Procescalculus : de vertexmachine

Procescalculus: de vertexmachine

In dit artikel, het achtste in een serie, geven de auteurs aan hoe geïntegreerde schakelingen (ic's) worden vervaardigd. De firma ASM in Bilthoven ontwikkelt zogenoemde Advance-machines voor het vervaardigen van deze geïntegreerde schakelingen. Een vereenvoudigde versie, de Vertexmachine, wordt gebruikt voor het onderzoek naar het ontwikkelen van real-time besturingen met behulp van de procescalculus. In dit artikel aandacht voor het modelleren van de verschillende fysieke componenten van de machine en aan het modelleren van de besturing. De auteurs gaan speciaal in op het minimaliseren van de doorlooptijden van de produkten.

Prof.dr.ir.J.E.Rooda en ir. H.J.Vaes zijn verbonden aan de faculteit der Werktuigbouwkunde van de Technische Universiteit te Eindhoven.

Geïntegreerde schakelingen worden op grote schaal toegepast in verschillende elektronische apparaten. Computers spreken hierbij wellicht het meest tot de verbeelding.

De produktie van geïntegreerde schakelingen verloopt globaal als volgt: een cilindrische staaf van zuiver silicium met een diameter van 82,5 of 150 mm wordt in dunne plakken gezaagd. Iedere plak, ook wel wafer genoemd, ondergaat vervolgens enkele honderden bewerkingen.

De bewerkingen kunnen in vijf categorieën worden verdeeld: diffusiebewerkingen, lithografische bewerkingen, etsbewerkingen, implantatiebewerkingen en metalliseerbewerkingen.

Een diffusiebewerking brengt een laag aan op het oppervlak van een plak. Zo'n laag ontstaat door de plak in een oven te verhitten en bloot te stellen aan bepaalde gassen.

Een lithografische bewerking brengt een patroon aan op de plak. Hiertoe wordt de plak voorzien van een fotogevoelige laklaag, waarna deze laklaag wordt "belicht" met het patroon en de niet belichte delen van de laklaag tijdens het "ontwikkelen" weer worden verwijderd.

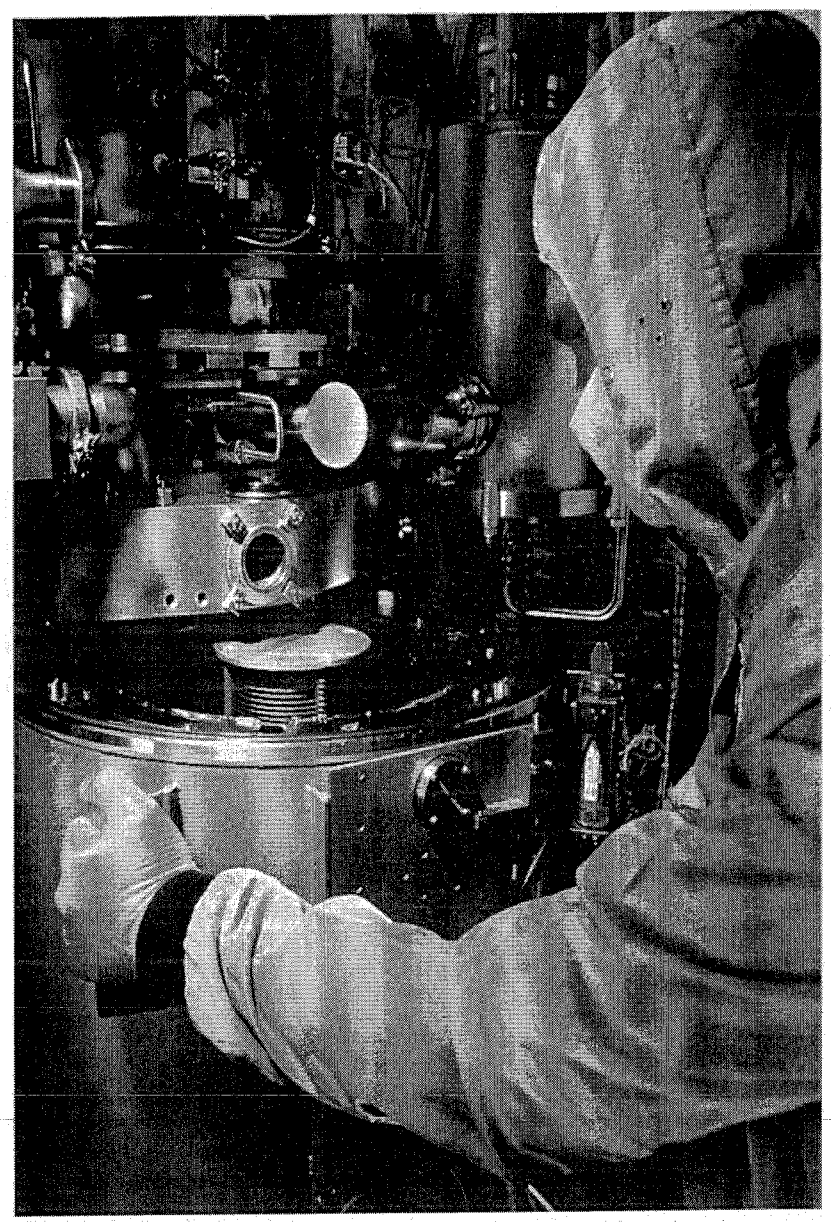
Een etsbewerking verwijdert delen van de oppervlaktelaag. De laklaag beschermt de onderliggende oppervlaktelaag tegen het etsen. Door het patroon van de laklaag worden de juiste delen verwijderd.

Bij een implantatiebewerking worden ionen geïmplanteerd in de oppervlaktelaag van de plak, terwijl een metalliseerbewerking nieuwe oppervlaktelagen van metaal op de wafer aanbrengt. Dit gebeurt door middel van het opdampen van metaal.

Een uitvoerige beschrijving van de

verschillende bewerkingen wordt gegeven door Sze [1983].

Iedere bewerkingsstap wordt meestal voorafgegaan door een reinigingsstap en gevolgd door een inspectie stap. Door vele van deze bewerkingen achter elkaar uit te voeren ontstaan complexe geïntegreerde schakelingen op de plak. Na afloop van alle bewerkingen wordt de plak in stukjes gezaagd. Ieder stukje wordt voorzien van contactpootjes en een behuizing.
Omdat de plakken tijdens de produktie zeer gevoelig zijn voor stof en oxidatie, vindt de produktie plaats in schone ruimtes, "clean rooms". In deze ruimtes bevinden zich naast de machines die nodig zijn voor de verschillende bewerkingen ook opslagmiddelen en transportinrichtingen.



Advance-machines
De behoefte aan geïntegreerde schakelingen met fijnere patronen noopt de producenten van deze produkten om de condities waaronder de bewerkingen plaatsvinden nog beter te beheersen. Om aan deze wens tegemoet te komen worden door leveranciers van produktiemiddelen clustermachines ontwikkeld. Met deze machines is het mogelijk om meerdere bewerkingsstappen direct na elkaar uit te voeren. De firma ASM brengt deze machines onder de naam Advance-machines op de markt. Foto 1 toont een zijaanzicht van een Advance-machine. De machine is modulair opgebouwd en wordt gevormd door een zeshoekig frame, waarin zich een transportsysteem bevindt en waaraan enkele reactoren kunnen worden bevestigd. Figuur 1 en 2 geven schematisch een bovenaanzicht en een zijaanzicht van de Advance-machine [Brugman, 1989]. De verschillende delen van de machine kunnen vacuüm worden gepompt. Omdat de plakken onder deze geconditioneerde omstandigheden niet worden blootgesteld aan de buitenlucht en omdat geen menselijke tussenkomst is vereist tussen de opeenvolgende bewerkingsstappen, wordt de kwaliteit van de produkten verhoogd, en zijn dus fijnere patronen mogelijk.

De bewerkingen gebeuren in reactoren. Het centrale transportsysteem wordt gevormd door een robot die de vorm heeft van een kikkerpoot. Deze robot kan de plakken één voor één transporteren tussen de verschillende reactoren, de in- en de uitvoermodule (speciale reactoren). Liften in de reactoren zorgen voor transport van de plakken in de reactor.
Omdat er meerdere reactoren zijn, die parallel bewerkingen uit kunnen voeren, kan de Advance-machine worden gebruikt als een job-produktie-machine [Rooda, Arentsen, Smit, 1992].

Vertexmachine
In het laboratorium voor Machinebesturingstechnologie van de leerstoel Automatisering van de Produktie is een logistiek equivalent van een Advance-machine aanwezig. Deze opstelling, de Vertexmachine, wordt gebruikt voor onderzoek naar real-time besturingsarchitecturen en naar de prestatie van de Advance-machine.
Foto 2 geeft een beeld van deze Vertexmachine. De opstelling bevat 6 liften en een transportsysteem, de kikkerpoot. Met de liften worden zowel de reactoren als de in- en uitvoermodules nagebootst.

De werking van de Vertex-machine is vergelijkbaar met de werking van de Advance-machine. De (gewenste) cyclus is in beginsel als volgt:
Een cassette met plakken wordt door de operateur in de invoermodule geplaatst. De plakken worden één voor één door de

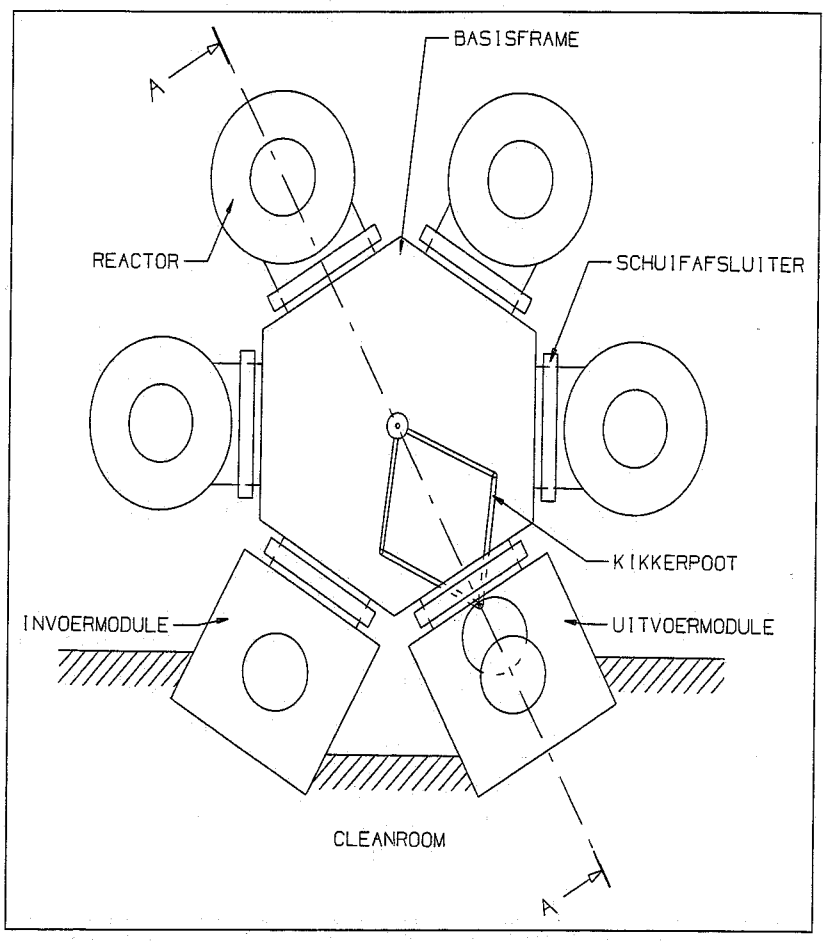


Fig. 1. Bovenaanzicht van de Advance-machine

kikkerpoot naar een reactor getransporteerd. Nadat de plakken hier zijn bewerkt, worden ze door de kikkerpoot naar een volgende reactor getransporteerd. Als alle benodigde bewerkingsstappen zijn uitgevoerd, worden de plakken door de kikkerpoot in de cassette geplaatst die aanwezig is in de uitvoermodule. De gevulde cassette wordt tenslotte door de operateur uit de uitvoermodule gehaald en vervangen door een lege cassette.

Specificatieblok 1

```
Processor Reactor
instance variable names: lift

initializeTasks
  lift ← OrderedCollection new

neemPlak
  self workDuring: 0.7 seconds.
  lift addLast: (self receiveFrom: 'kikkerpoot')

geefPlak
  self workDuring: 0.6 seconds.
  self send: lift removeLast to: 'kikkerpoot'

body
  | commando |
  commando ← self receiveFrom: 'machineBestuurder'.
  commando = #neemPlak ifTrue: [↑self neemPlak].
  commando = #geefPlak ifTrue: [↑self geefPlak].
  self workDuring: commando
```

Model van de Vertexmachine

Met behulp van de procescalculus is een model beschreven dat de relevante aspecten (van het gedrag) van de fysieke machine modelleert [Vaes, 1989]. Dit model is gebruikt voor de ontwikkeling van een real-time besturing voor de fysieke machine. De beschreven besturing bevat enkele besturingsprocessoren. Deze vormen tezamen een gesloten lusbesturing. Dit wil zeggen dat een besturingprocessor na het zenden van een stuuractie een bevestiging wenst te ontvangen. Hiermee is het mogelijk om fouten, bijvoorbeeld storingen, terug te koppelen naar hogere besturingsniveaus.

Ten behoeve van dit artikel is dit model vereenvoudigd. Door de aanname dat de machine foutloos functioneert, is het mogelijk om een open lusbesturing te gebruiken: er is geen terugkoppeling. De architectuur van de besturing komt evenwel overeen met de besturingsarchitectuur zoals beschreven door Vaes.

Het hier gepresenteerde model van de fysieke machine en zijn besturing beschrijven we nu top-down.
Het bovenste niveau vormt een abstract model van de Vertexmachine en zijn omgeving. Op dit niveau worden drie processoren onderscheiden: de bladprocessoren Leverancier en Afnemer en de geëxpandeerde processor VertexMachine, zie figuur 3.

Materiaal- en informatiestroom

In figuur 3 is te zien dat in het model rekening wordt gehouden met een materiaalstroom die loopt van de Leverancier via de Vertexmachine naar de Afnemer, en een informatiestroom die de omgekeerde weg volgt. De Afnemer (een andere afdeling of klant) genereert orders en stuurt deze naar de Vertexmachine. Deze reageert daarop door de Leverancier (een andere afdeling of ander bedrijf) te vragen om materiaal. De Leverancier levert het materiaal aan de Vertexmachine, die de bewerkingen doet en het materiaal vervolgens verstuurt naar de Afnemer.
Het materiaal wordt in dit model beschreven door objecten [Rooda, 1992] van de klasse:

De instantiaties van deze klasse hebben geen specifieke kenmerken zoals een naam of status. Vandaar dat bij de klasse geen instance variabelen zijn gedefinieerd.

De informatiestroom wordt gevormd door informatie-objecten, waar we in een later stadium op ingaan.
Op de processoren Leverancier en Afnemer gaan we in dit artikel niet verder in.
De geëxpandeerde processor Vertexmachine is een model van de Vertexmachine. In figuur 4 is te zien dat de Vertexmachine twee processoren bevat, namelijk de geëxpandeerde processor Machine en de geëxpandeerde processor MachineBestuurder. Op dit niveau wordt de scheiding aangebracht tussen bestuurder en bestuurde machine. De MachineBestuurder zal de Machine gaan besturen.

Machine

Het doel van het beschrijven van een model van de fysieke machine is om inzicht te krijgen in de werking van deze machine en om de te ontwikkelen besturing te kunnen testen aan de hand van dit model. De processor Machine modelleert de relevante aspecten van het gedrag van de fysieke machine. Als dit model niet correct is, kan het zijn dat de besturing, die getest is met behulp van het model van de fysieke machine, niet is staat is de fysieke machine zelf correct te besturen.
Figuur 5 beschrijft de processor Machine. Het model is analoog aan de werkelijke machine en bestaat uit zeven processoren: een Kikkerpoot, een InvoerModule, een UitvoerModule en vier Reactoren. De Kikkerpoot is een model van de fysieke kikkerpoot zoals aanwezig in de fysieke machine. Zo geldt dat ook voor de andere processoren.

Naast de zeven processoren is in figuur 5 ook de materiaalstroom gemodelleerd door middel van de interactiepaden. Over deze paden kunnen objecten van de klasse Plak worden verstuurd. De plakken komen door de poort 'leverancier' binnen in de InvoerModule, worden door de Kikkerpoot getransporteerd, komen uiteindelijk terecht in de UitvoerModule, en verlaten vervolgens de VertexMachine door de poort 'afnemer'.
De processoren In- en Uitvoermodule worden in dit artikel niet behandeld omdat ze veel overeenkomen met de processor Reactor. De processor Reactor zal nu worden behandeld, gevolgd door de processor Kikkerpoot.

Reactor

De fysieke reactor, zoals aanwezig in de fysieke machine, heeft een machinebesturing die ervoor zorgt dat hij door de te ontwikkelen besturing kan worden aangesproken met commando's die een drietal waarden kunnen hebben: #neemPlak, #geefPlak en een getal dat de bewerkingduur aangeeft. De machinebesturing interpreteert deze commando's en zorgt dat de bijbehorende acties worden uitgevoerd.
Bij het beschouwen van de processor Reactor, een model voor zo'n fysieke reactor, zijn we aangekomen op het niveau van de bladprocessoren. De beschrijving van processor Reactor is te vinden in specificatieblok 1.
De processor Reactor maakt gebruik van een instance variable met de naam lift. Op deze wijze wordt gemodelleerd dat een fysieke reactor een lift bevat om plakken in op te slaan. Deze lift werkt als een stapelaar: de plak die het laatst is ingebracht wordt het eerste verwijderd (LIFO). De instance variabele lift wordt geïnitialiseerd als een Ordered-Collection, waaraan een plak kan worden toegevoegd of worden verwijderd met de methoden add:, removeLast:, etc. Deze methoden zijn standaard geïmplementeerd in de klasse Ordered-Collection in de taal Smalltalk.

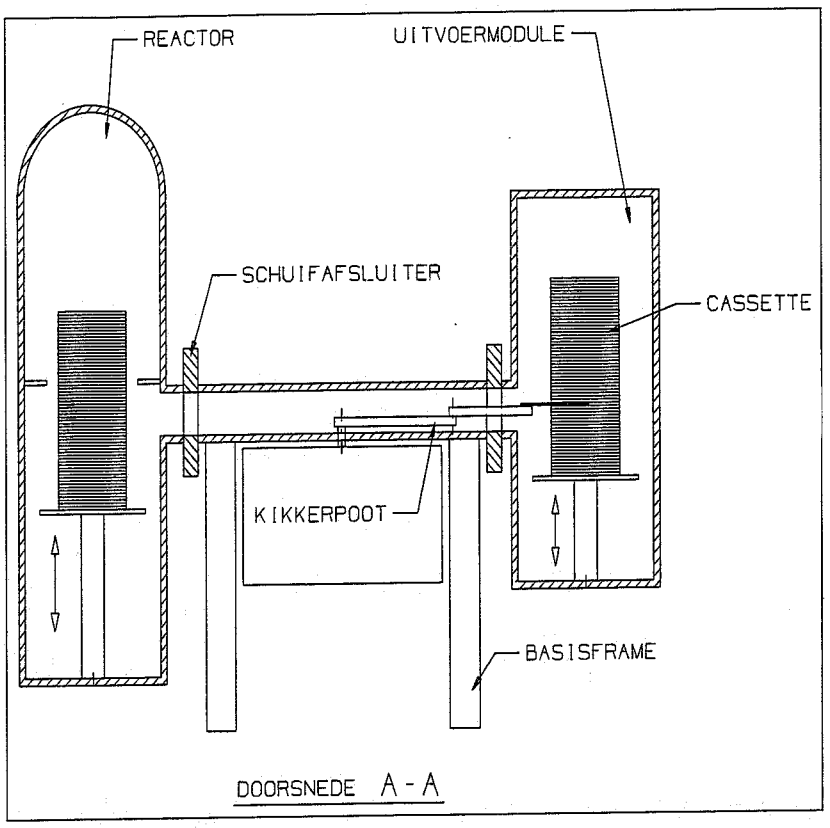


Fig.2. Doorsnede van de advance-machine

Kikkerpoot

Ook de fysieke kikkerpoot bevat nog een machinebesturing. Deze is in staat om de volgende commando's te interpreteren: #neemPlak, #geefPlak en een karakterreeks die de positie (een reactor) aangeeft waar de kikkerpoot naartoe moet. Het gedrag van de fysieke kikkerpoot wordt gemodelleerd door de Kikkerpoot, zoals beschreven in specificatieblok 2.
De beschrijving van de processor Machine is nu compleet. Er is een model gepresenteerd dat het gedrag van de fysieke machine zo goed mogelijk beschrijft. Het model bevatte vier soorten processoren, waarvan twee soorten nader zijn beschouwd. Nu zal ingegaan worden op de besturing.

MachineBestuurder

De processor MachineBestuurder vormt de besturing voor de processor Machine, en in een later stadium voor de fysieke machine zelf. Over het gewenste gedrag van de MachineBestuurder is reeds het één en ander bekend.

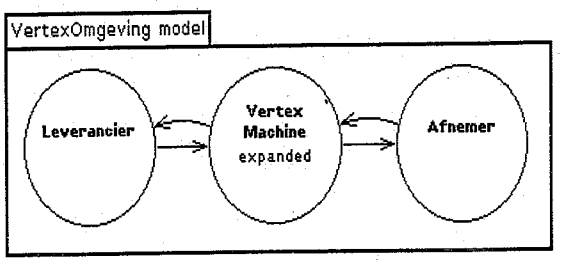


Fig.3. Beschrijving van het model van de vertexmachine en zijn omgeving

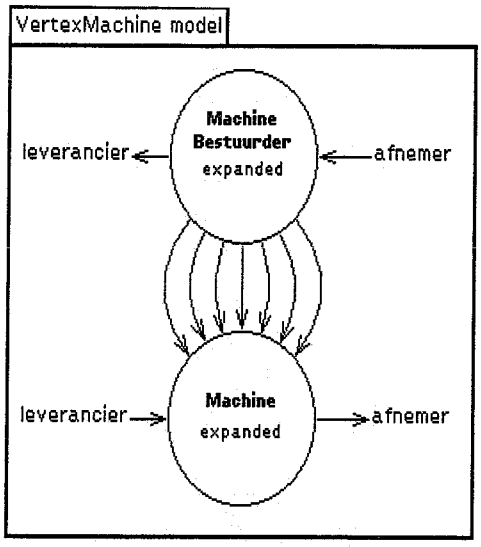


Fig.4. Beschrijving van de processor VertexMachine

Zo ontvangt hij orders van de Afnemer en moet hij sub-orders, voor levering van onbewerkte plakken, sturen naar de Leverancier. Verder zorgt hij dat de plakken ingevoerd, getransporteerd, bewerkt en uitgevoerd worden. Daartoe verstuurt hij commando's naar de verschillende machine-onderdelen.
Het is dus de taak van de MachineBestuurder om de vertaling te maken van de orders naar de verschillende benodigde sub-orders en commando's die noodzakelijk zijn voor het afhandelen van de orders.
Alvorens in te gaan op de ontwikkelde besturing, zal eerst worden beschreven hoe een order eruit ziet.

Order

Een order is een object dat door de Afnemer wordt opgesteld. Een order beschrijft hoeveel plakken de Afnemer wil en welke bewerkingen deze plakken moeten hebben ontvangen. Zo'n order is een object van de klasse:

Object Order
instance variable names: aantal Plakken recept

De instance variable recept representeert een geordende lijst (OrderedCollection) van receptstappen, die tezamen de bewerkingen beschrijven die de plakken in volgorde moeten ondergaan. Een receptstap beschrijft op welke reactor gedurende hoelang de bewerking moet plaatsvinden en is een object van de klasse:

Object ReceptStap
instance variable names: reactor Naam bewerkingsDuur

Een voorbeeld van een order is:

Order new
aantalPlakken: 5;
recept: (OrderedCollection new
add: (ReceptStap new
reactorNaam: 'reactor3';
bewerkingsDuur: 10 minutes);
add: (ReceptStap .....);
add: (ReceptStap ....))

Als de MachineBestuurder zo'n order ontvangt, moet hij zorg dragen dat de plakken aanwezig zijn en dat ze de opgegeven receptstappen ondergaan.

Tijdens het ontwikkelen van de besturing is gekozen om deze taak niet door één processor te laten uitvoeren, maar te verdelen over meerdere processoren. De besturingsarchitectuur die dan ontstaat is te zien in figuur 6.
In deze figuur is te zien dat de besturingstaak opgedeeld wordt over processoren op drie verschillende niveaus. De besturingsprocessoren zullen nu nader worden bekeken.

Verroosteraar

De Verroosteraar ontvangt de orders, bestelt de onbewerkte plakken en definieert voor iedere receptstap een opdracht. Een opdracht beschrijft over hoeveel plakken het gaat (alle plakken van de order), waar ze zich bevinden en welke receptstap ze als volgende moeten ondergaan. Een opdracht is een object van de klasse:

Object Opdracht
instance variable names: aantal Plakken bron receptStap

De Verroosteraar stuurt de, voor de order benodigde, opdrachten één voor één naar de Uitvoerder. De beschrijving van de Verroosteraar komt verderop in dit artikel uitgebreid aan bod.

Uitvoerder

De Uitvoerder ontvangt zo'n opdracht en zorgt ervoor dat hij wordt uitgevoerd. Hiervoor moet hij zorgen dat de plakken worden getransporteerd van de bron naar de bestemming waar ze de bewerking moeten ondergaan, en hij moet de bewerking "opstarten". Aangezien de kikkerpoot maar één plak tegelijkertijd kan transporteren, moeten de commando's voor het transport voor iedere plak van de order apart worden gegeven.
De informatie die de Uitvoerder nodig heeft, bevindt zich in de ontvangen opdracht. Zo zijn het aantal plakken en de bron expliciet vermeld en kunnen de

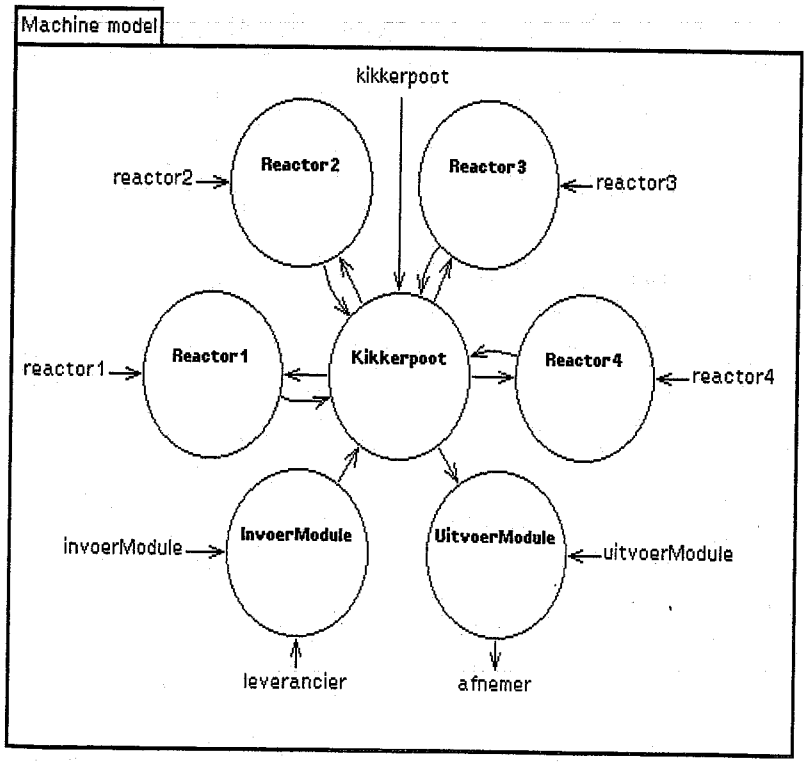


Fig.5. Beschrijving van de processor Machine

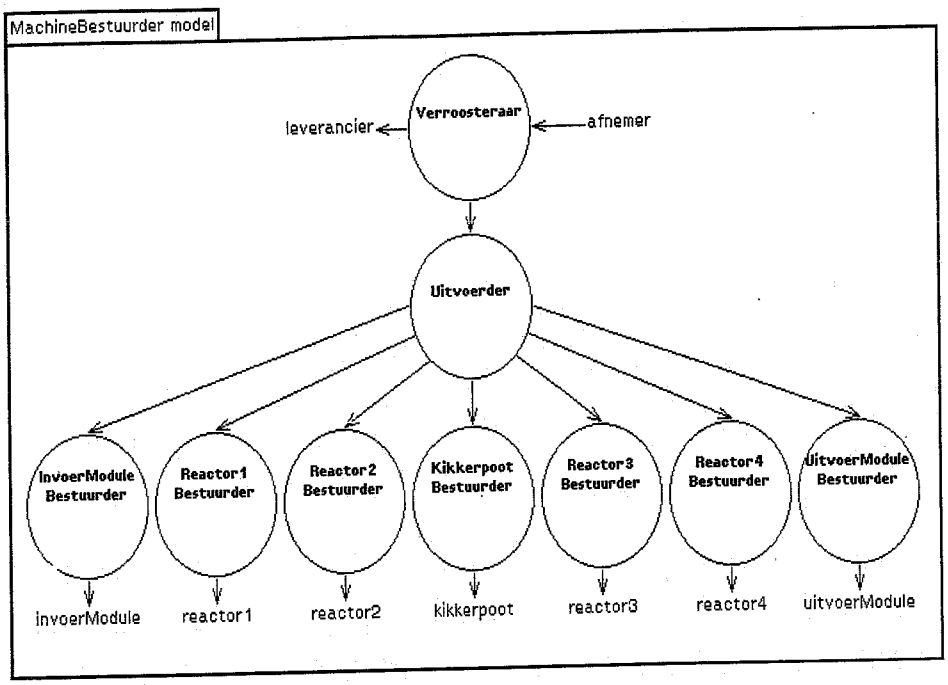


Fig.6. Beschrijving van de processor MachineBestuurder

bestemming en de bewerkingsduur worden verkregen door inspectie van de receptstap.
Om de taak van de Uitvoerder te vereenvoudigen, is een derde laag besturingsprocessoren toegevoegd. Dit maakt het mogelijk om het transport te besturen met objecten van de klasse:

Object TransportCommando
instance variable names: bron bestemming

Zo'n commando bevat de naam van de bron en de naam van de bestemming van de te transporteren plak.
Voor de procesbeschrijving van de Uitvoerder zie specificatieblok 3.
Door het invoeren van de derde laag processoren is de complexiteit van de Uitvoerder verminderd. In specificatieblok 3 is te zien dat de Uitvoerder tracht zoveel mogelijk aktiviteiten parallel te laten plaatsvinden.

KikkerpootBestuurder

De KikkerpootBestuurder ontvangt een transportcommando en verstuurt enkele nieuwe commando's naar de Kikkerpoot in een format dat de Kikkerpoot kan interpreteren. De procesbeschrijving van de KikkerpootBestuurder is te vinden in specificatieblok 4.

ReactorBestuurder, In- en UitvoerModuleBestuurder

De ReactorBestuurder, In- en UitvoerModuleBestuurder treden in dit model slechts op als doorgeefluik naar de Reactor, In- en UitvoerModule. Ze ontvangen een object en sturen het meteen door. De procesbeschrijving van deze processoren zijn erg eenvoudig, ze bevatten slechts één ontvangaktie en één zendaktie. De processoren zijn in het model opgenomen omdat ze in een later stadium de koppeling moeten vormen met de fysieke machine (hardware).

Verroosteraar

De Verroosteraar is de meest complexe processor van het gehele model. Alvorens hij zal worden beschreven wordt kort ingegaan op de wijze waarop de VertexMachine kan worden gebruikt. Hierin kan onderscheid worden gemaakt tussen drie verschillende situaties. In situatie 1 komt steeds maar één order tegelijk bij de VertexMachine binnen. Pas als deze is verwerkt, komt een eventuele nieuwe order. In deze situatie is de Verroosteraar zeer eenvoudig en die zal hier dan ook niet worden behandeld. In situatie 2 wordt de VertexMachine gebruikt in een lijnproduktie situatie. Er komen steeds meerdere orders binnen, die alle hetzelfde recept hebben. Ook de Verroosteraar voor deze situatie zal niet worden behandeld. In situatie 3 komen steeds meerdere orders binnen, ieder met een eigen recept. Deze situatie is de reeds eerder behandelde job-produktie situatie [Rooda, Arentsen, Smit, 1992]. Hier zal nu verder op worden ingegaan.

In een job-produktie situatie ontvangt de Verroosteraar van de Afnemer steeds een lijst met orders. Ieder van deze orders leidt tot een aantal opdrachten die uitgevoerd moeten worden. Het is

Specificatieblok 2

```
Processor Kikkerpoot
instance variable names: positie plak

initializeTasks
  positie ← 'invoerModule'

neemPlak
  self workDuring: 1 seconds.
  plak ← self receiveFrom: positie.
  self workDuring: 1 seconds

geefPlak
  self workDuring: 1 seconds.
  self send: plak to: positie.
  plak ← nil.
  self workDuring: 1 seconds

roteer: eenPositie
  self workDuring: 4 seconds.
  positie ← eenPositie

body
  | commando |
  commando ← self receiveFrom: 'machineBestuurder'.
  commando = #neemPlak ifTrue: [↑self neemPlak].
  commando = #geefPlak ifTrue: [↑self geefPlak].
  self roteer: commando
```

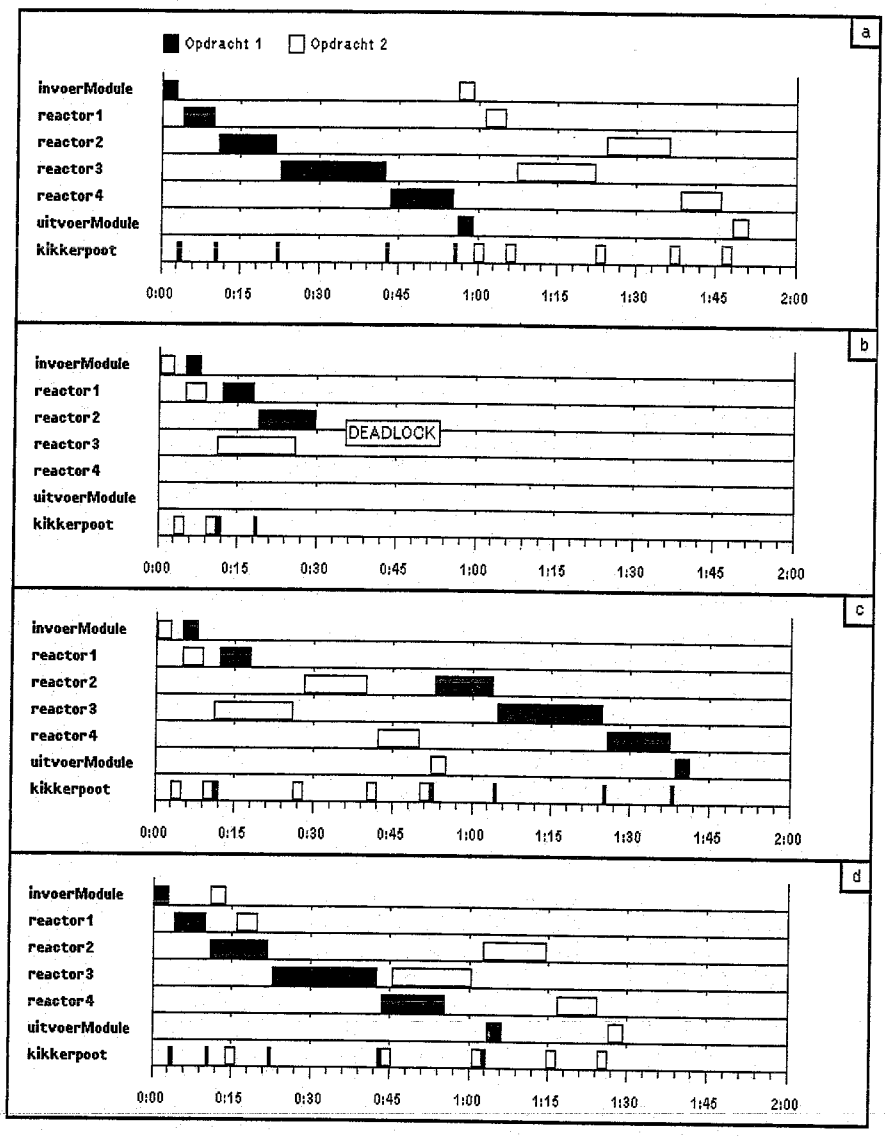


Fig. 7. Een gantt-chart van een voorbeeld voor vier verschillende verroosteraars

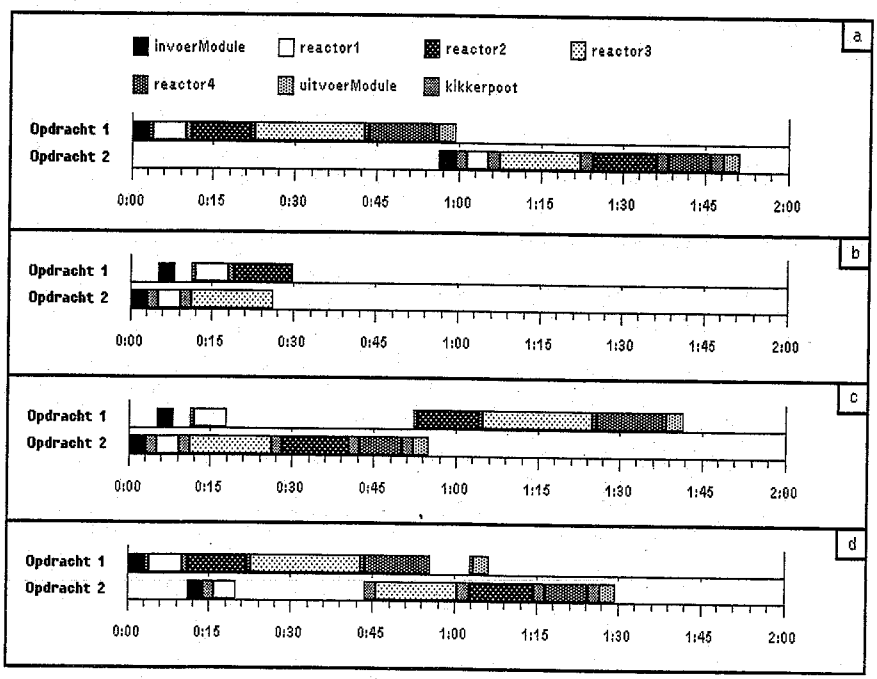


Fig. 8. De gantt-chart van figuur 7 geïnverteerd

nu de functie van de Verroosteraar om een volgorde te bepalen waarop hij deze opdrachten één voor één naar de Uitvoerder stuurt. Het is duidelijk dat hier geen sprake is van één mogelijke volgorde, maar dat de Verroosteraar voor de volgende te versturen opdracht steeds meerdere mogelijkheden heeft. De Verroosteraar zal dus steeds moeten kiezen voor een volgende opdracht. Welke opdracht hij kiest en tot welke volgorde dit leidt is (meestal) niet bepalend voor de kwaliteit van de produkten. Wel bepaalt de volgorde andere aspecten, zoals de doorlooptijd van de orders en de bezettingsgraad van de reactoren. Het kiezen van een volgorde wordt in de literatuur scheduling (verroosteren) genoemd.

Nu zullen drie verschillende verroosteraars worden gepresenteerd, die alle op basis van andere criteria een keuze maken voor de volgende opdracht die ze versturen. Met behulp van de simulator [Wortmann, 1991] zal het gedrag van deze verroosteraars nader worden onderzocht. De eerste verroosteraar is eenvoudig; de tweede verroosteraar probeert de doorlooptijden te minimaliseren, maar blijkt niet naar behoren te functioneren; de derde verroosteraar is wel functioneel bij het minimaliseren van de doorlooptijden van de orders.

Verroosteraar 1

De meest eenvoudige Verroosteraar (1) verstuurt de benodigde opdrachten in een zodanige volgorde naar de Uitvoerder dat er steeds maar aan één order tegelijkertijd wordt gewerkt. De beschrijving van deze Verroosteraar is te vinden in specificatieblok 5.

De volgorde die door deze Verroosteraar wordt aangehouden kan worden weergegeven met behulp van een zogenaamde "gantt-chart". Een gantt-chart geeft (grafisch) activiteiten weer van verschillende machines in de tijd. Iedere lijst met orders zal een andere gantt-chart te zien geven. Figuur 7a is een gantt-chart, gemaakt door Verroosteraar (1) voor twee, hier niet nader vermelde, orders. Duidelijk is te zien dat er nauwelijks sprake is van een parallelle verwerking van de orders.

Verroosteraar 2

Een tweede Verroosteraar is in staat de totale bewerkingstijd (= tijd waarop de laatste order klaar is) terug te brengen door een andere volgorde te kiezen. Zo kan hij proberen in het voorbeeld van figuur 7 de beide orders zoveel mogelijk tegelijk te laten verwerken. Dit kan door niet alle opdrachten van een order achter elkaar te versturen, maar door een ander criterium te gebruiken voor het bepalen van de volgorde. Een voorbeeld van zo'n criterium kan zijn om, over alle orders gezien, de receptstap met de kortste bewerkingsduur voorrang te verlenen (shortest processing time first = SPT). Een ander criterium kan zijn om de volgende receptstap van de order die nog de langste tijd nodig heeft tot zijn voltooiing voorrang te verlenen (longest remaining processing time first = LRPT). Er bestaat uitvoerige literatuur over het gebruik van verroosteraars en de effecten van verschillende van deze criteria [Lenstra, Rinnooy Kan, 1983; Lawler, et. al., 1989]. Het effect van deze criteria kan vervolgens aan de hand van een gantt-chart worden uitgebeeld. De aangepaste Verroosteraar (2), die werkt volgens het SPT-criterium wordt beschreven in specificatieblok 6.

Deze Verroosteraar is in staat een beter volgorde te bepalen dan de eerste Verroosteraar. Als deze Verroosteraar echter de orders uit figuur 7a moet verwerken, dan blijkt "deadlock" op te treden. Deadlock is een situatie waarin geen uitweg meer mogelijk is, omdat één of meer acties niet kan worden voltooid voordat een andere voltooid is. Figuur 7b geeft de gantt-chart van de deadlock situatie die hier optreedt. In dit geval moeten de plakken van order1, die zich in reactor2 bevinden, worden getransporteerd naar reactor3. Dit kan niet, want daar bevinden zich de plakken van order2 nog. Die kunnen daar niet worden weggehaald, want ze moeten naar reactor1 en die was juist bezet. Uit deze situatie kan alleen de operateur nog een uitweg bieden door de recepten van de orders te veranderen.

```
Processor  Uitvoerder

body
  | opdracht bron bestemming bewerkingsDuur |
  opdracht ← self receiveFrom: 'verroosteraar'.
  bron ← opdracht bron.
  bestemming ← opdracht receptStap reactorNaam.
  bewerkingsDuur ← opdracht receptStap bewerkingsDuur.
  taak aantalPlakken timesRepeat:
    [self send: #geefPlak to: bron.
    self send: #neemPlak to: bestemming.
    self send: (TransportCommando new
                  bron: bron;
                  bestemming: bestemming)
          to: 'kikkerpoot'].
  self send: bewerkingsDuur to: bestemming
```

Specificatieblok 3

```
Processor  KikkerpootBestuurder

body
  | transportCommando bron bestemming |
  transportCommando ← self receiveFrom: 'uitvoerder'.
  bron ← transportCommando bron.
  bestemming ← transportCommando bestemming.
  self send: bron          to: 'kikkerpoot'.
  self send: #neemPlak     to: 'kikkerpoot'.
  self send: bestemming    to: 'kikkerpoot'.
  self send: #geefPlak     to: 'kikkerpoot'
```

Specificatieblok 4

Verroosteraar 3

Een manier om deadlock te voorkomen is de introductie van bufferposities zoals in [Rooda, Arentsen, Smit, 1992] is beschreven. Dit is door de uitvoeringsvorm van deze machine uitgesloten.

Een andere manier om deadlock te voorkomen is om te voorspellen of deadlock zal optreden, alvorens een volgorde-beslissing te nemen. Dit is een complex probleem omdat deadlock kan volgen uit de combinatie van een groot aantal factoren. Een vaker toegepaste manier om deadlock te voorkomen is het vooraf opstellen van een compleet plan. Dit

Specificatieblok 5

```
Processor  Verroosteraar  (1)

body
  | orderLijst bron bestemming |
  orderLijst ← self receiveFrom: 'afnemer'.
  orderLijst do:
    [ :order |
    self send: order aantalPlakken to: 'leverancier'.
    bestemming ← nil.
    order recept do:
      [ :receptStap |
      bron ← bestemming.
      bestemming ← receptStap reactorNaam.
      self send: (Opdracht new
                    aantalPlakken: order aantalPlakken;
                    bron: bron;
                    receptStap: receptStap)
            to: 'uitvoerder']]
```

```
Processor Verroosteraar (2)
instance variable names: reactors

initializeTasks
  reactors ← Dictionary new.
  reactors at: 'reactor1' put: nil.
  reactors at: 'reactor2' put: nil.
  reactors at: 'reactor3' put: nil.
  reactors at: 'reactor4' put: nil.
  reactors at: 'invoerModule' put: nil.
  reactors at: 'uitvoerModule' put: nil

body
  | orderLijst order receptStap bestemming bron |
  orderLijst ← self receiveFrom: 'afnemer'.
  orderLijst ← orderLijst asSortedCollection: [ :x :y |
                    x recept first bewerkingsDuur <
                    y recept first bewerkingsDuur].
  [orderLijst reSort.
  orderLijst isEmpty]
    whileFalse:
      [order ← orderLijst detect: [ :o |
                                receptStap ← o recept first.
                                bestemming ← receptStap reactorNaam.
                                (reactors at: bestemming) isNil]
                        ifNone: [self error: 'deadlock'].
      bron ← reactors keyAtValue: order ifAbsent: [nil].
      bron isNil ifFalse: [reactors at: bron put: nil].
      reactors at: bestemming put: order.
      reactors at: 'uitvoerModule' put: nil.
      order recept removeFirst.
      order recept isEmpty ifTrue: [orderLijst remove: order].
      bron isNil ifTrue: [self send: order aantalPlakken to: 'leverancier'].
      self send: (Opdracht new
                  aantalPlakken: order aantalPlakken;
                  bron: bron;
                  receptStap: receptStap)
        to: 'uitvoerder']
```

Specificatieblok 6

kan bijvoorbeeld door de volgorde-beslissingen te simuleren, en zodra dit leidt tot deadlock terug te komen op deze beslissing. Als op deze manier een goed plan is opgesteld kunnen de opdrachten zonder risico volgens dit plan worden uitgevoerd.
Op de laatst beschreven manier is de derde Verroosteraar ingericht. Door middel van een recursief algoritme (verrooster) stelt hij een rooster (plan) op, en vervolgens voert hij dit plan uit. Dit algoritme voert 'backtracking' uit in geval van een deadlock. Dat wil zeggen dat hij terugkeert op zijn laatst genomen beslissing.
Verroosteraar (3) is een deadlock-vrije Verroosteraar, die gebruik maakt van het SPT-criterium: zie specificatieblok 7.
De gantt-chart van deze Verroosteraar met een SPT-criterium is gegeven in figuur 7c. Duidelijk is te zien dat het plan deadlock vermijdt door het uitstellen van de derde bewerking van de eerste order. Het blijkt dat dezelfde Verroosteraar met een LRPT-criterium bij deze orders een nog beter resultaat geeft: figuur 7d. De verschillende relevante gantt-charts zijn geïnverteerd weergegeven in figuur 8. Het blijkt dat door toepassing van een "slimme" Verroosteraar de prestatie, voor de twee orders van het voorbeeld, kan worden verbeterd met circa 20% (De prestatie is uiteraard afhankelijk van de aangeboden orders).

Tot dusver is het gedrag van de machine en zijn bijbehorende besturing onderzocht met behulp van een model. Hiervoor is het niet nodig om te beschikken over een fysieke Vertex-machine. Voor het verder toetsen van de juistheid van het model wordt nu gebruik gemaakt van de Vertex-machine.

Besturing van de Vertexmachine

De hier gepresenteerde besturing wordt nu gebruikt voor de besturing van de Vertexmachine. Hierbij worden de model-processoren, die de fysieke machines modelleren, vervangen door koppelingen ("interfaces") met de fysieke machines. In het laboratorium is hierbij zoals eerder beschreven gebruik gemaakt van een meer robuust model met terugkoppelingen [Vaes, 1989; Aarts, 1990; Klaassen, 1990]. Voor het handig invoeren van de verschillende orders is daarnaast een zogenaamde recept-editor ontwikkeld [Neerrijnen, 1990]. Het bleek dat de machine real-time kon worden bestuurd zonder de beschrijving van de besturingsprocessoren te veranderen of aan te passen. Het voordeel van de hierboven geschetste werkwijze is dat programmatuur uitvoerig kan worden getest op zijn correctheid en gedrag zonder dat men beschikt over de fysieke machine.

Nabeschouwing

In dit artikel is aangegeven hoe men een complexe bewerkingsmachine en zijn besturing kan modelleren met behulp van object-georiënteerde technieken en de procescalculus.
De in dit artikel gebruikte werkwijze voor het ontwerpen van machines en hun besturingen verschilt van de traditionele wijze waarop dit gebeurt.
De traditionele werkwijze kan grofweg worden onderverdeeld in de volgende aktiviteiten: het vastleggen van het doel van de te realiseren machine, het functioneel specificeren van de machine en het functioneel specificeren van de besturing, het technisch specificeren van de machine en het technisch specificeren van de besturing, het realiseren van de machine en het realiseren van de besturing en het testen van de machine met zijn besturing. Deze aktiviteiten vinden doorgaans na elkaar, sequentieel, plaats.
De hier geschetste werkwijze kan worden beschouwd als een vorm van gelijktijdig, parallel of concurrent, ontwerpen van de machine en zijn besturing en het realiseren van de besturing. Deze werkwijze heeft een aantal voordelen: de doorlooptijd van doel tot realisatie wordt bekort, en de kwaliteit van de machine en zijn besturing is beter. In een vroeg stadium worden specificatiefouten in de machine en in de besturing vastgesteld. Met name fouten in de architectuur van de besturing worden vroegtijdig hersteld. Wel dient een model van de fysieke machine te worden gemaakt. Een dergelijk model verschaft inzicht in het logistieke gedrag van de machine: men begrijpt de werking van de machine en zijn besturing

voordat de machine "in ijzer" is gerealiseerd.

Opgaven

1. Beschrijf de processor Afnemer en de processor Leverancier. Gebruik hiervoor de andere artikelen uit deze serie en de gegevens uit dit artikel.
2. Beschrijf de processor Invoermodule en de processor Uitvoermodule.
3. Onderzoek de werking van Verroosteraar (2) en Verroosteraar (3) aan de hand van zelfgekozen voorbeelden.

```
Processor Verroosteraar (3)
instance variable names: reactors orderLijst rooster

initializeTasks
  <zie Verroosteraar (2)>

verrooster
  | receptStap bestemming vre t1 t2 bron klaar |
  orderLijst isEmpty ifTrue: [↑true].
  orderLijst reSort.
  orderLijst do:
    [ :order |
    receptStap ← order recept first.
    bestemming ← receptStap reactorNaam.
    (reactors at: bestemming) isNil
      ifTrue:
        [vre ← rooster reverse detect: [ :re | re order = order] ifNone: [nil].
        bron ← vre isNil ifTrue: [nil] ifFalse: [vre receptStap reactorNaam].
        t1 ← rooster isEmpty  ifTrue: [self time]
                              ifFalse: [rooster last transportEindTijd].
        t2 ← vre isNil ifTrue: [self time] ifFalse: [vre bewerkingsEindTijd].
        t1 ← t1 max: t2.

        bron isNil ifFalse: [reactors at: bron put: nil].
        reactors at: bestemming put: order.
        reactors at: 'uitvoerModule' put: nil.
        order recept removeFirst.
        order recept isEmpty ifTrue: [orderLijst remove: order].
        rooster add: (RoosterElement new
                        aantalPlakken: order aantalPlakken;
                        bron: bron;
                        receptStap: receptStap;
                        order: order;
                        startTijd: t1).

        klaar ← self verrooster.

        bron isNil ifFalse: [reactors at: bron put: order].
        reactors at: bestemming put: nil.
        order recept addFirst: receptStap.
        order recept size = 1 ifTrue: [orderLijst add: order].
        klaar ifTrue: [↑true] ifFalse: [rooster removeLast]]].
  ↑false

body
  | order |
  orderLijst ← self receiveFrom: 'afnemer'.
  orderLijst ← orderLijst asSortedCollection: [ :x :y |
                  x recept first bewerkingsDuur <
                  y recept first bewerkingsDuur].
  rooster ← OrderedCollection new.
  self verrooster.
  rooster do:
    [ :roosterElement |
    order ← roosterElement order.
    bron ←roosterElement bron.
    bron isNil ifTrue: [self send: order aantalPlakken to: 'leverancier'].
    self workDuring: roosterElement transportStartTijd - self time.
    self send: roosterElement opdracht to: 'uitvoerder']
```

Specificatieblok 7

Literatuur

Aarts A.A.M., 1990,
Machinebesturing met de Smalltalk-80 proces-interactie omgeving,
Eindstudieverslag WPA 0915,
Technische Universiteit Eindhoven.

Brugman T.R.A.M., 1989,
Besturing van een Vertex in S84,
Eindstudieverslag WPA 0700,
Technische Universiteit Eindhoven.

Klaassen R.J.L.W., 1990,
Realisatie van besturingen met de proces-interactie omgeving gekoppeld aan een bitbusnetwerk,
Eindstudieverslag, Faculteit Wsk/I,
Technische Universiteit Eindhoven.

Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Shoys D.B., 1989,
Sequencing and Scheduling: Algorithms and complexity,
Centre for Mathematics and Computer Science, Amsterdam.

Lenstra J.K., Rinnooy Kan A.H.G., 1983,
Scheduling theory since 1981: an annotated bibliography,
Centre for Mathematics and Computer Science, Amsterdam.

Neerrijnen R.J.M. van, 1990,
De realisatie van een trajectory editor voor de Vertex-machine,
Eindstudieverslag, WPA 0934,
Technische Universiteit Eindhoven.

Rooda J.E., 1991a,
Procescalculus 2: Systemen, modellen en geschiedenis van de procescalculus,
i² Werktuigbouwkunde 7(8), 36 - 39.

Rooda J.E., 1991b,
Procescalculus 3: Definities en begrippen,
i² Werktuigbouwkunde 7(10), 35 - 40.

Rooda J.E., Arentsen J.H.A., Smit G.H., 1992,
Procescalculus 5: Modelleren van job-produktie fabrieken,
Mechanische Technologie 2(2), 36 -45.

Rooda J.E., 1992,
Procescalculus 7: Object-georiënteerde modelleertechnieken,
Mechanische Technologie aug. '92, p. 31-37.

Smit G.H., 1992,
A hierarchical control architecture for job-shop manufacturing systems,
Proefschrift,
Technische Universiteit Eindhoven.

Sze S.M., 1983,
VLSI technology,
McGraw-Hill Book Company, Singapore.

Vaes H.J., 1989,
Vertex en scheduling,
Eindstudieverslag, WPA 0786,
Technische Universiteit Eindhoven.

Wortmann A.M., 1991,
Modeling and simulation of industrial systems,
Proefschrift,
Technische Universiteit Eindhoven.