijken hoe men beslissingstabellen leest. Dit geschiedt rij per rij. Om dit te vergemakkelijken is het zinvoller ze te presenteren in de vorm van figuur 13 in plaats van deze in figuur 12.

Orderbehandeling						
C1. Klant = groothandelaar	Ja					Neen
C2. Bestelde hoeveelheid (**H**)	H<10	10≤H<15			H≥15	-
C3. Afstand tussen fabriek en woning klant (in km) (**A**)	-	A<50	50≤A<100	A≥100	-	-
A1. Korting in %	0	10	5	2	10	0
A2. Spoorvervoer	-	-	-	-	X	X
A3. Wegvervoer	X	X	X	X	-	-
A4. Type factuur	A	A	A	A	B	A

Figuur 13: Beslissingstabel in een vorm om ze rij per rij te lezen

3.6.2 Het nut van beslissingstabellen

Beslissingstabellen zijn een techniek waarvan de toepassing algemener is dan alleen de informatica. Aangezien zij uiteindelijk te volgen procedures voorstellen, kan men er nut uit halen telkens men met procedures te maken heeft. Dit is onder andere het geval voor de wetgeving en de rechtspraak, reglementen in bedrijven, verenigingen en andere organisaties, de automatisering van rekenregels, KBS (knowledge based systems), voorstelling van procedures in de specificatie van informatiesystemen en automatische opstelling van programma's.

Uiteindelijk kan men uit beslissingstabellen voordeel halen bij drie soorten activiteiten: (1) bij het toepassen van procedures; (2) bij de controle van procedures op volledigheid, contradictie en juistheid; (3) bij het opstellen van procedures. Het klassieke alternatief voor beslissingstabellen bij het uitvoeren

van deze activiteiten zijn teksten. Om de voordelen van beslissingstabellen op teksten te illustreren, zullen we uitgaan van de tekst van figuur 14, die het exacte equivalent is van de (juiste en volledige) beslissingstabel van figuur 13.

Orderbehandeling.

1. Korting
Enkel aan groothandelaren wordt korting toegestaan, en dit inzover zij tenminste een hoeveelheid van 10 stuks bestellen. De kortingpercentages zijn 10%, 5% en 2%: 10% voor de groothandelaren die minstens 15 stuks bestellen, of die op minder dan 50 km van de fabriek wonen en minstens 10 stuks bestellen; 5% voor de groothandelaren die tenminste 10 maar minder dan 15 stuks bestellen en die tenminste 50 km, maar minder dan 100 km van de fabriek verwijderd wonen; 2% voor de groothandelaren die tenminste 10, maar minder dan 15 stuks bestellen en die tenminste 100 km ver wonen.

2. Wijze van vervoer
Wij vervoeren per spoor indien de bestelling niet afkomstig is van een groothandelaar of als een groothandelaar tenminste 15 stuks heeft besteld. In alle andere gevallen wordt wegvervoer toegepast.

3. Type factuur
Het normale type is A. Per uitzondering moet een factuur van het type B worden opgesteld, namelijk voor de groothandelaar die minstens 15 stuks heeft besteld.

Figuur 14: Een tekst met voorwaardelijke uitspraken

3.6.2.1 Toepassing van procedures

De tekst van figuur 14 en de tabel van figuur 13 beschrijven precies dezelfde procedure. We stellen ons nu het volgende experiment voor.

1. Kijk niet naar de beslissingstabel, maar enkel naar de tekst. Bepaal op grond van de tekst de korting, de vervoerswijze en het factuurtype voor het volgende geval:
   - de klant is groothandelaar;
   - de bestelde hoeveelheid is 12 stuks;
   - de afstand tussen de fabriek en de woonplaats van de klant is 120 km.
2. Kijk nu niet meer naar de tekst, maar enkel naar de beslissingstabel van figuur 13. Bepaal op grond van de beslissingstabel de korting, de vervoerswijze en het factuurtype voor het volgende geval:
   - de klant is groothandelaar;

- de bestelde hoeveelheid is 14 stuks;
- de afstand tussen de fabriek en de woonplaats van de klant is 80 km.

Na dit experiment zal u het volgende hebben opgemerkt. Allereerst bleek dat het ontdekken van de toe te passen acties veel minder tijd kostte bij raadpleging van de tabel dan bij lezing van de tekst. Bij gebruik van de tabel volstonden drie vragen, namelijk één voor elke conditie, om te bepalen wat er moet gebeuren. Bij het eerste experiment echter, waar men slechts de tekst had, heeft men zich maximaal 12 vragen moeten stellen. Sommige van deze vragen waren identiek, en dus overbodig. Zo heeft men zich een aantal malen moeten afvragen: is de klant een groothandelaar ? Dit illustreert een fundamenteel voordeel van de beslissingstabel: **versnelling van de beslissingsprocedure**. De grondreden hiervoor is - en hierop zijn nauwelijks uitzonderingen te vinden - dat teksten *actiegericht* zijn. Met andere woorden, een tekst die voorwaardelijke uitspraken beschrijft, is doorgaans ingedeeld in stukken die elk het toepassingsgebied van één bepaalde actie weergeven. Beslissingstabellen daarentegen zijn *conditiegericht*: eerst worden alle condities behandeld (maar iedere conditie slechts éénmaal) en hieruit vloeit automatisch het geheel van toepasselijke acties voort.

Merk ook op, dat bij gebruik van de beslissingstabel er zich gevallen zullen voordoen waarbij één vraag volstaat. Als bijvoorbeeld de klant geen groothandelaar is, dan ziet men direct dat de acties van R6 gelden. De tabel vertoont ook gevallen waarin twee vragen volstaan. Veronderstellen we nu even dat de frequentie waarmee ieder van de zes beslissingsregels zich in de praktijk voordoet even groot is. Dan kan worden berekend dat de behandeling van één geval door middel van de tabel gemiddeld 2,33 vragen vergt, en door middel van de tekst 7,66 vragen. Gebruikt men de tabel, dan verloopt het beslissingsproces dus minstens ongeveer 3,3 maal sneller dan bij gebruik van de tekst.

We mogen een nog grotere versnelling verwachten wegens het feit dat teksten vaak breedsprakig zijn; daarentegen staan in tabellen geen overbodige woorden. Bovendien kan de gebruiker van een tabel slechts één weg volgen en dit is altijd een korte, zoniet de kortst mogelijke. In een tekst moet hij zijn weg zelf zoeken; hij zal dus waarschijnlijk soms een stuk van de tekst lezen, dat niet toepasbaar is op het voorliggend geval. Ter illustratie: beschouw het geval dat de klant geen groothandelaar is. Gebruikt men de tekst, dan is voor de actie ‘korting’ alleen de eerste helft van de eerste zin toepasbaar: ‘Enkel aan groothandelaren wordt korting toegestaan’. Daarna kan de lezer overgaan naar punt 2 en mag hij de rest van punt 1 negeren. Dit wordt hem echter niet gezegd. Hij/zij zal tenminste doorlezen tot aan het einde van de eerste zin en de kans is groot dat hij/zij doorleest tot aan het einde van punt 1. Trouwens de voorzichtige lezer van de tekst weet dat hij/zij er goed aan doet verder te lezen, want in teksten schuilen veel addertjes onder het gras. Typische voorbeelden hiervan zijn wetteksten en

verzekeringscontracten, waarin de uitzonderingen verscholen zijn in de kleine lettertjes aan het einde. In beslissingstabellen is dit vermeden.

Er is echter nog meer. Niet alleen neemt men met een tabel sneller beslissingen; de **beslissingen** zullen ook **juister** zijn. Vooreerst stelt men met de tabel veel minder vragen; telkens wanneer een vraag wordt gesteld kan men zich vergissen en het antwoord verkeerd geven. Hoe minder vragen men zich stelt, hoe minder vergissingen men begaat. Een tweede reden voor het juistheidseffect is de onduidelijkheid of interpreteerbaarheid van teksten. In de orderbehandelingstekst komen uitdrukkingen voor zoals 'enkel', 'en dit in zover', 'of', 'maar', 'in alle andere gevallen', 'het normale type', 'per uitzondering', 'namelijk'. Deze termen zijn dikwijls voor interpretatie vatbaar ofwel worden zij verkeerd begrepen. (Wetteksten zijn hiervan sprekende voorbeelden.) In de overeenkomstige tabel zijn al deze uitdrukkingen verdwenen en allemaal vervangen door het ondubbelzinnige woord *en*, dat bij conventie de conditietoestanden binnen eenzelfde beslissingsregel verbindt.

Hiermee is op concrete wijze verduidelijkt dat beslissingstabellen bij de toepassing van procedures twee voordelen bieden: een verbetering van de **snelheid** en van de **juistheid** van de beslissingsname. Het gebruikte voorbeeld is bewust eenvoudig gehouden, maar in de praktijk is de versnelling en de juistheidstoename bij gebruik van tabellen doorgaans nog veel groter dan uit dit eenvoudig voorbeeld blijkt. Snelheid en juistheid zijn altijd na te streven, maar soms zijn ze van cruciaal belang. Een voorbeeld hiervan zijn procedures voor hulp bij eerste ongevallen bij personen die een giftig product hebben ingenomen.

3.6.2.2 Controle op volledigheid, contradictie en juistheid

De gegeven tekst omtrent de orderbehandeling (figuur 14) is volledig. Met andere woorden, hij geeft uitsluitsel over alle mogelijke gevallen die zich in de praktijk kunnen voordoen binnen het beschouwd probleemgebied. Hij dankt dit kenmerk aan het feit dat hij werd afgeleid uit de overeenkomstige beslissingstabel. Het normale geval is anders. Immers, doorgaans specificeert men voorwaardelijke procedures enkel in tekstvorm. Onder deze vorm worden zij aan o.a. systeemontwerpers voorgelegd en dan is het meestal onmogelijk en/of uiterst moeilijk na te gaan of de desbetreffende tekst exhaustief is. Trouwens, het is de ervaring van de auteur dat in de praktijk exhaustiviteit van teksten een uitzondering vormt. Niet-exhaustieve teksten brengen mee dat de lezer voor gevallen komt te staan die niet zijn voorzien. Hij of zij moet dan voor zichzelf beslissen welke acties moeten worden ondernomen. Met andere woorden, hij of zij moet de tekst interpreteren. En deze interpretatie kan ongewild verschillen van de bedoeling van de opsteller van de tekst.

Er bestaat een goede methode om uit een bestaande tekst een overeenkomstige beslissingstabel te maken. Deze methode kan men o.a. vinden in [9]. Vertoont de

op die wijze geconstrueerde tabel kolommen zonder actie-aanduidingen, of acties die in geen enkele kolom zijn aangestreept, dan is hij onvolledig. Aan de opsteller van de tekst kan dan worden gevraagd welke acties in ieder van deze gevallen gelden. Aldus is de beslissingstabel een uitstekend middel om procedures, door middel van teksten beschreven, exhaustief te maken.

Ten tweede zijn teksten vaak contradictorisch. Voor de tekst 'Orderbehandeling' hierboven geldt dit weer niet. Ook om dezelfde reden: de tekst is uit een beslissingstabel afgeleid. Teksten die niet werden gemaakt op grond van een tabel vertonen bijna altijd dit verwerpelijk kenmerk. Concreet betekent dit dat een bepaalde zin of paragraaf van de tekst voorschrijft dat in zekere gevallen een actie moet worden ondernomen, terwijl een andere zin of paragraaf voor een gedeelte van dezelfde gevallen een tegenstrijdige actie aangeeft. Voorbeeld van twee tegenstrijdige acties: korting = 5% en korting = 10%.

Tegenstrijdige uitspraken in teksten zijn veelal uiterst moeilijk op te sporen. Zet men een tekst echter om in een beslissingstabel, dan springen zij onmiddellijk in het oog: een uitspraak is tegenstrijdig als in dezelfde kolom twee tegenstrijdige acties zijn aangegeven.

Een concrete illustratie moge deze belangrijke voordelen van beslissingstabellen verduidelijken. Veronderstel dat een orderbehandelingsprocedure werd opgesteld volgens de tekst van figuur 15.

Orderbehandeling.

1. Korting
Enkel aan groothandelaren wordt korting toegestaan. De kortingpercentages zijn 10%, 5% en 2%: 10% voor de groothandelaren die minstens 15 stuks bestellen, of die op minder dan 50 km van de fabriek wonen en minstens 10 stuks bestellen; 5% voor de groothandelaren die tenminste 10 maar minder dan 15 stuks bestellen en die tenminste 50 km, maar minder dan 100 km van de fabriek verwijderd wonen; 2% voor de groothandelaren die tenminste 10, maar minder dan 15 stuks bestellen en die tenminste 100 km ver wonen.

2. Wijze van vervoer
Wij vervoeren per spoor indien de bestelling niet afkomstig is van een groothandelaar of als een groothandelaar tenminste 15 stuks heeft besteld. In alle andere gevallen wordt wegvervoer toegepast.

3. Type factuur
Het normale type is A. Per uitzondering moet een factuur van het type B worden opgesteld, namelijk voor de groothandelaar die minstens 15 stuks heeft besteld.

Figuur 15. Een niet-exhaustieve en tegenstrijdige tekst

Deze procedure verschilt zeer weinig van die uit figuur 14. Ze is echter niet exhaustief en bevat tegenstrijdigheden. Tracht eens, en dit natuurlijk zonder beide teksten te vergelijken, de niet-voorziene gevallen en contradicties in de tekst van figuur 15 te vinden. Misschien lukt het, maar de kans is veel groter dan het niet lukt. Moest men echter de tekst met de eerdergenoemde methode om teksten in beslissingtabellen om te zetten, behandelen, dan heeft men de absolute zekerheid die onvolledigheden en contradicties te vinden. De genoemde methode zou dan de tabel van figuur 16 produceren.

Vermits we hier de tabel gebruiken om onvolledigheden en contradicties te vinden, werden alle acties geformuleerd in een zodanige vorm dat de beslissingsregels in het actiegedeelte enkel streepjes en kruisjes bevatten.

Orderbehandeling	R1	R2	R3	R4	R5	R6
C1. Klant = groothandelaar	Ja	Ja	Ja	Ja	Ja	Neen
C2. Bestelde hoeveelheid (**H**)	$H<10$	$10 \leq H<15$	$10 \leq H<15$	$10 \leq H<15$	$H \geq 15$	-
C3. Afstand tussen fabriek en woning klant (in km) (**A**)	-	$A<50$	$50 \leq A<100$	$A \geq 100$	-	-
A1.1. Korting = 0%	-	-	-	-	-	X
A1.2. Korting = 2%	-	-	-	X	-	-
A1.3. Korting = 5%	-	-	X	X	-	-
A1.4. Korting = 10%	-	X	-	-	X	-
A2. Spoorvervoer	-	-	-	-	X	X
A3. Wegvervoer	X	X	X	X	-	-
A4.1. Type factuur= A	X	X	X	X	-	X
A4.2. Type factuur = B	-	-	-	-	X	-

Figuur 16: Beslissingstabel die toelaat volledigheid en contradictie te controleren

Beschouwen we nu alle acties die een korting toestaan. Voor R1 is geen enkel kortingspercentage opgegeven, zelfs niet 0%. Dit is een onvolledigheid, voortvloeiend uit het feit dat de eerste zin van figuur 15 verschillend is van deze van figuur 14. In figuur 15 werd immers de volgende zinsnede weggelaten: ‘en dit inzover zij tenminste een hoeveelheid van 10 stuks bestellen’. Verder springt in het oog, dat er een geval bestaat waarvoor zowel een korting van 2% als een korting van 5% wordt toegestaan (R4). Dit is een contradictie, veroorzaakt door het feit, dat we in figuur 15 ten opzichte van figuur 14 op een bepaalde plaats de zinssnede ‘maar minder dan 100 km’ hebben weggelaten.

Het geheel van de uitspraken vervat in een tekst moge volledig zijn en geen enkele uitspraak contradictorisch, maar daarmee zijn de uitspraken nog niet

noodzakelijk juist. In een beslissingstabel kan de juistheid van iedere uitspraak gemakkelijk worden geverifieerd door de tabel kolom per kolom te onderzoeken.

3.6.2.3 Opstelling van voorwaardelijke uitspraken

Een beslissingstabel is niet alleen een uitstekend middel om in teksten neergelegde voorwaardelijke uitspraken te toetsen op volledigheid, mogelijke tegenstrijdigheid en juistheid. Ze kan ook worden gebruikt om waterdichte voorwaardelijke uitspraken te maken. Dan gaat men omgekeerd tewerk: men maakt eerst de beslissingstabel en leidt daaruit de tekst af. Veelal blijkt het zelfs totaal overbodig na de constructie van de tabel nog een tekst te maken. De tabel is immers veel efficiënter dan de corresponderende tekst voor de toepassing van de voorgestelde uitspraken.

In [9] wordt een methode behandeld om in deze context de foutenvrije beslissingstabel op te stellen.

3.6.3 Relevantie voor de informatica

In de informatica kunnen beslissingstabellen worden gebruikt in zowel de fasen van de analyse en het ontwerp als bij de programmering.

In de eerste fasen van een project wordt men vaak geconfronteerd met teksten die te automatiseren procedures voorschrijven. Voor de *betrouwbaarheid* van het systeem is het van het grootste belang (1) dat deze teksten foutenvrij worden gemaakt, en (2) dat zij begrepen worden door de analyst. Deze teksten omzetten tot beslissingstabellen volgens de methode gepresenteerd in [9] is hiervoor aangewezen. Wil men hierbij feilloos, comfortabel en snel werken, dan gebruikt men best het tool PROLOGA (Procedural Logic Analyzer), dat gemaakt is binnen de researchgroep van Prof. Jan Vanthienen [jan.vanthienen@econ.kuleuven.ac.be].

Het komt ook vaak voor dat de te automatiseren materie niet in teksten vastgelegd werd, maar moet achterhaald worden door gesprekken. Dan kan de methode vermeld onder 3.6.2.3 vaak met veel nut worden gebruikt om op zinvolle wijze de ware toedracht van de zaak te achterhalen. De hierbij te gebruiken methode is ook vervat in PROLOGA.

Ook bij de programmatie kan veel nut gehaald worden uit het gebruik van beslissingstabellen. En wel in twee opzichten. Vooreerst kunnen zij automatisch omgezet worden in programma-onderdelen. In PROLOGA kan dit in meerdere programmeertalen. De voordelen zijn snelheid en *betrouwbaarheid*. Maar ook de *flexibiliteit* wordt bevorderd, want beslissingstabellen zijn onafhankelijke modules, die gemakkelijk kunnen gewijzigd worden en op dezelfde plaats zonder verdere risico's terug ingebracht worden.

3.7 Soberheid

Men heeft de zaken veel te ingewikkeld gemaakt. Programmeertalen bevatten veel te veel instructies. (In dit verband vergeleek Dijkstra destijds de programmeertaal PL/1 met een barok monster). Technieken voor systeemanalyse en –ontwerp bevatten veel te veel mogelijkheden (als voorbeeld hiervan moge gelden UML (Universal Modeling Language), die een onvoorstelbaar aantal grafische voorstellingswijzen bevat).

Het is sterk aan te bevelen zich te beperken tot enkele essentiële elementen uit het gamma van voorgestelde instructies en schema's en enkel déze te gebruiken. MERODE bvb. bezit voor de specificatie van systemen slechts drie schema's (een entity-relationship grafiek, een event-objecttabel en een finite state machine schema), en kan hier alles mee modelleren dat noodzakelijk is. Het voordeel van zich op deze wijze te beperken is drievoudig: (1) de ontwerpaandacht wordt niet afgeleid door de complexiteit van de voorstellingswijze; (2) de juistheid van het ontwerp is veel beter na te gaan; (3) men heeft meer vrijheidsgraden in het ontwerp zelf, hetgeen de flexibiliteit van het systeem bevordert. Het zou nuttig zijn ontwerpteams een stricte discipline op te leggen en bepaalde (op zichzelf eventueel krachtige) technieken te verbieden.

4 Besluit

De wetenschap (?) van de systeemontwikkeling heeft een ganse evolutie meegemaakt. Het gevaar is zeer reëel dat daarbij fundamentele en onveranderlijke principes uit het oog verloren worden. In dit artikel is een poging gedaan de voornaamste hiervan onder de aandacht te brengen. Wij zijn van mening dat in dit opzicht een grote verantwoordelijkheid rust op alwie zich bezighoudt met onderricht in de systeemontwikkeling, niet alleen binnen de scholen, maar ook daarbuiten. Het wordt steeds belangrijker nieuwe ontwikkelingen te toetsen aan de genoemde onveranderlijke constanten vooraleer men er onderricht over organiseert. Dit is in de praktijk moeilijk, want de verleiding is groot snel met het nieuwe uit te pakken, zonder verder na te denken. Een dergelijke houding is te betreuren.

Referenties

[1] Booch, G., Rumbaugh, J.and Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.

[2] Dedene, G., *Collegenota's Systems Analysis*, K.U.Leuven, 2001.

[3] Dijkstra, E. G., *Go To Statement Considered Harmful*, Communications of the ACM, Vol. 11, nr 3, 1968, pp. 147-148.

[4] Jackson, M.E., *Principles of Program Design*, Ac. Press, London, 1975.

[5] Jackson, M. E., *Software Requirements & Specifications*, A-Wesley, 1995.

[6] Naur, P. and Randall, B, editors, *Software Engineering, Report on a Conference*, NATO Scientific Affairs Division, Garmisch, 1968.

[7] King, J.E, *Logtab - A logic table technique*, General Electric Co., Schenctady, 1959.

[8] Snoeck, M., Dedene, G., Verhelst, M. and Depuydt, A., *Object-Oriented Enterprise Modelling with MERODE*, Leuven University Press, 1999.

[9] Verhelst, M., *De praktijk van beslissingstabellen*, Kluwer-Deventer/Antwerpen, 1980.