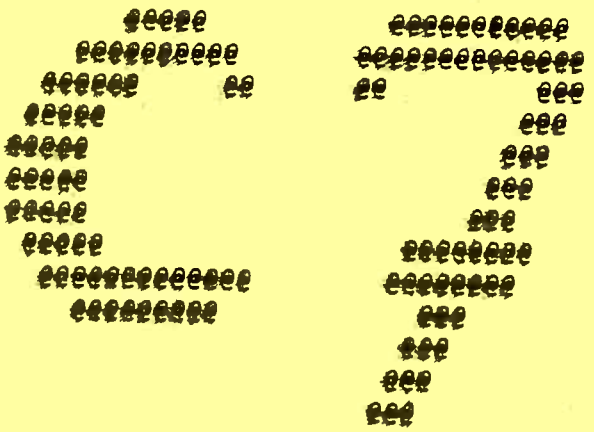
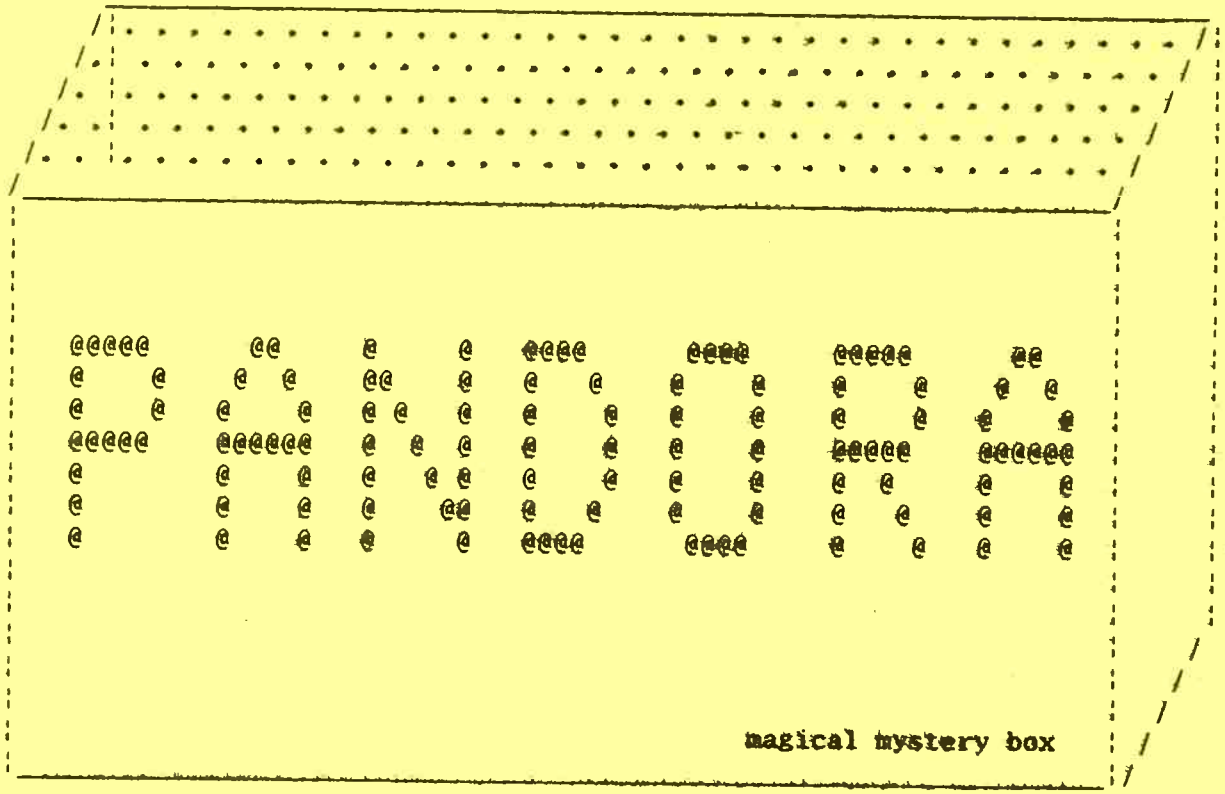
validatie van



laag 2 en 3 met



Nu bezat de mens de weldoende, zegenrijke kracht van het vuur.

Met spijt echter zag de wereldheerser Zeus, hoe het menselijk geslacht met zulk een gave was toegerust. Onmiddellijk zond hij de mensen een hevig kwaad om hun macht te beperken. Hij bracht tot hen een beeldschone jonge vrouw, die Hefaistos, de god van het vuur en de smidskunst, had geschapen en aan wie alle goden een onheilbrengende gave hadden meegegeven. Pandora, de draagster van alle gaven, heette zij. Zij verscheen te midden van de argeloze mensen en vond alom bewondering. Geen kwaad vermoedend aanvaarde Epimetheus, ondanks de waarschuwing van zijn broeder Prometheus, haar geschenk, een fraaie doos. Doch hoe zwaar zou de goedgelovigheid zich aan heel de mensheid wreken. Want nauwelijks was de deksel van de doos van Pandora geopend, of allerhande ziekten, rampen en smarten vlogen eruit en verspreiden zich bliksemsnel onder de mensen over heel het rond der aarde, terwijl zij tot nu toe vrij van moeiten en ziekten hadden geleefd. Aldus strafte Zeus Prometheus' roof. Een enkele goede gave bleef in de doos verborgen: de hoop. Maar voordat zij kon ontkomen sloeg de boze boodschapster der goden de deksel toe en hield ze voor altijd gesloten.

Naar de sage 'Prometheus'

Afstudeeropdracht bij het
Dr Neher Laboratorium van
de PTT, sectie ST-B1.

OPDRACHT:

Verificatie van het gemenewegsignalerings systeem No.7 (C7) m.b.v. PANDORA.

TOELICHTING:

In CCITT verband wordt het gemenewegsignaleringssysteem No. 7 gespecificeerd met als doel een internationaal gestandariseerd gemenewegsignaleringssysteem te krijgen.

Via het C7 signaleringssysteem kunnen SPC centrales communiceren t.b.v. het telefonieproces.

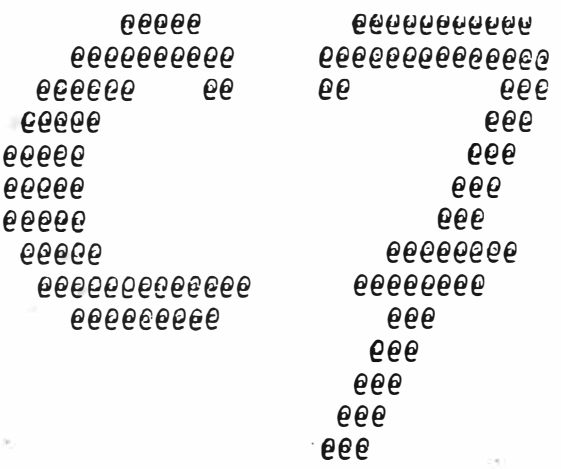
Dit communicatiesysteem is een betrouwbaar transportmiddel, voor de overdracht van informatie, d.m.v. protocollen op de lagen 2 en 3 (Retransmissie, changeover, rerouting, etc).

De specificatie van de lagen 1 t/m 3 is nagenoeg afgerond en nu toe aan een grondige verificatie .
Deze lagen moeten m.b.v. PANDORA aan een grondig onderzoek worden onderworpen.

In deze opdracht zijn de volgende aspecten te onderscheiden:

- Orientatie in C7 (laag 1 t/m 3)
- Reductie van de specificatie.
- Beschrijving van de C7 protocollen in PDL.
  (Pandora Description Language).
- Analyse van de resultaten.

Als uit de analyse wijzigingen in de specificatie volgen worden deze in het internationale overleg ingebracht.



Overige gegevens.

Aantal studenten	:	1.
Voorkennis	:	OSI model. Data communicatie.
Begeleiding	:	TH Delft_ PANDORA. DNL - C7.
Tijdsduur	:	9 a 12 maanden.
Aanvang	:	zo spoedig mogelijk.
Plaats	:	TH Delft i.v.m. plaats van het PANDORA systeem.
Bijzonderheden	:	Salariering volgens regeling van afstudeerders.
	:	

SAMENVATTING

In een datacommunicatie protocol worden afspraken vastgelegd die nodig zijn voor de besturing van datacommunicatie. Dergelijke afspraken dienen volledig, en vrij van fouten te zijn. Het is daarom gewenst deze afspraken op hun juistheid te toetsen. Met het PANDORA-systeem is het mogelijk deze validatie geautomatiseerd uit te voeren. Vanuit het ontwerp van een protocol wordt ingegaan op de definities van validatie en specificatie en de invloed van systeemstructuren daarop.

Het CCITT telefonie signalerings protocol nummer 7 (C7) kent een gelaagde opbouw welke in de lagen 1, 2 en 3 vergelijkbaar is met het OSI-model. Er is getracht de lagen 2 en 3 van dit protocol met PANDORA te valideren. De lagen 2 en 3 van C7 zijn gespecificeerd met tekst en geillustreerd met SDL diagrammen. Gebleken is dat de complexiteit van het C7-protocol de validatie erg moeilijk maakt. Daardoor konden slechts delen van laag 2 gevalideerd worden en is de validatie van laag 3 geheel achterwege gelaten.
Er wordt ingegaan op de manier van specificeren van het C7 protocol en de daarvan te maken afbeelding op een model welke geschikt is om met PANDORA te valideren.

Bij de validatie zijn zowel fouten in PANDORA als C7 (laag 2) aan het licht gekomen.

ABSTRACT

In a data communication protocol rules which are necessary for controling the information exchange are given. These rules have to be free of errors. Therefore it is desirable to check this rules, described in a protocol specification, on their correctness. This validation can be done by the PANDORA system, as PANDORA supplies a computer-aided validation for protocols.
The design of protocols is the basis from which the definitions of validation and specification , including system structures, are investigated.

CCITT signalling system number 7 (C7) has a layered structure which is comparable to the OSI model in layers 1,2 and 3. An attempt was made to validate layers 2 and 3 of C7 with PANDORA. These layers are specified with text and illustrated by SDL-diagrams. The complexity of layers 2 and 3 made the validation difficult. Ultimately, only parts of layer 2 appeared to be suitable for a validation with PANDORA, while layer 3 wasn't validated at all.
The way of specifying the C7 protocol, and the mapping of this specification on a suitable 'PANDORA-model' are both considered.

While carrying out this validation, errors in C7 (layer 2) as well in the PANDORA system were brought to light.

INHOUDSOPGAVE

LIJST VAN AFKORTINGEN

AERM	Alignment Error Rate Monitoring
CC	Congestion Control
CCITT	Comité Consultatif International de Telegraphique et Telephonique
CH	Channel
CHILL	CCITT High Level Language
DAEDR	Delimitation, Alignment, And Error Detection (Receiving)
DAEDT	Delimitation, Alignment, And Error Detection (Transmitting)
FIFO	First In First Out
FSM	Finite State Machine
FSMIB	Finite State Machine met FIFO IngangsBuffer.
IAC	Initial Alignment Control
ISO	International Standards Organisation
LSC	Link State Control
LSCP	Link State Control Part
LSSU	Link State Signalling Unit
MD	Message Dicrimination
MR	Message Routing
MSU	Message Signalling Unit
MTP	Message Tranfer Part
OSI	Open Systems Interconnection
PANDORA	Protocol ANalysis, Design and OpeRation Assessment
POC	Processor Outage Control
PSL	Pandora Specification Language

RC	Reception Control
RP	Recetion Part
SDL	Specification and Description Language
SMH	Signalling Message Handling
SNM	Signalling Network Management
ST..	State (toestand)
STIDLE	State idle
STIS	State In Service
STLPO	State Local Processor Outage
STPO	State Processor Outage
STRPO	State Remote Processor Outage
SU	Signalling Unit
SUERM	Signalling Unit Error Rate Monitoring
TP	Transmission Part
TXC	Transmission Control
UP	User Part

DEFINITIES

bestek
: Bouwbeschrijving; gedetaileerde beschrijving van de technische uitvoering van het ontwerp.

certificatie
: Test of de implementatie (uitvoering) aan de specificaties voldoet.

diensten specificatie
: Deel van de specificaties waarin de geboden diensten beschreven worden.

extentionele specificatie
: Specificatie waarin een black-box benadering wordt toegepast; het te specificeren element wordt vanaf de buitenkant beschreven.

functionele specificatie
: Deel van de specificatie waarin de functionele structuur van een element beschreven wordt. Functies met hun onderlinge relaties worden beschreven.

intentionele specificatie
: Specificatie waarin het te specificeren element in zijn inwendige opbouw beschreven wordt.

interface specificatie
: Gebruiksaanwijzing; deel van de specificatie waarin beschreven wordt hoe het beschreven element bedient moet worden.

ontwerp
: Omzetting van een aantal eisen in een functionele specificatie en/of bestek.

ontwerpcyclus
: Proces waarin een behoefte in een concreet, in de behoefte voorzienend, resultaat wordt omgezet.

protocol
: Verzameling regels

specificatie
: Beschijving van een element of systeem.

systeem
: Verzameling elementen waartussen relaties bestaan.

validatie
: Test of een specificatie aan algemene eisen voldoet.

verificatie
: Test of een specificatie aan de gestelde (algemene en specifieke) eisen voldoet.

1. INLEIDING

Bij alle vormen van communicatie is een aantal regels nodig voor het besturen van de communicatie. Deze regels hebben betrekking op de wijze waarop de communicatie tot stand wordt gebracht, wordt onderhouden en weer wordt beeindigd, en voorts de procedures die gevolgd moeten worden als een bericht verkeerd verstaan of begrepen wordt. Het geheel van deze, voor een bepaalde wijze van communiceren geldende, regels wordt een protocol genoemd [9].

Bij directe (mondelinge) communicatie tussen mensen onderling zijn deze regels min of meer informeel vastgelegd. Dankzij het improviserend vermogen van de mens kunnen eventuele problemen "ad-hoc", d.w.z. zonder dat daar vooraf vastgestelde regels voor nodig zijn, worden opgelost.

Bij communicatie tussen machines en tussen mensen en machines dienen deze regels op strikt formele wijze te worden vastgelegd. Het ontwerpen en testen van deze regels is een zeer moeilijke taak. Machines bezitten geen improviserend vermogen, er moeten dus regels zijn voor het oplossen van alle mogelijke problemen die zich bij het tot stand komen, onderhouden en beeindigen van de communicatie kunnen voordoen.

De eerste communicatie protocollen voor machines zijn reeds lang geleden ontstaan in het vakgebied van de automatische telefonie. Voor de besturing van een telefoonnet moest informatie uitgewisseld worden tussen de knooppunten van dat net, door middel van signalering. De technieken in de telefonie staan niet stil, zo ook niet de technieken op het gebied van de signalering. Om de zoveel tijd worden nieuwe signalerings protocollen ontwikkeld welke aan telkens nieuwe eisen voldoen, eisen die vaak ontstaan zijn door vooruitgang in de technologie.

Signalerings-protocollen worden in internationaal verband door o.a. de CCITT ontwikkeld en gestandaardiseerd. Momenteel is men bij de CCITT bezig met de ontwikkeling van signalerings-protocol nummer 7 (C 7).

De afgelopen tien jaar heeft zich onafhankelijk van de telefonie een wetenschap rond protocollen ontwikkeld. Deze wetenschap vindt zijn oorsprong in de computertechniek. Computers communiceren met de buitenwereld en met elkaar en gebruiken

daarbij protocollen. De eerste computer protocollen waren zeer eenvoudig, ze hadden betrekking op de communicatie tussen randapparatuur en computer. Met de begin van de jaren zeventig ontstane behoefte computers te koppelen, teneinde informatie te kunnen uitwisselen en gezamelijk gegevens te verwerken, ontstond er een vraag naar complexere protocollen, protocollen die aan zwaardere eisen voldeden. Het ontwerpen van dergelijke protocollen bleek niet zo eenvoudig te zijn zodat zich een geheel nieuw vakgebied ontwikkelde; Protocol-wetenschappen.

Een onderdeel van dit nieuwe vakgebied betrof het valideren van protocollen. Bij de validatie van protocollen wordt onderzocht of de protocollen consistent en betrouwbaar zijn. Dit probeert men door logisch te redeneren op grond van de beschrijving van het protocol, een formele analyse.

Het PANDORA systeem, ontwikkeld op de vakgroep Automatische Verkeers Systemen van de afdeling Elektrotechniek van de TH-Delft, biedt mogelijkheden om een dergelijke formele analyse geautomatiseerd uit te voeren.

Getracht is de lagen twee en drie van het C7 signalerings protocol met behulp van PANDORA te valideren. Laag twee en vooral laag drie van C7 betreffen echter zeer complexe, grote protocollen. Het valideren van grote protocollen is zeer moeilijk. De moeilijkheden bij het valideren van complexe protocollen komen in dit verslag naar voren. Om deze moeilijkheden te kunnen illustreren zal zoveel mogelijk van de systeemleer worden uitgegaan. In het volgende hoofdstuk zal daarom eerst op een aantal begrippen en principes uit de systeemleer worden ingegaan.

2. SYSTEEMLEER

Systeemdenken is de basis bij de ontwikkeling en analyse van complexe systemen. Een datacommunicatie netwerk is een zeer complex systeem, vele begrippen in de datacommunicatie zijn dan ook afkomstig uit de systeemleer[8]. In deze paragraaf worden enkele van deze belangrijke begrippen belicht vanuit de systeemleer.

In de systeemleer wordt een systeem gedefinieerd als een verzameling entiteiten (of elementen) waartussen onderling relaties bestaan.

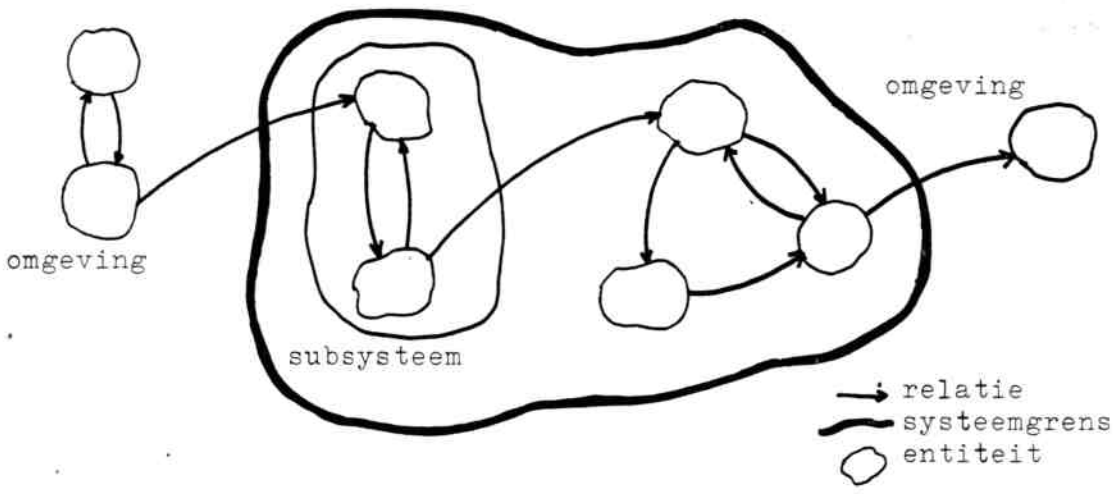


fig 2.1 Systeem en omgeving

De entiteiten zijn de elementen of delen van een systeem. Het zijn de kleinste delen van een systeem die een onderzoeker als een onderdeel van een systeem wenst te beschouwen. Aan entiteiten kunnen eigenschappen worden toegekend. Een entiteit kan op zichzelf ook een systeem zijn. Waar de grens getrokken wordt hangt af van het beschouwingsniveau.

De relaties beschrijven een bepaalde samenhang tussen de

entiteiten. In abstracte systemen zijn dit begripsmatige samenhangen (bijv. groter, kleiner). In concrete systemen is er sprake van een interactie of wisselwerking; elementen kunnen elkaar beinvloeden, d.w.z. veranderingen in de waarde van de eigenschappen van een entiteit kunnen veranderingen in de waarde van de eigenschappen van andere entiteiten tot gevolg hebben.

De samenhang kan voor een balans bijvoorbeeld als volgt omschreven worden. Beschouw zowel van A als B de massa's en de posities als eigenschappen. Wanneer we de eigenschap van A, die massa is genoemd, verminderen dan verandert, verondersteld dat er zwaartekracht is, de eigenschap van B die we positie hebben genoemd.

fig 2.2 Balans systeem

In communicatie systemen komen relaties tot stand door een uitwisseling van berichten. Het communicatiebegrip houdt in dat de beinvloeding wederzijds is. In andere systemen kunnen relaties ook eenvoudig zijn. (bijv. een mechanische overbrenging)

De verzameling van relaties die binnen een systeem aanwezig zijn wordt de structuur van het systeem genoemd. Vele bestaande systemen (waaronder datanetwerken) zijn zo complex dat een opsomming van alle relaties geen duidelijk beeld verschaft van de structuur van het systeem. Om in dat geval toch een hanteerbaar geheel te verkrijgen wordt het noodzakelijk entiteiten met hun relaties te groeperen tot subsystemen.

Subsystemen kunnen zelf weer met hun onderlinge relaties beschouwd worden als een systeem, maar dan vanuit een ander abstractieniveau. Het herhaald groeperen van subsystemen tot nieuwe grotere subsystemen leidt tot een hierarchische systeemstructuur.

Wat door de onderzoeker precies als systeem wordt beschouwd is afhankelijk van zijn probleemstelling. Hij zal een grens moeten trekken om een deel van de entiteiten die hij binnen de totale werkelijkheid wil onderzoeken. Deze grens wordt de systeemgrens genoemd. Na het trekken van een systeemgrens kan besloten worden wel of niet de relaties met entiteiten buiten het systeem bij de beschouwing te betrekken. Men kan spreken van resp. open en gesloten systemen. De voor ons bekende systemen zijn per definitie open, d.w.z. er bestaan relaties met entiteiten buiten de systeemgrens. In sommige gevallen kan het nuttig zijn een

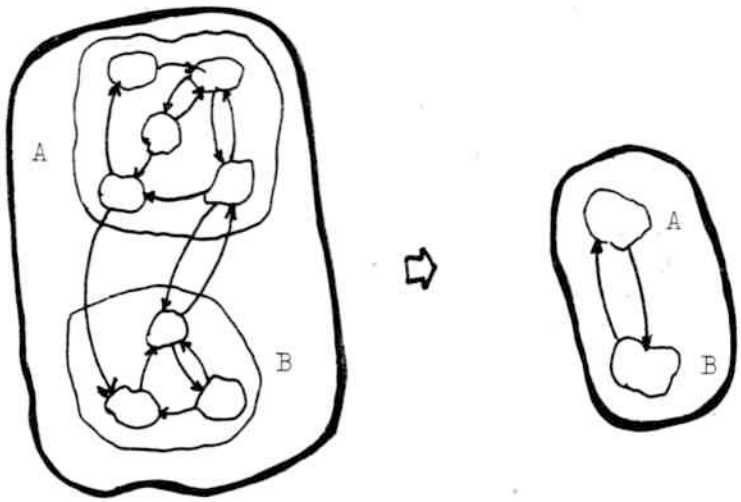


fig 2.3 Systeem met subsystemen

systeem als gesloten te beschouwen, bijvoorbeeld als men in een eerste beschouwing externe invloeden wenst uit te sluiten.

In het systeemdenken is het belangrijk een onderscheid te maken tussen de begrippen functie en taak. Onder een functie van een systeem (subsysteem, entiteit) wordt verstaan de gewenste bijdrage die het levert aan een groter geheel, waarvan het deel uit maakt. Een taak houdt datgene in wat moet gebeuren om die bijdrage tot stand te laten komen, zodat de functie wordt vervuld. Bij het functiebegrip gaat het om resultaten naar buiten, bij een taak gaat het om de activiteiten die zich binnen het (sub)systeem of de entiteit afspelen.

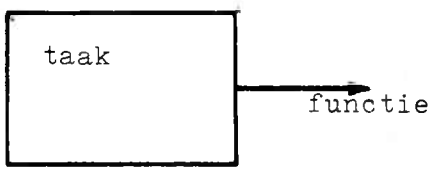


fig 2.4 Functie vervullen; taak verrichten

Het denken in functies is het uitgangspunt in moderne ontwerp- en analyse methoden. Men stelt zich eerst de vraag welke functies een systeem moet vervullen en hoe de relaties tussen deze functies moeten liggen, om het doel van het systeem te kunnen verwezenlijken. Men bekommert zich daarbij dus in eerste aanleg niet om de vraag welke taken verricht moeten worden om de

functies te vervullen, d.w.z. de functionele beschouwing wordt los gezien van de uitvoering. De achtergrond van deze beschouwingswijze is dat een functie veel minder aan tijd gebonden is dan een taak. De wijze waarop een taak wordt verricht wordt bepaald door de technologische ontwikkelingen en economische factoren. Functies komen voort uit behoeften en zijn daardoor minder aan een tijdperk gebonden.

Onder het gedrag van een systeem wordt verstaan de wijze waarop het systeem reageert (via de uitgang) op bepaalde in- en uitwendige omstandigheden en op veranderingen daarin (de ingang). Het gedrag van een systeem kan in veel gevallen gezien worden als het verloop van de toestand van het systeem in de tijd. Bij de gedragsbeschrijving van een systeem is de interne structuur van het systeem (nog) niet van belang, of zelfs geheel onbekend (black box benadering). Systemen met een verschillende interne structuur kunnen een identiek gedrag vertonen (bijv. een computer)

Het architectuurbegrip sluit direct aan bij het functiebegrip en het systeemgedrag. We verstaan onder de architectuur van een systeem een nauwkeurig beschrijving van de functies die een systeem vervult, zoals deze in de omgeving van het systeem 'zichtbaar' zijn.

Een belangrijk hulpmiddel bij het bestuderen van systemen is het model. Een model is een materiele voorstelling of formele beschrijving van een te onderzoeken systeem, dat eenvoudiger kan zijn dan het systeem zelf, vaak doordat een aantal aspecten buiten beschouwing worden gelaten. Uiteraard moet een model voor het gestelde probleem in voldoende mate overeenstemmen met het werkelijke systeem.

DEEL I

VALIDATIE, COMPLEXITEIT EN PANDORA

Het ontwerp van een communicatie protocol is een passend uitgangspunt om validatie toe te lichten. Validatie heeft een plaats in de ontwerpcyclus. Alle begrippen en technieken die bij validatie gebruikt worden, kunnen vanuit het ontwerp helder in hun oorspronkelijke betekenis worden benaderd. Dit biedt ook mogelijkheden om de problemen die bij validatie als gevolg van complexiteit optreden fundamenteel te benaderen.

Dit deel is uitgebreid opgezet, het is tevens het resultaat van een literatuuronderzoek.

3. ONTWERPEN

3.1 Inleiding

Het ontwerp van een protocol verloopt in principe als elk ander ontwerp. Aan elke totstandkoming van een nieuw object (abstract of concreet) gaat immers een ontwerpcyclus vooraf. Soms spreekt men in plaats van ontwerp ook wel van ontwikkeling. Ik probeer de ontwerpcyclus vanuit de systeemleer te beschouwen.

3.2 De ontwerpcyclus vanuit de systeemleer

De mens leeft in interactie met zijn omgeving (fig 3.1). Die omgeving omvat vele deelgebieden; zijn medemens, zijn leefmilieu, de maatschappij, de techniek. De door de mens bevonden gebreken (of gesignaleerde toekomstige gebreken) in de interactie tussen mens en omgeving kunnen globaal beschouwd worden als zijn onvervulde behoeften(fig 3.2).

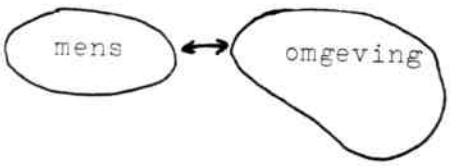


fig 3.1
Interactie mens-omgeving

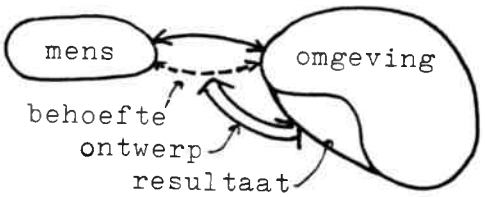


fig 3.2
behoefte-ontwerp-resultaat

In die behoefte kan voldaan worden door een object aan de omgeving toe te voegen welke de gemiste interactie mogelijk maakt. Als dit object nog niet bestaat moet het ontworpen worden. Het resultaat van een ontwerp wordt ingepast in de omgeving. Doel van het ontwerp is in de behoefte te voldoen.

Een ontwerpcyclus heeft met twee ingangen te maken; de behoefte en de omgeving. Deze combinatie wordt vaak "het probleem" genoemd (fig 3.3). Het probleem heeft niet op de hele omgeving betrekking maar slechts op een deel ervan. De eerste taak van de probleemanalyse is te bepalen welk deel van de omgeving dit is, hierbij lettend op de uitvoerbaarheid van het ontwerp.

v.b. Voor boten kan de relevante omgeving de zee zijn, of een rivier, of een haven, of alle wateroppervlakten, etc..

Het tweede deel van de probleemanalyse bestaat uit het opstellen van een aantal eisen waaraan het ontwerp moet voldoen opdat aan de behoefte voor het bepaalde deel van de omgeving voldaan is. In de eisen staat als het goed is nauwkeurig omschreven met welke invloeden van de omgeving de ontwerper rekening dient te houden. Deel van de eisen vormt ook de minimale gewenste betrouwbaarheid voor deze bepaalde omgeving. De eisen kunnen verder onderstreept worden door toevoeging van enkele tests waar het resultaat aan moet voldoen.

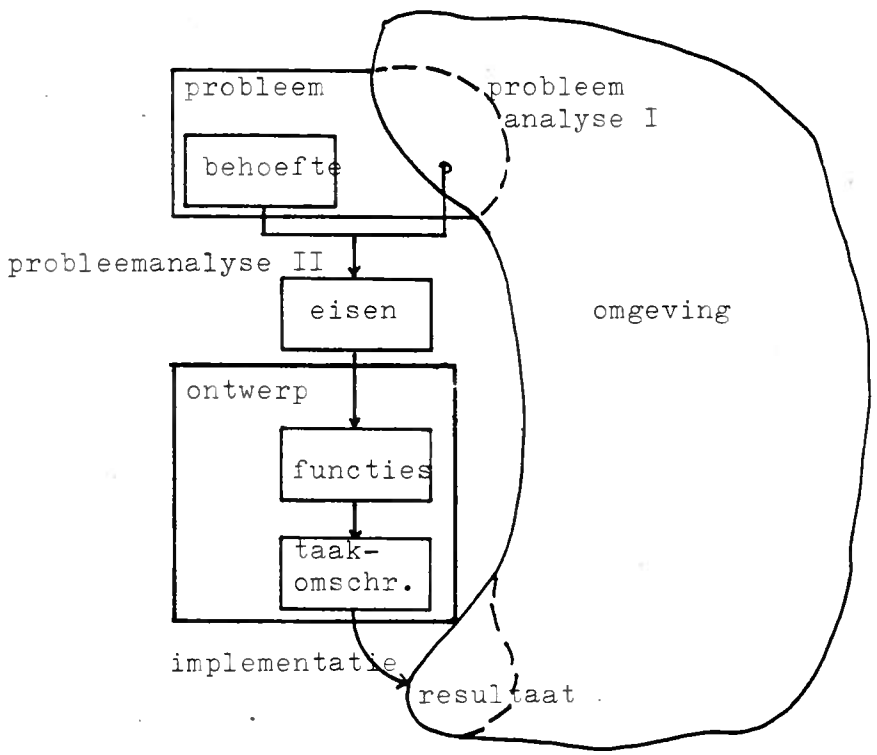


fig 3.3 probleemanalyse-ontwerp-implementatie

Het resultaat (hier een systeem) dient de vereiste eigenschappen te bezitten (eis = wat men verlangt alvorens tevreden te zijn). Een deel van de eisen beschrijft de functies die het object naar buiten toe moet vervullen, de geboden diensten of services. Voor de beschrijving (specificatie) van het systeem zijn deze functies belangrijk, de mogelijkheden van het systeem worden ermee beschreven. Deze geboden diensten maken deel

uit van de specificaties en kunnen als deel van de specificaties ook wel 'diensten specificatie' worden genoemd. De diensten specificatie heeft een hoog abstractieniveau.

v.b. Voordat met het ontwerp van een radio wordt gestart, worden er eisen opgesteld waaraan de toekomstige radio zal moeten voldoen. Een eis kan bijvoorbeeld "een uitgangs-vermogen van 10 watt" zijn. Als de radio uiteindelijk in de winkel staat wordt er in de specificatie gezegd; "uitgangsvermogen : 10 watt" (ontwerp geslaagd).

Tijdens het eigenlijke ontwerpen wordt, uitgaande van de opgestelde eisen, een opsplitsing van het gehele systeem gemaakt in subsystemen, componenten en modulen (fig 3.4). Het moet uitmonden in een overzichtelijke, uitvoerbare structuur. Alle relaties tussen de systeemdelen moeten aangegeven worden, evenals de door hen vervulde functies in het geheel. Bij het opsplitsen in delen streeft men naar zo weinig mogelijk relaties tussen die delen. Dan kan men zich tijdens het verdere ontwerp goed op een onderdeel concentreren; om de toenemende grootte aan te kunnen is dit essentieel.

Tijdens het eigenlijke ontwerp wordt voor de communicatie over, de beschrijving en de bestudering van het ontwerp gebruik gemaakt van modellen (NB; de hele volgorde van probleem, eisen, specificatie, implementatie noem ik ontwerpcyclus, het eigenlijke ontwerp is het omzetten van eisen in een specificatie). Op hoog abstractie niveau zijn dat systeem-modellen. Ze gebruiken dezelfde begrippen als in hoofdstuk 2 geintroduceerd zijn. Naarmate het ontwerp vordert worden de modellen uitgebreider en gedetailleerder. Om de groeiende hoeveelheid details te kunnen beschrijven worden deelmodellen, beschrijvingen van systeemdelen, gebruikt. Op het allerlaagste niveau wordt aangegeven door wat voor taakverrichtingen de functies van de kleinste systeemelementen vervuld kunnen worden.

v.b. De aandrijving van een auto kan op het hoogste niveau aangegeven worden bestaande uit motor-frictie-versnellingsbak-differentieel-wielen, en op het allerlaagste niveau met gedetailleerde constructietekeningen.

Op bovenbeschreven wijze ontstaat een functionele structuur of systeemstructuur. De functies van alle systeemdelen worden zo nauwkeurig mogelijk beschreven evenals hun relaties. De entiteiten, of elementen van de systeemstructuur worden vervolgens ingevuld met taken, waardoor een bestek (bouwbeschrijving) ontstaat. Het bestek is een-eenduidig te implementeren, het bestek dient als leidraad voor de uitvoerders of implementeurs. Het verschil tussen de systeemstructuur en het bestek is dat de systeemstructuur tijdlozer is.

vb. De systeemstructuur van een oude buizenradio ziet er hetzelfde uit als de systeemstructuur van een transistorradio (fig 3.5) De bestekken zijn echter totaal verschillend, de technische uitvoering van een radio is immers met z'n tijd meegegaan.

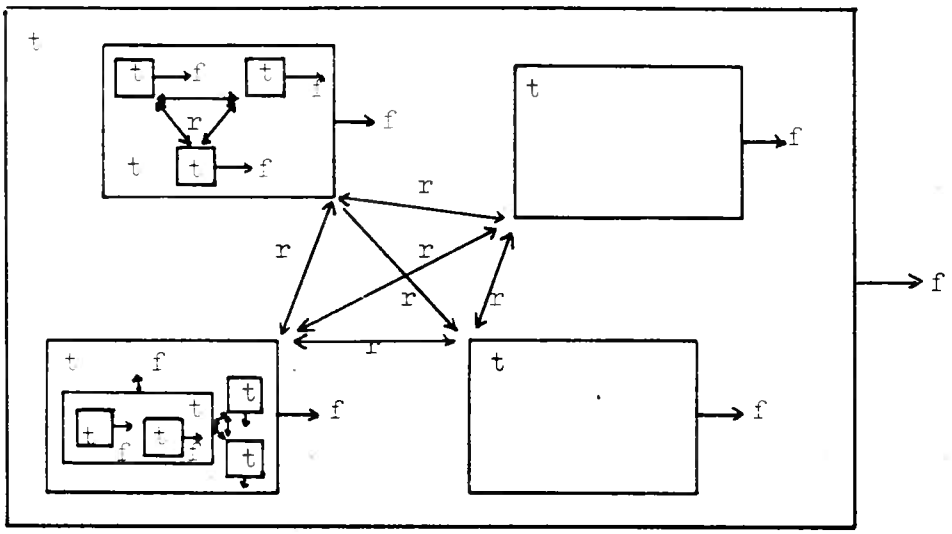


fig 3.4 Systeem opsplitsing

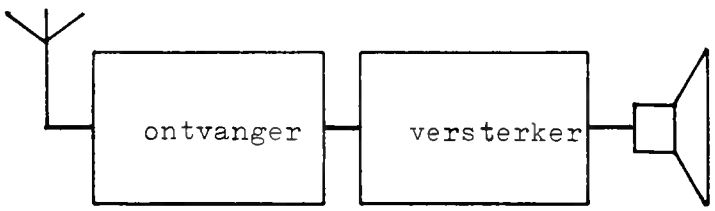


fig 3.5 Systeemstructuur van een radio

De exacte beschrijving van de uiteindelijke systeemstructuur, of 'functionele specificatie' vormt een volgend deel van de specificatie. Het laatste deel van de specificatie wordt gevormd door de gebruiksaanwijzing of 'interface specificatie', het bevat informatie over de bediening (zit de volumeknop linksonder of rechtsboven). De interface specificatie ligt relatief dicht op het implementatie niveau, het is echter wel zo dat de eisen ook delen kan bevatten die betrekking hebben op de bediening.

Men kan functies beschouwen tot op het niveau waarop men dat nodig vindt (H 2). Dit maakt het abstractieniveau (dus de uitvoering) van de functionele specificatie variabel. Men zou van een radio het schema (bestek) ook als funcionele specificatie kunnen beschouwen daar het schema relaties tussen functies beschrijft en de elementen op eigenschappen (functies) aangeeft (element met versterking B, element met weerstand r, etc). Het verschil tussen de functionele specificatie en het bestek kan daardoor erg vaag zijn. Men zou er naar kunnen streven de

functionele specificatie op een niveau te beschrijven waarop er relevante inwendige verschillen kunnen optreden tussen de ervan afgeleide implementaties (afhankelijk van gebruikte techniek).

Verder is het zo dat de gebruiker nauwelijks in de functionele specificatie of het bestek geinterresseerd is, de diensten specificatie biedt hem genoeg informatie ten aanzien van zijn behoeften. Dit maakt de functionele specificatie voor algemeen gebruik relatief onbelangrijk; de gebruiker benadert zijn radio bijvoorbeeld toch als black-box.

3.3 Testen

Uiteindelijk doel van een ontwerp is dat de implementatie in de behoefte zal voldoen. Men wil daarom graag weten of, en in hoeverre de implementatie aan die wens zal voldoen. Dit doet men door testen.

Tijdens testen probeert men fouten op te sporen. Het is niet goed mogelijk een precieze eenduidige definitie van het begrip 'fout' te geven. Het begrip fout is namelijk relatief. Als bijv. een implementatie fout is, is het steeds fout ten opzichte van iets (de eisen, de specificatie). Tijdens het testen moet men zich dan ook voordurend bewust zijn ten opzichte van wat (de testparameters) men probeert te testen.

Net als een fout is een test een relatief begrip. In onze ontwerpcyclus zijn 6 relatieve testen mogelijk.

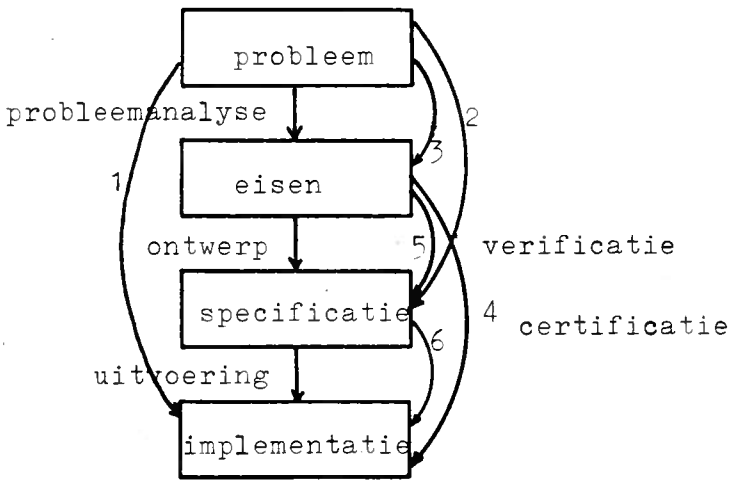


fig 3.6 Relatieve testen

Meestal worden onder testen de relaties tussen de implementatie en de overige delen bedoeld (Appendix A). Men test of de implementatie het probleem oplost, in de behoefte voldoet, het

uiteindelijke doel van het ontwerp. Het is echter niet verstandig bij een ontwerp dit als enige test te nemen. Als uit de test namelijk blijkt dat de implementatie niet goed is, dan moet men terug naar de ontwerp- of probleemanalysefase, om vandaar af de ontwerpcyclus opnieuw te doorlopen. Dit zal onnodig veel tijd kosten en zal daardoor veel te duur zijn. Men wil reeds eerder zekerheid hebben. De andere testen uit de figuur zijn daarom ook erg belangrijk.

Voor een aantal van de relatieve testen bestaan aparte namen. Het testen of de specificatie aan de eisen voldoet (5) kan 'verificatie' worden genoemd. Het testen of de implementatie aan de specificatie voldoet (6) noemen we 'certificatie'. Verder kennen we nog de term validatie (validatie = op waarde taxeren) welke op alle testen van toepassing kan zijn. Validatie is niet relatief in de ontwerpcyclus zelf, maar relatief ten opzichte van buitenliggende (bekende, grijpbare) testparameters. De testparameters zijn bij validatie van algemene aard.

Validatie maakt veelal deel uit van de verificatie. Validatie heeft betrekking op algemene karakteristieken, karakteristieken waarover dus veel bekend is. Validatie is veelal wetenschappelijk onderbouwt en heeft het karakter van een correctheidsbewijs. Voor de omgeving gelden bijv. fysische en wiskundige wetten. Validatie kan kijken of de specificatie aan deze wetten voldoet. Eis daarbij is dat de specificatie een formeel karakter heeft.

v.b. Aan een huis wordt als (algemene) eis gesteld dat het niet instort. Daarom wordt de specificatie (bouwtekeningen) voor de eigenlijke implementatie (bouw) plaatsvindt geheel doorgerekend (door constructietechnici). Dit is een vorm van validatie in de verificatie. Deel van de verificatie kan verder (niet algemeen) het voldoen aan eisen in bijv. vorm/stijl zijn.

4. PROTOCOL ONTWERP

4.1 Inleiding

In het eerste hoofdstuk is een protocol omschreven als een aantal voor een bepaalde wijze van communiceren geldende regels. Een protocol is nodig voor een geordend verlopende communicatie. Deze communicatie verloopt als we over datacommunicatie spreken tussen computers.

De eisen die er aan protocollen gesteld worden komen voort uit de eisen aan datacommunicatie. Ze worden gesteld door (behoefte van) de gebruikers. De eisen die aan datacommunicatie systemen gesteld worden betreffen prestatie (verwerkingscapaciteit en snelheid), omvang, betrouwbaarheid en mogelijkheden. Het protocol dat bij de datacommunicatie gebruikt wordt heeft invloed op deze factoren. Deze invloed is voor eenvoudige communicatiesystemen (weinig besturing) gering maar neemt toe met de complexiteit van het communicatiesysteem. Factoren die er ook invloed op hebben zijn netwerktopologie en netwerkcapaciteit (fig 4.1).

De eisen die er aan een protocol gesteld worden komen dus voort uit de eisen die aan het communicatiesysteem gesteld worden en de andere communicatiesysteem karakteristieken als topologie en capaciteit (samenhang tussen subsystemen). De omgeving die van belang is voor het ontwerpen van een protocol kan dus omschreven worden als het communicatienet met gebruikers. Men probeert een protocol onafhankelijk van de topologie en de capaciteit te ontwerpen. Hierdoor krijgt de specificatie van het protocol algemenere geldigheid. Later bij de implementatie van een protocol worden deze factoren pas meegenomen. Bij het protocolontwerp let men slechts op algemene technische karakteritieken als packet-switched, circuit-switched, LAN, point to point etc. De omgeving die er voor het protocol daardoor overblijft is 'een communicatienet met gebruikers'.

Protocollen hebben als functie het besturen van de communicatie. In het volgende wordt communicatie verder bestudeerd, weer zoveel mogelijk vanuit de systeemleer[8][11]. Hierdoor is ook precies aan te geven wat de betekenis van een protocol vanuit de systeemleer is.

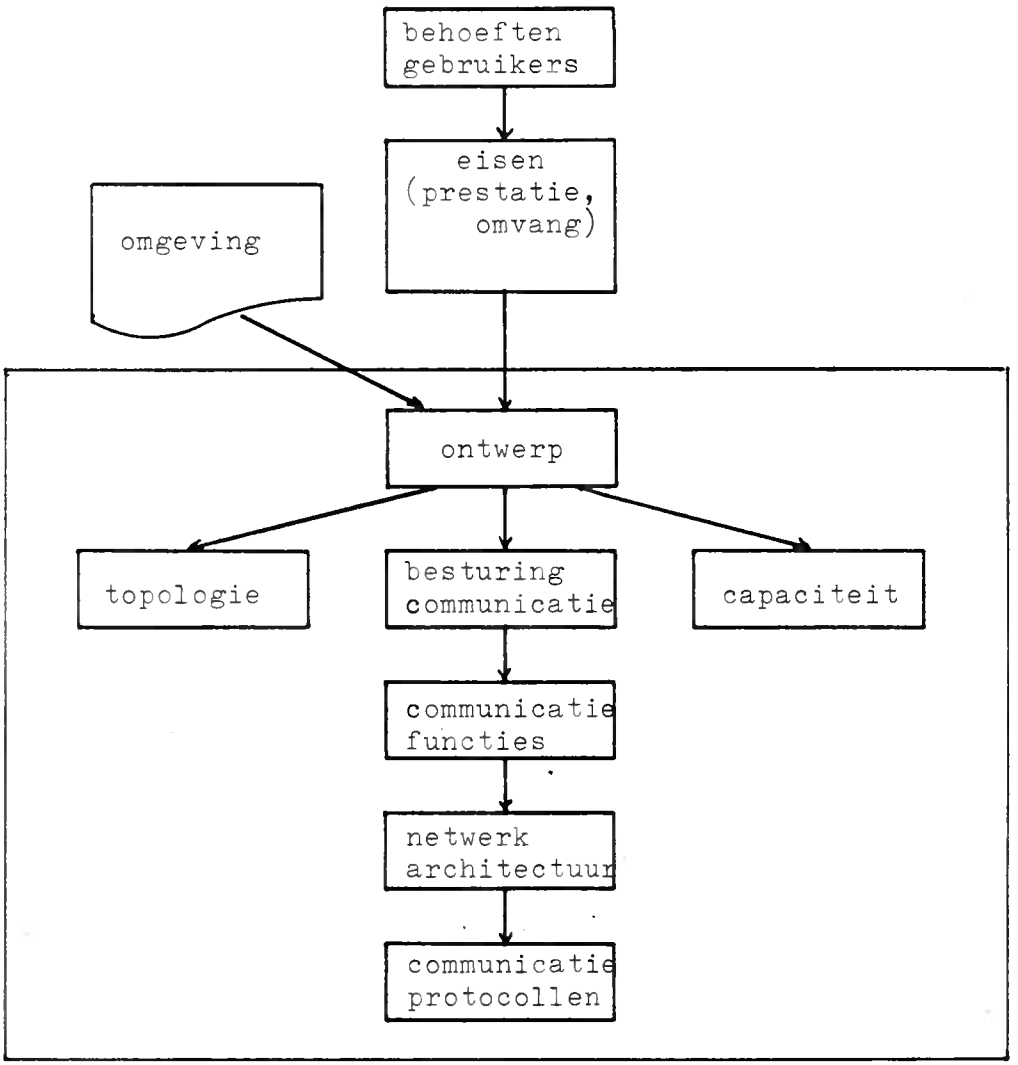


fig 4.1 Ontwerp datacommunicatie-netwerk

4.2 Het communicatie model

Een communicatienet is te omschrijven als een gedistribueerd systeem met communicatie tussen communicatiestations. Onder een communicatiestation verstaan we dat deel van een systeem waarin de communicatiefuncties zijn ondergebracht. Een communicatie station kan een geheel netwerkelement zijn (terminal, telefooncentrale) of een deel daarvan. Bij elke vorm van communicatie zijn altijd twee of meer communicatiestations aanwezig. Communicatie tussen stations komt tot stand via een communicatiemedium (fig 4.2). In principe kan een communicatiemedium varieren van een geheel netwerk tot een rechtstreekse verbinding tussen twee stations.

Men kan dit model uitbreiden door onderscheid te maken tussen de eigenlijke uitwisseling van berichten (data) tussen de stations en de uitwisseling van de informatie die nodig is voor

de besturing van het geheel. Evenzo wordt het communicatie station verdeeld in een deel dat bron en bestemming is van data, en een besturingsdeel. In de afzonderlijke besturingsdelen zijn regels geimplementeerd die de gewenste dataoverdracht tussen de bron/bestemmingsdelen mogelijk maken. De verzameling regels noemen we het protocol. De besturingsdelen noemen we protocolstations (fig 4.3).

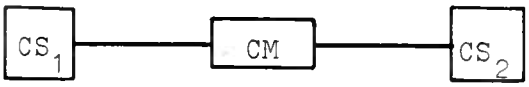


fig 4.2 Communicatie-model

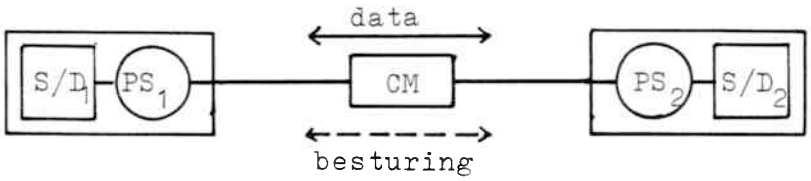


fig 4.3 Communicatie-model met protocolstations

CM= communicatiemedium
CS= communicatiestation
PS= protocolstation
S/D= bron/bestemming
CR= communicatiemiddel

Als we daarnaast de protocolstations samen met het communicatiemedium als communicatiemiddel beschouwen dan kan figuur 4.2 omgezet worden in fig 4.3.

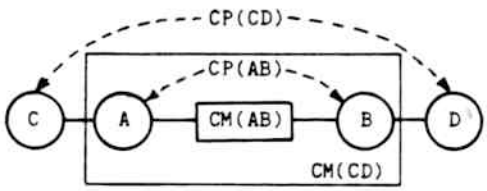
Met behulp van het communicatiemiddel kan een gebruiker, iemand die behoefte heeft aan communicatie, communiceren. Het voldoet aan een communicatie-eis en vervult voor de gebruiker enige functies, het biedt hem diensten of services. Bij het communicatiemiddel hoort ook een gebruiksaanwijzing. Hiermee hebben we twee delen van de specificatie (H 3.2) gevonden; de diensten en de interface specificatie. De functionele specificatie beschrijft hoe het communicatiemiddel als systeem werkt.

4.3 Netwerk architectuur

In een netwerk zijn erg veel functies door het protocol te vervullen, er worden vele eisen gedefinieerd. Om aan deze vele eisen te kunnen voldoen wordt het protocol tijdens het ontwerp,

als in andere ontwerpcycli, in delen opgeplitst.
In principe is de keuze van opsplitsing vrij. Voor algemeen gebruik is dit echter niet zo praktisch. Een vrije opsplitsing betekent namelijk dat ieder ontwerp in zijn geheel overgenomen dient te worden. Een modulaire structuur, waarbij delen van het ontwerp ook voor andere ontwerpen gebruikt kunnen worden, is wat dit betreft aantrekkelijker. Een universele (modulaire) functionele structuur zal ook makkelijker hanteerbaar en toekomstvaster zijn. Ook in andere ontwerpen worden functionele structuren gestandaardiseerd. Bij stereoinstallaties zijn de aansluitingen van de componenten bijvoorbeeld ook gestandaardiseerd.
Een universele functionele structuur voor communicatieprotocollen is onder andere vastgelegd in het 'Reference Model of Open System Interconnection', in het kort OSI-model. Het OSI-model biedt een raamwerk voor het ontwerp van protocollen, en is ontwikkeld door de ISO (International Standards Organisation). Er zijn meer van dergelijke raamwerken bedacht (IBM heeft het SNA-model), het OSI-model wordt echter gezien als de toekomstige standaard. Dergelijke structuren vallen onder de noemer netwerk architectuur.

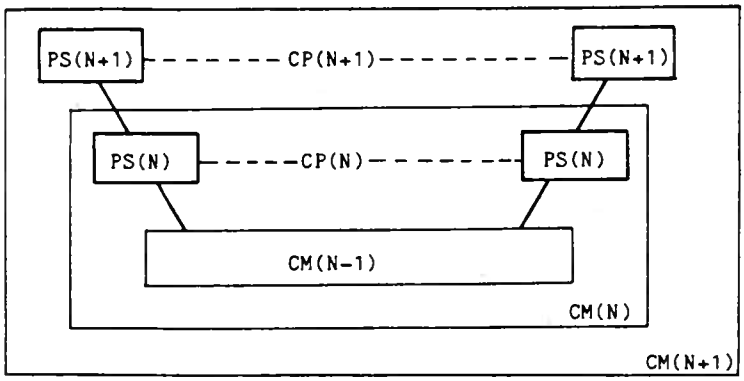
Het OSI-model is gebaseerd op het communicatie model als boven beschreven, deze vormt het basiselement. Er wordt gestructureerd op basis van structureringstechnieken die bekend zijn uit de systeemleer. De belangrijkste structureringstechniek die van deze technieken in het OSI-model gebruikt wordt is het wrappen (omhullen).
Het principe van wrapping wordt toegelicht met behulp van fig 4.4.



(CM=communicatiemedium, CP=communicatieprotocol)

fig 4.4 Wrapping

Een basiselement bestaande uit twee protocolstations(entiteiten) A en B, die via een communicatiemedium CM(AB) met elkaar samenwerken volgens een protocol CP(AB), vormt een communicatiemedium CM(CD) voor twee protocolstations C en D. Het stelsel C-CM(CD)-D maakt gebruik van diensten die geboden worden door het basis-element A-CM(AB)-B en voegt toe de diensten die het resultaat zijn van de interactie tussen C en D.
Dit principe kan herhaald toegepast worden waardoor er verschillende schillen onstaan (fig 4.5).
Groot voordeel van deze methode is dat de omgeving voor elk



(CM=communicatiemedium, PS=protocolstation
CP=communicatieprotocol)

fig 4.5 Schillenstructuur

protocol(N) hierdoor beperkt is tot de onderliggende schil(N-1) en de bovenliggende schil(N+1).

Enititeiten gelegen in dezelfde schil worden peer enitities genoemd. De interactie tussen peer entities wordt bestuurd door protocollen. Deze interactie komt tot stand door gebruik te maken van de services geboden door de onderliggende schil. De interactie tussen entiteiten in aangrenzende schillen wordt beschreven met de term interface. Tussen een interface en een protocol hoeft geen wezenlijk verschil te bestaan, beide beschrijven precies de wijze waarop twee (of meer) entiteiten met elkaar samenwerken.

In het OSI model zijn de diensten die de schillen moeten bieden ook vastgelegd. De schillen kunnen beschouwd worden als subsystemen en dienen in die zin zo min mogelijk relaties te hebben. Aan de communicatie tussen de schillen over de grenzen (interfaces) zijn in het OSI-model daarom ook eisen gesteld. Dit biedt de mogelijkheid om voor een schil uit meer protocollen, welke voor die schil beschikbaar zijn, te kiezen, de protocollen worden uitwisselbaar. Dit biedt op den duur zeker voordeel, omdat er steeds betere protocollen ontwikkeld zullen worden, welke dan ook daadwerkelijk ingezet kunnen worden.

Het OSI model kent zeven schillen of lagen [8,9,22] (fig 4.6)
Globaal gezien kan gezegd worden dat laag 1 tm 3 betrekking hebben op het communicatienet zelf, terwijl de lagen 4 tm 7 meer betrekking hebben op het gebruik van het net. In Appendix B wordt de communicatie tussen de lagen in het model beschouwd.

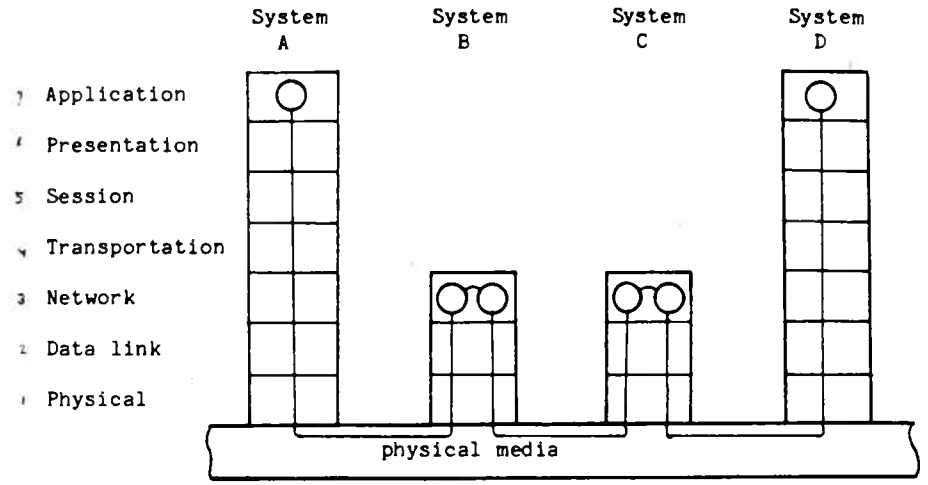


fig 4.6 OSI-model

4.4 Functies van een laag

Getracht is in het OSI-model op elke laag een of meer functies te leggen. Deze functies moeten geheel door de laag vervuld worden. De omgeving voor die functies is, als beschreven, voor elke laag steeds anders. Voor iedere laag biedt de onderliggende laag (omgeving) immers andere services.

De diensten die de onderliggende laag (omgeving) aan de lagen 2 tm 4 biedt worden als voorbeeld achtereenvolgens beschouwd.

Laag 2 heeft als communicatiemedium het fysische transportmedium, de fysieke laag 1. Laag 2 heeft daardoor te maken met

1 kanaal

dat

a.stoort

b.vertraagd

Laag 2 moet aan laag 3 een transparant kanaal met een geeiste betrouwbaarheid bieden, de functies die het daartoe moet vervullen betreffen vooral foutendetectie/correctie, hetgeen storingen opvangt. Aan vertraging is helaas niets te doen.

Laag 3 heeft als communicatiemedium laag 2. Laag 2 biedt

meer kanalen

die

vertragen.

Laag 3 moet aan laag 4 een net i.p.v meer kanalen bieden, de functie die het daartoe kan vervullen is bijv. routering. Daarnaast kan laag 3 bijv. de verkeersstromen in diverse richtingen verdelen.

De specificatie van een OSI-laag komt in het volgende hoofdstuk aan de orde.

5. SPECIFICATIE

5.1 Inleiding

In de inleiding is een protocol omschreven als een aantal voor een bepaalde wijze van communiceren geldende regels. Communicatie heeft onder andere parameters in de tijd, communicatie vindt altijd plaats tussen processen. Kenmerk van een proces is dat het parameters in de tijd heeft, er is altijd sprake van volgorde. Hier hebben we het over datacommunicatie. De communicerende processen bestaan in dat geval uit processen (software en/of hardware) welke zich in computers afspelen. Dit zijn informatie verwerkende processen.

De verschillende soorten specificaties die er rond een OSI protocol te vinden zijn kunnen we ook weer vanuit het ontwerpproces aangeven. Het ontwerpproces uit hoofdstuk 3 kunnen we uitwerken naar een ontwerp dat in een 'OSI-omgeving' plaatsvindt. Naar de benaming in de software engineering noem ik het ontwerpproces de 'OSI-protocol-life-cycle'.

In de OSI-protocol life cycle, waarbij algemene termen uit het ontwerpproces vervangen zijn door specifieke protocol termen, zijn een aantal delen van de specificatie te herkennen.

In OSI is een protocol gedefinieerd in een omgeving met behulp van een aantal service-primitieven (Appendix B), en een tweetal grenzen. De primitieven en de grenzen definieren een abstracte interface waarbinnen het protocol gaat gelden. Daarnaast worden door OSI ook enkele diensten van een laag verlangt, Deze diensten worden beschreven in de diensten-specificatie van de betreffende laag. Daarnaast kan het protocol ontworpen worden voor een specifiek soort omgeving, bijv. lokale communicatie, of satellietcommunicatie. Deze omgeving stelt een aantal aanvullende eisen aan het protocol.

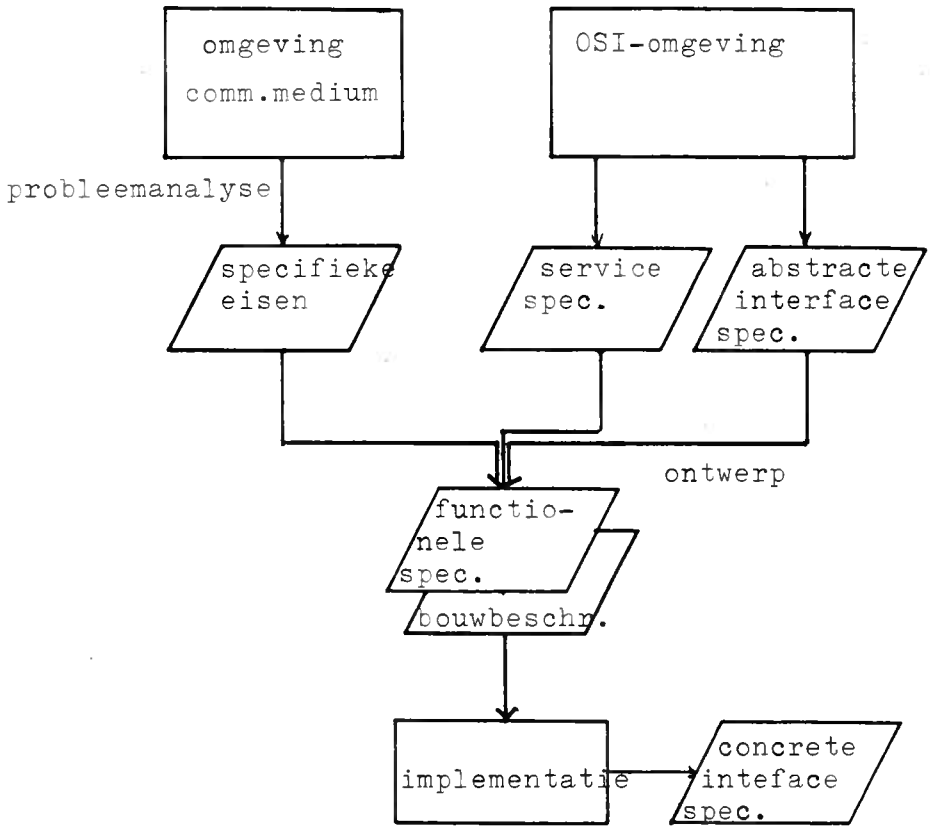


fig 5.1 OSI-Protocol-life-cycle

Bovengenoemde elementen worden tijdens het ontwerp van een protocol verwerkt tot een functionele specificatie (fig 5.1). Het protocol wordt functioneel gespecificeerd door een protocol-entiteit, een van de communicerende processen, te beschrijven *).

Hiermee is de schil uit hoofdstuk 4.3 in tweëen gedeeld (point to point) en zijn werkelijk twee lagen ontstaan (fig 5.2). Een schil betreft de gehele communicatie tussen twee knooppunten, terwijl een laag slechts een knooppunt betreft.

Door slechts één entiteit te beschrijven worden de aan een functionele specificatie gestelde eisen echter wel erg zwaar. Men dient het gedrag dat het proces ten opzichte van de doorsnijding

---

*) Een protocol is een verzameling regels. Een regel is een na te volgen handels wijze. Men schrijft een regel aan een entiteit of individu voor. Het te vertonen gedrag van een entiteit of individu wordt daartoe beschreven.

v.b De verkeersregels (een protocol) worden door middel van individueel te volgen gedrag beschreven: Als er een auto van rechts komt ; stoppen.

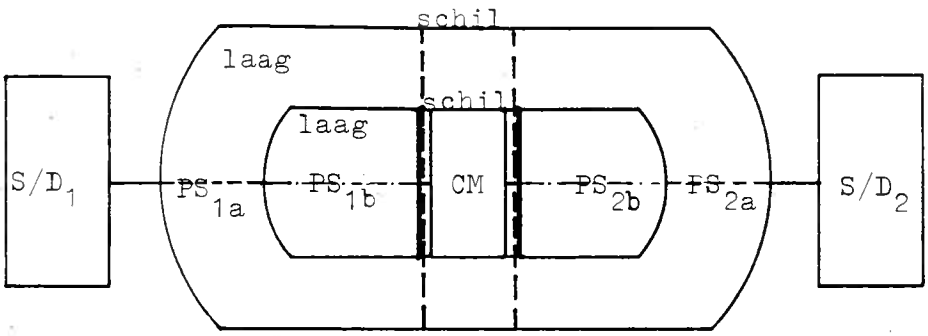


fig 5.2 Van schil naar laag

vertoont zeer nauwkeurig vast te leggen, en twee processen tegenover elkaar die het beschreven gedrag vertonen dienen aan de eisen van de gebruikers te voldoen. De functionele specificatie heeft daardoor een tweedelige omgeving, het ene deel wordt gevormd door de gebruikers (hogere lagen), het andere door het communicatiemedium (lagere lagen).

Tijdens het ontwerp van een protocol worden de eisen (eerste deel omgeving) omgezet naar complexere structuren die aan de kant van het communicatiemedium (tweede deel omgeving) te voorschijn komen. Dit maakt de functionele specificatie zeer belangrijk, het gedrag (eigenschappen, functies) aan de communicatiemedium-kant dient nauwkeurig beschreven te worden om betrouwbaar te kunnen communiceren. Dit betekent dat er zware eisen aan de functionele specificatie worden gesteld.

Er is naar te streven de technische invulling van de functionele specificatie vrij te laten (H 3), de functionele specificatie moet op meer bestekken af te beelden zijn. Met de beschrijving van de functionele specificatie hoort het ontwerp dus nog niet klaar te zijn. De bestek is weer wel een-eenduidig af te beelden op een implementatie.

Nu is het functioneel specificeren van een complex communicerend proces als geheel dermate moeilijk (iets heel complex dien je als een entiteit te benaderen) dat men snel gedwongen wordt aspecten van de implementatie mee te nemen. De scheidingslijn tussen functionele specificatie en bestek is daardoor moeilijk te trekken. In paragraaf 4 van dit hoofdstuk wordt verder ingegaan op de benadering van de complexiteit van een protocol (zie ook Appendix E).

Het doel van de functionele specificatie kan nu als volgt gedefinieerd worden:

De functionele specificatie moet een entiteit zodanig beschrijven dat alle implementaties die aan de specificatie voldoen, met elkaar communicerend, aan alle gestelde eisen voldoen.

Hiertoe zal de functionele specificatie aan de volgende algemene eisen moeten voldoen.

-Voldoende sterk/compleet.
Alle aspecten van het communicerende proces moeten beschreven worden.
-Doorzichtig
Naar aanleiding van de functionele specificatie wordt een implementatie ontworpen. De begrijpbaarheid van de specificatie zal dit gemakkelijker maken.
-Kanoniek
De functionele specificatie dient geen overbodige gegevens te bevatten, er moet naar de eenvoudigste vorm gestreefd worden, of kanonieke vorm (naar de term in de netwerktheorie).

De functionele specificatie kan op formele en informele wijze (soort model; vgl. schaalmodel, wisk.model [45]) beschreven worden. Het verschil tussen formele en informele wijzen van beschrijving kan vooral gezocht worden in de eenduidigheid, geldigheid en vooral grondslag van de afspraken die erover gemaakt zijn.

5.2 Informele beschrijvings-technieken voor de functionele specificatie

Bij de informele beschrijvings-technieken wordt gebruik gemaakt van natuurlijke taal. Men probeert het communicerende proces op een of andere manier te beschrijven, in z'n mogelijkheden, en vaak ook in z'n inwendige structuur.
Bij het gebruik van natuurlijke taal kunnen echter subtiele verschillen in de interpretatie van de specificatie optreden, die men zich niet bewust is en die tot moeilijk te ontdekken fouten kunnen leiden in de verschillende implementaties. Volgend nadeel is dat de specificatie niet geverifieerd kan worden (H 6). Gebruik van natuurlijke taal voor een specificatie is daarom af te raden.
Bij de beschrijving van het communicerende proces maakt men ook gebruik van illustraties. Dat kunnen blokstructuur- of flow-chart-achtige tekeningen zijn. Omdat er voor deze tekeningen ook geen echt strakke afspraken gemaakt zijn resulteert de toevoeging van de illustraties ook niet tot een complete en eenduidige specificatie. Binnen een flow-chart zijn bijvoorbeeld enorme verschillen in abstractie in de blokjes mogelijk. Voor de beschrijving van de blokjes wordt verder weer natuurlijke taal gebruikt. Er is dan geen goede grondslag voor de illustraties.

5.3 Formele beschrijvings-technieken voor de functionele specificatie

Een formele beschrijvings-techniek heeft een wiskundige grondslag, is logisch onderbouwt. Natuurlijke taal blijft naast de functionele beschrijvingstechniek bruikbaar, een formele beschrijvings-techniek kan met toegevoegde tekst bijvoorbeeld sneller interpreteerbaar zijn.

Het te beschrijven communicerende proces heeft twee kenmerken.

1. berichten
2. volgordes

Om deze kenmerken te beschrijven bestaat een functionele specificatie uit twee delen, een statisch deel en een dynamisch deel.

Het statische deel bevat uitvoerings informatie. Het kan bijvoorbeeld aangeven waar bepaalde berichten in een frame worden gezet. Ook datastructuren worden in het statische deel beschreven.

Voor de specificatie van het dynamische deel zijn er twee methoden.

1. Bij de eerste methode probeert men het gedrag of de uitwendige eigenschappen van het communicerende proces te beschrijven. Een eigenschap kan ook als functie van het proces beschouwd worden, er worden dus functies beschreven (H 2). Deze benadering wordt ook wel 'extentioneel' genoemd, het proces wordt met een extentionele specificatie beschreven.

Er wordt bij extentionele specificatie technieken gebruik gemaakt van recente inzichten die men in programmeer talen heeft. Men beschrijft generatieve regels en legt op die manier de syntax van het protocol (taal) vast. Deze methode staat de laatste jaren erg in de belangstelling daar de implementeur (de techniek) de vrijheid wordt gegeven het proces verder zelf in te vullen. Veranderende technische inzichten kunnen zo steeds toegepast worden.

2. Bij de tweede methode kan men eigenlijk niet van een functionele specificatie spreken, daar de taakvolgordes beschreven worden in de inwendige structuur van het systeem. Het proces wordt intentioneel benaderd, de binnenkant wordt beschreven. Intentionele specificaties voor protocollen bestaan al langer dan extentionele, ze sluiten aan op technieken die in het verleden ontwikkeld zijn om het inwendige van processen te beschrijven in bijv. de schakeltechniek.

Een formele beschrijvings-techniek bestaat ook uit twee delen. Het eerste deel geeft gereedschappen aan waarmee het statische deel te beschijven is, het tweede deel geeft gereedschappen om het dynamische deel mee te beschrijven.

Voor het statische deel maken de formele beschrijvings-technieken gebruik van moderne technieken uit de software engineering.

In beide benaderingswijzen voor het dynamische deel van de specificatie zijn krachtige en minder krachtige formele beschrijvings-technieken te onderscheiden, de gebruikte modellen verschillen in aantal dimensies.

5.3.1 Intentionele beschrijvings-technieken

Bij de intentionele beschrijvingstechnieken wordt veelal gebruik gemaakt van Finite State Machine technieken (Appendix C). Met Finite State Machines (FSM) kunnen eenvoudige transitie modellen gemaakt worden. FSM zijn al zeer lang bekend, waardoor er al veel theorie rond opgebouwd is [5]. Ze zijn al veelvuldig toegepast in de schakeltechniek [38], en ook de relatie met talen is onderzocht [37][103].

Een groot bezwaar van FSM is dat er geen variabelen mee te beschrijven zijn. Dit betekent dat de variabelen in de vorm van toestanden verwerkt moeten worden. Afhankelijk van het gebruik van de variabelen kan daarmee iedere waarde van een variabele het aantal toestanden verdubbelen. Voor een variabele met een bereik n kan het aantal toestanden daardoor 2**n groeien. Dit verschijnsel staat bekend onder de naam toestandexplosie.

Om dit probleem te ondervangen wordt in de intentionele beschrijvingstechnieken veelal gebruik gemaakt van Extended Finite State Machines. Hierin worden over het proces en de toestanden heen scope variabelen gedefinieerd. Een voorbeeld van een dergelijke beschrijvingstechniek is SDL (13).

Men probeert beschrijvingstechnieken ook te standaardiseren. Het is voor de implementeurs namelijk onbegonnen werk voor iedere specificatie een andere beschrijvingstechniek te leren. Schema symbolen zijn in de elektrotechniek daarom bijvoorbeeld ook gestandaardiseerd. SDL (Specification and Description Language) is een intentionele beschrijvingswijze die ontwikkeld en gestandaardiseerd is door de CCITT.

Een andere intentionele benadering maakt gebruik van Petri-netten. Men zou kunnen zeggen dat in Petri-netten de FSM uitgebreid wordt met meervoudige transitie condities. In een Petri-net stelt een knooppunt een conditie voor en een actie een staaf ('bar'). In een staaf ontmoeten verschillende condities elkaar. Als alle condities gelden wordt de staaf uitgevoerd en worden andere condities ingesteld (een 'token' in geplaatst).

Een toestand in een Petri-net (dus het systeem) is bepaald door de combinatie van condities, het toestand begrip is daardoor al gedeeltelijk verlaten. Petri-netten zijn oorspronkelijk afkomstig uit de theorie van parallelle programmering (veel verwantschap met protocollen).

In figuur 5.3 is een voorbeeld van een Petri-net gegeven. Hierin zijn alle staven te vervullen functies en zijn de cirkels condities. Als bijvoorbeeld conditie M en B beide gelden wordt staaf f2 uitgevoerd en wordt conditie C ingesteld.

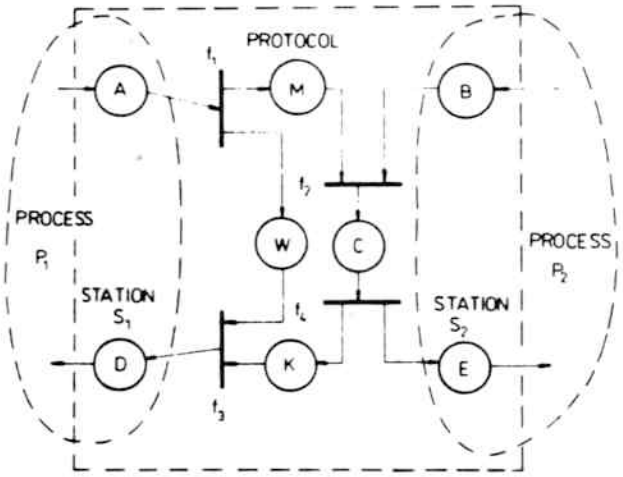


fig 5.3 Voorbeeld van een Petri-net

5.3.2 Extentionele beschrijvings-technieken

Extentionele specificatie technieken zijn vooral gebaseerd op de theorie van de programmeertalen. Men probeert het protocol te beschrijven zoals men de syntax en semantiek van een taal beschrijft. Dit gebeurt met generatieve regels. De moeilijkheidsgraad bij het opstellen van de regels ligt door de complexiteit van een protocol echter hoger dan bij programmeertalen.

Men moet volgorde relaties voor berichten definieren. De technieken die daarvoor ontwikkeld zijn maken gebruik van temporale logica [42] of reguliere expressies [6][43][105]. Ook in PANDORA wordt gebruik gemaakt van een extentionele specificatie, welke als grondslag reguliere expressies (algebra) kent (H 7).

Er wordt ook getracht extentionele beschrijvings-technieken te standaardiseren. Bij de ISO is men bezig de taal LOTOS (Language Of Temporal Ordering Specification) [44] te ontwikkelen.

5.4 Systeem structuur

Met het introduceren van een lagenmodel is de complexiteit van protocollen enigszins omvatbaar gemaakt. De functies die een OSI-laag moet vervullen zijn echter nog niet altijd in 1 systeemdeel voldoende nauwkeurig te beschrijven. Voor de functionele specificatie moet een verdere opsplitsing worden beschreven, tot op het niveau waarop voldoende nauwkeurig beschreven kan worden. Voor de functionele specificatie wordt dus een vaste (definitieve) systeem structuur ontworpen. De systeemdelen (modulen) hebben elk geringere complexiteit dan het geheel, terwijl de modulen tesamen aan alle gestelde eisen voldoen. De modulen kunnen bijvoorbeeld communicerende processen

zijn. In dat geval kunnen er binnen een knooppunt, op een laag meer communicerende processen bestaan. De functionele opbouw of systeemstructuur heeft grote invloed op de functionele specificatie.

Om de complexiteit van de losse delen te kunnen overzien, is het van belang dat de interactie tussen de delen niet al te ingewikkeld is. Dit is bovendien van groot belang voor de testbaarheid (H 3). Voor het bepalen van opsplitsingen bestaat geen echte methode. Er wordt opgesplitst op grond van bepaalde kriteria, die nog veel creativiteit vergen. Deze kriteria zijn samenhang in en koppeling tussen de subsystemen (modulen).

Samenhang is een maat voor de verwantschap tussen de componenten van een moduul. Meestal streeft men naar een zo groot mogelijke samenhang. Op deze wijze hoopt men een geringe samenhang tussen de modules onderling te krijgen, en daarmee een geringe interactie. De samenhang in een moduul of subsysteem kan vaak met de volgende kriteria worden verklaard.

- Toevallige samenhang. Hier zijn de componenten op een toevallige wijze in modulen gegroepeerd. Er is geen significant verband tussen de componenten.

- Logische samenhang. Hier verrichten de componenten van een moduul taken die logisch samenhangen (Moduul met alle invoerroutines).

- Samenhang in tijd. Een goed voorbeeld hiervan is een initialisatie moduul. De componenten van de moduul hebben weinig samenhang maar worden op het zelfde moment geactiveerd.

- Communicatie samenhang. Deze vorm van samenhang treedt op als de componenten op gemeenschappelijke gegevens opereren en dus via deze gegevens met elkaar kunnen communiceren.

- Functionele samenhang. Alle componenten van een moduul dragen bij aan een moduulfunctie (bijv. rekenfunctie).

Het zal natuurlijk lang niet altijd eenvoudig zijn een zo sterk mogelijke samenhang tussen de componenten van een moduul te verkrijgen. Ook zal men in een systeem vaak mengvormen aantreffen. Zo zal op de lagere niveaus van het systeem vaak communicatiesamenhang het hoogst haalbare zijn, terwijl op niveaus dichter bij de top functionele samenhang kan optreden.

Het tweede structurele kriterium is koppeling. Koppeling is een maat voor de sterkte van de verbindingen tussen de modulen onderling. Een hoge mate van koppeling treedt op als men het totale systeem op een willekeurige manier opsplitst. Het systeem krijgt daardoor een mozaiek structuur. Het andere extreem is en zuivere boomstructuur, waarbij van hierarchie sprake is.

Men streeft naar een zo gering mogelijke koppeling, dus veelal naar de hierarchische boomstructuur. Helaas impliceert een boomstructuur veelal veel overhead, en dus geringere performance, hetgeen de ontwerper tot een compromis dwingt. Het is niet altijd

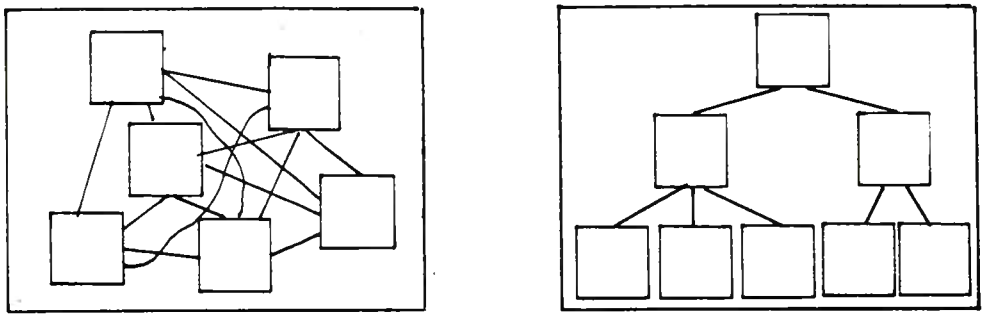


fig 5.4 Mozaiek structuur en boomstructuur

zo gemakkelijk de doorsnijdingen te kiezen.

In de volgende twee paragrafen worden twee uitwerkingen van het structurerings probleem beschouwd. Beide gaan uit van een bepaalde opsplitsing in deelprocessen of procedures.

5.4.1 Onderling structureren van communicerende processen

Men kan de communicerende processen onderling berichten laten uitwisselen. Hierdoor zijn binnen een OSI-schil twee soorten communicatie te onderscheiden: peer-communicatie (Communicatie tussen twee processen in verschillende knooppunten) en knooppunt-communicatie (Communicatie tussen twee processen op hetzelfde knooppunt).

De processen kunnen eventueel gegroepeerd worden in subsystemen (Bijlage V). De communicatie tussen de subsystemen verloopt echter via willekeurige communicatie tussen de elementen, de processen. Hierdoor komt de communicatie tussen de subsystemen niet zo goed uit de verf, en als men niet oppast, de subsystemen zelf ook niet. Het zal lijken dat er voor de vorm om wat groepjes elementen lijntjes zijn getrokken (fig 5.5).

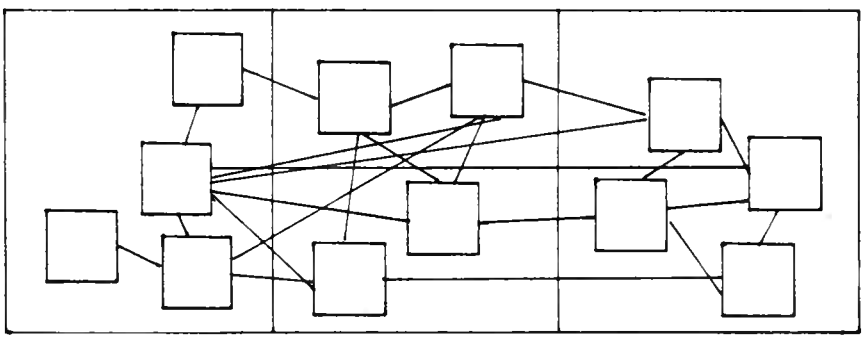


fig 5.5 Chaotische processtructuur

Dit is eventueel op te vangen door elk subsysteem hierarchisch

op te bouwen. Het proces dat boven in de hierarchie ligt verzorgt de communicatie tussen de subsystemen, het vertegenwoordigt het eigen subsysteem (fig 5.6).

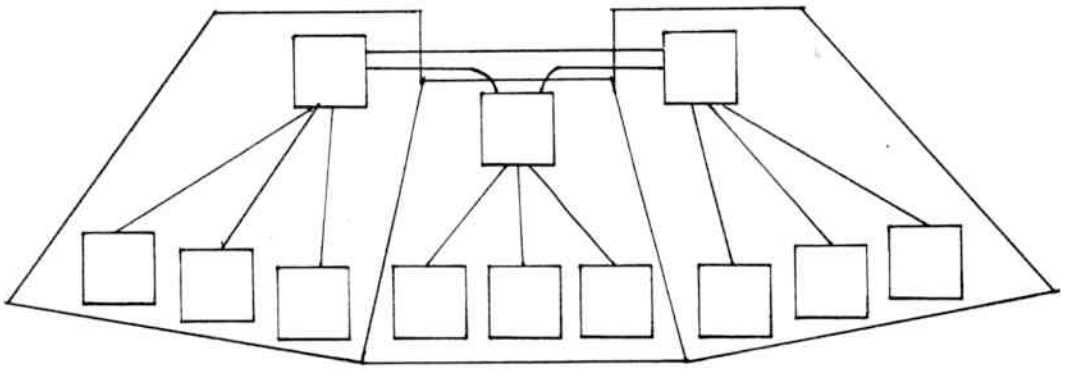


fig 5.6 Hierarchische processtructuur

Om het functioneren van elk subsysteem verder te onderstrepen moet van het gedrag van het subsysteem een vereenvoudigde formele specificatie gemaakt worden. Dit maakt de subsystemen tevens afzonderlijk testbaar/verifieerbaar. Het vergt echter een zeer consequente benadering de subsystemen zo uitvoerig te definieren.

5.4.2 Toestand/stimulus tabellen

Een recente protocol systeemstructuur maakt gebruik van zo geheten toetstand/stimulus tabellen [15][16][40]. Deze wordt beschouwd vanuit het OSI model (H 4.3). waarbij het OSI-model nog iets verder uitgewerkt wordt.

De gegevensstromen binnen een OSI laag worden bij deze systeemstructuur als volgt zichtbaar gemaakt.

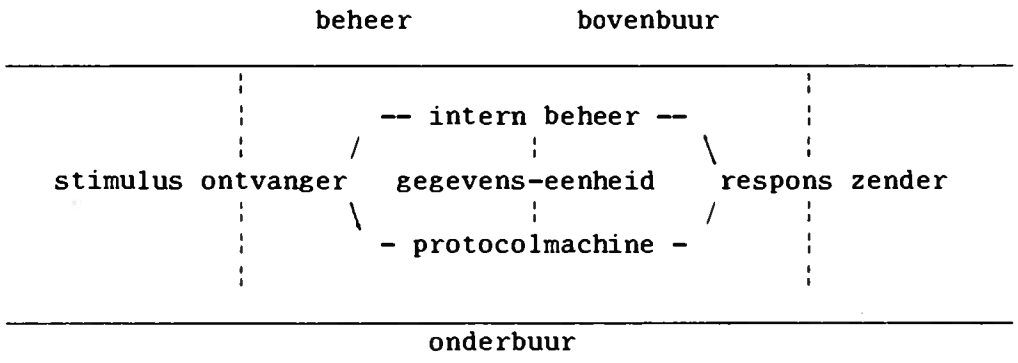


fig 5.7 Protocol structuur met toestand/stimulus tabel

De gegevens worden van de boven- en onderliggende laag ontvangen door de stimuli ontvanger en verzonden naar deze lagen door de respons zender. Beheersgegevens lopen daarbij van/naar het interne beheer, overige gegevens lopen van/naar de protocolmachine. Verder is er nog een gegevenseenheid, die

administratieve en toestandsgegevens bevat. Zowel het interne beheer als de protocolmachine hebben hier kontakt mee.

De protocolmachine wordt gevormd door de toestand/stimulus tabel. De protocolmachine werkt als volgt.
De protocolmachine ontvangt stimuli (Protocol Data Units en Service Data Units) van de stimulus ontvanger, stuurt responses via de respons zender en gebruikt gegevens uit de gegevens eenheid.
Binnen de protocolmachine loopt de informatie stroom als volgt: Stimuli komen binnen bij de ontvanger. Deze kiest al naar gelang de protocolstatus en stimulus een tabel. Al naar gelang de grootte van het protocol kunnen er een of meer tabellen zijn. In de tabel wordt aan de hand van stimulus en status een protocol procedure gekozen. De verschillende procedures kunnen zijn;

- gegevens van de gegevens-eenheid lezen en wijzigen,
- response zenden (via bovenbuur-, benedenbuur- of peer-bericht procedures, die op hun beurt gebruik maken van een zender),
- predikaten testen (vergelijken met gegevens uit de gegevenseenheid)
- speciale akties uitvoeren, taken verrichten.

Na afloop van de procedure wordt een volgende toestand gekozen. De opvolgende toestand kan bepaald worden door de procedure. Als de procedure de opvolgende toestand niet bepaalt, dan wordt de volgende toestand bepaald door een default-transitietabel. Dit is een toestand/stimulus tabel met op de locaties default-opvolgende toestanden. De door de procedure bepaalde toestand heeft prioriteit ten opzichte van deze tabel.
Bij deze methode staat de toestand stimulus tabel boven de verschillende processen (procedures) en is dus sprake van een zekere hierarchie.

5.5 Standaardisatie

Protocollen hebben pas echt waarde als ze door velen gebruikt worden. Dit vergroot de mogelijkheden van de gebruikers, welke immers behoefte hebben aan een zo omvangrijk mogelijk communicatiebereik. Dit geldt ook in internationaal opzicht. Op internationaal niveau wordt daarom intensief overleg gevoerd om protocollen en protocolgereedschappen te standaardiseren.
Moeilijkheid bij standaardisatie is echter dat vele gebruikers vele wensen (behoeften, eisen) hebben. Hierdoor gaat bij de standaardisatie van protocollen de politiek en de belangenafweging een grote rol spelen.

Veelal ontstaan hierdoor compromissen waarbij de invulling van delen van het protocol geheel vrij of naar keuze gelaten worden. Verdere vrijheid wordt gecreerd door de specificatie in natuurlijke taal te beschrijven. In politieke termen wordt zoiets vervolgens een aanbeveling genoemd.

De formaliteit van de standaards is daardoor ver te zoeken. Dit is niet verwonderlijk, formaliteit en vrijblijvendheid zijn strijdig.

6. VERIFICATIE

6.1 De aan een protocol gestelde eisen

De eisen die aan het protocol gesteld worden zijn afgeleid van de eisen die aan een datacommunicatie netwerk gesteld worden. De eisen gelden voor het communicatiemedium met protocollen in het algemeen, en gelden dus voor communicatie met twee stations of meer (fig 6.1).

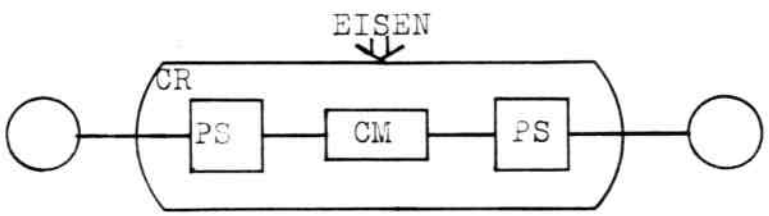


fig 6.1, Eisen aan communicatie

De eisen aan een protocol hebben globaal betrekking op prestatie, geboden diensten en veiligheid of consistentie. Prestatie is een real-time eigenschap en is daardoor vooral een eigenschap die betrekking heeft op de implementatie. Bij de gevolgde procedure in het communicerende proces zou men hoogstens van doelmatigheid kunnen spreken.

Verificatie (H 3.3) tracht aan te tonen dat de functionele specificatie aan de eisen voldoet. De functionele specificatie kan dus op doelmatigheid, geboden diensten en veiligheid of consistentie geverifieerd worden. In plaats van geboden diensten en consistentie spreekt men ook wel van semantiek (juiste betekenis) en syntax (juiste structuur). Deze termen hebben vooral betrekking op een extentionele specificatie (H 5.3.2).
Verificatie is een vorm van bestudering van een systeem, men

werkt met modellen. Voor die toepassing zijn alleen formele modellen, dus formele functionele specificaties zinvol. Als de eisen ook formeel zijn opgesteld dan is de verificatie eenduidig vastgelegd.

Voor een eenvoudige beschouwing van de eisen aan een protocol ga ik even uit van een intentionele functionele specificatie door middel van een procesbeschrijving. De eisen kunnen hierop als volgt worden geprojecteerd.

Een proces kan beschouwd worden als een volgorde van handelingen of taken. In een proces kunnen beslissingen genomen worden wat betreft de volgorde en de keuze van de taken. Een proces kan een eenmalig resultaat opleveren. Het resultaat is dan gerealiseerd als het proces geheel, met de juiste volgorde van de juiste taken, doorlopen is.

Het geheel doorlopen van het proces kan gerelateerd worden aan de veiligheid, voor de veiligheid moet continuiteit gewaarborgd zijn. De juistheid van de volgorde en de taken kan gezien worden als de correcte werking, een eigenschap die de juiste geboden diensten (het resultaat) garandeert. Maat voor de doelmatigheid is de hoeveelheid onnodige volgordes van taken.

Bij een protocol is het moeilijker om van resultaten van processen te spreken. Een protocol geldt continu (regels), de beschreven processen zullen continu werken. Bovendien wordt een protocol door minstens twee communicerende processen vertolkt (H5.4.2). Het resultaat ligt niet eenduidig bij het doorlopen van een proces.

De vertaling van eisen naar eigenschappen is daardoor voor een protocol veel complexer.

Met bovenstaand voorbeeld heb ik getracht de moeilijkheid van protocolverificatie enigszins aan te geven. Er wordt veel onderzoek naar protocolverificatie gedaan. Dit onderzoek is op te splitsen naar de eisen die er aan protocollen gesteld worden [31].

Het onderzoek naar de verificatie van de correcte werking van een protocol gaat uit van theorieen die in de software engineering ontwikkeld zijn. De gebruikte theorieen zijn daarbij afkomstig uit het onderzoek naar de semantiek van software. Bij protocolspecificaties wordt voornamelijk bij het extentionele type van semantiek gesproken, er wordt voor het onderzoek naar de correcte werking van een protocol dan ook voornamelijk van extentionele protocol specificaties gebruik gemaakt. Het semantisch onderzoek van protocollen is in een beginstadium, er wordt nu vooral nog onderzoek gedaan naar de bruikbaarheid voor protocollen van bestaande semantische software verificatie systemen [34][35]. De verificatie is gebaseerd op het onderzoeken of (zelf te bedenken) beweringen voor een protocol blijven gelden (analoog met bijv. invarianten).

Het onderzoek naar de prestatie van protocollen gebeurde tot voor kort voornamelijk door berekeningen op grond van het statische deel van de functionele specificatie en de bekende (gewenste) eigenschappen van het protocol. Daarbij werd gebruik

gemaakt van de klassieke wachttijdtheorie. Zeer recentelijk is onderzoek gestart naar het uitbreiden van deze prestatie voorspellingen op basis van het dynamische deel van de specificatie.

6.2 Validatie

Het onderzoek naar de veiligheid of consistentie van protocollen is relatief het meest ver gevorderd. De verificatie van de veiligheid of consistentie van protocollen wordt veelal aangeduid met validatie (v.g.l. huis (H 3.2) berekeningen aan veiligheid, of consistentie), er kan namelijk gesproken worden van algemene eigenschappen.

De veiligheid of consistentie van protocollen kan gerelateerd worden aan de volgende eigenschappen van de gespecificeerde communicerende processen.

1. Vrij zijn van deadlocks

Als een van de processen definitief in zijn voortgang stokt, ofwel ergens blijft hangen, dan is dit proces in deadlock. Het proces zal daardoor stoppen met communiceren, iets wat we zeker niet willen. Deze deadlock wordt ook wel statische deadlock genoemd. De statische deadlock is iets wat mogelijk slechts bij een proces kan optreden en niet voor de gehele communicatie hoeft te gelden (communicatie tussen andere processen kan wellicht evengoed doorgaan).

Globaal zijn er drie oorzaken van statische deadlocks bij communicerende processen te onderscheiden.

a. Het eerste type deadlock wordt intern in het communicerende proces veroorzaakt. Dit type deadlock is te vergelijken met deadlocks welke in gewone programma's kunnen vookomen. Het programma voert een handeling uit (cyclisch of statisch) welke nooit zal eindigen. Hierdoor zal de communicatie van het programma (in- en uitvoer) ook stokken.

b. Een deadlock kan ook veroorzaakt worden door een verstopte (FIFO) ingangsbuffer van het in deadlock verkerende proces. Het proces heeft dan een bericht gekregen waar het niets mee kan doen, het herkend het bericht niet, en het bericht blijft staan.

c. Als niet herkende berichten uit de ingangsbuffer verwijderd worden dan zal dit type deadlock niet meer voorkomen. Een communicerend proces met een dergelijke ingangsbuffer verwacht een bericht uit een bepaalde verzameling berichten. Deze verzameling wordt na elke berichtontvangst bijgesteld. Als geen van de berichten uit de verzameling daarna kan binnenkomen (door de toestand van de omgeving van het proces), dan kent het proces geen voortgang meer en is het ook in deadlock. Er is dan sprake van een onbalans tussen het proces en de rest van het systeem.

De laatste twee types deadlock zijn typerend voor communicatie.

Naast de statische deadlock kennen we ook de dynamische deadlock. Bij een dynamische deadlock zijn meer (hoeft weer niet

alle) processen via de onderlinge communicatie in een onvermijdelijke oscillatie. De oscillatie is niet produktief en kent geen uitweg [33] (Appendix F,H 5.1.3).

2. Bestendigheid

In de loop van de tijd moet een proces altijd hetzelfde gedrag kunnen vertonen. Men wil niet dat er in de loop van de tijd delen van het gedrag niet meer kunnen voorkomen. In het proces zelf betekent dit dat alle delen van het proces altijd bereikbaar moeten blijven.

Bovenstaande twee eigenschappen zijn essentieel voor het goede functioneren van het systeem. De volgende eigenschap betreft vooral tijdsaspecten.

3. Vrij zijn van ongewenst herhaald gedrag

Als een proces hetzelfde gedrag herhaald vertoont, dus herhaald dezelfde berichtenvolgordes produceert en consumeert dan kan dit proces niet produktief wat betreft communicatie zijn. Men zou van een (eindige) oscillatie tussen een proces en de rest van het systeem kunnen spreken. In het proces zelf is dit herhaalde gedrag te herkennen aan lussen in de procesgang of taakafhandeling.

Naast bovenstaande proces eigenschappen zijn nog een aantal protocol specifieke eigenschappen te herkennen die betrekking hebben op de veiligheid van het protocol. Deze gelden dus niet voor alle OSI-protocollen. Onderstaand worden een aantal voorbeelden gegeven. Het rijtje kan daarbij naar protocol aangevuld worden, er worden enige voorbeelden gegeven. De voorbeelden zijn in het verleden opgesteld voor ongelaagde protocollen[41].

- Goede verdeling van geboden voorzieningen.

Een eigenschap die bijvoorbeeld voor LAN (Local Area Network) protocollen erg belangrijk is. De eigenschap ligt dicht bij de implementatie omdat er real-time aspecten bij komen kijken en de capaciteit ook een rol speelt.

- Voorzien in foutherstelprocedures.

Een fout kan extern en intern beschouwd worden. Intern is het een situatie welke niet verwacht is (het ontstaan van de situatie wordt in het midden gelaten) en niet wenselijk, extern kan een fout door de omgeving veroorzaakt worden (storend kanaal). In het OSI-lagenmodel zijn op laag 2 foutherstelprocedures te vinden voor externe fouten. Om op alles voorbereid te zijn kunnen foutherstelprocedures na interne ongewenste situaties op meer lagen gelegd worden. Het voorzien in foutherstelprocedures is een algemenere eigenschap, maar heeft wel betrekking op de semantiek, valt dus niet onder de term validatie.

Het is mogelijk de eerste drie (proces)eigenschappen met geautomatiseerde hulpmiddelen te onderzoeken. Al deze geautomatiseerde hulpmiddelen hebben gemeen dat er beweringen op een bepaald model worden losgelaten, die betrekking hebben op de onderzochte proceseigenschappen. De geautomatiseerde hulpmiddelen zijn voor een deel specifiek voor protocollen ontwikkeld, en voor een deel afkomstig uit de software engineering.

De lijst van beweringen die bij de software verificatie systemen opgesteld wordt ten behoeve van procesvalidatie heeft betrekking op specifieke proces- en protocol eigenschappen. De gebruikte modellen zijn daarbij echter bedoeld voor software en behoeven daardoor niet zo geschikt te zijn voor protocollen [35]. De geschiktheid van bestaande software verificatiesystemen is verkend maar blijkt niet optimaal te zijn [34]. Het onderzoek ernaar wordt voortgezet.

De specifiek voor protocollen ontwikkelde validatiesystemen van proceseigenschappen gebruiken speciale protocol modellen. Deze modellen zijn, of zijn afgeleid, van het dynamische deel van de functionele specificatie.

De vroegste resultaten werden, gebruik makend van theorieen rond de van oudsher bekende Finite State Machines, met intentionele specificaties behaald. Relatief eenvoudige protocolletjes zijn met behulp van FSM technieken succesvol gevalideerd [30]. Bij grote protocollen stuit men echter toch al snel op de geringe kracht van het FSM model. Deze zwakheid probeert men de laatste jaren te omzeilen door theorieen op te zetten waarmee het te valideren model kleiner te maken is dan de werkelijke protocolspecificatie, veelbelovende resultaten zijn daarbij echter nog niet bereikt [3].

De laatste jaren wordt ook onderzoek gedaan naar validatie van protocollen op grond van een extentionele specificatie (bijvoorbeeld PANDORA) Dit onderzoek heeft al aardige resultaten opgeleverd [4][108], men heeft echter ook moeilijkheden met het valideren van grotere protocollen. Ook hier wordt dus onderzoek gedaan naar het verkrijgen van kleinere validatie modellen [10].

Er wordt ook onderzoek gedaan naar validatie met behulp van andere modellen, als Petri-netten en hybride modellen. Daarbij stuit men echter ook steeds op het complexiteits probleem[24][25].

6.3 Validatie techniek: bereikbaarheidsanalyse

De technieken die specificiek voor protocol validatie ontwikkeld zijn berusten grotendeels op dezelfde filosofie: bereikbaarheidsanalyse. Bij de bereikbaarheidsanalyse wordt gebruik gemaakt van een model waarin de processen via oneindig grote FIFO-buffers met elkaar communiceren, een meta-model van de communicatie (fig 6.2). Er wordt getracht alle mogelijke executie volgorden van dit model te onderzoeken op het voldoen aan bepaalde kenmerken, kenmerken die betrekking hebben op een of meer van de onderzochte eigenschappen. Een dergelijke techniek bestaat uit twee delen;

1. Vindt alle executievolgorden
2. Test alle executievolgorden op een bepaald kenmerk (predicaat) en deel de executievolgorden op naar kenmerk.

De testen op een bepaald kenmerk kunnen zowel na als tijdens het opbouwen van een bepaalde executievolgorde gedaan worden.

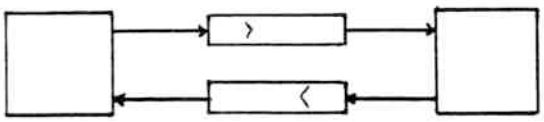


fig 6.2 Meta-model t.b.v. validatie

Omdat het model met het toevoegen van een derde proces direct al zeer complex wordt door de optredende afhankelijkheid wordt bij de meeste technieken uitgegaan van slechts twee processen. Twee communicerende processen voeren een onafhankelijke communicatie, drie communicerende processen zijn bij een vermaasde constructie al zeker niet meer twee aan twee onafhankelijk (fig 6.3), omdat er buiten de bestudeerde communicatie om nog meer communicatie plaatsvindt.

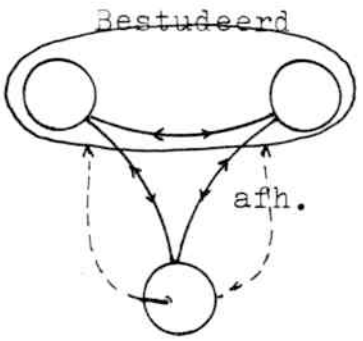


fig 6.3 Afhankelijkheid drie vermaasde processen

Het aantal executie volgorden (men kan spreken van een executie-boom of executie-ruimte)) is zelfs voor 2 eenvoudige met elkaar communicerende processen erg groot. Men probeert daarom in de executie-boom bepaalde volgorden af te breken, men 'reduceert' de omvang van de boom. Het afbreken van bepaalde volgorden moet echter onder strenge condities gebeuren, opdat de analyse zijn geldigheid niet verliest [47]. Ook in PANDORA wordt een dergelijke techniek toegepast [106].

De kenmerken van de executie volgorde die men onderzoekt, verschillen per soort model. Bij een intentioneel model kan men immers naar de inwendige procestoestanden kijken, bij een extentioneel model naar de uitwendige.

Het aantal mogelijke executievolgorden -zelfs gereduceerd- groeit zeer snel met de toenemende grootte van het model. Dit probleem is fundamenteel, zelfs bij gebruik van de allerkrachtigste computers zal het probleem zich voordoen. Men is daardoor gedwongen om modellen met beperkte grootte te

valideren. Dit conflicteerd helaas met de toenemende complexiteit van protocollen. Daarnaast is het moeilijk om een model met meer als twee communicerende processen te valideren. Door systeemstructurering kan er echter in een complex protocol in elk communicatie knooppunt meer als een proces (procedure) in een laag aanwezig zijn.

6.4 De plaats van validatie in het ontwerp

Men probeert het complexiteitsprobleem bij validatie op te lossen door nog meer relatieve testen (fig 3.6) gedurende het ontwerp te doen. De laatste jaren is men daarom bezig met het ontwikkelen van zogenaamde ontwerpsystemen, een soort expert systems voor protocol ontwerp [29][49]. Er bestaan al verschillende van dit soort systemen, die het karakter hebben van een ontwikkelomgeving. Er zijn in deze systemen diverse gereedschappen ('tools') beschikbaar die het inzicht van de ontwerper in het protocol kunnen vergroten en de mogelijkheid bieden bepaalde mechanismes te valideren/verifieren. Bovendien wordt er naar gestreeft na het ontwerp automatisch een implementatie te genereren. Ook PANDORA is bedoeld om protocollen mee te ontwerpen[104].

Omdat de ontwerper van een protocol het meeste inzicht in het protocol zal hebben gaat men er van uit dat deze in staat is goede validatie modellen op te zetten. Dit garandeert een doordachte opzet. De sequentiele geisoleerde validatie gedurende het gehele ontwerp hoeft echter nog geen foutloos protocol op te leveren (fig 6.4).

De sommatie eigenschap zal zeker niet hoeven gelden, als de met a gemerkte validaties kloppen dan hoeft de met b gemerkte valdatie nog niet op te gaan. Dit vindt zijn oorzaak in de afhankelijkheid van de verschillende systeemdelen. Het is waarschijnlijk wel mogelijk theorieen te bedenken die restricties opleggen aan het ontwerp, zodanig dat stapsgewijze validatie wel validatie van het gehele systeem inhoud. Hier is men op het gebied van de synthese, het opleggen van ontwerpregels[26][50]. De ontwikkelingen op dit gebied zijn echter nog in het prille beginstadium.

Men wil synthese ook graag automatiseren. Hiertoe onderzoekt men of het mogelijk is een syntaxgenerator te maken. Men voert in deze generator de semantiek in en laat de generator vervolgens zelf de syntax bedenken. In de praktijk kan dit er bijvoorbeeld op neer komen dat voor ieder verzonden bericht alle mogelijke delen van het ontvangende proces waar dit bericht ontvangen kan worden gegeven worden, waarna bij de bedoelde ontvangsten een actie toegevoegd wordt en de overige een dummy ontvangst.

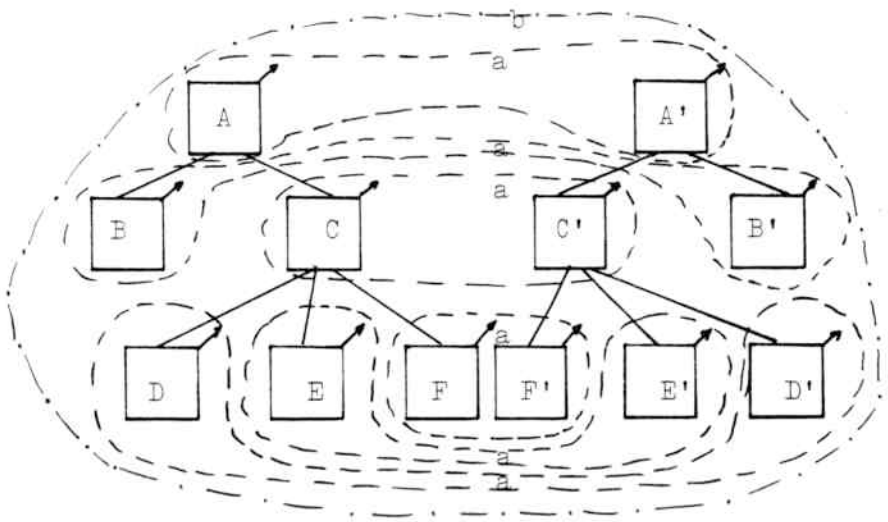


fig 6.4 Sommeren van validatie

6.5 Naar kleinere modellen

Met een gelaagde structuur als in OSI is reeds bereikt dat de grootte van de protocollen enigszins hanteerbaar is geworden. Toch is een protocol op een laag ook al zo groot dat deze als geheel geen bruikbaar validatiemodel vormt (H 6.3, H 5.4). Men wordt daardoor gedwongen om slechts delen van het protocol te valideren. De validatie heeft echter pas het karakter van een correctheidsbewijs als bewezen kan worden dat de samenstelling van de verschillende correcte delen een correct geheel vormt. Dit is echter door de afhankelijkheid van delen (H 6.4) van het protocol vrijwel onmogelijk (Appendix E).

Het heeft toch zin om delen van een protocol te valideren, ook al wordt daarmee niet de algehele correctheid van het protocol bewezen. Een deel afzonderlijk dient toch ook naar behoren te functioneren (niet goed functionerende ddelen hoeven niet catastrofaal te zijn, zie Appendix E).
In de methoden om kleinere modellen te verkrijgen kunnen methoden die een wetenschappelijke basis hebben en methoden die veel eigen creativiteit vergen, worden onderscheiden.

Veelal zal een deel van een protocol een bepaalde functie voor het protocol vervullen (H 5.4). De vervulling van deze functie is te projecteren uit het model. Men probeert deze projectie fundamenteel aan te pakken [7] door te bewijzen dat ze toegestaan is. Hierbij gaat men uit van de executie-ruimte en bewijst daarvandaan dat het mogelijk is een concreet 'beeld'protocol te

creeren. Het is in de praktijk echter niet eenvoudig om dergelijke theorieen toe te passen.

Andere technieken proberen 2 processen onder bepaalde voorwaarden (eenzijdige afhankelijkheid) te combineren tot 1 proces [3][10]. Dit noemt men directe koppeling. Helaas kan dit in het gros van de gevallen niet omdat er aan de voorwaarde niet voldaan is.

Verder is het ook nog mogelijk het protocol te reduceren, of optimaliseren. Doordat er veel bekend over is kan dit bij Finite State Machines goed (Appendix C).

De meeste bruikbare technieken om kleinere validatie modellen te verkrijgen gaan uit van een grote dosis creativiteit en interpretatie vermogen. Er wordt bij deze technieken van de volgende principes uitgegaan.

1. In een techniek wordt geprobeert op het laagste abstractieniveau te valideren. Om de grote omvang aan te kunnen probeert men functies uit het model te projecteren. Deze projectie geschiedt op basis van eigen interpretatie. Soms kan daarbij direct gebruik gemaakt worden van de structuur van het protocol, een functie kan als gevolg van de systeem structuur van het protocol bijvoorbeeld geheel in een proces vervuld worden (bijv. initialisatie moduul).

2. Een andere methode probeert op een hoger abstractieniveau te valideren. Hiertoe verwijdert men gevoelsmatig onbelangrijke zaken (visie vanuit hoger abstractieniveau) uit het protocol. Deze techniek kan eenvoudig met projectie gecombineerd worden.

Om een grote hoeveelheid processen aan te kunnen is ook gebruik te maken van deze technieken (H 9).

Genoemde bruikbare methoden hebben een praktische grondslag, geen wetenschappelijke. Hierdoor zal voor validatie hetzelfde gaan gelden als voor testen van een implementatie; men kan alleen de aanwezigheid van fouten aantonen, niet de afwezigheid. Validatie kan het beste al tijdens het ontwerp van het protocol worden uitgevoerd. De ontwerper kent het protocol immers beter dan anderen, en de validatie dwingt hem wellicht tot doordachte systeemstructuren.. Een algehele validatie achteraf kan vervolgens door derden worden uitgevoerd (belangen verdeling).

7. PANDORA

7.1 Inleiding

Het PANDORA systeem wordt ontwikkeld op de vakgroep Automatische Verkeers Systemen van de afdeling Elektrotechniek aan de TH-Delft, met steun van het Dr.Neher Laboratorium van de PTT. Het PANDORA project is in 1982 opgezet met de bedoeling onderzoek te doen naar een systeem waarmee het mogelijk is protocollen te valideren, te ontwerpen en te testen. PANDORA kan daarmee beschouwd worden als een protocol ontwikkel-omgeving. PANDORA staat voor Protocol ANalysis, Design and OpeRation Assessment. PANDORA draait onder het UNIX operating systeem.

Sinds enige tijd is het mogelijk protocollen op het systeem te valideren [102][108]. Het onderzoek naar de mogelijkheden om protocollen te testen (operation assessment) is slechts in het beginstadium. Ook naar ontwerpregels (synthese) voor protocollen is in het PANDORA project nog weinig onderzoek gedaan. De synthese is beperkt tot het opsporen van grovere fouten d.m.v. een compiler [101]. Deze compiler kan vinden of bepaalde processen geisoleerd zijn, berichten ontvangen kunnen worden die nergens verzonden kunnen worden, berichten verzonden worden welke nergens ontvangen kunnen worden etc. Van synthese met het doel bijvoorbeeld deadlocks te vermijden (6.4) is nog geen sprake.

7.2 PSL

PANDORA werkt met een extentioneel beschreven model. Dit model wordt beschreven in de beschrijvingstaal PSL (Pandora Specification Language). Met PSL kan het gedrag van processen beschreven worden, de taal kent geen (interne) toestanden (H 5.3.2). Met PSL (een meta-taal) is het mogelijk een meta-model van communicerende processen te beschrijven [103]. De belangrijkste elementen van PSL staan in de volgende tabel beschreven.

```
proc X.. ..end X    /* proces definitie */
bestemming!bericht  /* verz. bericht naar bestemming; event*/
bron?bericht        /* ontvangst bericht van bron; event*/
timeout             /* aflopende timer; event */
default             /* ontvangst ander bericht; event */
->                  /* volgorde; na elkaar */
;                   /* volgorde; na elkaar */
if.... ...fi        /* ongeconditioneerde keuze; statement*/
do.... ...od        /* herhaalde ongecond. keuze; statement*/
break               /* spring uit do_od */
::                  /* vlag voor statement */
goto                /* ..... */
skip                /* non-event*/
```

De taal kent geen variabelen in de processen en in de berichten. Een voorbeeld van een PSL-model is in figuur 7.1 gegeven.

```
proc greg
   doug!init;
   if
   :: doug?ack -> doug!request;
       do
       :: doug?data -> doug!request
       :: doug?data -> doug!enough ; break
       :: timeout   -> doug!abort  ; break
       od
   :: default -> doug!abort
   :: timeout -> doug!abort
   fi;
   doug?done
end greg;

proc doug
   do
   :: default   -> skip
   :: greg?init -> greg!ack;
       do
       :: greg?request -> greg!data
       :: greg?request -> skip
       :: greg?enough  -> break
       :: greg?abort   -> break
       od;
       greg!done; break
   od
end doug.
```

fig 7.1 PSL voorbeeld

Het is in de figuur zichtbaar dat het in PSL mogelijk is om te nesten. In de PANDORA handleiding [107] is te vinden hoe deze PSL specificatie geinterpreteerd moet worden.

Doordat met PSL geen variabelen beschreven kunnen worden kan de kracht van een PSL-model vergeleken worden met de kracht van het gewone Finite State Machine-model. Door enige aanvullende taalconstructies toe te voegen is het mogelijk om met PSL Finite

State Machines te beschrijven. Het is ook mogelijk om uit een PSL-model een Finite State Machine model op te stellen [103]. Hiertoe dient men zelf enige aanvullende syntactische regels op te stellen, en te volgen bij het maken van een PSL-model. Enige gereedschappen ('tools') in het PANDORA systeem zijn in staat om uit het PSL-model SDL-achtige plaatjes en transitie tabellen te genereren [101][103][107][109].

7.3 Het beschreven model

PSL beschrijft een model waarin ieder gedefinieerd proces een ingangsbuffer heeft (berichten wachtrij) waarin alle overige processen voor dat proces bestemde berichten kunnen deponeren. De buffers worden eindig maar van ongedefinieerde grootte gesteld. De buffers werken volgens het FIFO principe. In figuur 7.2 is een PSL model met drie processen weergegeven.

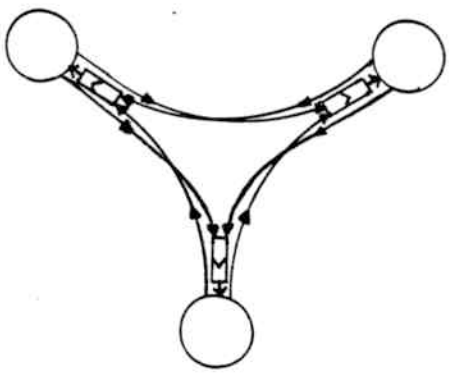


fig 7.2, PSL-Model

Dit model komt overeen met gebruikelijke modellen voor communicerende processen. Alle operaties op een buffer sluiten elkaar uit in de tijd. In lege buffers wordt door een 'systeem-duivel' (system-demon) timeout berichten gedeponeerd (na op te geven tijd? [6]), hiermee wordt gesteld dat het timeout bericht het oudste bericht is. Het is met enige restricties ook mogelijk meer ingangsbuffers per proces te definieren.

Bij de beschrijving van het model is het niet mogelijk rekening te houden met tijd. Hierdoor kan aan volgorden als gevolg van parallelle acties geen restricties op basis van tijd gesteld worden, alle volgorden (hoe onwaarschijlijk ook) zijn mogelijk. Dit is vooral van belang voor timeout berichten, deze kunnen op de meest ongewenste momenten binnenkomen.

7.4 Analyse

Het validatiedeel van PANDORA werkt op basis van reguliere expressies[6], het PSL-model wordt door een compiler ten behoeve van de validatie omgezet in een algebraisch model. Voor ieder

proces wordt een algebraische notatie opgesteld, bijv.:

[(1/a)(1/b)a(1/c)b(d+e)]

Door deze expressies uit te werken (vergelijkbaar met in de gewone algebra a(b+c)=ab+ac), ontstaan de volgordes die een proces in zijn berichtenbehandeling kan hebben. Voor de algebra zijn daartoe enige regels (axioma's) bedacht.

Met behulp van een cross-operator worden de uitwerkingen van de processen vervolgens gecombineerd. Hierdoor ontstaan de volgordes die het model als geheel kan hebben. Dit is een vorm van bereikbaarheidsanalyse op basis van combinatie van mogelijkheden.

Elke volgorde wordt op een paar kenmerken onderzocht. Een volgorde kan normaal, in een (statische) deadlock of in een lus eindigen. Een volgorde wordt na optreden van een lus niet voortgezet, hoewel dit in werkelijkheid wel mogelijk kan zijn (vorm van reductie). Een volgorde wordt met normaal gekenmerkt als alle proces expressies volledig uitgewerkt zijn voor de betreffende volgorde. Als een van de proces expressies niet is uitgewerkt, dus niet door zijn gehele specificatie heen is, is dit proces in deadlock. In de algebra heeft een deadlock in de uitwerking een bepaalde verschijningsvorm. Een lus wordt gekenmerkt door een optredende herhaling in de expressies tijdens de opbouw van een volgorde. Bij de normale en in lus eindigende volgorden kunnen na afloop van een volgorde in de buffers van het model nog berichten aanwezig zijn. Deze berichten (residueen) worden gemeld. De analyse moet opnieuw opgestart worden, waarbij de residueen vooraf in de buffers geplaatst worden.

Het aantal gegenereerde volgorden wordt met een selectietechniek op basis van equivalenties gereduceerd [106][47]. Alle overgebleven volgorden worden uitgevoerd, en worden naar kenmerk gescheiden in files geschreven (Appendix J). Ook onaangeraakte delen van het PSL-model worden aangegeven.

Alle resultaten van de analyse worden omgezet naar een redelijk interpreteerbare vorm (Charts). Deze resultaten moeten vervolgens handmatig geinterpreteerd worden. In hoofdstuk 5 van appendix J worden enige wenken gegeven hoe dit te doen.

Met PANDORA zijn de eigenschappen 1 (afwezigheid van deadlocks) (gedeeltelijk) en 3 (afwezigheid ongewenst herhaald gedrag) van hoofdstuk 6.2 te controleren. Ook eigenschap 2 (bestendigheid) wordt gedeeltelijk (delen van het gedrag die nooit vertoont worden), en onder bepaalde voorwaarden volledig *) gecontroleerd. Met enige moeite (Appendix J, hoofdstuk 5.1.3) is eigenschap 1 ook volledig te valideren.

Voor de huidige PANDORA implementatie gelden enige beperkingen voor de grootte van het model [107]. De beperkingen zijn opgelegd met het oog op de geringe kracht van de doelmachine (PDP 11/24).

---

*) Als het model dusdanig opgesteld wordt dat na het doorlopen van een proces dit proces weer in het begin zijn specificatie is, dan betekent dit dat alle delen van het proces bereikbaar blijven als er geen deadlock

DEEL II

VALIDATIE LAAG 2 EN 3 VAN COMMON CHANNEL SIGNALLING SYSTEM NO. 7 MET PANDORA

Voor het opbouwen en verbreken van een verbinding tussen twee telefoonabonnee's is een uitwisseling van gegevens nodig tussen a) de telefoonabonnee's en de telefooncentrale(s) en b) tussen de telefooncentrales onderling. Deze uitwisseling van gegevens noemt men signalering. Signalering wordt beschreven in een signaleringssysteem. In een signaleringssysteem worden protocollen gedefinieerd, protocollen die speciaal ontworpen zijn voor de communicatie tussen twee telefooncentrales.

Daar telefoonverbindingen wereldwijd zijn en via centrales van verschillend fabrikaat lopen zijn op internationaal niveau (CCITT) een aantal signaleringssystemen gestandaardiseerd. De CCITT stelt hiertoe Recommendations (aanbevelingen) op. De ontwikkeling van een signaleringssysteem neemt vele jaren in beslag. Elke vier jaar (een studie periode) worden boeken uitgebracht waarin de bijgewerkte of verder ontwikkelde aanbevelingen worden beschreven.

De CCITT is momenteel bezig met de ontwikkeling van signaleringssysteem nummer 7.

8. C7 - MESSAGE TRANSFER PART

8.1 Inleiding

CCITT Signalling System No. 7 (C7) is een gemeneweg signaleringssysteem. Hiermee wordt bedoeld dat de informatieuitwisseling tussen centrales bij de opbouw van een telefoonverbinding niet door de (variabele) spreekweg plaatsvindt, maar door een vast, voor meer spreekwegen geldend, signaleringskanaal. De signalering is geheel gescheiden van de spraak. Gemeneweg-signalerings (Common channel) systemen zijn speciaal ontworpen voor processor gestuurde telefooncentrales. Gemeneweg signaleringssystemen kunnen goed vergeleken worden met gewone datacommunicatiesystemen.

C7 is modulair opgezet opdat het met de veranderende techniek en behoeftes mee kan evolueren. Het kent als basis een "Message Transfer Part" (MTP) welke dient als communicatiemedium voor verschillende "User Parts" (UP). De UP's zijn bedoeld voor specifieke toepassingen en maken gebruik van de diensten van het MTP (fig 8.1).

Het MTP is, "vergelijkbaar" met de onderste lagen van het OSI-model, opgebouwd uit drie lagen.

	C7		OSI
laag 1	: Signalling Data Link Layer	-	physical layer
laag 2	: Signalling Link Layer	-	data link layer
laag 3	: Signalling Network Layer	-	'network layer'

De diensten die laag 1 en 2 bieden zijn gelijk aan de diensten die laag 1 en 2 in het OSI model bieden (H 4.4), de functionele opsplitsing is daar gelijk.

8.2 Specificatie MTP

De specificatie van de lagen 1 tm 3 is gegeven in de Recommendations Q702, Q703 en Q704, terwijl in Q701 het MTP algemeen functioneel wordt beschreven.

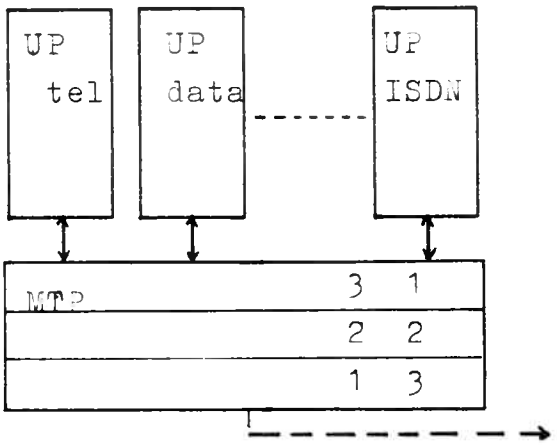


fig. 8.1, Modulaire opbouw, MTP en UP's

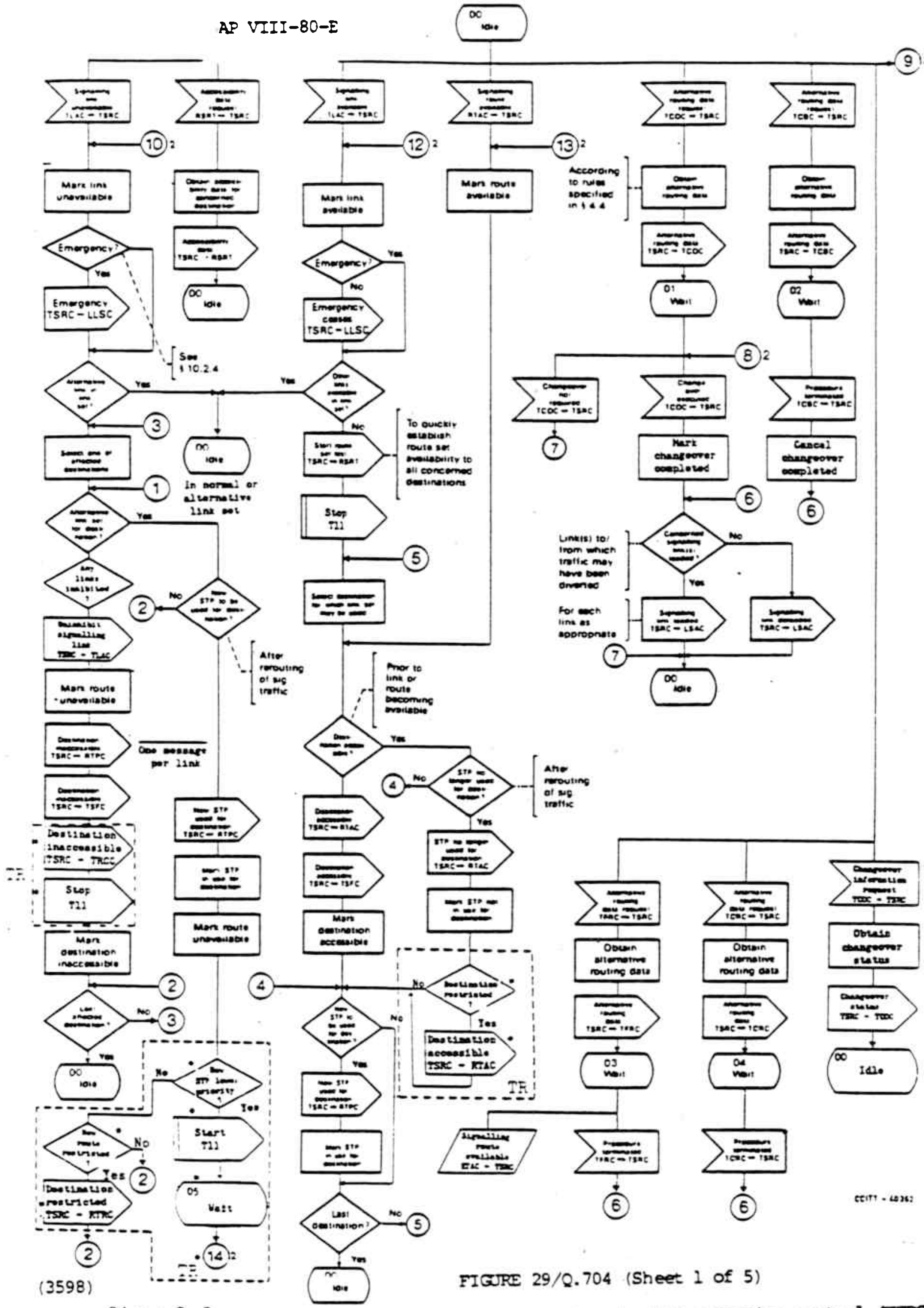
In 1984 is bij de CCITT de studieperiode '80-'84 afgesloten. De nieuwste aanbevelingen zullen in de zogenaamde 'Red Books' verschijnen ook wat betreft C7 (De nieuwe aanbevelingen na afloop van een studieperiode worden altijd in een anders gekleurde kaft uitgegeven, in 1980 was deze geel, in 1984 is het rood).

Alle specificaties zijn in tekst gesteld. Bij Q702 tm Q704 zijn aan de tekst SDL diagrammen toegevoegd, deze zijn echter niet bindend. Hierdoor houden de specificaties een informeel karakter (H 5). Laag 2 beslaat 28 paginas tekst en 30 pagina's SDL diagrammen, laag 3 beslaat 90 pagina's tekst en ong. 60 pagina's SDL diagrammen.

In de SDL specificaties van laag 2 worden verschillende parallel werkende processen gespecificeerd (Bijlage I), terwijl in laag 3 de processen verder opgedeeld zijn door middel van subsystemen (Bijlage V) (H 5.4). Aan ieder proces worden een of meer pagina's SDL diagrammen gewijd. Laag 2 kent 10 processen, laag 3 kent er 20. In figuur 8.2 is een voorbeeld van een bladzijde van een proces bestaande uit meer sheets afgebeeld.

Door het gebruik van meer dan een bladzijde is men gedwongen om connectoren te gebruiken. In figuur 8.3 is te zien dat dit niet altijd even zorgvuldig gedaan wordt, de connectoren worden hierin ook onnodig vaak op de enkele bladzijde zelf gebruikt.

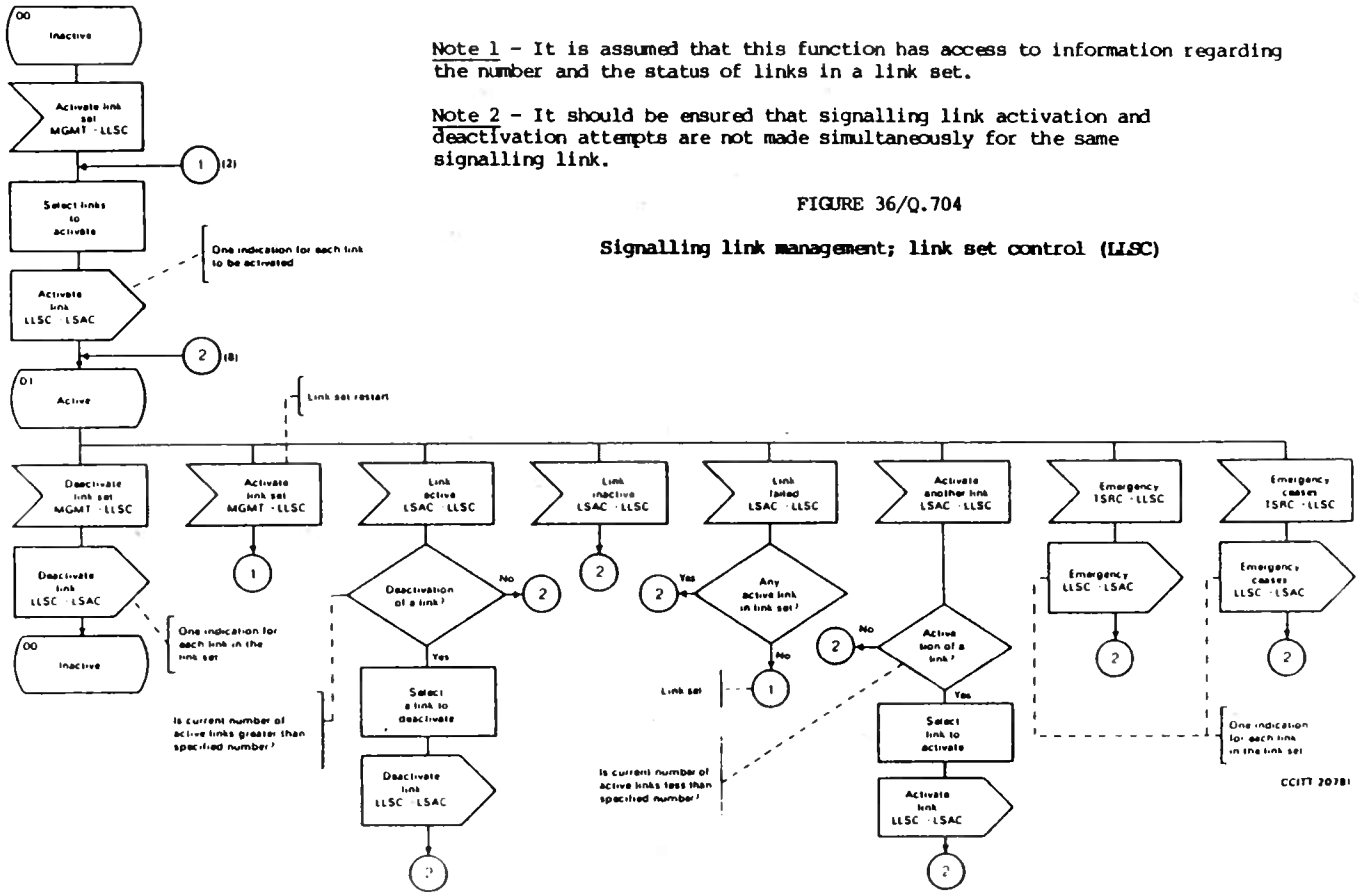
AP VIII-80-E



(3598)

FIGURE 29/Q.704 (Sheet 1 of 5)

fig. 8.2 Signalling traffic management; signalling routing control (TSRC)



Note 1 - It is assumed that this function has access to information regarding the number and the status of links in a link set.

Note 2 - It should be ensured that signalling link activation and deactivation attempts are not made simultaneously for the same signalling link.

FIGURE 36/Q.704

Signalling link management; link set control (LLSC)

fig. 8.3

DEEL II

9. MODELLEERMETHODIEK

9.1 Inleiding

Het blijkt niet eenvoudig te zijn om op basis van interpretatie direct PSL-modellen uit de tekst en SDL diagrammen te maken. Er worden dan vrijwel alleen fouten gevonden die het gevolg zijn van slecht modelleren.

Er is daarom getracht de SDL-specificaties met een bepaalde methodiek om te zetten in een PSL-model. Voor de omzetting moeten een aantal belangrijke handelingen verricht worden.

- De Extended Finite State Machine die met SDL beschreven wordt moet omgezet worden in een Finite State Machine met ingangsbuffers (FSMIB). Daarbij kunnen slechts zeer beperkt variabelen in toestanden worden omgezet, vanwege de dan optredende toestandexplosie.

- Door middel van projectie en abstractie moet het aantal processen van de specificatie op een gering aantal processen in het model afgebeeld worden.

Bovenstaande handelingen worden parallel verricht. De variabelen zijn immers ook deel van de abstractie of de te projecteren functie. Bovenstaande handelingen resulteren in een Finite State Machine Model. Dit model wordt beschreven in PSL.

9.2 Aanvullingen bij de SDL-specificatie

Om de vele SDL-processen op enkele FSMIB-processen af te beelden zijn zeer vele handelingen noodzakelijk. Iedere handeling moet daarbij compleet en volledig zijn om de waarde van de validatie niet te doen afnemen. Naast de SDL-diagrammen is daarbij gebruik gemaakt van enige tabellen welke relevante gegevens van de verschillende processen op een overzichtelijke manier weergeven.

De volgende tabellen worden voor ieder proces opgesteld:

1. Toestanden
   Alle toestanden die het proces kan aannemen worden op een rijtje gezet met de eventueel bijbehorende nummers. Onder toestanden worden hier de onder SDL gedefinieerde toestanden verstaan.
2. Inkomende berichten
   met de acceptatietoestanden en de oorsprong;

bericht	van	in toestand 0 1 2 3 4 5	opmerkingen
hallo	piet	X X	

Eventueel kan ook aangegeven worden wat voor toestandstransitie het binnenkomende bericht veroorzaakt. De tabel ziet er dan als volgt uit;

bericht	van	in toestand 0 1 2 3 4 5	opmerkingen
hallo	piet	2 4	

3. Uitgaande berichten
   met de generatietoestanden en de bestemming;

bericht	naar	in toestand 0 1 2 3 4 5	opmerkingen
hallo	jan	X X	

Ook hier kan eventueel aangegeven worden met wat voor toestandsovergang het uitzenden van het bericht gepaard gaat.

bericht	naar	in toestand 0 1 2 3 4 5	opmerkingen
hallo	jan	1 1	

De opmerkingen zijn bedoeld voor opvallende kenmerken van de berichten. Later in het modelleer proces wordt de ruimte ook gebruikt om aan te geven dat het bericht vervalt of er iets aan gewijzigd wordt, bijvoorbeeld de bestemming.

4. Variabelen
   Bij variabelen wordt aangegeven van welk type ze zijn; boolean, binary, integer (+bereik) of real (+bereik). Bij boolean variabelen wordt verder aangegeven of de waarde geschakeld wordt in de verschillende toestanden (alleen aan [mark] alleen uit [cancel] of beide). Verder wordt getracht aan te geven wat de waarde van de variabele is bij ingaan van de toestand en of de variabele getest (= gebruikt) wordt in die toestand. Daarmee ziet een overzicht van de boolean variabelen er als volgt uit;

Variabele

toestand	evt geschakeld	bij ingaan	getest
1	on/off	off	X
2	off	on/off	
3	on	off	X

Binaire variabelen worden op dezelfde manier beschreven. Integer variabelen zijn moeilijker te interpreteren. Zodra er integer variabelen zijn zullen er dusdanig veel mogelijkheden zijn dat een overzicht niet meer mogelijk is. Bij integer variabelen wordt dan ook enkel de functie aangegeven, later worden besluiten genomen over hoe de variabele gemodelleerd wordt.

5. Timers

worden aangegeven. Daarbij wordt gezegd in wat voor toestand ze kunnen binnenkomen (= aflopen) en in welke toestanden ze gestart/gestopt worden.

		toestand								
		1	2	3	4	5	6	7	8	9
Timer 1	start	X			X		X	X	X	
	stop		X		X	X				
	binnen			X	X	X	X	X	X	

De tabellen zijn bijvoorbeeld bij het weghalen van berichten bijzonder praktisch. Als een berichtontvangst in een proces niet van belang is voor de gemaakte SDL Spec. -> PSL-Model afbeelding, dan kan deze weggehaald worden. Met tabel 3 van het zendende proces kan direct gevonden worden waar in het proces het bericht verzonden wordt. Dit scheelt veel zoekwerk in de SDL-diagrammen en voorkomt dat er berichten worden overgeslagen.

9.3 Afbeelding

De afbeelding op een aanvaardbaar model is een handeling die erg veel creativiteit vraagt. Het is niet mogelijk van het afbeelden een standaard recept te geven.

9.3.1 Projectie en abstractie

De SDL-diagrammen in deze specificatie staan op een laag abstractie niveau. Het eenvoudigste is om dit lage abstractie niveau even aan te houden. Om tot een kleiner model te komen moet dan een functie of deel uit het model geprojecteerd worden. De projectie moet tot een bevatbaar model leiden, iets waarmee bij de keuze van de projecties rekening gehouden moet worden. Een projectie voor laag 2 kan bijvoorbeeld betrekking hebben op initialisatie, of sequence control of flow control etc.

Bij de uitvoering van de projectie worden door elkaar heen de volgende handelingen verricht.

- Processen die voor de projectie niet van belang zijn worden weggehaald. Alle berichten van en naar die processen worden weggehaald of omgezet in andere berichten naar andere processen. v.b. Een proces dat niet van belang is voor de projectie kan wel diensten voor een proces verrichten. Door deze diensten, welke slechts een klein deel van het weg te halen proces vormen, direct uit te voeren kan het proces weggehaald worden. In figuur 9.1 kan proces B weggehaald worden door het bericht MESSAGE direct door proces A te laten verzenden.

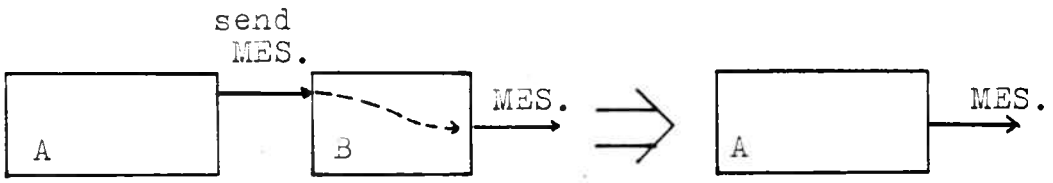


fig. 9.1 Diensten verlenend proces

- Berichten welke niet van belang zijn voor de projectie worden overal weggehaald.

Het model dat overblijft kan evengoed nog veel processen bevatten. Om een kleiner model te verkrijgen wordt de projectie vanuit een hoger abstractie niveau bekeken. Hierbij kunnen een aantal van de volgende handelingen verricht worden.

- Er kunnen processen zijn die een zeer gelimiteerde functie hebben. Er zijn in laag 2 van C7 bijvoorbeeld een aantal processen welke bepaalde fouten in de berichtenstroom van het communicatie medium controleren (Bijlage A, de processen SUERM, AERM, DAEDR). Deze processen genereren in geval van fouten een bericht. De berichten generatie is afhankelijk van fouten, heeft dus een stochastisch karakter.
Deze berichten komen het proces op een willekeurig ogenblik binnen. Het bericht kan ook daarom vervangen worden door een timeout in het proces zelf. Een timeout kan in het PANDORA model immers op ieder ogenblik binnenkomen. Het stochastisch reagerende proces kan dan weggehaald worden.

- Er kunnen in de processen toestanden weggehaald worden. Bepaalde toestanden geven een bepaalde fase in de procesgang aan

welke niet van belang hoeft te zijn voor de projectie. Hierbij zullen ook 'idle' of 'Out Of Service' toestanden zijn. Deze toestanden worden weggehaald.

Alle handelingen die verricht worden voor de afbeelding worden, om het overzicht te bewaren, nauwkeurig in de tabellen bijgehouden (door bepaalde rijen/kolommen weg te strepen) en gedocumenteerd. Hierdoor kan later opgezocht worden waarom bepaalde handelingen verricht zijn, en wat de invloed op het model zal zijn.

9.3.2 Eliminatie van variabelen

Tot slot moeten de variabelen weggehaald worden. Kenmerk van variabelen is dat ze in taken veranderd worden en in beslissingen getest (fig 9.2). Er zijn boolean en integer variabelen.

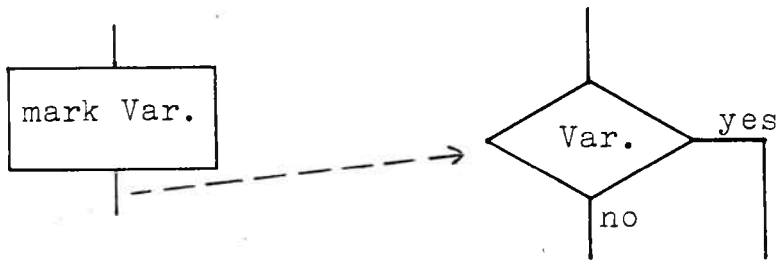


fig. 9.2 Variabelen in SDL: taken en beslissingen

Taken kunnen in het PSL-model niet beschreven worden.
Beslissingen (if.. ..fi) zijn altijd ongeconditioneerd.
Variabelen kunnen op twee manieren geelimineerd worden.
1. Door ze om te zetten in toestanden van de beschreven FSMIB.
2. Door ze gewoon weg te halen en de beslissingen niet, of ongeconditioneerd te nemen.

Ad.1.
Eigenlijk komen alleen boolean variabelen in aanmerking voor omzetting in toestanden. Een boolean variabele heeft twee waarden, kan dus het aantal toestanden hoogstens verdubbelen (H5.3.1). Dit kan nog acceptabel zijn, een integer variabele met een bereik n kan het aantal toestanden met 2**n vermenigvuldigen (zeker niet acceptabel).
De omzetting van een boolean variabele in toestanden kan als volgt voorgesteld worden (fig 9.3).
Bij ieder verandering van de variabele wordt naar de andere specificatie gesprongen.

Ad.2.
Hierbij wordt de variabele vanuit een hoger abstractieniveau beschouwd; bij een beslissing kunnen bepaalde variabelen ten alle

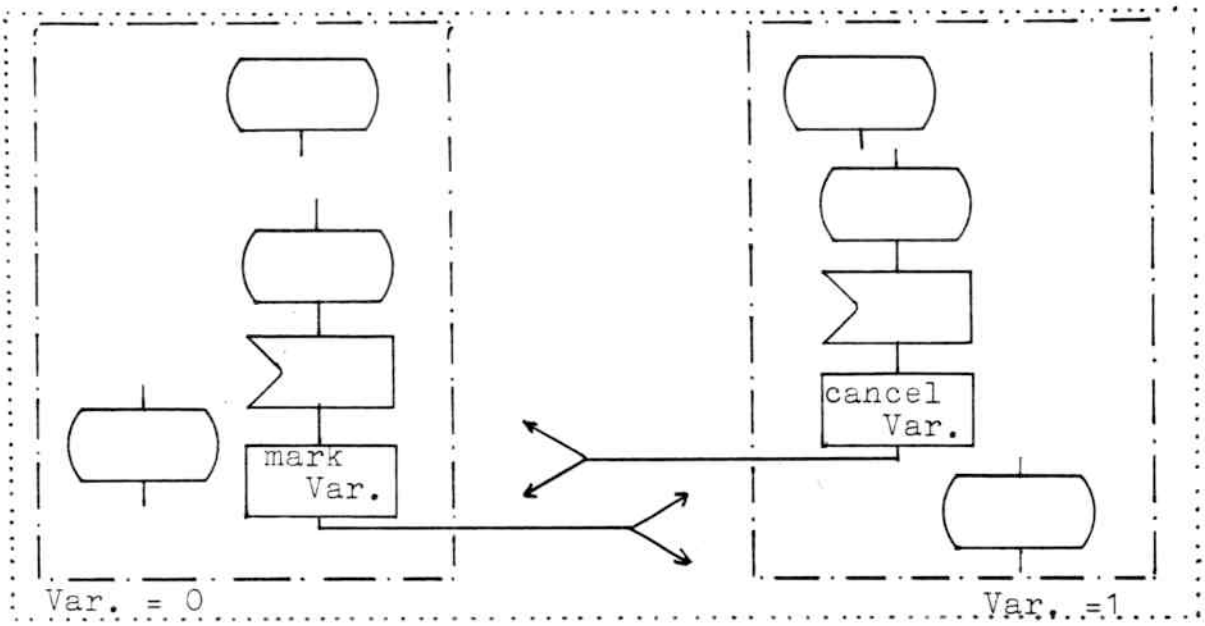


fig 9.3 Boolean variabele omzetten in toestanden

tijde zowel wel als niet aan een conditie voldoen. Op zich levert deze methode een eenvoudiger model, het vergt echter veel meer interpretatie van degene die aan 't valideren is. De plaats van de beslissing heeft immers een bepaalde betekenis, een bepaalde semantische waarde. Deze kan niet altijd zo makkelijk over het hoofd gezien worden. Het betekent dat de resultaten van de analyse door de gedeeltelijk semantische onjuistheid van het model handmatig op juiste semantiek in volgorde gecontroleerd moeten worden. Bij grote specificaties zal dit teveel energie vergen.

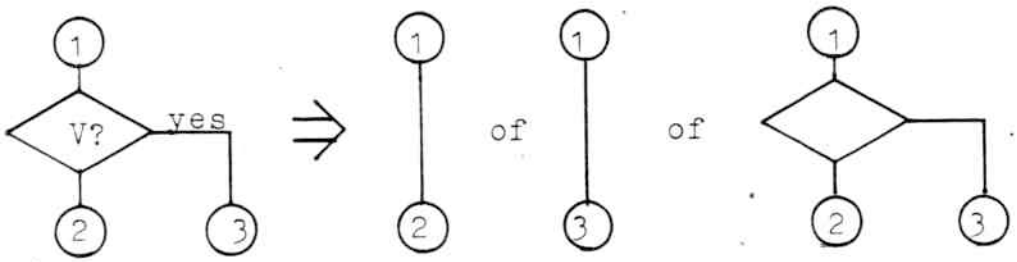


fig. 9.4 Geconditioneerde keuze modelleren

Er moet gekozen worden of een beslissing op grond van de variabele van te voren ingevuld moeten worden (dus geen beslissing, de beslissing is vooraf bepaald) of ongeconditioneerd (dus willekeurig) moet plaatsvinden (fig 9.4).

9.4 PSL-like SDL

De afbeelding van de vorige paragraaf resulteert in een FSMIB met enige extra features als timeouts en ongeconditioneerde beslissingen [103]. De processen zijn te beschrijven met een aangepaste subset van SDL welke ik PSL-like SDL heb genoemd. Dit wordt gedaan om de resultaten van de analyse gemakkelijker te kunnen interpreteren.

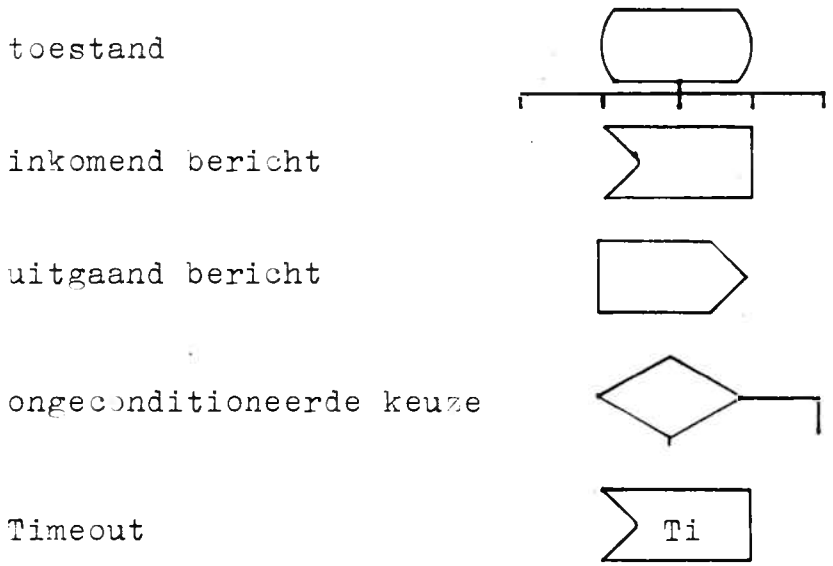


fig 9.5 PSL-like SDL

In figuur 9.5 zijn de in PSL-like SDL gebruikte symbolen weergegeven. Verder gelden dezelfde constructie regels als in SDL.

9.5 Omzetregels PSL-like SDL naar PSL

Het is zeer eenvoudig om het met PSL-like SDL beschreven model om te zetten in PSL.

9.5.1 Toestanden

Een toestand in SDL kan gezien worden als een vast punt in de procesdoorgang, waarvandaan actie ondernomen wordt bij het binnenkomen van bepaalde invoer. Dit vaste punt wordt nog eens extra geaccentueerd door de aanwezigheid van de SDL default-clausule. In een toestand in een SDL-proces wordt ieder niet gespecificeerd ontvangen bericht uit de ingangsbuffer genomen en weggegooid, zonder dat er verdere acties worden ondernomen.
Om deze reden is er voor de definitie van een toestand gekozen voor het do-od statement. Een toestand gaat er hiermee als volgt uitzien.

```
;
/* toestand begroeting */
 do
 :: sien?dagot -> sien!dagsien /* blijf in deze toestand */
 :: sien?hoezo -> sien!niks ; goto stverbazing
 :: sien?hallo -> sien!vraag ; break
                          /* ga naar opvolgende toestand */
 :: default -> skip
 od;
```

Met de default event worden alle overige berichten naast de gespecificeerde beschreven (niet allemaal [101][107]).

Meestal hebben toestanden een naam. Als ergens vanuit een willekeurig deel van het proces naar een toestand gesprongen wordt, dient deze naam tevens als label. Deze moet dan vooraf aan het do-od statement gegeven te worden.
Als niet naar de toestand gesprongen wordt en de toestand slechts in een sequentiele procesdoorgang voorkomt dan is een comment met de toestandnaam reeds voldoende. In de specificatie is de toestand met een comment aangegeven, in de specificatie van 9.5.3 is de toestand met een label aangegeven..

9.5.2 Inkomende en uitgaande berichten

Inkomende berichten, uitgaande berichten en timeouts zijn elementen van PSL en behoeven geen nadere uitleg [107].

9.5.3 Ongeconditioneerde beslissingen

Ongeconditioneerde beslissingen worden afgebeeld op het if-fi statement.

```
    ;
stbegroeting :
  do
  :: sien?hoezo -> sien!niks ; goto stverbazing
  :: sien?dagot ;
          if
          :: sien!vraag ; goto stantwoord
          :: sien!dagsien
          fi
  od;
```

In het voorbeeld kunnen na het ontvangen van dagot twee dingen gebeuren. Hij stelt direct een vraag en gaat vervolgens naar de toestand antwoord, of hij groet (nogmaals) terug met dagsien.
Door de keuze van het if-fi statement zullen er in een if-fi statement alleen maar berichten verzonden worden, en in een do-od statement alleen maar berichten ontvangen. Dit is overzichtelijk.

9.6 Optimaal gebruik PSL

Het is ook mogelijk om van de specifieke eigenschappen van PSL in PANDORA specificatie gebruik te maken. In PSL kan veel beter gestructureerd worden dan in SDL (H 7). Een SDL specificatie vol met goto's kan in PSL met meer moderne structurering technieken als nesting, veel netter beschreven worden. Dit is gevolg van het feit dat PSL over toestanden heen kan kijken (extentioneel), niet vastgebakken zit aan toestand constructies.
Het valt echter niet mee om een SDL specificatie in een minimale PSL specificatie om te zetten zonder dat daarbij moeilijkheden optreden. De default-clause van SDL is daar de hoofdzakelijke oorzaak van [101].

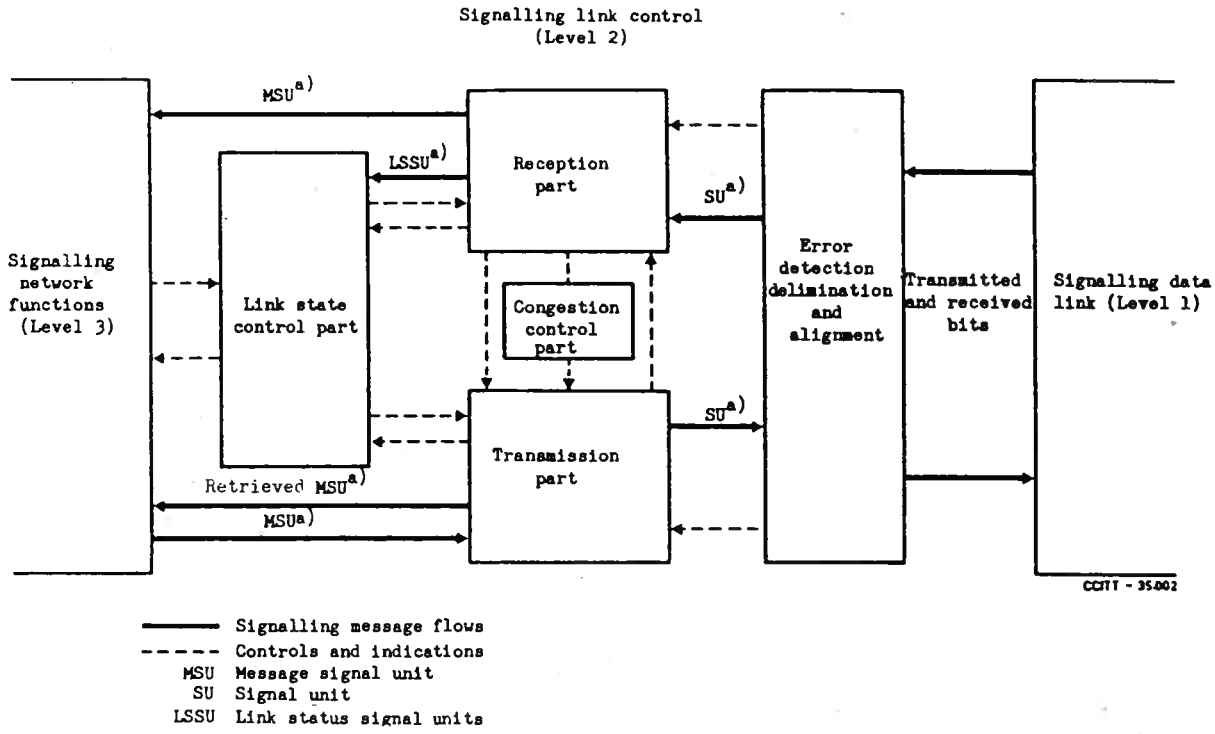
Op het Dr. Neher Laboratorium wordt C 7 geimplementeerd in een STP (Signalling Transfer Point). Voor deze implementatie worden de SDL-diagrammen rechtstreeks omgezet in CHILL. Om zo goed mogelijk de juistheid van de implementatie te testen zijn de opgestelde modellen zo gehouden als in de vorige paragraaf genoemd is. Nadeel van deze methode is dat er in de specificatie nogal wat overhead aanwezig is.

10. VALIDATIE SIGNALLING LINK LAYER

10.1 Inleiding

Er is getracht de signalling-link laag te valideren. De signalling-link laag vervult een aantal functies die, gebruik makend van de fysieke laag, de bovenliggende laag een betrouwbare signallerings verbinding tussen twee signallerings punten biedt.

De opbouw van de signalling-link laag is schematisch voorgesteld in figuur 10.1.



a) These signal units do not include all error control information.

fig 10.1, Blokstructuur van C7-laag 2

De verschillende blokken hebben hierin de volgende functies;

Link State Control Part (LSCP) is verantwoordelijk voor het interne beheer van de laag. Het initieert opbouw procedures en verwerkt foutmeldingen van andere delen van de laag. LSCP is de voornaamste communicatiepartner voor de bovenliggende laag 3, wat betreft de service primitieven.

Transmission part (TP) verwerkt peer-protocol berichten van LSCP (in de vorm van Link State Signalling Units [LSSU]) en berichten van de bovenliggende laag (in de vorm van Message Signalling Units [MSU]) tot verzending van Signalling Units [SU] (packets).

Reception Part (RP) ontvangt dergelijke SU's en scheidt de informatie om de delen vervolgens aan LSCP en laag 3 door te geven.
Samen zijn TP en RP verantwoordelijk voor de sequence control en retransmissie.

Error Detection Delimitation And Alignment stelt de frames in de bitstroom samen. Het voegt de Cyclic Redundancy Check (CRC) code toe aan te verzenden packets, controleert deze code van ontvangen packets, zet vlaggen, verzorgt bit-stuffing etc. Deze functies staan dicht bij het fysieke niveau en worden in de implementatie grotendeels in hardware geimplementeerd.
Congestion Control beheert de doorstroomcapaciteit van laag 2.

10.2 Aandachtsveld van de validatie

De validatie heeft zich voornamelijk gericht op het beheer, of Link State Control Part. De procedures die daarin gevolgd worden wat betreft opbouw, bewaking etc. zijn het meest aantrekkelijk om te valideren. Dit proces zorgt door middel van LSSU's ook voor een groot deel van de peer-communicatie van laag 2.

De overige delen van de laag zijn minder geschikt of ongeschikt voor de validatie.
In EDDAA worden voornamelijk taken verricht (CRC, bitstuffing en foutbewaking) die ongeschikt zijn voor validatie met PANDORA. De validatie zou zich wat deze processen betreft hoogstens op de juiste aansturing van de verschillende processen kunnen toespitsen (is een proces niet in rust als er diensten van verwacht worden).
In RP en TP worden taken voor de rest van de laag verricht (verzenden en ontvangen van LSSU's, verzenden en ontvangen SU's) en wordt een peer-protocol wat betreft de sequence control en foutafhandeling afgehandeld. Dit peer protocol zou eventueel ook geschikt zijn voor validatie. De processen die de peer-procedures van RP en TP afhandelen kennen slechts een actieve toestand en kunnen daarin aldoor dezelfde verzameling berichten ontvangen. Gevonden fouten in het peer-mechanisme zullen daardoor niet in deadlocks tot uiting kunnen komen (6.2). Dit vermindert de noodzaak tot validatie met PANDORA. Bovendien kent

deze peer-procedure erg veel variabelen (de sequence control werkt modulo 128, en is gecombineerd met vlaggen voor retransmissie), iets wat met PANDORA veel problemen zal opleveren.

Congestion control (CC) is nauwelijks onderzocht op mogelijkheden tot validatie met PANDORA. De functie heeft veel te maken met real-time eigenschappen van het protocol en komt als zodanig niet direct voor validatie met PANDORA in aanmerking.

10.3 SDL-proces opbouw van de laag

In figuur 10.2 is de SDL-proces opbouw van de laag weergegeven. Een aantal van de voorgaand besproken blokken zijn daarin direct (onveranderd) terug te herkennen (Reception Control (RC), Transmission Control (TXC) en Congestion Control (CC)). De overige blokken zijn verder onderverdeeld in meer processen.

Zo is Error Detection Delimitation And Alignment onderverdeeld in een ontvangend en een verzendend deel (DAEDR en DAEDT). Het Link State Control Part bestaat uit 5 processen, waarvan:

- twee fouten bewakende/tellende processen
  Alignment Error Rate Monitoring (AERM) en
  Signal Unit Error Rate Monitoring (SUERM),
- een proces welke de Processor Outage Status van de beide kanten van de link bijhoudt. Processor Outage is een status welke aangenomen wordt als in andere delen van C7 storingen optreden.
  Processor Outage Control (POC)
- een proces speciaal bedoeld voor de opbouw van een verbinding
  Initial Alignment Control (IAC)
- en een algemeen beherend proces
  Link State Control (LSC)

In Bijlage I is de procesopbouw volgens SDL nauwkeuriger uitgewerkt weergegeven. De betekenis of functies van de processen zijn gedetailleerd te vinden in de Recommendations.

Voor de validatie zijn de Link State Control Part processen geisoleerd van de overige processen. Daarna was het mogelijk in LSCP zelf AERM en SUERM weg te abstraheren (H 9.3.1). Veel scope-variabelen waren er in de LSCP processen niet aanwezig. Ze waren bovendien van gering belang voor de validatie (bijv. tijdafhankelijke variabelen als "Emergency"), hetgeen de opstelling van een model vergemakkelijkte.

Het model dat op deze manier is opgebouwd is in PSL-like SDL weergegeven in bijlage II. Het model is symmetrisch opgebouwd, er is getracht de peer functies op een reele manier te valideren.

Ook dit geisoleerde model is te groot voor validatie. Daarom is het model verder gesplitst in een deel Opbouw (IAC) functies en een deel Processor Outage (POC) functies.

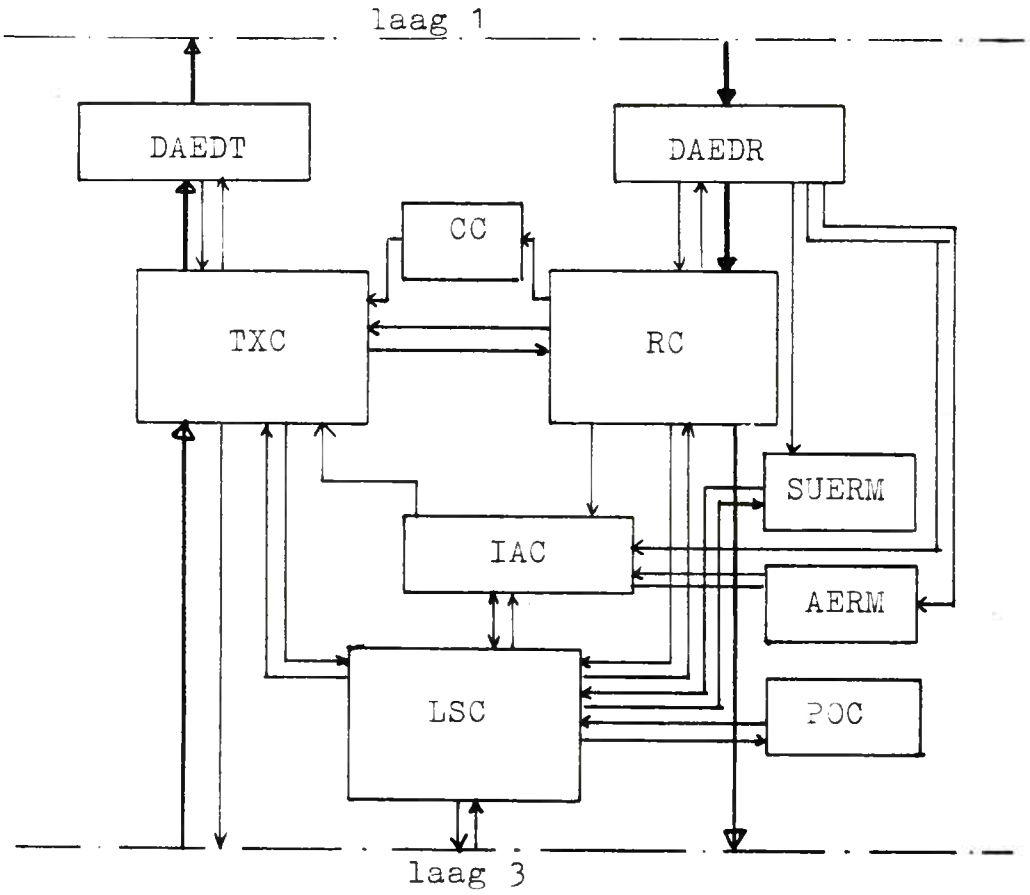


fig 10.2 Proces opbouw laag 2

10.4 Opbouw

Er is getracht de opbouw met behulp van twee soorten modellen te valideren. Het eerste soort was een gesloten model. Het bevatte alleen het proces IAC. In het tweede model werd geprobeerd de opbouw procedure meer bij zijn omgeving te betrekken, een open model, wat betekent dat LSC in het model werd opgenomen.

10.4.1 Gesloten model; IAC

Het model met alleen IAC is zeer eenvoudig, het PSL-model bevatte slechts twee processen (fig 10.3). In dit model werd de opstart procedure (het doorlopen van IAC) een keer doorlopen.
Dit model kan als een gesloten model beschouwd worden en

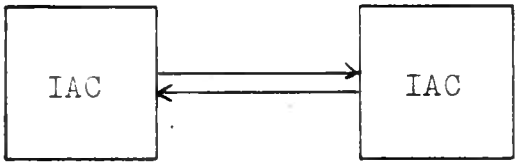


fig 10.3 Enkelvoudig IAC model.

valideert een bepaald mechanisme. Het model is daardoor een gesloten systeem. Dit model is gevalideert met een foutvrij kanaal en met een eenzijdig verstoort kanaal. Er zijn bij deze analyse geen deadlocks gevonden. In bijlage III is de listing van dit model opgenomen.

10.4.2 Open Model; IAC en LSC

Door LSC bij de validatie van de opbouw te betrekken wordt het model meer een open systeem (fig 10.4). LSC heeft redelijk veel relaties met de omgeving (de rest van de laag en laag 3). De invloeden van de omgeving zijn zoveel mogelijk gesimuleerd. Het Processor Outage mechanisme is in dit model geheel weggelaten, daarop betrekking hebbende berichten en toestanden zijn weggehaald.

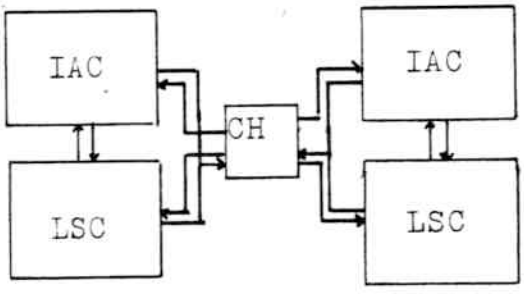


fig 10.4 Meervoudig model

Alle uitwendige factoren zijn getracht mee te nemen, waardoor een redelijk waarheidsgetrouw model is ontstaan.
Voor de analyse zijn meer PSL-modellen opgesteld. Aanvankelijke

modellen bevatten veel redundantie waardoor de analyse te lang duurde of bepaalde grenzen overschreedt (7.4). Bij volgende modellen werd de analyse echter voortijdig afgebroken of waren er andere moeilijkheden, in de vorm van bijv. teveel residueen (die er niet konden zijn vanwege de aard van het gebruikte model (defaults) - [fout van het analyseprogramma]). Een echt geschikt model voor de analyse is niet gevonden.

10.5 Processor Outage Control

Vervolgens is getracht een model op te stellen van LSC samen met POC. Voor dit model werd IAC geabstraheerd naar resultaat; geslaagde of niet geslaagde initialisatie. In figuur 10.5 wordt de opbouw van het model weergegeven. Hierin komen processor outage meldingen van laag 3 in de vorm van timeouts in de LSC's binnen.

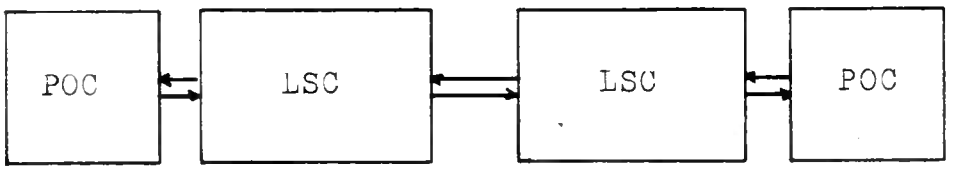


fig 10.5 Opbouw model met POC.

Ook de analyse van dit model verliep niet zonder problemen. Er is weer geexperimenteerd met diverse modellen, de analyse bereikte echter aldoor de maximale lengte van de te analyseren volgordes. In Bijlage IV is een gebruikt PSL-model weergegeven.

Er werd echter wel een fout gevonden. Deze fout betrof de communicatie tussen POC en LSC.

POC dient min of meer als geheugen element voor LSC. In POC wordt de processor outage status (po.) van de beide kanten van de link bijgehouden. Als aan een van beide kanten een Processor Outage optreedt (dit wordt door de betreffende laag 3 gemeld), dan wordt de normale MSU communicatie gestopt. Het Processor Outage Mechanisme werkt als volgt:

Als het LSC-proces aan een kant (LSCa) een po.melding van laag 3 krijgt, dan meldt LSCa dit aan POCa, wordt de normale MSU

communicatie gestopt en worden er SIPO's (Status Indication Processor Outage) naar de andere kant van de link gezonden. LSCa gaat naar de toestand "Processor Outage" ("PO"), POCa gaat naar de toestand "Local Processor Outage" ("LPO").
LSC aan de andere kant (LSCb) stopt na de ontvangst van de SIPO ook de normale MSU-communcatie, gaat invul SU's verzenden (FISU's), en meldt dit aan POCb. LSCb gaat naar de toestand "Processor Outage" ("PO"), POCb gaat naar de toestand "Remote Processor Outage" ("RPO").

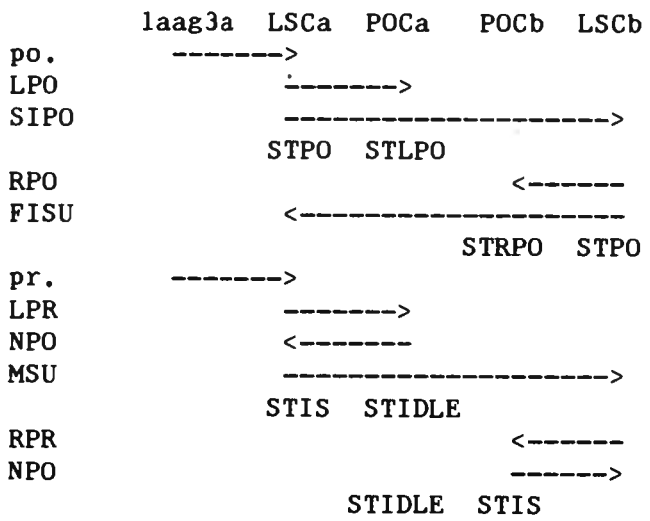


fig 10.6 Processor Outage Mechanisme

Als LSC na enige tijd van laag 3 een melding krijgt dat de po. is opgeheven, dan meldt deze dat alleen aan POC. POC zendt vervolgens een bericht No Processor Outage ('NPO') naar LSC terug, waarna LSC de rest van de laag in het knooppunt er toe zet de communicatie te vervolgen.
Het proberen te vervolgen van de normale communicatie wordt aan de andere kant gedetecteerd, waarna LSCb een bericht 'Remote Processor Recovered' ('RPR') aan POCb stuurt. POCb retourneert LSCb vervolgens een 'No Processor Outage' ('NPO') waarna LSCb de MSU-communicatie ook wat deze kant betreft hervat.
Het is ook mogelijk dat beide kanten in po. zijn, daarvoor kennen de POC's de toestand "Both Processor Out" ("BPO"). Het bestaan van deze mogelijkheid moet het bestaan van POC rechtvaardigen, POC houdt bij welke kanten in po. zijn.

Het volgende kan echter mis gaan. Stel POCa is in de toestand "RPO" en LSCa dus in "PO", LSCb in "PO" en POCb in "LPO". LSCb meld dat de Processor Outage is opgeheven, waarna LSCb de communicatie probeert te hervatten. Net als dit aan de a-kant doorkomt wordt echter aan de a-kant tegelijkertijd een Processor Outage gemeld. Als gevolg van deze toevallige omstandigheid kunnen bij de communicatie tussen LSC en POC de berichten 'NPO' en 'LPO' elkaar kruisen. In figuur 10.7 is weergegeven hoe PANDORA dit detecteert. Deze 'CHARTS' zijn daarbij voorzien van toestandtransities om de leesbaarheid te verhogen (Appendix G).
De 'NPO' wordt door LSC geinterpreteerd als herstel van de net

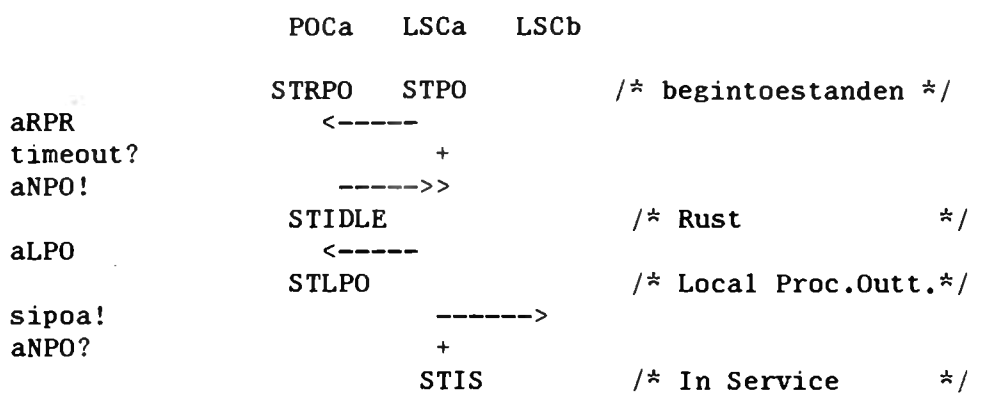


fig 10.7 Optredende Fout in het Processor Outage Machanisme

verzonden 'LPO', waardoor LSC "In Service" komt, hetgeen betekent dat de communicatie, ongewild, gewoon voortgezet wordt. Dit betekent dat in andere lagen er vreemde dingen kunnen gebeuren.

Deze fout kent ook een andere versie. Als de b-kant Processor Outage meldt tegelijkertijd met dat de a-kant, die in "LPO" was, de Processor Outage hersteld, gebeurt hetzelfde. De berichten 'RPO' en 'NPO' kunnen elkaar dan kruisen (fig 10.8).

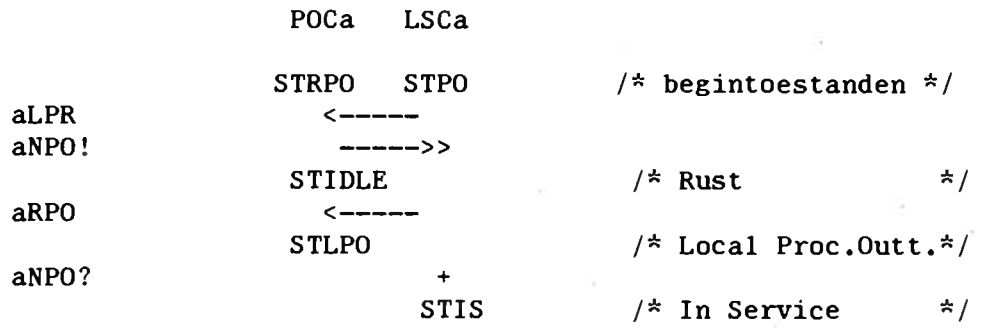


fig 10.8 Andere verschijningsvorm van de POC-fout.

Deze fout zal echter geen rampen veroorzaken in de voortgang van de communicatie. LSCa zal namelijk direct weer merken dat er door de andere kant SIPO's verzonden worden, zal dit aan POC melden en gaat toch weer naar de toestand "Processor Outage".

Dit voorbeeld is illustratief voor hetgeen in Appendix E wordt gezegd. De kans dat bij meer 'slimme' processen een deadlock optreedt neemt af. Fouten worden snel gesignalleerd en hersteld.
Er behoren echter in een protocol dat als 'zeer betrouwbaar' is ontworpen geen fouten te zitten, ook al veroorzaken deze geen deadlock.

Er is ook een model opgesteld waarbij POC geheel in LSC is opgenomen, hiervoor zijn aan LSC enige toestanden toegevoegd. Deze handeling zou men directe koppeling kunnen noemen (H 6.5). In dit model werd (zover analyseerbaar) geen deadlock meer gevonden.

DEEL II

11. VALIDATIE NETWERK LAAG

11.1 Inleiding

De netwerklaag vervult functies gerelateerd aan het transporteren van berichten van een knooppunt naar een bepaald ander knooppunt over een net met meer knooppunten. De netwerklaag moet daartoe functies als routering, verkeersbeheersing, netbeheer ed. vervullen.

De netwerklaag van C7 is ten behoeve van deze functievervulling in twee delen gesplitst:

Signalling Message Handling (SMH)

Signalling Network Management (SNM).

De functie van Signalling Message Handling is er voor te zorgen dat een bericht afkomstig van een bepaald User Part in een bepaald knooppunt (bron) wordt afgeleverd aan eenzelfde User Part van een ander aangegeven knooppunt (bestemming). Dit kan via een of meer links geschieden, er wordt daarbij gebruik gemaakt van aan de berichten meegegeven 'routing labels'. SMH is opgesplitst in drie delen (fig 11.1).

De datastroom is in het Signalling Message Handling deel goed zichtbaar. De functies die in de 3 delen van SMH vervuld worden kunnen daarmee goed zichtbaar gemaakt worden.

Het deel Message Discrimination (MD) detecteert of een bericht dat het knooppunt (via laag 1 en 2) binnenkomt voor het knooppunt zelf of voor een ander knooppunt bestemd is (houden of doorsturen).

Het deel Message Routing (MR) bepaalt de uitgaande signalerings link waarop een bericht zijn bestemming kan bereiken. Dit doet het zowel voor berichten afkomstig van MD als voor berichten afkomstig van de User Part's.

Het deel Message Distribution moet de berichten voor het knooppunt in het juiste User Part afleveren.

De functie van Signalling Network Management is bedoeld voor reconfiguratie van het netwerk in geval van storingen, en om verkeer te regelen in geval van optredende congestie. Een reconfiguratie wordt door middel van een aantal procedures uitgevoerd, met deze procedures kunnen in het geval van

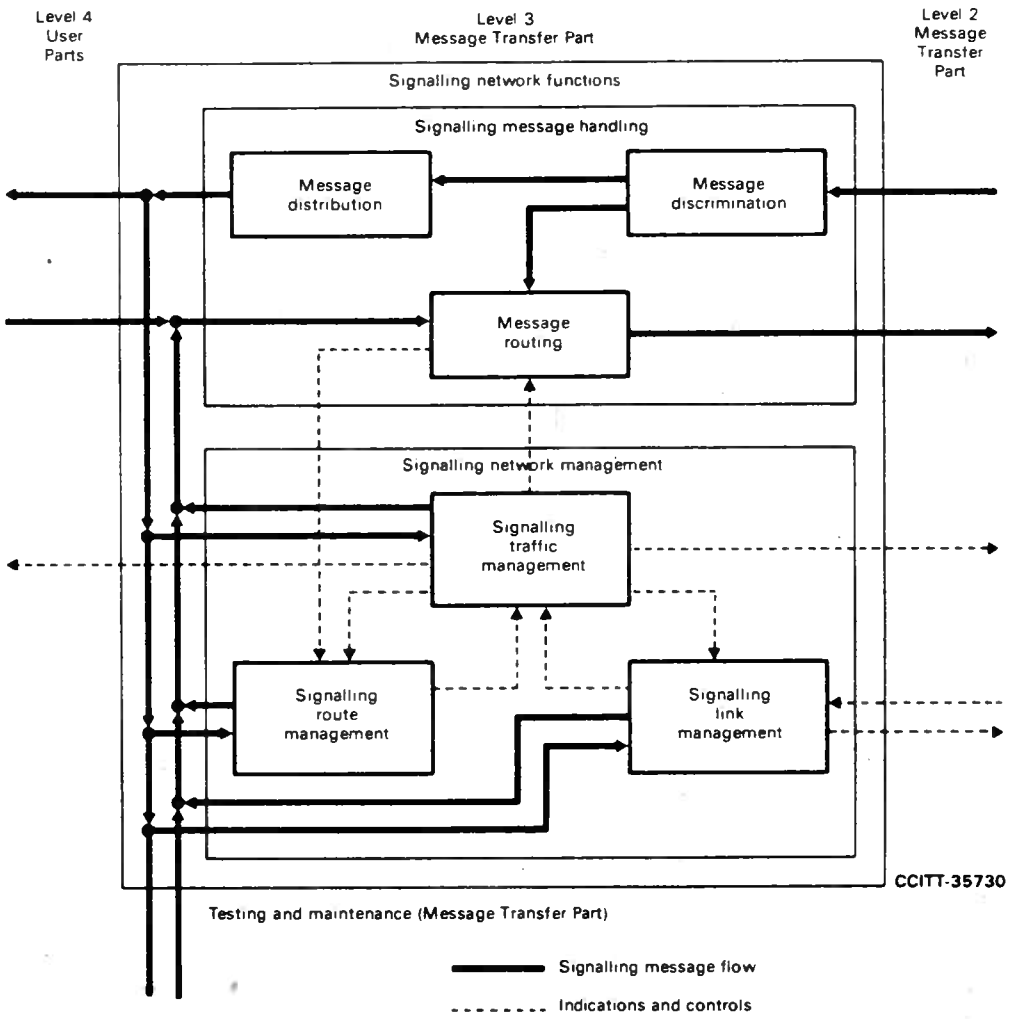


fig 11.1 Signalling Network functies

uitgevallen links omleidingen gemaakt worden. Hiervoor is peer-communicatie (communicatie tussen de knooppunten) noodzakelijk. Deze communicatie betreft het opsporen van fouten en het eventueel opstarten van links. Wanneer een uitgevallen link hersteld is dan moet de link herstart worden, en moet de omgekeerde procedure gevolgd worden om de normale configuratie van het net te krijgen.

In figuur 11.1 is al te zien dat Signalling Network Management ook in drie delen verdeeld is.

- Signalling Traffic Management
- Signalling Link Management
- Signalling Route Management

De delen worden pas geactiveerd als er een gebeurtenis, een storing of een herstel, optreedt. In de figuur is te zien dat veel wederzijdse relaties tussen de delen bestaan.
In elk deel zijn een aantal procedures gedefinieerd.

1. Signalling Traffic Management
   - Changeover
   - Changeback
   - Forced Rerouting
   - Controlled Rerouting
   - Signalling Traffic Flow Control
2. Signalling Link Management
   - Restoration
   - Activation
   - Inactivation
   - enige procedures betreffend de signalling terminal
3. Signalling Route Management
   - Transfer prohibited
   - Transfer allowed
   - Transfer restricted
   - Transferred controlled
   - Signalling route set test
   - Signalling route set congestion test

Dit is veel vergeleken met laag2, in laag 2 zijn slechts enkele van dit soort procedures te vinden (opbouw, congestie, processor outage).

Elk deel (subsysteem) is door middel van enige processen (entiteiten) in SDL gespecificeerd. Deze opbouw is in bijlage V te vinden. De systeemstructuur die er ontstaat is echter nogal chaotisch omdat de communicatie tussen de subsystemen door alle processen tegelijk geschiedt. Hierdoor lijken, als de entiteiten-structuur op zich wordt bekeken, de subsystemen niet meer te zijn als een paar lijntjes die om wat groepjes entiteiten heen zijn getrokken (5.4.1). De relaties tussen de subsystemen zijn erg complex.

11.2 Welke Projectie ?

Alleen het Signalling Network Management deel komt in aanmerking voor validatie, het Signalling Message Handling deel is meer een taakverrichtend deel, de daar verrichte functies liggen meer op het gebied van de uitvoering.
Er is al opgemerkt dat de uitwerking van de systeemstructuur van Signalling Network Management erg chaotisch is, er liggen teveel relaties tussen teveel entiteiten. Het wordt daardoor bijvoorbeeld moeilijk om op het abstractieniveau van de entiteiten functies te projecteren. Als men een projectie

uitvoert moet de omgeving een voorspelbaar gedrag kunnen vertonen (om het eventueel te kunnen simuleren in het model) en moet de projectie een aanvaardbaar formaat hebben. Door de chaotische relaties tussen de entiteiten zal dit namelijk niet lukken.

Doordat het zeer moeilijk is projecties op abstractieniveau van de entiteiten uit te voeren wordt de modelleermethodiek van hoofdstuk 9 vrijwel onbruikbaar. Een validatie zal op een hoger abstractieniveau gedaan moeten worden, vrijwel onafhankelijk van de SDL diagrammen. Daarbij zal de tekst als leidraad moeten dienen, de SDL diagrammen zijn door de ingewikkelde relaties zeer ondoorzichtig, en daardoor vrijwel onbruikbaar. Gevonden fouten hoeven in een model op hoger abstractieniveau nog geen ernstige gevolgen te hebben. Het kan de inzichten in de procedures echter wel vergroten.

Veel tijd is er voor de validatie van laag 3 niet overgebleven. Het is in de beschikbare tijd niet gelukt om modellen op te stellen voor Signalling Network Management.

12. CONCLUSIES

1. Het is gebleken moeilijk te zijn een complex protocol als C7 met PANDORA te valideren. De oorzaken hiervan moeten zowel bij de beperkingen van PANDORA, de mogelijkheden van validatie en de specificatie van C7 gezocht worden.

2. Validatie van complexe protocollen berust (nog) op veel creativiteit. Men kan met de huidige validatietechnieken de aanwezigheid van fouten in een protocol aantonen, niet de afwezigheid. Validatie heeft nog niet het karakter van een correctheids-bewijs.

3. De systeemstructuur van een protocol heeft grote invloed op de bruikbaarheid van validatietechnieken en de kans op het aanwezig zijn van deadlocks.

4. Het is verstandig tijdens het ontwerp van een protocol validaties te doen. Dit zal het ontstaan van een doordachte systeemstructuur stimuleren.
Bij validatie na afloop van het ontwerp kost het erg veel tijd om het protocol nauwkeurig te leren kennen. Er geldt eigenlijk hetzelfde als bij een software ontwerp, de ontwerper zal zijn systeem het beste doorzien. De validatie achteraf biedt wel de gelegenheid een oordeel te vellen over de kwaliteit van het ontwerp (systeemstructuur). Helaas is het erg moeilijk een eenmaal gekozen systeemstructuur bij een internatinonaal gestandaardiseerd protocol te veranderen.

5. Het is moeilijk de analyse van complexe protocollen op PANDORA probleemloos te laten verlopen. Het gedrag van de analyse is onvoorspelbaar en complex. De eigenschappen van de analyse moeten grondiger onderzocht worden voordat opnieuw geprobeerd wordt bestaande complexe protocollen te valideren.

6. De afbeelding van een complex protocol gespecificeerd in SDL op PSL modellen is erg moeilijk. Het is niet altijd mogelijk zinvólle redukties en projecties te vinden.

7. PANDORA kan uitgroeien tot een goede ontwikkelomgeving voor protocollen, het biedt nu al een aardige set gereedschappen waarmee iets eenvoudigere protocollen goed geanalyseerd kunnen worden. Voor die tijd zullen er veel gereedschappen aan PANDORA toegevoegd kunnen worden en kunnen een aantal dingen sterk verbeterd worden.

8. De grote hoeveelheid uitvoer van pan is met enige nieuw toegevoegde en toe te voegen gereedschappen te comprimeren tot handzaam formaat.

9. De analyse van modellen met meer dan twee vermaasde processen is niet volledig.

10. Er zijn enige fouten in PANDORA gevonden. Dit kan betekenen dat er nog meer belangrijke fouten in het systeem aanwezig zijn. Hier moet men op bedacht zijn. Het systeem is nog erg nieuw en experimenteel, relevante fouten zijn niet uitgesloten.

11. In de SDL specificaties van C7 laag 2 is een fout in de communicatie tussen de processen Link State Control en Processor Outage Control gevonden. Geadviseerd wordt het proces Processor Outage Control te verwijderen en de daarin vervulde functie in toestanden of variabelen van proces Link State Control op te nemen.

12. De SDL specificatie van C7 laag 3 is niet doorzichtig. De systeemstructuur is erg complex en de specificatie in SDL te groot. Validatie van laag 3 wordt daardoor erg moeilijk.

13. SDL is een beschrijvingstaal die voor eenvoudige processen en protocollen zeer doorzichtig en duidelijk is. Bij toenemende complexiteit wordt dit echter snel minder.

14. De SDL default clausule voorkomt zeer veel deadlocks.

15. Een beschrijvingstechniek moedigt een bepaald soort gebruik aan. Zo ook SDL.
- In SDL beschreven processen worden snel te klein gekozen. Er wordt (te) snel een parallel proces gespecificeerd. Hierdoor worden seriele systeemstructuren onhandig uitgevoerd. Er wordt in de C7 specificaties geen enkele procedure gedefinieerd. De parallelle processen vormen een bron van fouten en zullen de prestatie van een protocol nadelig beinvloeden. De kans op deadlocks wordt geringer.
- Er wordt snel naar het connector symbool van SDL gegrepen. De 'in' connector van SDL is erg moeilijk van de 'out' connector te onderscheiden. Deze twee factoren tesamen beinvloeden de doorzichtigheid van de specificaties zeer nadelig.

13. EXAMENZITTING

Tijdens de examenzitting zijn vragen gesteld naar aanleiding van een voorlopige versie van dit afstudeerverslag. Voor de definitieve versie zijn de punten van het verslag waarover vragen gerezen zijn (waaronder ook onjuistheden) deels gewijzigt of aangevuld.

Enige vragen en opmerkingen uit de examenzitting komen in onderstaand verslag naar voren.

Vragen J.van Loon:

Vraag: Kun je het verschil tussen de begrippen architectuur en interne structuur wat nader uitleggen?

Antw.: Op het eerste gezicht is dit verschil er niet, beide kunnen een structuur beschrijven. Na enige discussie wordt geconcludeerd dat het begrip architectuur een systeem vanaf de buitenkant beschrijft, terwijl het begrip interne structuur dit vanaf de binnenkant doet.

Vraag: In je verslag zeg je dat het OSI-model opgezet is voor packet-switched netwerken. Is dit wel zo?

Antw.: Ik dacht het wel, voor andere technieken (bijv. circuit-switching) zijn waarschijnlijk andere, beter geschikte architecturen te bedenken. Volgens de overige aanwezigen is dit een misvatting, OSI beschrijft een functioneel model welke voor alle communicatie tussen open systemen geldt. Toegepaste technieken zijn hierbij niet van belang (in definitieve versie verslag gewijzigd).

Vraag: Hoe is het op laag 1 mogelijk dat berichtenvolgordes verwisseld of verdubbeld worden?

Antw.: Vooral als gevolg van echo, eventueel looptijdverschillen. In de daaropvolgende discussie wordt vastgesteld dat het OSI-model fysisch van een FIFO-kanaal uitgaat. Berichten verdubbelingen en verwisselingen kunnen daardoor niet voorkomen (in definitieve versie verslag gewijzigd).

Vragen A.Gierveld

Vraag: Kun je je uiteenzetting over ontwerpen van systemen enigszins toelichten.
Antw.: Tijdens het ontwerp wordt het probleem afgebakend en wordt vervolgens op gerichte wijze in abstractieniveau gedaald, hetgeen tot enige volledige ontwerp-beschrijvingen leidt.
Vraag: In H 4.3 beschrijf je 'globaal' de verdeling tussen de lagen 1,2 en 3 en de lagen 4 tm 7. Is deze naast globaal, ook niet wat onnauwkeurig?
Antw.: De lagen 4 tm 7 beschrijven enige gebruikers (end-to-end) protocollen, terwijl de lagen 1,2 en 3 protocollen gericht op het net beschrijven. Het valt niet mee om in enkele woorden een eenduidig te interpreteren beschrijving van dit verschil te geven.
Vraag: In de conclusies schrijf je dat de analyse van C7-laag3 moeilijk is. Is het niet zo dat de analyse van C7-laag3 onmogelijk is?
Antw.: Dit zou een erg krachtige uitspraak zijn die ik alleen in geval van concrete bewijzen zou willen doen. Misschien is het met de huidige specificaties van C7 inderdaad onmogelijk.

Vragen R.Beukers

Vraag: Volgens jou is het mogelijk met PSL een Finite State Machine beschrijven. Is dit wel zo?
Antw.: Ja, dit is mogelijk, of beter: dit is niet onmogelijk. Men dient daartoe zelf syntactische constructies voor toestanden te bedenken, zoals beschreven kent PSL geen toestanden.
Vraag: Je schrijft in je conclusies dat de ontwerper het beste zelf de valdatie kan uitvoeren. Is dit geen verkeerd belang?
Antw.: Dat is het inderdaad, ja. Ik heb geprobeerd echter de nadruk op validatie tijdens het ontwerp te richten. Validatie tijdens het ontwerp zal doordachte systeemstructuren opleveren. Het zal zeer veel eenvoudiger zijn een goede protocol systeemstructuur alsnog door derden te laten valideren.
Vraag: In H 3.2 beschrijf je dat er 'relevante verschillen moeten kunnen optreden tussen de verschillende implementaties van een functionele specificatie'. Wat bedoel je daar precies mee, is het tegengestelde niet bedoeld?
Antw.: Er is naar te streven een functionele specificatie op een dusdanig abstractieniveau te leggen, dat met bijv. verschillend toegepaste technieken voor de implementatie, geen belangrijk verschillende eigenschappen tussen verschillende implementaties optreden. Kritieke punten daarbij zijn; welke eigenschappen zijn belangrijk en welke verschillen zijn belangrijk. Hiervan is ook het abstractieniveau afhankelijk. Daarnaast is te streven naar een vanzelfsprekende afbeelding van de specificatie op de implementatie, opdat weinig fouten bij het implementeren gemaakt zullen worden.
Vraag: In H 9 schrijf je dat de SDL specificatie op een Finite State Machine afgebeeld moet worden. Een FSM heeft toch geen ingangsbuffer?
Antw.: Dat is waar, het is inderdaad in dit geval onzorgvuldig de term Finite State Machine te gebruiken. Bedoeld wordt

een Finite State Machine met enkele aanvullingen. In de nieuwe versie is deze term gewijzigt in Finite State Machine met ingangsbuffer (FSMIB).

Vragen D.Sparreboom

Vraag: In figuur 10.1 wordt tussen RP en LSCP een LSSU berichtenstroom gedefinieerd. Mist deze berichtenstroom niet tussen TP en LSCP?

Antw.: Op het eerste gezicht mist deze berichtenstroom inderdaad, de figuur is daarin nogal verwarrend. In de nadere uitwerking van het systeem zal echter naar voren komen dat LSCP aan TP opdracht geeft herhaald (elke SU (frame)) LSSU's te verzenden. Deze opdrachtgeving wordt gezien als besturing (gestippelde lijn). Aan de overzijde worden deze herhaalde LSSU's ontvangen door RP. Elke ontvangen LSSU wordt gemeld aan LSCP. Door dit herhaalde karakter wordt deze berichtenstroom gezien als signalerings berichtenstroom (dikke lijn). Hiermee is de figuur toch juist.

Vraag: In H 10.4.2 wordt gesproken over residueen die er niet konden zijn. Hoe kan dit?

Antw.: Dit is achteraf waarschijnlijk het gevolg van een fout in het residu-afleid-algoritme van PANDORA.

Vraag: Is het proces Channel in figuur 10.4 inderdaad gemodelleerd met meervoudige ingangs- en uitgangsbuffers?

Antw.: Neen, hier wordt een proces bedoeld met een ingangs- en een uitgangsbuffer. Achteraf was het misschien beter een proces voor de ene richting en een proces voor de andere richting te nemen. Gescheiden kanalen tussen de IAC's en de LSC's is echter zeker niet juist.

Opmerkingen Prof. de Kroes

Opm.: Het in H 3 gebruikte begrip 'bouwbeschrijving' kan vervangen worden door het meer passende begrip 'bestek' (in definitieve versie toegepast).

Opm.: Waarschijnlijk is het wel mogelijk laag 2 in meerdere lagen op te splitsen. Een laag kan zich rond 'Error Detection Delimitation And Alignment' vormen, en een laag kan zich rond 'Link State Control Part' vormen. Dit zou de validatie kunnen vereenvoudigen.

Opm.: Tijdens het ontwerp (H 5.4) wordt alleen van de hierarchische systeemstructuur afgeweken als dit (economisch) voordeel biedt, de orde wordt zover economisch noodzakelijk doorgevoerd.

LITERATUURLIJST

1 Beukers R.
Kroes de J.L.
Dictaat Elektronische Automatische Telefonie
Uitgave 1984/85 behorend bij college 1100 van de afd. Elektrotechniek, TH-Delft
Bevat onder andere een uitgebreid hoofdstuk wat signalering behandeld

2 Brand D. &
Zafiropulo P.
On communicating Finite-State Machines.
Journal of the ACM
Vol 30, No 2, april 1983

3 Budowski S. &
Najm E.
Structured Finite State Automata
Conf. on protocol specification, testing and verification, held at IBM Zurich, Switserland, 1983
Er wordt een aan meer-proces communicatie aangepaste beschrijvingstechniek voor Finite State Automaten gedefinieerd, waardoor directe koppeling van processen onder bepaalde condities mogelijk wordt.

4 Gustavsson R. &
Pehrson .E
The power of some formal models of ditributed computing.
Proc. 3rd INWG/IFIP Conf. on protocol specification, testing and verification, held at IBM Zurich, Switserland, 1983
Vergelijking van Milner's CCS en Zafiropulo's Finite State Machines in de praktijk.

5 Hennie F.C.
Finite state models for logical machines
Wiley & sons, New York, 1968
Beschrijft theorie FSM.
Reductietechniek voor transitietabellen.

6 Holzmann G.
A theory for protocol validation
IEEE Transactions on Communications
Vol C-31, No 8, Aug 82
De theorie waarop de Pandora analyse is gebaseerd.

7 Lam S.S. &
Shankar A.U.
Protocol verification via projections
IEEE Transactions on Software-Engineering
Vol SE 10, No 7, July 1984
Uitwerking en bewijs van een techniek welke uit de Global-state space functies projecteerd zonder dat we de Global-state

space behoeven te kennen. Er wordt een zich normaal gedragend beeld(image)-protocol uit de originele protocol specificatie geprojecteerd en geminimaliseerd.

8 Nijhof J.A.M.
De architectuur van datanetten
Proc. KIVI/NGI ASI symposium Informatie over communicatie 24 oktober 1984, TH Delft
Netwerkarchitectuur belicht vanuit de systeemleer.

9 Nijhof J.A.M.
Dictaat Telecommunicatienetten
Voorlopige versie 1982/83 Behorend bij college 1102 afd. Elektrotechniek, TH-Delft (College gewijzigd-opgesplitst)
Algemene introductie in de telecommunicatie met het accent op de datacommunicatie.

10 Pehrson B.
Abstraction by structural reduction
Proc. 3rd INWG/IFIP Conf. on protocol specification, testing and verification, held at IBM Zurich, Switserland, 1983
Techniek welke de functionele beschrijvingen van een set verbonden componenten versimpelt. Demonstratie m.b.v. Alt.Bit.Pr.

11 Puzman J. & Porizek P.
Communication control in computer networks Part 4
Wiley & sons, New York, 1980 (Aanwezig op afdelingsbibliotheek Elektrotechniek, TH-Delft)
Heldere beschrijving van de algehele protocol problematiek, inclusief verificatie.

12 Simon G.A. & Kaufman D.J.
An extended Finite State approach to protocol specification
Proc. 2nd INWG/IFIP (ed.Sunshine) Conf. on protocol specification, testing and verification, held in Idylwild, California, USA, 1982

13 Schie van L.N.M.
SDL-Introductie
PTT Telecommunicatie, Centrale Afdeling Telefonie
Doelstellingen, ontwikkelingen, concepten en definities van SDL.

14 Vuong S.T. & Cowan D.D.
A decomposition method for the validation of structured protocols.
Proc. Infocom '82, Las Vegas USA, p 209-220

15 CCITT
Rec. X224 Transport-laag protocol
Specificatie met meerdere
toestand/stimulus tabellen.

16 CCITT
Rec. X225 Sessie-laag protocol
Specificatie met een toestandstimulustabel

17 Matchett E.
Glossary of terms used in Fundamental Design
Method.
Bijlage bij [18]

18 Malotaux P.Ch.A.
Produktontwikkeling; beleid en organisatie
Collegedictaat Bedrijfsleer bb1,
rapport no 125, Onderafdeling Bedrijfsleer
en Industriele Organisatie, TH-Delft

19 Vliet van J.C.
Software Engineering
Stenfert Kroese, Leiden, 1984
Overzicht state of the art van Software
Engineering. Zeer duidelijk en overzichtelijk.
Toekomstgericht.

20 Freeman & Lewis
Software engineering
Proc. of the Softw. Eng. Workshop
New York, 30 mei - 1 juni 1979
Academic Press 1980

21 Honderd G.
Systeemtechniek
collegedictaat, mei 1980
TH-Delft, afdeling elektrotechniek
Inleiding in de systeemleer, uitgewerkt
naar de cybernetica.

22 Pouzin L.
Zimmermann H.
The standard network architecture
deveveloped by ISO.
Data communication and computer networks.
Proc IFIP(TC-6)/CSI Conference on
Networks 80, Bombay, India, 4-6 Feb 1981
edited by S.Ramani, North Hol. pub

23 Balter R.
Decitre P.
Validation et reprise dans le systeme
transactionnel cooperant Scot.
Technique et Science Informatique
februari 1984 (in Frans)

24 Bochmann G.V.
A general transition model for protocols
and communication services
IEEE Trans. on Communications. Vol com. 28
Vol. Com. 28, No 4, April 1980 pp 643-650
Beschrijving transitie model die FSM-
en Programma-verificatie combineert.
Structureringstechnieken. Vb met HDLC en
X 25 virtual circuit service.

25 Bochmann G.V. & Gescei J.
A unified method for the specification and verification of protocols.
Proc. IFIP congr, Toronto, Ont. , Canada 1977 pp 229-234.
Hybride specificatie/verificatie techniek

26 Bochmann G.V. & Merlin P.
On the construction of communication protocols
ICCC No. 5, Oct 27-30 1980
Opsplits techniek voor protocollen

27 Bochmann G.V. & Sunshine C.A.
Formal methods in communications protocol design
IEEE Trans on commun., Vol C-28, april 1980
Goed overzicht van protocol-technieken.

28 Danthine A.A.S.
Protocol representation with finite state models
IEEE Trans on Commun., Vol C-28, april 1980
Sterk gericht op gestoord kanaal, goede uitleg basisprincipes.

29 Rafiq O. & Ansart J.P.
Validoc, a protocol validator and its applications.
Conf. on protocol specification, testing and verification, held at IBM Zurich, Switserland, 1983
Transformatie van spec. in nat. taal in Ext. FSM m.b.v. predicaten. Uitvoering op protocol-expert system VALIDOC.

30 Rudin H.
Automated protocol validation; some praktical examples.
Proc. ICCC No.6, London, 7-10-1982
Enige resultaten van protocolvalidatie met Zafiropulo's techniek. Voorbeelden worden ook gebruikt door Holzmann in [108].

31 Rudin H.
The ICC'83 communication protocols session: An overview.
Proc. ICC '83
Recent overzicht van de stand van techniek van protocol-wetenschappen.

32 Schultz G.D. & Rose D.B. e.a.
Executable description and validation of SNA
IEEE Trans. on commun., Vol C-28, april 1980
Beschrijft expert systeem rond het SNA (vegl.OSI)model. Genereert ook implementatie.

33 Sherman M.
Using automated validation techniques to detect lock-ups in packet switched networks
IEEE Trans on commun. Vol C-30, 7, july '82
Toevoeging van detectie van dynamische deadlocks aan het Zurich FSM model. Test m.b.v. effectiviteit predikaten. Met vb.

34 Sunshine C.A.
Experience with automated protocol verification
Proc. ICC '83, sessie E 4.4
Vergelijking van vier softwareverificatie systemen op geschiktheid voor protocol verificatie.

35 Sunshine C.A. & Thompson D.H. & Erickson R.W. e.a.
Specification and verification of communication protocols in AFFIRM using state transition models.
IEEE Trans. on Softw. Eng., Vol SE-8, No 5
Alg. verif. op service en veiligheid.
Tevens aandacht voor certificatie. AFFIRM is Softw. test systeem van het ISI project.

36 Piatkowski T.F.
Protocol Engineering
Proc. ICC '83, sessie E4.8
Pleidooi voor een nieuw vak.

37 Ollongren A.
Inleiding tot de leer van het programmeren
Syntax en semantiek van programmeertalen
Universitaire Pers Leiden, 1976
Duidelijke inleiding in de theorie van de abstracte automaten.

38 Eversdijk C.H. Thijssen A.P. Vink H.A.
Digitale schakeltechniek
Delftse Universitaire Pers

39 Tanenbaum A.S.
Computer networks
Prentice Hall 1981
Basistheorieen en functies in computer netwerken.

40 Jager W. e.a.
Implosie documenten map
PTT Dr. Neher Laboratorium

41 Merlin P.M.
Specification and validation of protocols
IEEE Trans. on Comm. Vol COM-27, nov '79
Overzicht, enigszins verouderd

42 Hailpern B. & Owicki S.
Verifying network protocols using temporal logic.
Proc. NBS Trends & Application Symp.
29 mei 1980, pp 18-28

43 Milner R.
A calculus for communication systems
Lecture notes in Computer-Science
Springer verlag
Complete beschrijvingstechniek voor protocollen d.m.v. reguliere expressies

44 ISO/TC97/SC21 — LOTOS - A formal description technique
ISO/TC97/WG16-1 ad hoc group on
FDT/ Subgroup C.

45 Verbruggen C.A. & Klesser H.W. — Modelvorming en simulatie
Inleiding, procesmodellen, Testsignalen
Collegedictaat vakgroep Regeltechniek
afd. Elektrotechniek TH-Delft.

46 West C.H. — Applications and limitations of automated protocol validation.
Proc. 2nd INWG/IFIP (ed.Sunshine)
Conf. on protocol specification,
testing and verification, held in
Idylwild, California, USA, 1982
Samenvatting bezigheden IBM Zurich, goede inleiding in hun werkzaamheden. Veel praktische tips. SNA model technieken. Mogelijkheden validatie LAN's.

47 Rubin H.& West C.H. — An improved Protocol Validation Technique
Computer Networks, April 1982
Beschrijft basis van de reductietechniek van PANDORA, en werking validatie techniek van Zurich.

48 Klaassen & van Dam — Bedrijfszekerheidstechniek
collegedictaat college 1115 81/82
vakgroep Elektronische Instrumentatie
Instrumentatie, afd Elektrotechniek
TH-Delft.

49 Yemini Y. & Nounou N. — CUPID: A protocol development environment
Proc. 3rd INWG/IFIP
Conf. on protocol specification,
testing and verification, held in
Zurich, Switserland, 1983
Ontwikkelomgeving voor protocollen
Op dit systeem zijn verschillende transformaties tussen Petri-netten CCS en FSM modellen mogelijk.

50 Sidhu D.P. — Synthesis of communication protocols
Proc. ICC '83 , 4C5

51 Yemini Y. & Kurose J.F. — Can current protocol verification techniques guarantee correctness
Computer networks 1982, No. 6
Pleidooi voor een meer probebalistische aanpak van protocolverificatie. Uitgewerkt aan de hand van Alternating Bit Protocol.

LITERATUUR PANDORA

101 Beukers R. & Sparreboom D. — Experience with a protocol specification and verification system. Proc. 2nd SDL users and Implementors forum 11-15 march 1985, Helsinki

102 Ruiter de O.C. — Verificatie van telefoonsignalering met behulp van PANDORA. Afstudeerverslag vakgroep Automatische Verkeers Systemen, afd. Elektrotechniek TH-Delft. nov. 83.

103 Yeu B.D. — Het verband tussen de protocolmetataal van het PANDORA-systeem en daaruit gegenereerde transitietabellen. Afstudeerverslag vakgroep Automatische Verkeers Systemen, afd. Elektrotechniek TH-Delft. mei 84.

104 Holzmann G.J. — The PANDORA system; An interactive system for the design of data communication protocols Computer Networks, Vol 8, No 2, april 1984 Opzet van PANDORA systeem, ontwikkelingsvisie

105 Holzmann G.J. — Algebraic Validation Methods, a comparision of three techniques Proc 2nd INWG/IFIP Conf. on protocol specification, testing and verification,

106 Holzmann G.J. — The PANDORA system; reduction techniques rep. No. 43, Vakgroep A.V.S. Afdeling Elektrotechniek, TH-Delft

107 Holzmann G.J. — Manual for the PANDORA protocol analyser rep. No. 44, Vakgroep A.V.S. Afdeling Elektrotechniek, TH-Delft

108 Holzmann G.J. — Automated protocol verification on PANDORA; four examples rep. No. 45, Vakgroep A.V.S. Afdeling Elektrotechniek, TH-Delft

109 Holzmann G.J. & Beukers R.A. — The PANDORA protocol development system Proc 3rd INWG/IFIP Conf. on protocol specification, testing and verification, held at I.B.M. Zurich, 1983

Appendix A

Het testen van een implementatie

Helaas is het niet zo makkelijk (zoniet onmogelijk) sluitende correctheidsbewijzen voor de testen van de implementatie te bedenken. De implementatie is concreet, in tegenstelling met de specificatie, welke formeel kan zijn. Men moet de implementatie daardoor altijd van de buitenkant benaderen. De test krijgt daardoor het karakter van een stimulus -> verwachte respons. De stimulus is het testgeval, de respons is erdoor bepaald. Dit kan men echter niet uitputtend doen. Men kan moeilijk om een calculator te testen aan deze calculator alle mogelijke combinaties van getallen en operaties aanbieden om vervolgens te controleren of de berekende antwoorden correct zijn. Dit is volstrekt irreeel, te duur en kost veel te veel tijd. Men moet zich beperken tot een beperkt aantal (zorgvuldig geselecteerde) testgevallen.

Men wordt gedwongen een keuze te maken. Men wil graag dat deze keuze zo goed mogelijk is, dat wil zeggen, tot zoveel mogelijk ontdekte fouten leidt. Het is echter ook nodig, om economische redenen, de verzameling testgevallen zo klein mogelijk te houden.

Het bepalen van een optimale verzameling testgevallen is een zeer kritische stap in het totale testproces. Een bijkomende moeilijkheid hierbij is dat men voor elke geselecteerde invoer ook de bijbehorende verwachte uitvoer moet bepalen.

Globaal gezien onderscheiden we twee methoden voor het afleiden van testgevallen:

1. Black box, of functionele analyse. Hierbij worden, vanuit het ontwerpproces gezien, de testgevallen van boven naar beneden afgeleid, dus vanuit de testparameters. Functionele testgevallen moeten zo vroeg mogelijk in het ontwerpproces gedefinieerd worden. Dit helpt mee het te ontwerpen systeem testbaar te houden (indien uberhaupt testbaar).

2. White box, of strucuturele analyse. Dit is een complementaire methode waarbij men van beneden naar boven testgevallen afleidt, dus vanuit het te testen object (bijvoorbeeld lettend op zwakke schakels).

Goed testen is een destructief proces. Men test met het doel zoveel mogelijk fouten te ontdekken. Een test wordt veelal pas succesvol genoemd als deze tot de ontdekking van een fout leidt. Een test toont de aanwezigheid van fouten aan, niet de afwezigheid.

Veel van de genoemde punten gelden ook voor verificatie. Wat geldt voor testgevallen, geldt veelal ook voor het maken van verificatiemodellen voor complexe protocollen.

Appendix B

Communicatie in OSI

De berichtuitwisseling in het OSI-model kan worden voorgesteld door een laag met zijn omgeving te beschouwen. In elke laag zijn een of meer entiteiten gedefinieerd. Deze entiteiten bieden, zonodig in samenwerking met andere entiteiten van dezelfde orde, diensten aan entiteiten in hogere ordes, in hogere lagen. De samenwerking tussen peer-entiteiten komt tot stand door de uitwisseling van informatie via een logische verbinding (OSI: connection), die wordt gevormd door een paar protocolstations (van lagere orde) samen met het communicatie medium. Op ieder punt waar de gebruiker communiceren wil is van elke laag een entiteit aanwezig, de entiteiten staan in serie.

De berichtuitwisseling via de logische verbinding bestaat uit twee delen:

een (N)-service-data-unit, deze bevat de data die (N)-entiteiten nodig hebben om de functies te kunnen vervullen voor de diensten gevraagd door de (N+1)-entiteiten;

(N)-protocol-control-information, deze bevat de informatie die door de (N)-entiteiten wordt uitgewisseld t.b.v. de onderlinge coordinatie.

De combinatie van een (N)-service-data-unit (SDU) en de (N)-protocol- control-information wordt een (N)-protocol-data-unit (PDU) genoemd (fig 1).

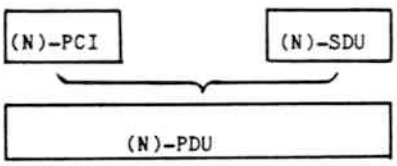


fig 1 Protocol Data Unit

Om de overdracht van de data-units tussen twee (N)-entiteiten mogelijk te maken moet een beroep worden gedaan op de diensten van een (N-1) connection tussen de beide eniteiten. Dit komt tot stand door het verzenden van een (N-1)-Interface-Data-Unit (IDU) via het grensvlak tussen de (N)-layer en de (N-1)-layer. Ook een interface-data-unit bestaat weer uit twee delen:

(N-1)-interface-control-information, deze dient voor de coordinatie van de (N)- en (N-1)-entiteiten;

de (N-1)-interface-data, deze bevat gewoonlijk de hiervoor genoemde (N)-PDU.

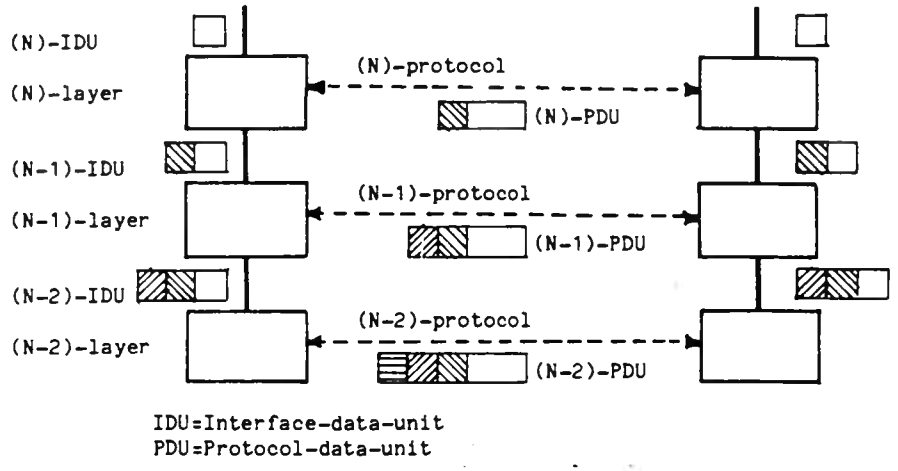


fig 2 Data-units in gelaagde structuur

De logische structuur van het model wordt weerspiegeld in de opbouw van de berichten die worden uitgewisseld tussen de entiteiten (fig 2). Bij de verzending van een bericht wordt aan de ene kant herhaald envelop om envelop gedaan, bij de ontvangst worden deze enveloppes er stuk voor stuk weer afgehaald.

Een laag kan een service van een onderliggende laag opvragen met behulp van service-primitieven. De volgende service primitieven zijn gedefinieerd.

1. Request
2. Indication
3. Response
4. Confirmation

Deze worden als volgt gebruikt: Laag N in systeem A gebruikt een request om en bepaald service-element uit laag (N-1) te activeren. Laag (N-1) in systeem B gebruikt een indication om te melden dat het service-element uit de overeenkomstige laag in A is geactiveerd. Laag N uit systeem B gebruikt dan een response om te reageren op de indication. Uiteindelijk gebruikt laag (N-1) uit A een confirmation om de ontvangst van een request te bevestigen (fig 3).

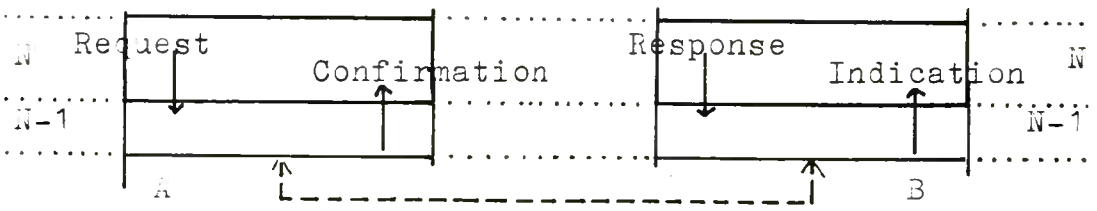


fig 3. Het gebruik van service-primitieven

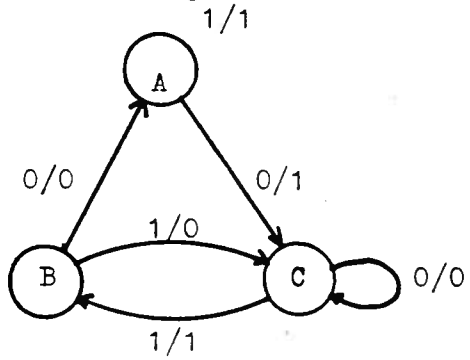
Appendix C

1. De Finite State Machine

Een Finite State Machine (FSM) is een abstract model om een sequentiele machine met een invoer en uitvoer functie te beschrijven. Met dergelijke machines worden synchrone digitale mode schakelingen bedoeld die met 1 kloksignaal werken en een bepaalde geheugenwerking hebben. Hierin zijn drie typen signalen te onderscheiden; ingangssignalen i, uitgangssignalen u en inwendige signalen z. De inwendige signalen geven de toestand aan waarin de schakeling zich bevindt.

Het gedrag kan o.a. beschreven worden in toestand tabellen of toestand diagrammen. Daarbij onderscheidt men puls-mode schakelingen en level-mode schakelingen, waarbij het uitgangssignaal resp. tijdens de overgang van de ene toestand naar de andere gegenereerd wordt (een puls) of gedurende de gehele toestand (een level). In figuur 1 is aangegeven hoe voor de verschillende uitvoeringen de tabellen en diagrammen eruit zien.

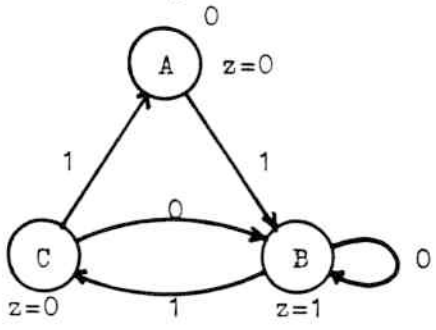
a) Transitie gerichte machine



Present State	Input 0	Input 1
A	C1	A1
B	A0	C0
C	C0	B1

Next State, Output

b) Toestand gerichte machine



Present State	Input 0	Input 1	Output z
A	A	B	0
B	B	C	1
C	B	A	0

Next State

Fig. 1 Representatie van een Finite State Machine

Alle inwendige toestanden dienen stabiel te zijn en voor een praktische bruikbaarheid van de schakeling worden daarnaast

deterministisch bepaalde toestanden geeist en worden overgangstoestanden niet meegeteld (dit gebeurt ook niet in het model). Als er met digitale codes gewerkt wordt, zijn er bij n signalen 2**n combinaties. De inwendige signalen (toestanden) worden door de inwendige toestanden bepaald. Soms zijn bepaalde ingangscombinaties in bepaalde inwendige toestanden uitgesloten; deze combinaties heten 'don't happen'. Naar gelang men de schakeling veiliger wil maken (i.v.m. storingen) kan in de schakeling hiervoor een 'don't care' of een 'don't use' geimplementeerd worden. Bij een 'don't use' blijft de schakeling in dezelfde toestand bij het optreden van de 'don't happen' ingangscombinatie, bij een 'don't care' kan de schakeling naar een andere toestand springen, iets wat zeker niet gewenst is.
Een Finite State Machine is een geordend vijftal

M = (I, U, S, g, y)

waarin

I, de verzameling ingangscombinaties (niet leeg), het ingangsalfabet;
U, de verzameling uitgangscombinaties (niet leeg), het uitgangsalfabet;
S, de verzameling inwendige toestanden (aftelbaar, niet leeg);
g, de toestandsfunctie, een afbeelding van S x I in S;
y, de uitgangsfunctie, een afbeelding van S x I op U.

Voor de interpretatie van het mathematisch model zijn twee mogelijkheden, zie figuur 2.

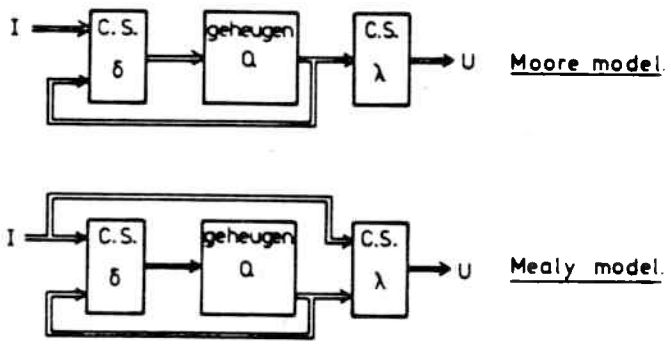


Fig. 2. Moore en Mealy model.

Het Moore model is voor puls-mode schakelingen bedoeld, d.w.z. transitie gerichte machines (is echter niet noodzakelijk), en het Mealy model voor level-mode schakelingen , d.w.z. toestand gerichte machines.

2. Reduceerbaarheid van Finite State Machines

De toestandtabellen van FSM kunnen reduceerbaar zijn, d.w.z. dat het aantal inwendige toestanden vaak te verminderen is. Dit gebeurt op grond van equivalentie. Twee toestanden zijn equivalent als voor elke mogelijke ingangsvolgorde de machine dezelfde uitgangsvolgorde genereerd.

Met een volgorde wordt hier een opeenvolging van willekeurige combinaties bedoeld, in de literatuur vaak een woord J = [a1 a2....an] met lengte L(J) genoemd. Alle voor toestand q gespecificeerde ingangswoorden worden aangeduid met Tq. Equivalentie kan hiermee beschreven worden als

$$y(qj,J) = y(qk,J)$$

voor alle J in Tqj door Tqk. Indien deze equivalente toestanden geelimineerd zijn spreekt men van een gereduceerde machine.

Reduceren doet men in de praktijk door opdeling (partitioning). Het gemakkelijkst kan daarbij met een toestandstabel (transitie model) en een rij toestanden gewerkt worden. De rij toestanden welke in de FSM mogelijk zijn worden opgedeeld in blokken door bepaalde deelregels.

De procedure verloopt als volgt. In de eerste deelregel P1 worden de toestanden die bij dezelfde ingangscombinatie i dezelfde uitgangscombinaties tot gevolg hebben bij elkaar gevoegd. Zo ontstaan enkele blokken. Zie het voorbeeld van figuur 3.

	0	1
A	B 1	C 0
B	C 1	D 0
C	B 1	D 0
D	E 0	C 1
E	D 0	A 1

$P_1 = (A, B, C)(D, E)$

$P_2 = (A)(B, C)(D, E)$

$P_3 = (A)(B, C)(D)(E)$

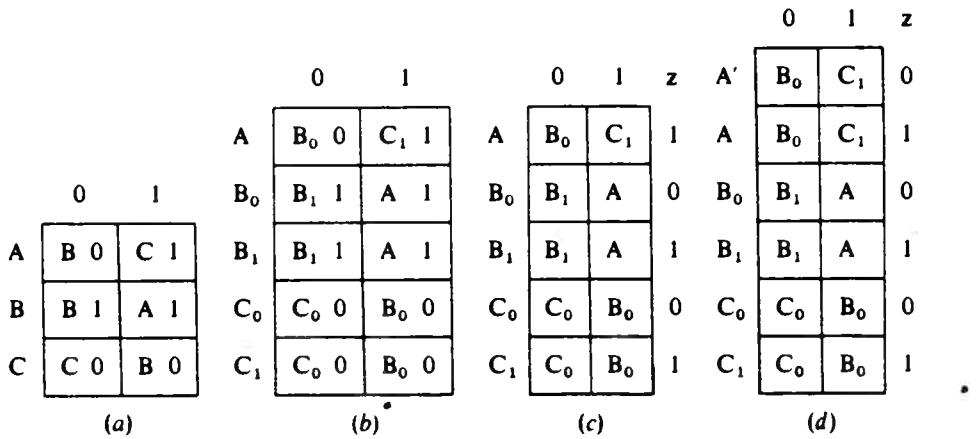
$P_4 = (A)(B, C)(D)(E)$

Fig. 3 Samenvoegen

Binnen deze blokken wordt vervolgens deelregel P2 toegepast. In elk blok worden die toestanden bij elkaar in nieuwe blokken gevoegd waarvan de opvolgende toestanden voor elke ingangscombinatie i in hetzelfde blok ligt (dit kunnen voor verschillende i verschillende blokken zijn). Ditzelfde wordt gedaan in de deelregels P3 tm Pn. De procedure eindigt als in de stap van Pi naar Pi+1 geen wijziging in de blokken optreedt. Een uitgebreidere uitleg van de procedure is te vinden in [5, blz 120].

3. Enkele stellingen rond Finite State Machines

* De uitvoer van een n-state FSM bij invoer met periode p is periodiek met ten hoogste periode np.
* Voor elke transitie gerichte machine bestaat er een corresponderende toestand gerichte machine die precies dezelfde input-volgorden herkent. In figuur 4 wordt hiervan een voorbeeld gegeven. Het omgekeerde is echter niet waar, van een toestand gerichte machine kan geen transitie gerichte machine worden gemaakt.



Appendix D

Fouten in pan

De moeilijkheden die er optraden bij het analyseren deden vermoeden dat er enkele belangrijke fouten in het analyseprogramma 'pan' konden zitten. Er zijn de afgelopen maanden inderdaad fouten gevonden.

1. Analyse meer dan 2 processen niet compleet

Het analyseprogramma pan werkt op basis van reguliere expressies. In 7.3 en 7.4 is globaal beschreven hoe deze algebra wordt gebruikt voor het valideren van communicatie. In de "Theory for Protocol Validation" [6] wordt de theorie nauwkeuriger beschreven. In dit stuk wordt met behulp van eenvoudige protocolletjes een aantal gestelde axioma's en regels geillustreerd.

1.1 De associatieve eigenschap

De volgende fout is recent door de heer Beukers ontdekt.
Voor elk proces worden een aantal expressies opgesteld welke de berichtenvolgordes voorstellen die dat proces kan verwerken. Met een 'Cross' operator worden deze expressies gecombineerd.

X(P1,P2,...Pn)

Voor twee processen wordt dit uitgewerkt tot:

P1 X P2

Om de vermenigvuldiging van minder eenvoudige expressies in de gewone algebra uit te voeren kennen we de associatieve eigenschap.

A x B x C = (A x B) x C = A x (B x C)

Hiermee kan bijvoorbeeld de volgende berekening in fasen uitgevoerd worden.

$$(a+b)(a+b)(a+c)=(a^2+2ab+b^2)(a+c)=a^3+2a^2b+a^2c+2abc+ab^2+b^2c$$

In de validatietheorie wordt gesteld dat de associatieve eigenschap ook voor protocolexpressies geldt (Bijlage VI). Het zou volgens de theorie mogelijk zijn paarsgewijs de

'Cross' operatie uit te voeren, waarbij de berichten die afkomstig zijn van, of bestemd zijn voor andere processen (even) buiten beschouwing worden gelaten. De theorie wat betreft de uitwerking van de algebra blijft dan kloppen. Er worden dus zeker volgorden gevonden. Het zullen echter niet ALLE volgorden meer zijn. Er wordt namelijk geen rekening gehouden met de veranderde berichtenvolgorde in de mailboxes van de bestudeerde processen. De vrijgelaten berichten worden pas nadat de communicatie tussen de twee processen is uitgewerkt (voor een volgorde) aan de ontstane enkele mailbox van de beide processen toegevoegd. Hierdoor is de berichtenvolgorde van de beide ingangsbuffers ernstig verstoord. De afhankelijkheid van de processen en de berichten in hun mailboxes wordt op deze manier ook niet meegenomen (H 6.3). Men kan de volgorde in de mailbox van een proces niet vaststellen zonder alle processen tegelijk erbij te betrekken.

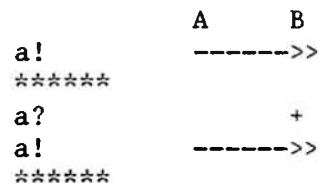
1.2 Volgorde-ruimte reduktie

De volgende fout heeft ongeveer dezelfde grondslag als de zojuist genoemde. Er is op PANDORA een bereikbaarheidsruimte-reduktie geimplementeerd, gebaseerd op een algoritme welke slechts voor twee processen correct is bewezen [106][47]. Om dezelfde redenen als boven genoemd zal dit betekenen dat de verzameling gegenereerde volgorden verre van compleet zal zijn, in dit geval eigenlijk extra incompleet. Waarschijnlijk is deze fout er ook de oorzaak van dat de analyse soms op onverklaarbare wijze gestopt wordt.

Genoemde problemen betekenen niet dat pan ineens onbruikbaar is geworden. Pan is gebaseerd op een zeer moderne analyse techniek, de techniek kent veel voordelen ten opzichte van andere analyse technieken. Andere technieken hebben als eerder gezegd ook moeite met meer dan twee processen.

2. Residueen algoritme

De residueen worden opgezocht door na afloop van de analyse de files met de in een lus eindigende- (PAN.LOOP) en normale volgorden (OUTPUT) te onderzoeken (Appendix F).
Het gebruikte algoritme onthoudt de verzending van een bericht in de volgorde totdat het bericht elders in de volgorde weer wordt ontvangen. Elk bericht dat aan het einde van de volgorde wordt overgehouden is een residu. Dit algoritme verwerkt de OUTPUT file correct. De PAN.LOOP file wordt echter niet goed verwerkt. In de PAN.LOOP file kunnen bijvoorbeeld de volgende volgorden voorkomen.



Doordat hier volgordetechnisch als laatste een bericht verzending genoemd wordt zal het algoritme onterecht een residu 'a' melden. Het algoritme houdt geen rekening met dit soort lussen.

Dit probleem is op te lossen door het algoritme enigszins te wijzigen.

Appendix E

Faalmodellen voor deadlocks

Inleiding

Deadlocks worden bij protocol validatie meestal beschouwd door middel van twee via een communicatie medium communicerende processen. Dit is een vereenvoudigd model van reele protocollen. In deze appendix wil ik enige aspecten van deadlocks in reele protocolsystemen trachten te beschouwen.

Bij het ontwerp van reele protocolsystemen worden bekende technieken uit de systeemleer toegepast. Deze technieken hebben onder andere geleid tot gelaagde modellen als OSI en SNA (lagenmodel van IBM). Vanwege de grote complexiteit van protocollen op een laag worden de laag-protocollen zelf ook weer verder opgesplitst (H 5.3).

In het eerste deel wordt bediscussieerd of de huidige validatietechnieken voor bepaalde structuren 100 % afwezigheid van deadlocks kunnen garanderen.

In het tweede deel wordt dit feit met enige probebalistische beschouwingen met de werkelijkheid gerelativeerd.

In de bedrijfszekerheidstechniek wordt de faalkans van wat complexere systemen uitgedrukt met behulp van faalmodellen (48). Naar mijn mening is het ook mogelijk voor deadlocks dergelijke systemen op te zetten. Hierover gaat het derde deel.

1. Validatie techniek

Ik wil de bewijskracht van een protocolvalidatietechniek welke gebruik maakt van een model met twee via een buffer communicerende processen beschouwen. Hierbij worden serie structuren en parallel structuren beschouwd. De afhankelijkheid van (of relaties tussen) de processen is hierbij erg belangrijk.

Een seriestructuur voor een protocol is een lagenmodel (fig 1).

In een lagenmodel is het mogelijk om de protocollen A en B

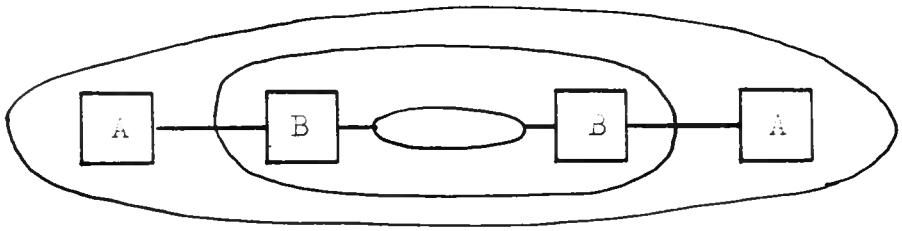


fig 1 lagenmodel

zo goed als onafhankelijk te maken.
Onze validatietechniek laat toe de protocollen A en B afzonderlijk te valideren. Als de protocollen onafhankelijk zijn, dan zal het bewijs dat beide protocollen geen deadlocks bevatten ook betekenen dat de protocollen serieel gecombineerd geen deadlocks bevatten.

Een parallelstructuur voor een protocol is in figuur 2a weergegeven.

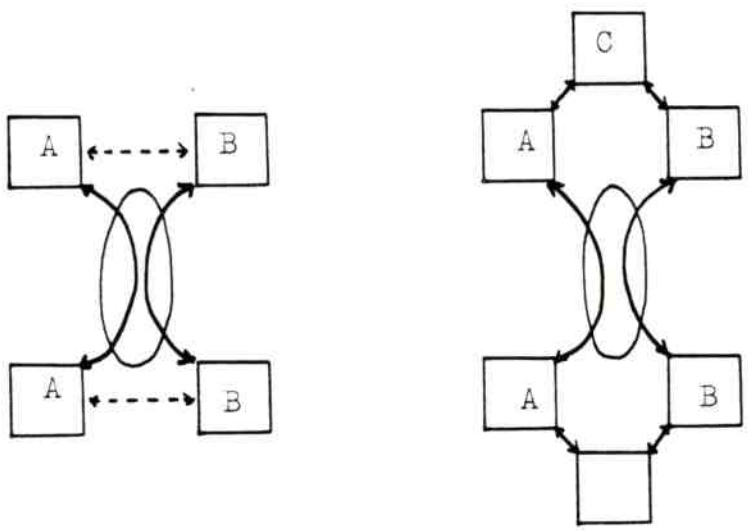


fig 2 parallel structuur

Als deze structuur wordt toegepast zullen de protocollen A en B vaak wel relaties met elkaar hebben, of wel afhankelijk

zijn.

Als de protocollen relaties hebben, dan moet direct een concessie gedaan worden voor de validatie. We kunnen protocol A en B apart valideren door de relaties buiten beschouwing te laten in de validatiemodellen. Als geen van beide protocollen deadlocks bevat dan zal dat echter geenzins garanderen dat de protocollen tesamen geen deadlocks bevatten. Omgekeerd zal het het feit dat een of beide protocollen deadlocks bevat niet betekenen dat de protocollen tesamen deadlocks bevatten. Beide protocollen beinvloeden elkaar immers aan beide communicatiekanten, de beinvloeding zal door de protocollen zelfs afhankelijk zijn voor beide kanten. Het ene protocol zou bijvoorbeeld kunnen dienen om storingen in het andere op te vangen! Ook als we de relaties tussen A en B als protocol gaan valideren behoeft dit nog niet te betekenen dat het protocol goed is. De afhankelijkheid van alle vier processen blijft immers steeds buiten beschouwing.

Alleen als we de relaties tussen de beide protocollen proberen te minimaliseren en strak te definieren kan het mogelijk zijn te bewijzen dat de protocollen tesamen ook geen deadlocks hebben. Zoiets kan bijvoorbeeld bereikt worden door twee derde processen te definieren, processen die geen relaties met elkaar hoeven te hebben. Deze processen staan hierarchisch boven de protocolprocessen (fig 2b) en kunnen de protocollen besturen. Zoiets wordt in de praktijk wel toegepast. De relaties tussen de protocollen zullen dan bijvoorbeeld parameters in de tijd hebben. Het vaststellen van regels waaraan de relaties zouden moeten voldoen (ontwerp regels, regels voor de koppeling (H 5.4)) is een punt van verder onderzoek.

2. Deadlocks bij zeer vele processen

In dit deel wordt het voorkomen van deadlocks in complexere processtructuren met enige kansbeschouwingen aangevuld.

Stel we hebben een processtructuur met meer dan twee processen, processen welke vermaasd met elkaar communiceren (fig 3).

Er wordt uit gegaan van processen die berichten die niet herkend worden uit hun ingangsbuffers verwijderen (als in SDL het geval is).

Een (statische) communicatie deadlock wordt gedefinieerd als een stilstand in een van de communicerende processen als gevolg van een disorientatie met de overige processen. De disorientatie bestaat eruit dat het in deadlock zijnde proces wacht op een bericht uit een verzameling door hem verwachte berichten. Als geen van de berichten uit deze verzameling door de omgeving van het proces (de overige processen) gegenereerd kan worden dan is het proces

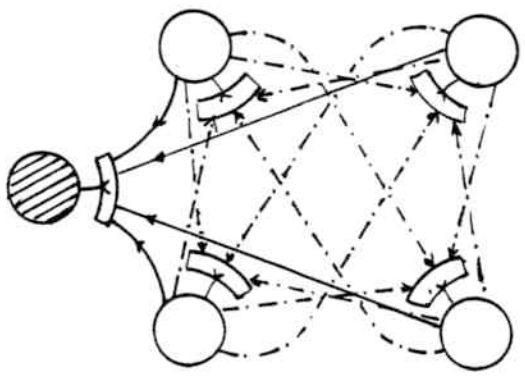


fig 3 Vermaasd communicerende processen

communicatie-technisch in deadlock.

Als we n processen hebben dan zullen n-1 processen kunnen blijven communiceren. Hoe meer processen de omgeving van het in deadlock zijnde proces kent, hoe dynamischer de omgeving zich zal gedragen en hoe groter het aantal mogelijke berichten wordt dat naar het in deadlock zijnde proces verstuurt kan worden. Als we uitgaan van een vast aantal berichten dat het in deadlock zijnde proces kan ontvangen, dan zal dit betekenen dat met een toenemend aantal processen in de omgeving de kans op het optreden van een deadlock afneemt. Dit zal dus ook betekenen dat bij toenemend aantal processen de kans op deadlocks vermindert (in de praktijk misschien sterk vermindert).

Als uitgegaan wordt van processen die berichten die niet herkend worden in hun (FIFO) ingangsbuffer laten staan, dan zal een omgekeerd effect optreden. Hoe meer processen er met een proces communiceren, hoe dynamischer de communicatie wordt en hoe groter de kans dat er een bepaald niet gedefinieerd bericht in de ingangsbuffer geplaatst wordt. Hoe groter dus ook de kans op een deadlock (kans zal waarschijnlijk snel groter worden met het aantal processen).

3. Katastrofale faalmodellen

In de bedrijfszekerheidstechniek (48) is een van de eenvoudigste en meest gebruikte modellen het zgn. katastrofale faalmodel. In dit model gaat men er van uit dat een entiteit slechts een faalwijze vertoont (single mode failures). Een entiteit wordt in faalmodellen vervangen door een black-box waarin men zich een schakelaar indenkt (fig 4a).

Als een entiteit goed is, staat de schakelaar gesloten en

Appendix F

1.INLEIDING

Tijdens mijn afstudeerwerk zijn enige wensen naar voren gekomen ten aanzien van het PANDORA systeem. Deze zijn in de conclusies naar voren gekomen. Een aantal van deze aanvullingen hebben een duidelijke plaats binnen de reeds bestaande software van PANDORA, en betreffen voornamelijk toevoegingen aan het analyse deel.

Een gedetaileerd overzicht van de gebruikte programma's en hun functies binnen de analyse en het PANDORA systeem is er echter niet. Tot nu toe is men hiervoor op de algemene rapporten betreffende het pandora systeem, waaronder een kleine manual, en de documentatie van de protocol-compiler 'proco' en het analyse programma 'PAN' aangewezen. Geen van allen bevat echter de informatie die ik zoek; de programmatische structuur van het systeem in detail.

Om enige van mijn opmerkingen te kunnen documenteren heb ik, wat het analyse deel betreft, aan de programmatische structuur in het eerste deel van deze bijlage enige paragrafen gewijd. De daarop volgende paragrafen betreffen het eigenlijke onderwerp van deze bijlage; de aanvullingen.

Naar mijn mening dient iedere gebruiker van PANDORA bekend te zijn met de inhoud van de eerste 3 hoofdstukken en hoofdstuk 5, In deze hoofdstukken staat waardevolle informatie over de algemene structuur van het systeem en het gebruik ervan.

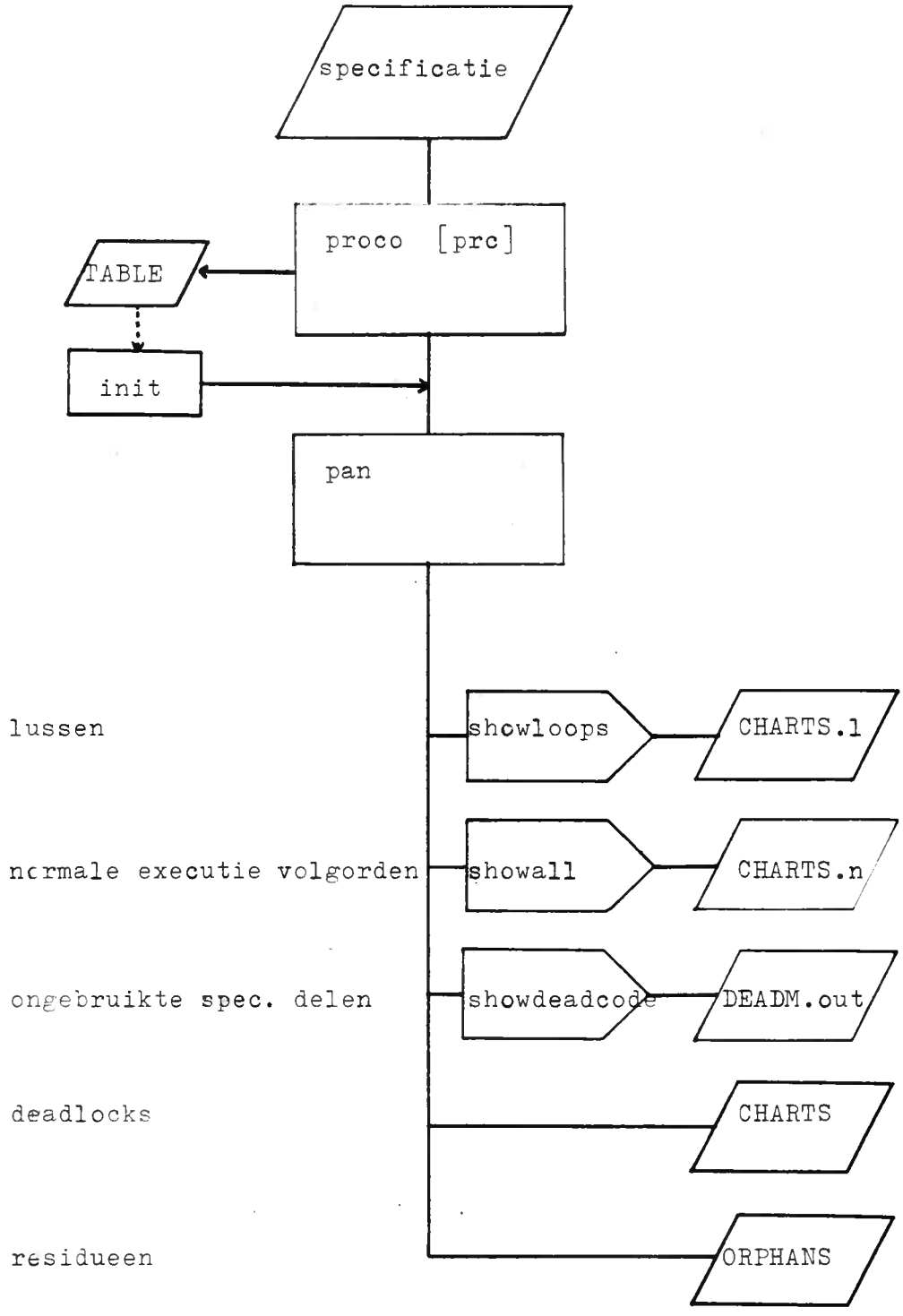
2. WERKEN MET PANDORA

Wanneer men een protocol op PANDORA analyseerd is de procedure voor te stellen als in nevenstaande figuur. Men stelt, al dan niet binnen prosy, een specificatie (spec) op welke men compileerd met 'proco' (prc). Wanneer de specificatie 'proco' zonder foutmeldingen doorkomt wordt de eigenlijke analyse met 'pan' gestart, eventueel voorafgegaan door een initiele berichtenrij welke met het init commando te definieren is. De initiele berichtenrij moet in gecodeerde vorm worden ingevoerd, de daarvoor te gebruiken codeomzettingen staan in de file 'TABLE'.

De analyse met 'pan' levert direct enige resultaten op; melding van het aantal gevonden deadlocks en een gemakkelijk leesbare file 'CHARTS' met volgorden die tot een deadlock leiden. Daarnaast wordt gemeld of er residueen zijn en wordt, indien deze aanwezig zijn, hiervoor naar de file 'ORPHANS' verwezen. 'ORPHANS' is niet zoals 'CHARTS' in een leesbare vorm gegoten, maar bevat code welke voor het init commando gebruikt kan worden. Er wordt gesteld dat de analyse bij aanwezigheid van residueen opnieuw gestart moet worden met de residueen in de initiele berichtenrij. Met de file 'TABLE' zijn ook de residueen te decoderen, eventueel is hiervoor het commando 'whatis'.

De overige resultaten zijn in sluimerende vorm aanwezig en worden indien gewenst naar voren gehaald door de naverwerk commando's 'showloops', 'showall' en 'showdeadcode'. Hiermee worden respectievelijk de verdachte lussen (file 'CHARTS.1'), de gewone executie-volgorden (file 'CHARTS.n') en de delen van de specificatie welke niet gebruikt zijn (file 'DEADM'.out) naar voren gehaald. Verdere verwerking is aan de gebruiker en betreft vooral de interpretatie van de resultaten.

Opmerking
Bij de naamgeving van PANDORA is aan alle data-files namen met hoofdletters gegeven (behalve 'prc.out'), en alle programma's en commands namen met kleine letters (behalve 'PAN').



PANDORA : Zichtbare systeemsoftware structuur van de analyse

3.FILESTRUCTUUR

Rond de geschetste proceduregang worden files door bepaalde commando's aangemaakt, welke dienen als gegevens voor andere commando's. Zo produceerd 'proco' een aantal files welke als invoer voor 'pan' en de 'show-' commando's dienen en zijn de versluimerde resultaten van 'pan' in gecodeerde files geschreven. Deze files dienen als invoer voor de 'show-' commando's. Ook voor de deadlock-file 'CHARTS' bestaat een gecodeerde voorloper.
Al deze files staan in de nevenstaande figuur. Daarin is ook aangegeven welke delen van het systeem deze files gebruiken. Met behulp van een eenvoudige specificatie wordt de betekenis van de files uitgelegt.

Deze specificatie luidt als volgt.

```
proc A
     B!m;
          do
          :: B?r -> B!m
          :: B?a -> break
          od
end A;

proc B
     do
     :: A?m ;
          if
          :: A!a -> break
          :: A!r
          fi
     od
end B.
```

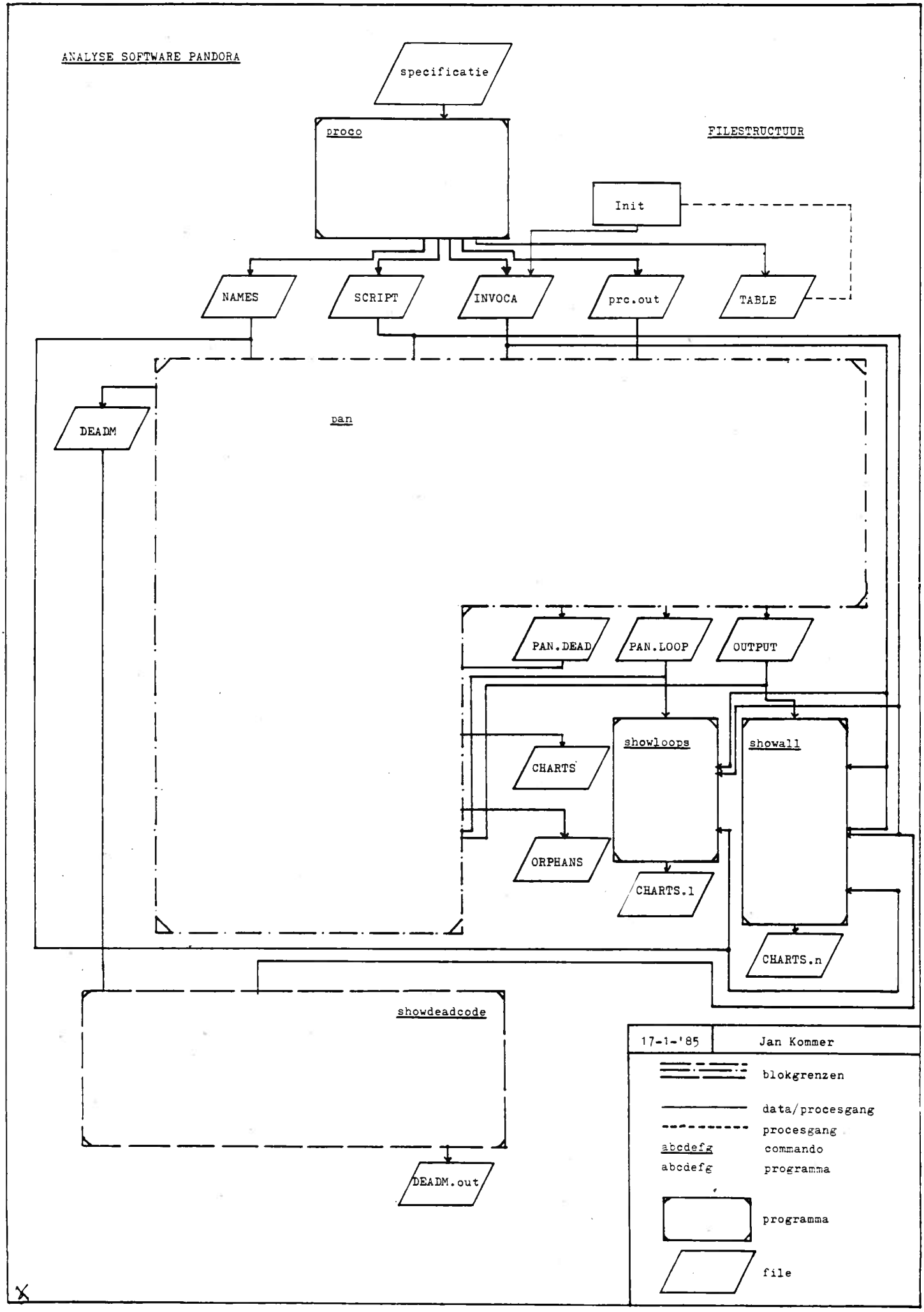
3.1 Files geproduceerd door 'proco'

Proco produceerd naar aanleiding van deze specificatie; zoals op de figuur te zien is, momenteel 5 files.

3.1.1 'prc.out'

In 'prc.out' staat de rechtstreekse vertaling van de specificatie in de validatiealgebra. Dit is m.b.v. nummers gedaan waarvan de betekenis in de file '/usr/pan/pan.doc/pan.numbers' te vinden is.
De gerbuiker van het systeem heeft vrijwel niets aan deze informatie, kan er ook niets anders mee doen als bekijken hoe de specificatie op de algebra is afgebeeld.

Als voorbeeld volgt de inhoud van 'prc.out' zoals deze voor de voorbeeld specificatie geproduceerd is, gevolgd door


ANALYSE SOFTWARE PANDORA
specificatie
proco
FILESTRUCTUUR
Init
NAMES
SCRIPT
INVOCA
prc.out
TABLE
DEADM
pan
PAN.DEAD
PAN.LOOP
OUTPUT
showloops
showall
CHARTS
ORPHANS
CHARTS.1
CHARTS.n
showdeadcode
DEADM.out
17-1-'85
Jan Kommer
blokgrenzen
data/procesgang
procesgang
abcdefg
commando
abcdefg
programma
programma
file

een kleine vertaling m.b.v. pan.numbers.

```
110,666,666,444,444,11,110,555,444,12,3,555,555,999,999,
666,666,444,444,10,444,444,112,3,555,444,111,555,555,
555,555,999,999,

110,do(,do(,if(,if(,11,110,if),if(,12,break,if),if),do),
do),
do(,do(,if(,if(,10,if(,if(,112,break,if),if(,111,if),
if),if),if),do),do),
```

In deze file hoort bij ieder proces uit de specificatie een regel. De regels zijn te lang om op een bladregel te kunnen, hier zijn ze afgebroken. De betekenis van de resterende nummers is te vinden in de file

3.1.2 'TABLE'

Hierin staan de nummers behorend bij de berichten uit de specificatie.

```
================
No:  Message:
================
110  A->B:m
111  B->A:r
112  B->A:a
```

Deze file is er voor het gerief van de gebruiker en wordt door het systeem zelf niet meer gebruikt.

3.1.3 'INVOCA'

Met 'INVOCA' is de verzameling berichten welke een proces in zijn ontvangst buffer kan krijgen gedefinieerd. Hieronder vallen ook 'default' ontvangsten en 'timeouts'. In dit geval ziet 'INVOCA' er als volgt uit.

```
11,12,
10,
```

Hierin staat dat proces A de berichten 11 en 12 (B->A:r en B->A:a) kan ontvangen en B het bericht 10 (A->B:m).

Het init commando werkt ook op de 'INVOCA' file. Als bijvoorbeeld Bij het opstarten van het proces meteen een bericht B!m (bericht 10) verzonden moet worden dan ziet na het 'init 110' commando 'INVOCA' er als volgt uit.

```
110,
11,12,
10,
```

Met de waarde 110 (100+10) wordt aangegeven dat het om een verzending van een bericht gaat. Zie hiervoor ook de verklaring van de file 'OUTPUT'. Met het toevoegen aan de INVOCA file wordt meteen een bericht in de mailbox ruimte

gebracht welke daar in de bijbehorende mailbox terecht zal komen aan het begin van de analyse.

4. 'NAMES'
In 'NAMES' staan de namen van de gespecificeerde processen. De gebruiker is vrij om wijzigingen in de volgorde van 'NAMES' aan te brengen om de leesbaarheid van de uitvoer (CHARTS) te verhogen.
Bij de voorbeeld specificatie hoort de volgende 'NAMES' file:

```
A
B
```

Met deze originele file 'NAMES' ziet een 'CHARTS' er als volgt uit;

```
MESSAGE       A      B

m               ----->
a               <-----
====
```

Als we de volgorde van 'NAMES' veranderen in

```
B
A
```

dan ziet dezelfde 'CHARTS' er omgekeerd uit;

```
MESSAGE       B      A

m               <-----
a               ----->
====
```

Hierdoor is men niet gebonden aan de volgorde die 'proco' bij het compileren ziet. Proco definieerd namelijk processen op het moment dat er iets (een bericht van/naar dat proces) van gevonden wordt, en zal daardoor niet altijd de volgorde van de processen in de specificatie aan houden.

5. 'SCRIPT'
In 'SCRIPT' staan edit-scripts welke gebruikt worden voor het terugvertalen van code naar leesbare berichten.

```
s/^10$/A->B:m?/g
s/^110$/A->B:m!/g
s/^11$/B->A:r?/g
s/^111$/B->A:r!/g
s/^12$/B->A:a?/g
s/^112$/B->A:a!/g
```

3.2 Files geproduceerd door pan

Pan gaat aan het werk met de door 'proco' geproduceerde files. Na afloop van de analyse produceerd 'pan' eerst een gecodeerd resultaat in de vorm van de files 'PAN.DEAD', 'PAN.LOOP', 'OUTPUT' en 'DEADM'. Vervolgens wordt de file 'PAN.DEAD' omgezet in de file 'CHARTS' met de gevonden deadlocks. Tenslotte wordt de file 'ORPHANS' geproduceerd.
Met uitzondering van 'CHARTS' zijn al deze files gecodeerd. De code die hierbij gebruikt wordt is met de hand te vertalen m.b.v. de files 'TABLE' en 'pan.numbers'.
PANDORA gebruikt voor het vertalen van deze files tot leesbare 'CHARTS*' files de files 'NAMES', 'INVOCA' en 'SCRIPT'. De laatste files worden door 'pan' gebruikt voor het omzetten van de 'PAN.DEAD' file in de 'CHARTS' file. Ook 'showloops', 'showall' en 'showdeadcode' gerbuiken deze files om de resp. files 'PAN.LOOP', 'OUTPUT' en 'DEADM' in de resp. files 'CHARTS.1', 'CHARTS.n' en 'DEADM.out' om te zetten.
De code-files met hun daaruit afgeleide leesbare files zien er als volgt uit;

3.2.1 'OUTPUT'

De file 'CHARTS.n' staat al als voorbeeld genoemd in de verklaring van 'NAMES'. Deze is geproduceerd naar aanleiding van de volgende 'OUTPUT' file.

```
110,10,112,12
```

In deze regel (er is maar een normale volgorde gevonden, anders waren het meerdere regels) stellen de getallen in de 100 het verzenden van berichten voor en de getallen onder de honderd (het verzendgetal - 100) de ontvangst van deze berichten.
In dit geval vallen het verzenden en ontvangen van berichten direct na elkaar. 10 volgt direct na 110 en 12 direct na 112. Deze directe ontvangst wordt vertaald in een enkel pijltje in de CHARTS.
Als de ontvangst door de bufferwerking niet direct op de verzending volgt en er eerst andere berichten verzonden/ontvangen worden dan wordt in 'CHARTS' eerst een dubbel pijltje voor de verzending gebruikt en vervolgens voor de ontvangst een plusje. Zo zal de 'CHARTS' van de 'OUTPUT'-regel

```
110,112,10,12
```

er als volgt uitzien.

```
MESSAGE        A       B

m!                 ------>>
a!               <<------
m?                       +
a?             +
```

3.2.2 'PAN.DEAD'
De file 'PAN.DEAD' is met de gegeven specificaties leeg, er zijn geen deadlocks in het voorbeeld. Als de bekende 'OUTPUT'-volgorde een deadlock was geweest dan zou deze er in de 'PAN.DEAD' file als volgt hebben uitgezien.

110,10,111,11%

Het enige verschil is dus het procent teken. In de 'CHARTS' maakt dit vrijwel niets uit.

3.'PAN.LOOP'
Aan de voorgaande cijfers zijn in de 'PAN.LOOP' file nog enige cijfers (2222 en 3333) toegevoegd. Deze cijfers markeren het begin en het eind van een loop. In de 'CHARTS' worden deze symbolen verwerkt tot sterretjes.
Bij de ene volgorde in onze 'PAN.LOOP' file

110,10,2222,111,11,110,10,3333,5%

zien de 'CHARTS.1' er als volgt uit.

```
1:
MESSAGE       A      B

m               ----->
****
r               <-----
m               ----->
****
====
```

4.'DEADM'
De file 'DEADM' lijkt enigzins op de algebraische specificatie in 'prc.out'. In 'DEADM' wordt wel altijd de proces volgorde van de compiler aangehouden. Dit is over het algemeen geen bezwaar, ,'DEADM'.out wordt slechts eenmalig uitgelezen. Mocht het wel een bezwaar zijn dan moeten voor het opstarten van 'pan' in de files 'prc.out', 'INVOCA' en 'NAMES' de regels zorgvuldig van volgorde veranderd worden.
In ons geval zijn alle delen van de specificatie uitgevoerd. Als er een bericht tussen zit welke niet uitgevoerd is, dan is dat aangegeven met een - teken.
Zo'n - teken is als voorbeeld toegevoegd.

```
PROC(1)
 110,666,666,444,444,-11,110,555,444,12,555,555,999,999,
END(1)

PROC(2)
 666,666,444,444,10,444,444,112,555,444,111,555,555,555,555,999,999,
END(2)
```

Vertaald met behulp van 'SCRIPT' tot 'DEADM'.out;

```
PROC(1)
 A->B:m!
   DO
   ::
      IF
      :: -B->A:r?
       A->B:m!
      :: B->A:a?
      FI
   OD
END(1)
PROC(2)
   DO
   ::
      IF
      :: A->B:m?
         IF
         :: B->A:a!
         :: B->A:r!
         FI
      FI
   OD
END(2)
```

All statements preceded by a minus sign have not be executed.
Some may be executable for different initial strings (use init).

Hieruit blijkt dat DEADM.out de terugvertaalde algebra voorstelt voorzien van - tekens bij niet geexecuteerde berichten.
Dit verklaart ook de volgende constructie [6], [7];

```
   DO
   ::
      IF
      ::
      FI
   OD
```

In de algebra is de do-loop namelijk een herhaalde or, ofwel

```
(a+b)* <- repeat
  ^------ or

 ==

do
:: !a
:: !b
od
```

De or is de representatie van het if-fi statement[3]

4. DE EIGENLIJKE SOFTWARE

4.1 De analyse software globaal

De vorige figuur uitgebreid met de globale interne structuur van 'pan' is nevenstaand aangegeven.
Binnen het omlijnde 'pan' commando vind de eigenlijke analyse in het rechterdeel plaats. De naverwerking vindt plaats in het linkerdeel.

Voor de analyse worden 3 programma's gebruikt; PAN, 'statespace' en 'ncompct'. Deze programma's draaien parallel. PAN en 'statespace' werken daarbij nauw samen, wat blijkt uit het wederzijdse dataverkeer tussen de beide blokken door de pipes. De analyse van de algebra wordt in deze twee processen gedaan. Alle volgorden worden, voorzien van een kenmerkje duidend of het een deadlock, een loop of een normale volgorde betreft, aangeboden aan 'ncompct' die deze volgorden vervolgens netjes uitschrijft in de juiste files. Vanwege deze relatief eenvoudige functie is 'ncompct' de kleinste van de drie. Voor de werking van 'PAN' en 'statespace' zelf wordt naar [1] verwezen.

De naverwerking van 'PAN.DEAD' vindt in 'translate' (een shell-script) en 'graff' (een programma) plaats.
De functie van 'translate' wordt duidelijk als we de tijdelijke file 'TRANSLATED' voor de fictieve deadlock-volgorde van 'PAN.DEAD' bekijken.

```
TRANSLATED:
A->B:m!
A->B:m?
B->A:a!
%
```

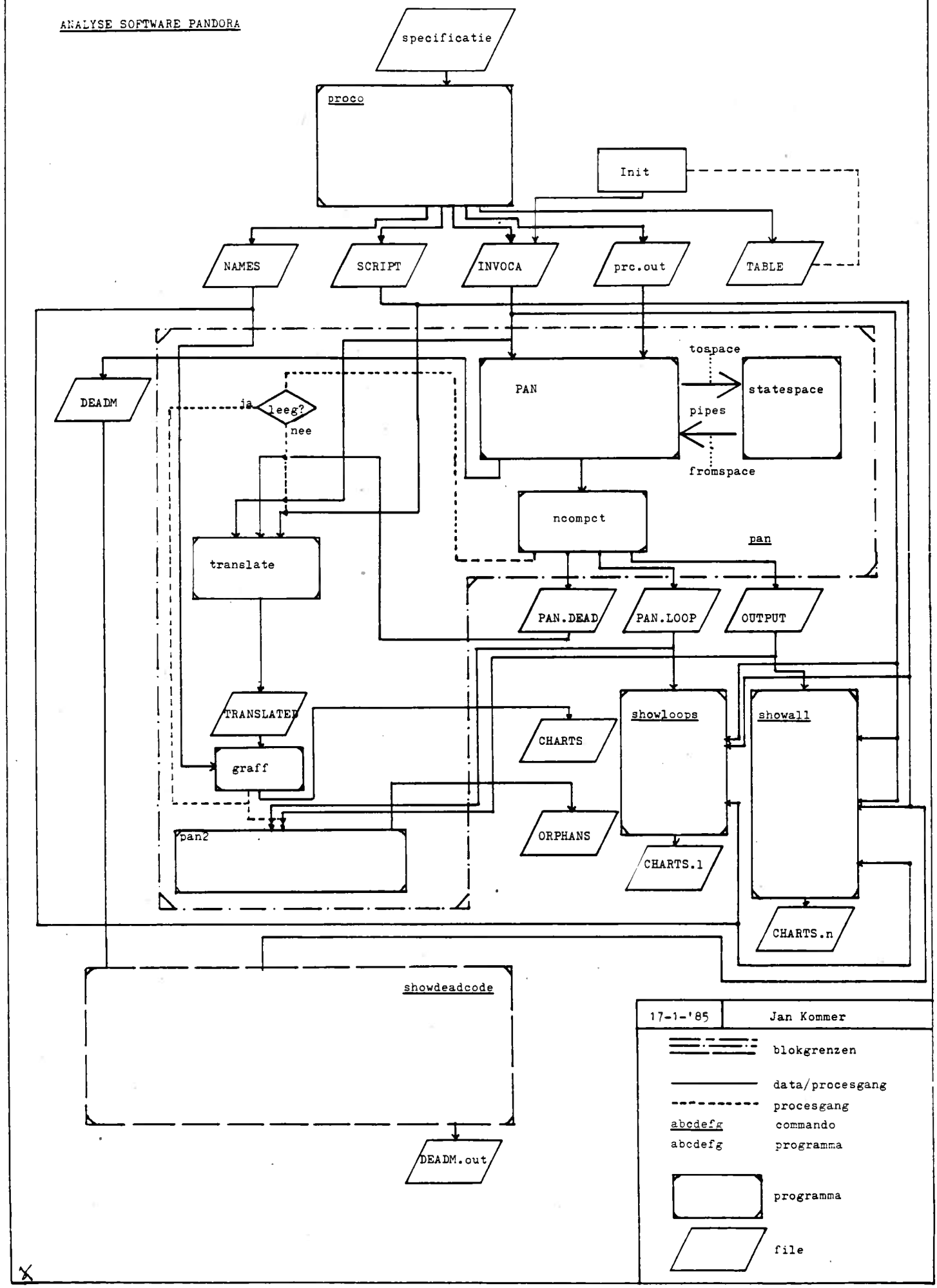
Hieruit blijkt dat 'TRANSLATED' al leesbaarder is geworden als de originele 'PAN.DEAD' file. Deze file is vrijwel rechtstreeks herkenbaar als de 'PAN.DEAD' file

```
110,10,111,11%
```

waarvan de betekenis van de nummers onder elkaar aangegeven is. Het toekennen van een betekenis aan de nummers geschiedt m.b.v. de file 'SCRIPT' en de 'sed' utility.

In 'graff' wordt 'TRANSLATED' op relatief eenvoudige wijze m.b.v. de file 'NAMES' ('graff' is de enige gebruiker van 'NAMES') vertaalt tot de bekende CHARTS.
Als de 'CHARTS' geheel gegenereerd zijn wordt onderzocht of er nog residueen zijn. Deze functie wordt door 'pan2' vervuld. Pan2 gebruikt daarbij de files 'PAN.LOOP' en 'OUTPUT'. Dit resulteert tenslotte in een file 'ORPHANS'.
Op dat moment is 'pan' klaar en geeft zijn meldingen.

ANALYSE SOFTWARE PANDORA



4.2 De analyse software in detail

De volgende figuur bevat de gedetailleerde programma opbouw van het analyse deel van het systeem. Daarin zijn onder andere ook de scripts 'proco', 'translate', 'pan2' en de show- commando's uitgewerkt. Zover dit noodzakelijk lijkt worden deze scripts besproken.

4.1 'translate'

Bij het vergelijken van 'PAN.DEAD' en 'TRANSLATED' moet opgevallen zijn dat er een berichtaankomst miste. Deze berichtaankomst wordt meteen in het begin van 'translate' weggehaald door het programma 'nequiv'. Nequiv doet dit m.b.v. file 'INVOCA'. [Reden onbekend]

Vervolgens worden de komma's weggehaald en wordt de 'PAN.DEAD' nummers onder elkaar (een per regel) i.p.v. naast elkaar geschreven. Vervolgens wordt met het programma 'counterrors' het aantal volgorden geteld (het aantal deadlocks wordt gemeld), en worden de nummers met 'sed', gestuurd door 'SCRIPT', vervangen door berichten. Het resultaat hiervan wordt in 'TRANSLATED' geschreven.

4.2 'showloops' en 'showall'

Showloops maakt van een enigzins gewijzigd 'translate' script gebruik, reden waarom de taken van 'translate' zelfstandig in het naverwerking van 'pan' zijn er niet, evenals in 'showall' welke wel van het 'translate' script gebruik maakt.

4.3 'showdeadcode'

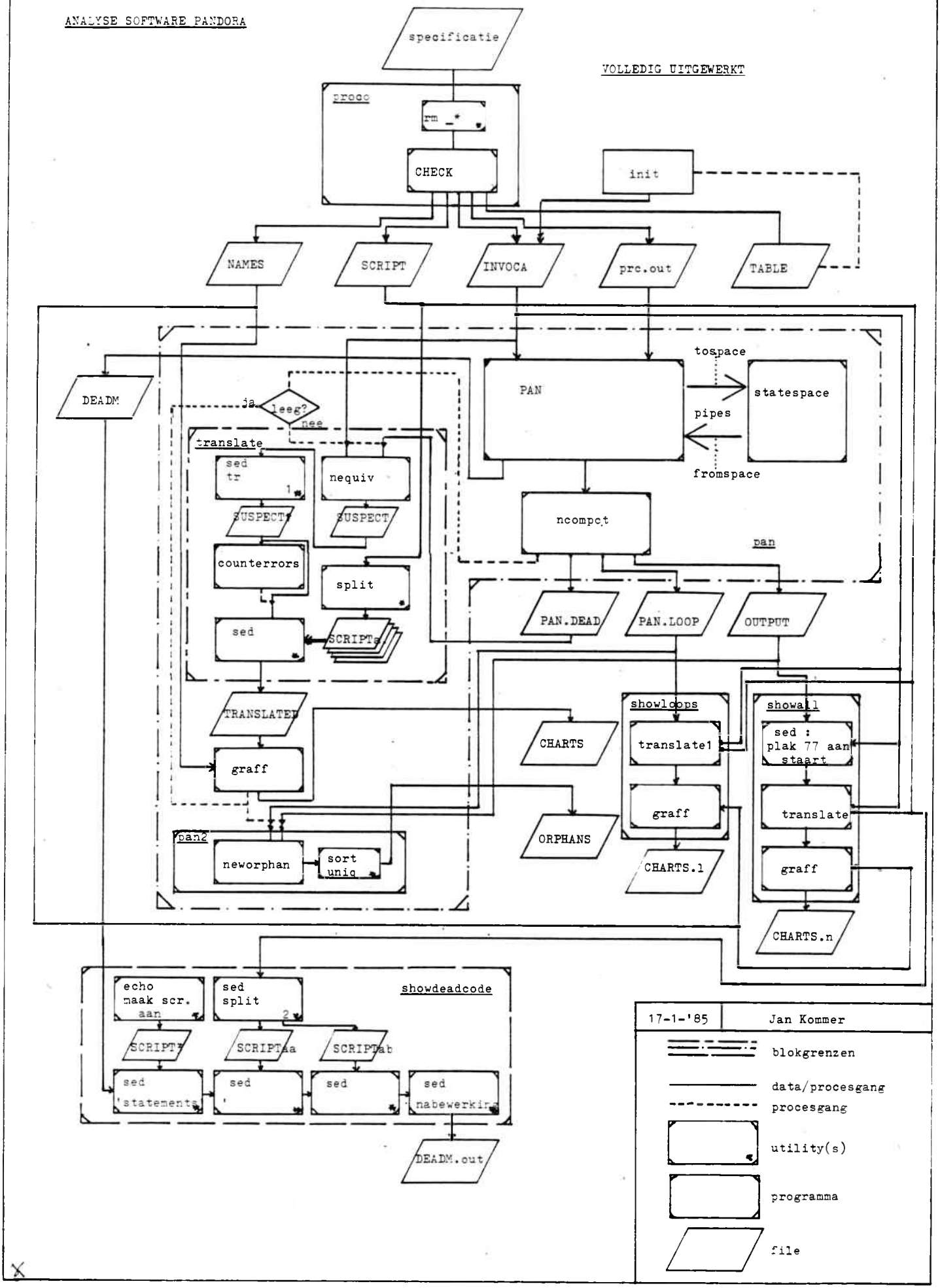
In 'showdeadcode' wordt door 'sed' m.b.v. een intern gegenereerd script en de file 'SCRIPT', de file 'DEADM' omgezet in 'DEADM'.out. Evenals bij translate wordt de file 'SCRIPT' opgesplitst in kleinere scriptjes. Hiermee wordt verwerkingssnelheid gewonnen.

4.4 'pan2'

Pan2 haalt d.m.v. het programma neworphan de residueen tevoorschijn. Dit doet het door de files 'PAN.LOOP' en 'OUTPUT' op compleetheid te analyseren. Bij iedere verzending behoort ook een ontvangst, anders is er sprake van een residu. De gevonden residueen worden vervolgens door een uniq filter gehaald waardoor alle residueen slechts een keer in de file 'ORPHANS' geschreven worden.

4.5 'proco'

Van 'proco' is mij weinig bekend. In het 'proco' script wordt het programma 'CHECK' aangeroepen. In 'CHECK' worden alle 'proco' functies vervuld. 'CHECK' is m.b.v. 'lex' en 'yacc' gegenereert.


ANALYSE SOFTWARE PANDORA
specificatie
VOLLEDIG UITGEWERKT
proco
rm _*
CHECK
init
NAMES
SCRIPT
INVOCA
prc.out
TABLE
DEADM
ja
leeg?
nee
translate
sed
tr
1
nequiv
SUSPECT
SUSPECT
counterrors
split
sed
SCRIPTs
PAN
tospace
statespace
pipes
fromspace
ncompct
pan
PAN.DEAD
PAN.LOOP
OUTPUT
TRANSLATED
graff
CHARTS
showloops
translate1
graff
CHARTS.1
showall
sed :
plak 77 aan
staart
translate
graff
CHARTS.n
pan2
neworphan
sort
uniq
ORPHANS
showdeadcode
echo
maak scr.
aan
sed
split
2
SCRIPTs
SCRIPTaa
SCRIPTab
sed
'statements
sed
'
sed
sed
nabewerking
DEADM.out
17-1-'85
Jan Kommer
blokgrenzen
data/procesgang
procesgang
utility(s)
programma
file

5. POSTPROCESSING

5.1 Inleiding

Bij een specificatie van enige omvang hikt men al snel tegen een grote hoeveelheid door 'pan' gegenereerde uitvoer aan. Deze uitvoer zou met de hand naverwerkt moeten worden, alle volgorden welke tot een deadlock of een lus leiden moeten geinterpreteerd worden. Dit karwei is over het algemeen echter ondoenlijk, men raakt zelfs geheel het gevoel kwijt met een geautomatiseerd systeem bezig te zijn.

Mij is gebleken dat grote delen van de uitvoer niet handmatig geinterpreteerd behoeven te worden. Deze delen kunnen namelijk door de computer worden overgenomen. Daarbij wordt voor de verwerking van deadlocks en lussen van dezelfde filosofie gebruik gemaakt.

Beide soorten fouten worden uitputtend gemeld. Iedere volgorde welke tot een loop of een deadlock leidt wordt beschreven. Daarbij zal het in vele gevallen om dezelfde lus of dezelfde deadlock gaan. Alle verschillende voorgeschiedenissen van een deadlock worden gemeld.

In aanvang is de protocol analysator niet in alle volgorden die naar een fout leiden geinteresseert, over het algemeen is een zo'n volgorde voldoende. Daaruit zal de werkelijke oorzaak van de fout meestal snel genoeg blijken.

De verder uitwerking van deze filosofie is voor de beide soorten fouten verschillend en wordt onderstaand besproken.

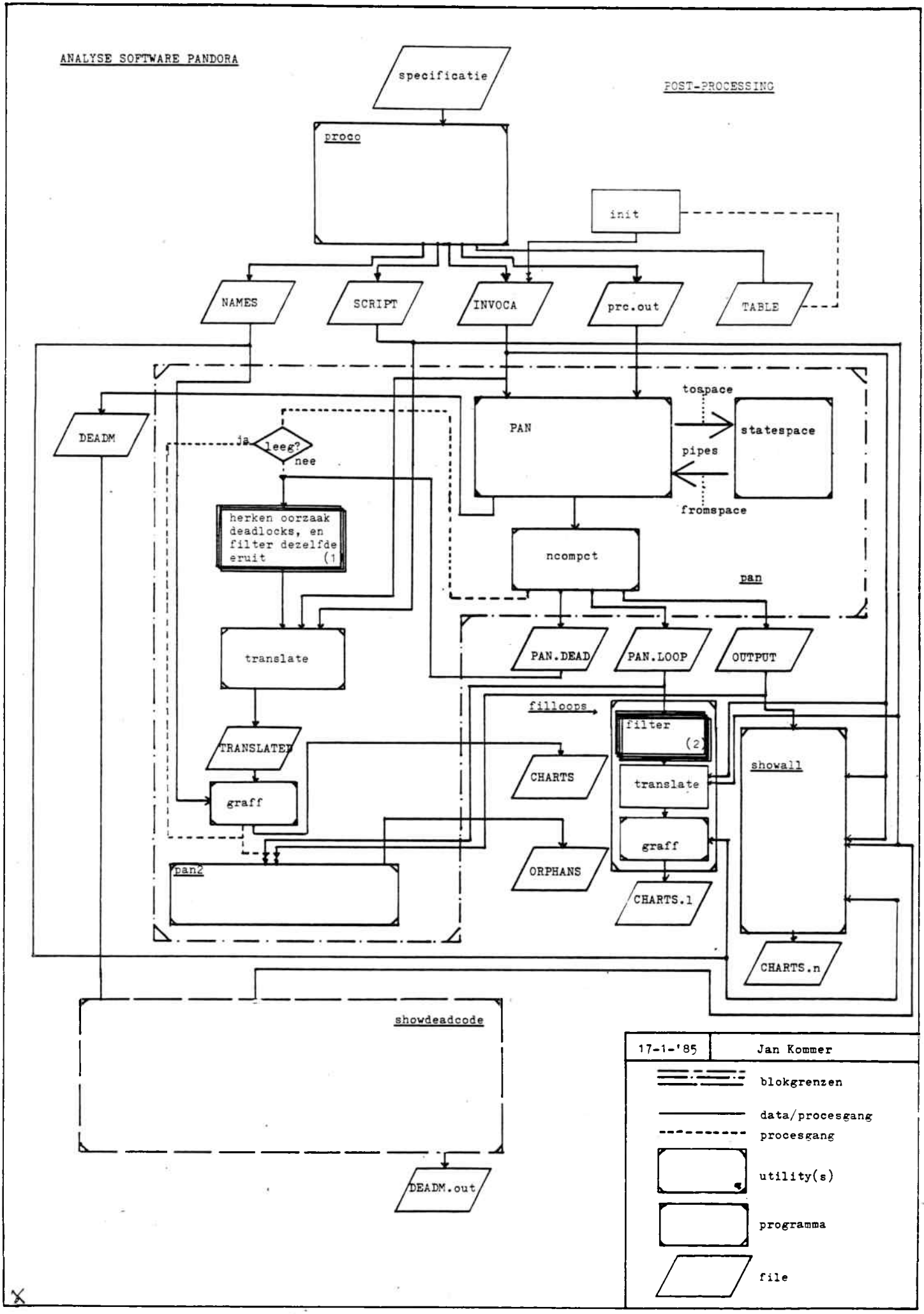
5.1 Postprocessing van loops.

5.1.1 Gelijke lussen

De naverwerking van lussen is gemakkelijk te verwezenlijken. Een lus is een duidelijk gedefinieerde volgorde van berichten die zich telkens herhaalt. Meestal is de omvang van de lus vergeleken met de grootte van de volgorde beperkt. Gelijke lussen zijn heel gemakkelijk te herkennen aan dezelfde volgorde van berichten, dus ook dezelfde volgorde van nummers in de 'PAN.LOOP' file. De grenzen worden in de PAN.DEAD file door gemakkelijk herkenbare speciale nummers gedefinieerd.

De naverwerking bestaat er in dat geval uit om ieder type lus slechts met een volgorde welke tot zo'n lus leidt in de 'CHARTS.l' file af te drukken. Dat zou iedere willekeurige volgorde kunnen zijn welke tot zo'n lus leidt, liefst de kortste.

Deze faciliteit is geheel automatiseerbaar en is reeds verwezenlijkt m.b.v. de unix utility's sed, sort en uniq. Het is een alternatief 'showloops' commando, 'filloops' genaamd. Filloops filtert de loops op de beschreven wijze en vertaalt ze tot 'CHARTS.fl'. Hiernaast is te zien hoe filloops geimplementeerd is in het PANDORA systeem.


ANALYSE SOFTWARE PANDORA
specificatie
POST-PROCESSING
proco
init
NAMES
SCRIPT
INVOCA
prc.out
TABLE
DEADM
ja
leeg?
nee
herken oorzaak deadlocks, en filter dezelfde eruit (1)
translate
TRANSLATED
graff
pan2
PAN
tospace
statespace
pipes
fromspace
ncompct
pan
PAN.DEAD
PAN.LOOP
OUTPUT
filloops
filter (2)
translate
graff
CHARTS
ORPHANS
CHARTS.1
showall
CHARTS.n
showdeadcode
DEADM.out
17-1-'85
Jan Kommer
blokgrenzen
data/procesgang
procesgang
utility(s)
programma
file

5.1.2 Lussen met timeouts

Het bovenbeschreven filter kan in de praktijk vrijwel altijd gebruikt worden. Toch kan het gemakkelijk zijn uit de dan gefilterde lussen doorsneden te maken door lussen op een bepaald kenmerk te bekijken.

Zo'n kenmerk kan bijvoorbeeld de aanwezigheid van een timeout in een lus zijn. PANDORA blijkt namelijk erg goed te zijn in het op de verkeerde manier afstellen van bewakende timers. Door fout afgestelde timers kunnen gemakkelijk lussen ontstaan. Over het algemeen is de protocol ontwerper in dat soort lussen minder geinteresseert. Vandaar dat voorzien is in een filter welke lussen met een timeout wegfiltert. Dit filter heb ik filtimo genoemd.

Door het resultaat van deze filtering met de originele 'PAN.LOOPS' file te doorsnijden kunnen ook all loops met een timeout naar voren gehaald worden. Dit kan met het unix 'diff' commando.

NB

Als door een verkeerd afgestelde timer een deadlock ontstaat dan is dit natuurlijk wel uiterst belangrijk. Dit dient ten alle tijde voorkomen te worden.

5.1.3 Dynamische deadlocks

De door pan geanalyseerde deadlocks staan bekend als statische deadlocks. Er is nog een ander type deadlock bekend; de dynamische deadlock. Als dit type deadlocks in de specificatie aanwezig zijn dan worden deze door 'PANDORA' gemeld in de 'PAN.LOOP' file. Ze moeten vervolgens handmatig uit de gemelde lussen gevonden worden. Een dergelijke deadlock zou er als volgt uit kunnen zien.

```
proc A
     B!m;
     do
     :: B?a -> B!einde -> break
     :: B?r -> B!m
     od
end A;

proc B
     do
     :: A?m -> A!r
     :: A?einde -> break
     od;
     A!a
end B.
```

Bij het analyseren worden hierin geen deadlocks gevonden.

Er zit echter wel een (zeer triviale) dynamische deadlock in, welke in de 'PAN.LOOP' file geschreven wordt. Deze deadlock is bewust gecreerd door proces B te veranderen.

```
MESSAGE        A        B

m                  ------>
****
r                  <------
m                  ------>
****
====
```

De processen komen in een loop waar ze nooit meer uit kunnen komen. In dit specifieke geval zou de gebruiker onraad ruiken omdat er residueen aanwezig zijn, en geen 'OUTPUT' file. Bij een groter protocol zal dit echter meestal niet het geval zijn.

Het is mogelijk een programma te schrijven dat dergelijke deadlocks uit de PAN.LOOP file opspoort. Een dergelijk programma zou een welkome aanvulling zijn op het PANDORA software pakket [4].

5.2 Postprocessing van deadlocks

Het naverwerken van deadlocks vergt voorlopig nog menselijke tussenkomst. Een deadlock manifesteerd zich lang niet zo duidelijk als bijvoorbeeld een loop. Een deadlock manifesteert zich meestal doordat ergens in de berichtenvolgorde iets gebeurt wat de processen bericht-technisch gezien uit balans brengt. Deze gebeurtenis kan bijvoorbeeld zijn het kruisen van berichten in verschillende buffers of het binnenkomen van een timeout op het moment dat een kwijtend bericht onderweg is.

Het streven is in de toekomst een filter te bouwen dat de deadlocks analyseert op hun oorzaak, en deadlocks met dezelfde oorzaak bij elkaar schraapt. Per oorzaak wordt vervolgens slechts een volgorde van een deadlock-oorzaak uitgevoerd met de werkelijke oorzaak gemarkeerd. Zoiets is in het schema getekend.

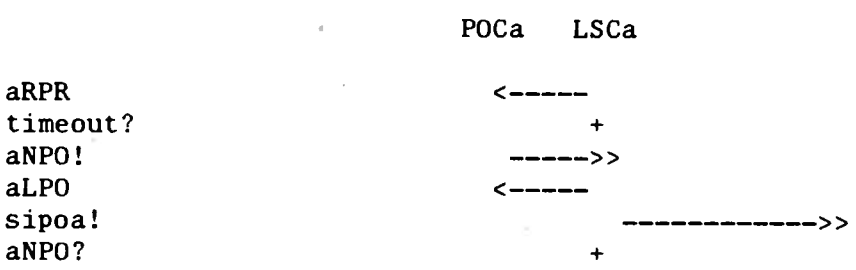
Het bouwen van een dergelijk filter is een bijzonder interressant punt van onderzoek maar zal zeer moeilijk blijken te zijn. Het zal voorlopig ook wel optioneel blijven. Tot die tijd is de gebruiker op zelf te specificeren filters aangewezen.

Moeiljkheid lijkt te zijn dat het zelf handmatig te specificeren filter zich ergens midden in 'pan' bevindt. Dit is op te lossen door 'pan' te splitsen in een deel 'jan' (!) (just analyses) welke 'PAN', 'statespace' en 'ncompct' bevat en een deel 'showdeadlocks' welke 'translate' en 'graff' omvat. Jan produceert alle 'pan' files behalve 'CHARTS', de laatste wordt door 'showdeadlocks' gemaakt. Voordat het

'showdeadlocks' commando gegeven wordt kan eerst de 'PAN.DEAD' file gefiltert worden, omgezet in een nieuwe kleinere 'PAN.DEAD' file. Het filteren gaat als volgt;

Op basis van de gevonden oorzaak wordt het filter gebouwd. Dit kan heel gemakkelijk met de 'sed'-utility van UNIX. Een voorbeeld.

Stel in de 'CHARTS' file wordt de volgende karakteristieke oorzaak van een deadlock gevonden (voorbeeld aan de praktijk ontleend);



De eigenlijke oorzaak ligt in het op dit moment in de procesgang kruisen van de berichten aNPO en aLPO. Dit 'CHARTS' deel zag er in de oorspronkelijke 'PAN.DEAD' file als volgt uit;

....,130,30,1133,127,120,20,136,27,....

Tussen deze berichten door kan in andere processen nog wat gebeuren zodat de 'CHARTS' van de deadlock wat dit betreft er niet altijd hetzelfde uit hoeven te zien.

Het filter dat de kruisende berichten 127,120,20,136,27 uit de volgende deadlockvolgorden haalt ziet er met 'sed' als volgt uit:

```
sed 's/.*,127,120,20,136,27,.*//' 'PAN.DEAD' > PAN.FIL
```

Daarmee zijn alle deadlockvolgorden waarin deze deadlock zich door deze berichtenvolgorde manifesteerd, eruit gefilterd. Ze zijn vervangen door een lege regel, en om deze te verwijderen moet nogmaals gefilterd worden. Hiervoor is het filter nlrm (new-line remove) gemaakt. Het totale filter ziet er nu als volgt uit:

```
sed 's/.*,127,120,20,136,27,.*//' PAN.DEAD | nlrm > PAN.FIL
```

Om de file-administratie te vergemakkelijken is in een aantal extra commando's voorzien. Om de naamgeving van de PAN.DEAD files wat te vergemakkelijken is in een 'showdl' command voorzien welke van een argument voorzien dient te worden waarin de om te zetten file staat. Een dergelijke afgeleide is er ook voor het 'peep' commando in de vorm van het 'pp' commando. Aan 'pp' moet als argument de te peepen file meegegeven worden.

Voor de originele 'PAN.DEAD' file weggehaald wordt moet eerst wel gezorgd worden dat er een voorbeeld van het deadlock in grafische (CHARTS) of code ('PAN.DEAD') vorm bewaard blijft. Als dat de eerste volgorde van CHARTS.p is dan kan dit gemakkelijk met de volgende commando's gedaan worden;

```
sed 'q' PAN.DEAD > PAN.DL1
showdl PAN.DL1
mv CHARTS.d CHARTS.DL1
```

Hiermee is de administratie ook in orde.
Waarschijnlijk zijn met het filter nog niet alle deadlock-volgorden met deze oorzaak weggehaald. De oorzaak van de deadlock kan in meerdere berichten-volgorden tot uiting zijn gekomen. Die andere berichten volgorden moeten naar voren komen in het resultaat van de filtering. Naar aanleiding van deze nieuwe berichten-volgorden moeten op dezelfde manier weer nieuwe filters gemaakt worden en op de overgebleven deadlocks losgelaten worden. Door dit verschillende keren te doen ontstaat een iteratief filterproces dat uiteindelijk tot -indien aanwezig- een andere deadlock of een lege 'PAN.DEAD' file leidt.

Om het vertalen van de 'CHARTS' file in code te vergemakkelijken is in een 'showcode' commando voorzien welke bij iedere regel in de 'CHARTS' aangeeft om wat voor code het gaat. Voor de duidelijkheid komen hierin echter geen enkele pijltjes meer voor, waardoor dit soort 'CHARTS' voor normaal gebruik minder geschikt zijn. De namen van de berichten worden snel afgebroken omdat ze te lang zijn (hier maximaal 4). Een 'CHARTS' uit deze file ziet er bijvoorbeeld als volgt uit:

```
130 aRPR!                    <<-----
30  aRPR?                   +
1133time?                         +
127 aNPO!                    ----->>
120 aLPO!                    <<-----
20  aLPO?                   +
136 sipo!                           ------------>>
27  aNPO?                         +
```

De gezochte code is nu dus direct af te lezen.

Wanneer de verwerking gescheiden geschiedt zal de analyse vrijwel altijd op de achtergrond gebeuren vanwege de lange verwerkingstijd. Het 'jan' commando draait daarom altijd op de achtergrond in 'nohup' mode, opdat de analysator tussendoor kan uitloggen.

6. TOEVOEGING VAN TOESTANDEN

6.1 Inleiding

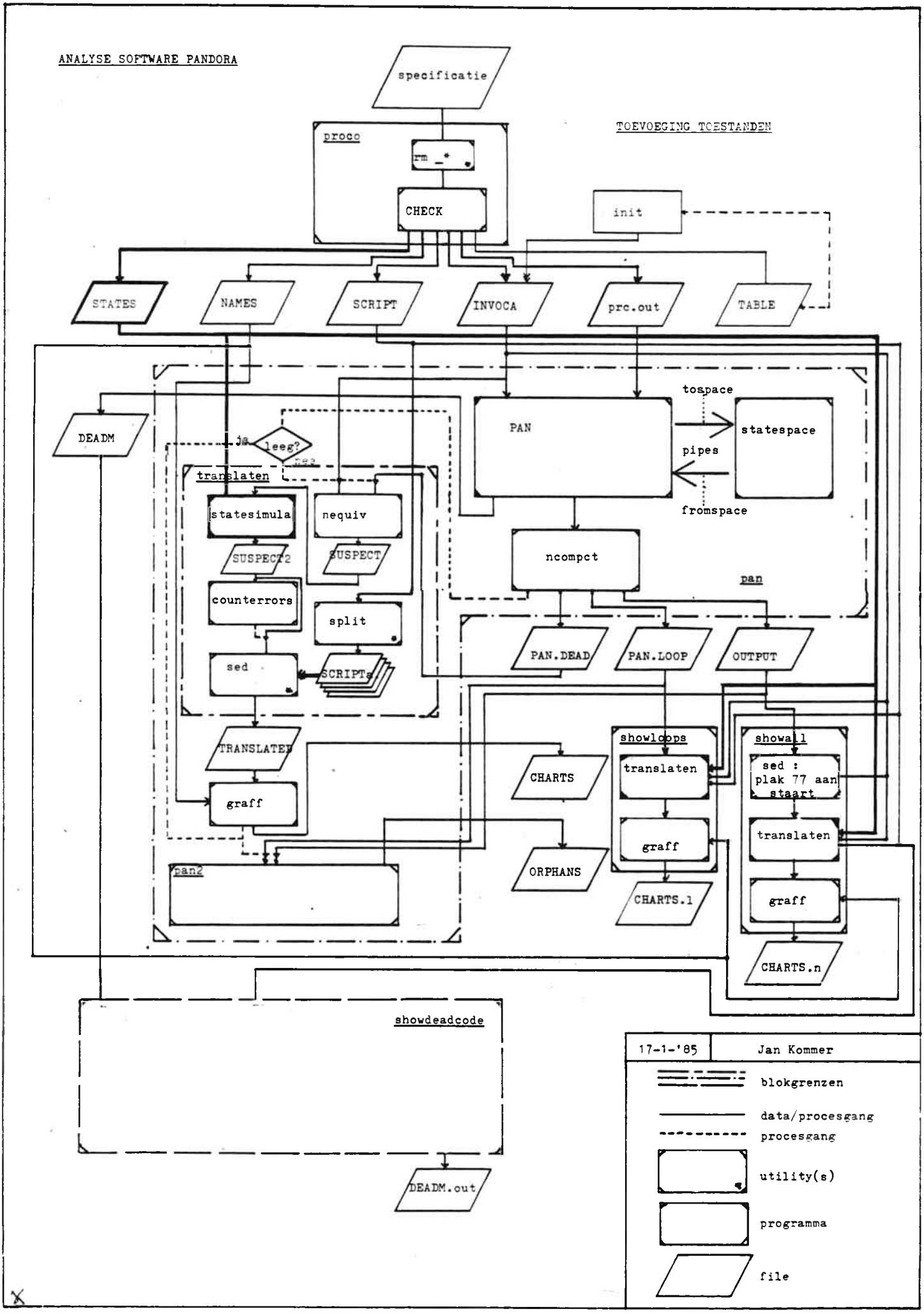
De specificatietaal van PANDORA, PSL, kent geen toestanden. De specificatie waarop het PSL-model gebaseert is kent echter vaak wel toestanden. Op zich is dit geen groot technisch bezwaar, door bepaalde berichtstructuren in de specificatie te leggen gedraagt de specificatie zich precies als het origineel. Dat PANDORA de toestanden niet nodig heeft kan alleen maar als voordeel worden gezien, het zal de analyse met een dimensie verlichten. Pan weet evengoed de deadlocks te vinden.

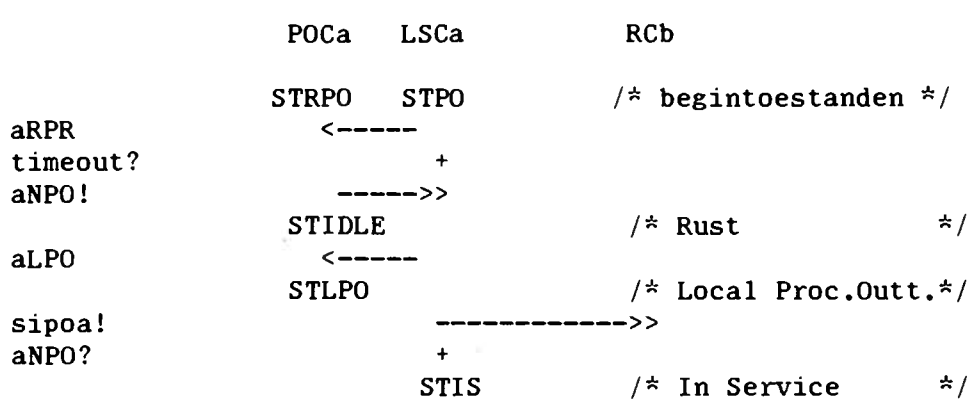
Voor de gebruiker is het echter soms wel een bezwaar. Deze denkt in toestanden en moet de toestandsloze uitvoer van 'pan' interpreteren. Voor de gebruiker zijn toestanden soms onontbeerlijk; de berichten kunnen in verschillende toestanden andere betekenis hebben en andere responses geven. Als een executievolgorde gevolgd wordt, bij het interpreteren van CHARTS, blijkt het niet zichtbaar zijn van de verschillende toestanden een ernstig bezwaar te zijn.

In dit hoofdstuk doe ik een voorstel omtrend de toevoeging van toestanden, met een technische beschrijving erbij. Voor de realisatie moeten op voor pan transparante wijze aan PSL toestanden worden toegevoegd. Hierdoor wordt de gebruiker namelijk niet gedwongen altijd toestanden te definieren, de structureringsmogelijkheden van PSL zouden dan teniet gedaan zijn (H 9.7). Toestanden zijn aardig voor het begrip maar horen verder overbodig te zijn (H 5.3).

6.2 Technische uitvoering

Omdat 'pan' geen toestanden kent, niet nodig heeft en het analyse programma dermate complex is dat wijzigen ervan onnodig veel tijd zal vergen, worden de toestanden om de eigenlijke analyse geleid. Dit is een weg die met de hand ook te volgen is. Met behulp van de specificatie is de uitvoer van toestanden te voorzien. Tussen de weergegeven berichten wordt met de hand de veranderingen in de toestand van de processen weergegeven. Tot slot kunnen dan de eindtoestanden gegeven worden. Zoiets komt er dan als volgt uit te zien:


ANALYSE SOFTWARE PANDORA
specificatie
proco
rm _*
CHECK
init
TOEVOEGING TOESTANDEN
STATES
NAMES
SCRIPT
INVOCA
prc.out
TABLE
DEADM
ja
leeg?
nee
translaten
statesimula
nequiv
SUSPECT2
SUSPECT
counterrors
split
sed
SCRIPTs
TRANSLATED
graff
pan2
PAN
tospace
statespace
pipes
fromspace
ncompct
pan
PAN.DEAD
PAN.LOOP
OUTPUT
showloops
translaten
graff
CHARTS.1
showall
sed :
plak 77 aan
staart
translaten
graff
CHARTS.n
CHARTS
ORPHANS
showdeadcode
DEADM.out
17-1-'85
Jan Kommer
blokgrenzen
data/procesgang
procesgang
utility(s)
programma
file



Dit kan wat onduidelijk overkomen omdat de specificatie van de processen niet bekend is. Het vergt te veel papier om beschrijven wat hier gebeurt (H 9). Het kan echter al duidelijker zijn dat er sprake is van een fout, als gezegd wordt dat proces LSCa alleen in toestand 'in service' (STIS) mag zijn als proces POC in de toestand 'idle' (STIDLE) is. Daarmee is het al duidelijker waarom dit een deadlock is als bij de uitleg van het bouwen van een deadlockfilter (daar betrof het hetzelfde voorbeeld).

Door de toevoeging van de toestanden is de leesbaarheid enorm verhoogd. Voor degene die aan het analyseren is is in een oogopslag te zien wat er aan de hand is en wat er gebeurt. Voordat ik ging kijken wat er werkelijk gebeurde in de beschreven volgorde liep ik alle processen af, van boven naar beneden. De toestandsveranderingen werden naar aanleiding van de ontvangen en verzonden berichten bijgeschreven. Pas daarna was namelijk te zien wat er nu eigenlijk gebeurde. Dit is enorm tijdrovend werk wat gemakkelijk door de computer overgenomen kan worden.

De computer moet daartoe de volgorden op dezelfde manier volgen als met de hand gebeurt. Er zal daartoe informatie aan de specificatie onttrokken moeten worden welke de berichten aan de toestanden koppelt. Met deze informatie moet een programma aan het werk die bij het vertalen van de PAN.* code naar 'CHARTS' de toestanden toevoegt. Dit is in de figuur weergegeven.

Het programma dat de toestanden toevoegd is daarin statesimula genoemd en neemt de plaats in van de script-module die binnen 'translate' de file 'SUSPECT' in de file 'SUSPECT1' omzet. De oorspronkelijke functies zijn in het blok statesimula ook vervuld.

Dit punt is volgens mij het makkelijkst te gebruiken voor de gestelde doelen. Er behoeven dan op andere punten geen wijzigingen te worden aangebracht.

De huidige 'SUSPECT' en 'SUSPECT1' files zien er als volgt uit:

```
SUSPECT:
114,14,110,%
114,14,115,15,114,110,%

SUSPECT1:
114
14
110
%
114
14
115
15
114
110
%
```

De laatste zal er na 'statesimula' als volgt uitzien;

```
114
14
A->A:STIA
110
%
114
14
A->A:STIA
115
15
114
B->B:STAR
110
%
```

Bij de verdere verwerking in 'translate' zullen de tussengevoegde regels onberoerd blijven staan en in de 'TRANSLATED' file terecht komen. Graff gebruikt deze 'TRANSLATED' en maakt er het volgende van.

```
MESSAGE      A      B
k             ------>
STIA         +     .
m             ------>>
```

Door het in de beschreven vorm toevoegen aan de 'SUSPECT' file en de 'TRANSLATED' file behandeld 'graff' de toestanden als een timeout. Deze worden in 'CHARTS' ook met het bovenstaande plusje gepresenteerd. Door wijzigingen in 'graff' aan te brengen is dit te veranderen in de volgende CHARTS;

```
MESSAGE      A      B
k             ------>
           STIA
m             ------>>
```

Als deze verandering wordt aangebracht zullen timeouts op dezelfde manier worden afgedrukt.

De informatie die het programma statesimula nodig heeft staat in de file 'STATES', welke door 'proco' gegenereerd wordt. 'STATES' bestaat uit een of meerdere tabellen welke ingelezen worden door het programma statesimula. Statesimula kan m.b.v. de tabellen heel gemakkelijk de procesgang volgen. Hoe de tabellen precies gedefinieerd worden is een keuze die samenhangt met het gekozen algorithme in het programma statesimula.

6.3 Toevoeging van toestanden aan PSL

De toestanden welke ik voorsta zijn reeds beschreven in hoofdstuk 9.6. De daar gebruikte toestanden zijn transparant voor pan, een van de gestelde eisen. Het is ook mogelijk andere construcies voor toestanden te bedenken, mits deze ook met het huidige pan programma verwerkt kunnen worden.

6.4 Nieuwe mogelijkheden door toevoeging van toestanden

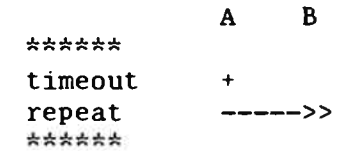
6.4.1. Deadlockdetectie

In het voorgaande hoofdstuk betreffende Post-processing is reeds gesproken over een programma welke automatisch de oorzaak van de gevonden deadlocks detecteerd en een programma dat uit de loops de dynamische deadlocks haalt.

Het maken van dergelijke programma's zal zeer vereenvoudigt worden als in PSL toestanden gedefinieerd zijn.

Voor het opsporen van deadlocks zal een behoorlijk gecompliceerd algorithme bedacht moeten worden, voor het opsporen van dynamische deadlocks is minder intelligentie benodigt. Een voorbeeld hoe dit zou kunnen werken.

Stel voor de eenvoud een specificatie bestaande uit twee processen. Een gedetecteerde loop zal minstens uit een zend bij het ene proces en een ontvangst bij het andere proces bestaan. In dat geval is er sprake van wat een zo genaamd gedwongen geef systeem.



Meestal zal er echter sprake zijn van meerdere zend- en

ontvangst handelingen. In de loop worden in beide processen cyclisch toestanden doorlopen of blijven een of beide processen in een en dezelfde toestand.

Er is sprake van een dynamische deadlock als in beide processen
in alle fasen van de loop geen andere mogelijkheden bestaan als de loop te blijven volgen, d.w.z. er zijn geen ontsnappingsmogelijkheden. Als er sprake is van twee processen is dit gemakkelijk te detecteren.

Als bij het doorlopen van de loop geen select statement gepasseerd wordt (er vanuit gaande dat het selectstatement alleen maar wordt gebruikt voor het kiezen uit verzendhandelingen) is er geen mogelijkheid uit de loops te komen, tenzij er in een van de twee processen spontaan een verzending kan plaatsvinden. In de toestanden die ik definieer kan alleen een spontane verzending plaatsvinden na een timeout. Er hoeft dan enkel gelet te worden op de timeouts in de gepasseerde toestanden.

Voor het detecteren van een dynamische deadlock moet gedurende de loop de specificatie gevolgd worden en gekeken worden naar de aanwezigheid van timeouts in de geldende toestanden en het passeren van select statements.

Het voorbeeld uit het vorige hoofdstuk kan hierin illustratief zijn.

```
proc A
    B!m;
    do
    :: B?a -> B!einde -> break
    :: B?r -> B!m
    od
end A;

proc B
    do
    :: A?m -> A!r
    :: A?einde ->  break
    od
    A!a
end B
```

Deze specificatie heeft de volgende loop-melding tot gevolg;

```
MESSAGE       A     B
m             ------>
****
r             <------
m             ------>
****
```

Binnen de loops is voldaan aan de eis dat in do-loops alleen berichten ontvangen worden. Dit is een dynamische deadlock omdat er geen ontsnappings mogelijkheden uit de lus zijn. Als we de specificatie veranderen in;

```
proc A
    B!m;
    do
    :: B?a -> B!einde -> break
    :: B?r -> B!m
    od
end A;

proc B
    do
    :: A?m;
         if
         :: A!a
         :: A!r
         fi
    :: A?einde -> break
    od
    A!a
end B
```

dan wordt dezelfde loop nog steeds gemeld maar is geen dynamische deadlock meer, er is door het select-statement een ontsnappings mogelijkheid gecreerd.

Tegen de gevolgde redeneringen is in te brengen dat de aanwezigheid van select-statements en timeouts nog geen ontsnappingsmogelijkheden garanderen. Dit klopt. In de tests moet ook gekeken worden of de binnen de if-fi statement of timeout gegenereerde andere berichten ook ontvagngbaar zijn in de geldende toestanden van het proces, en of die ontvangsten een actie tot gevolg hebben die de processen buiten de loop brengen. In het voorbeeld klopt dat.

Als sprake is van meer processen wordt de procedure aanzienlijk ingewikkelder. Er zijn dan twee soorten processen; processen binnen de loop en processen daarbuiten. Voor de processen in loop gelden onderling dezelfde criteria als voor twee processen. De buitenliggende processen oefenen ook invloed uit op de loop. Deze invloed is veel moeilijker om

te analyseren. De uitwendige processen gaan verder terwijl de loop blijft doorgaan. De loop kan op zeker moment gestopt worden door een uitwendig proces. Dit kan bedoeld zijn; bijv. een wachtlus. Het is echter zeker alarmerend als de uitwendige processen de loop ook niet kunnen stoppen.

Mij lijkt het voldoende om bij het opsporen van dynamische deadlocks alleen naar de bij de loop betrokken processen te kijken. Als hieruit reeds naar voren komt dat de loop niet uit zichzelf stopt, moet de loop gemeld worden. De protocol ontwerper zal het zeker willen weten.

6.4.2. Uitvoer eindtoestanden

Van de door pan geproduceerde uitvoerfiles kunnen specifiek op toestanden gerichte gegevens verstrekt worden. Dat kunnen voor de gewone volgorden de opvolgende toestanden zijn, en voor de deadlocks ook de eindtoestanden. Dit geeft globalere informatie als de eigenlijke berichten stroom.

Om de gebruiker tegen verkeerde opstart procedures te waarschuwen is nog een andere mogelijkheid in te bouwen. Als in het begin van de analyse de PAN.DEAD file volloopt zonder dat de OUTPUT file iets bevat, dus alle volgorden als deadlocks gemeld worden, moet een waarschuwing gegeven worden. Over het algemeen zal dit namelijk betekenen dat de analyse verkeerd opgestart is of dat de analyse onbedoeld niet klopt.

LITERATUURLIJST

[1]	notes.pan	G.J.Holzmann, oktober 1983. Intern rapport A.V.S.
[2]	notes.proco	G.J.Holzmann, oktober 1983 Intern rapport A.V.S.
[3]	Manual for the Pandora protocol analyser	G.J.Holzmann, augustus 1983 rapport no. 44 A.V.S.
[4]	Using automated validation techniques to detect lock-ups in packet switched networks.	M.Sherman IEEE Trans. on comm. Vol C-28 juli 1982
[5]	A theory for protocol validation	G.J. Holzmann, augustus 1982 IEEE Trans. on Computers, C-31, 8 ,p.730-738
[6]	Concise description of a protocol validation algebra.	G.J.Holzmann, rapport 38, A.V.S.

Appendix G

Ervaringen met PANDORA

Tijdens mijn afstudeerwerk is enige ervaring opgedaan met het maken van te analyseren PSL-modellen. Veel van deze ervaring is terug te vinden in hoofdstuk 9 of is reeds eerder beschreven in de PANDORA handleiding [107] en PANDORA voorbeelden [108]. Een punt, betreffende processoorten, komt daarin echter niet aan de orde. In het eerste hoofdstuk van deze appendix wordt hierop ingegaan.

Aan PANDORA valt nog een heleboel te sleutelen, en is nog een heleboel onderzoek te verrichten. Deze punten komen in het tweede hoofdstuk naar voren. Tot slot worden in het derde hoofdstuk enige suggesties gegeven voor verder onderzoek naar protocol-ontwerp in het algemeen.

1.proces-modellen specificeren, analyse opstarten

De proces specificaties in PSL kunnen zowel cyclisch als eindigend opgesteld worden. Een eindigend proces is na doorlopen definitief klaar, een cyclisch proces kan vele malen herhaald worden. Een cyclisch proces kan bijvoorbeeld gespecificeerd worden met een 'goto BEGIN' na de laatste instructie of kan een grote do-loop zijn, hetgeen beide betekent dat het proces herhaald doorlopen wordt.

Een model met alleen eindigende processen is het normale model voor de analyse. De processen zullen aan het begin van de analyse tegelijk opgestart worden.

Het kan echter moeilijk of onmogelijk zijn de processen allen eindigend te specificeren. Zo kan het zijn dat een proces uit een specificatie gedurende een analyse meermalen doorlopen moet worden, tegenover andere processen in die specificatie die slechts een keer doorlopen hoeven worden.

Een model waarin niet eindigende (cyclische) processen beschreven worden kan slechts onder een voorwaarde geanalyseerd worden: alle gespecificeerde processen in het model moeten niet eindigend zijn. De analyse zal bij een dergelijk model als gevolg van het interne reduktie mechanisme van PANDORA beeindigd worden. Dit mechanisme zorgt ervoor dat de beeindiging correct verloopt en geen fouten zal geven.

Als naast elkaar eindige en cyclische processen in het model zijn opgenomen, dan zal een groot deel van de gegenereerde volgorden als deadlock gedetecteerd worden, zonder dat er werkelijk van deadlocks sprake is.

Daar alle processen cyclisch gespecificeerd worden moeten er speciale maatregelen genomen worden om de analyse op te starten, en om (minstens) een proces kunstmatig niet

cyclisch te maken (bekorting analyseduur). Hiertoe voegt men een 'start'-bericht afkomstig van een fictief 'init'-proces aan de specificatie van het eenmalig te doorlopen proces toe. Het eenmalig te doorlopen proces is cyclisch gespecificeerd, maar wacht in het begin van zijn specificatie altijd op het bericht 'start' van het 'init'-proces. Dit start bericht wordt voor het opstarten van de analyse met het 'init'-commando in de initiele wachtrij geplaatst. Het init proces wordt in de werkelijke specificatie niet als proces gespecificeerd, hetgeen als warning bij de compilatie met proco gemeld zal worden (warning negeren). Bovenstaande wordt met een voorbeeld geillustreerd.

Het eenmalig te doorlopen proces:

```
proc beheer
STOOS:
        init?start -> rest!start ;
        do
        enz.
        .
        .
        ..... -> goto STOOS
        od
end beheer;
```

Bij de compilatie van deze specificatie zullen de volgende (te negeren) warnings gegeven worden:

```
Warning: init->beheer: start is received but not send.
Warning: unspecified process init
```

Voor de analyse met pan opgestart wordt moet met het 'init' commando het bericht 'start' in de initiele wachtrij van proces beheer gezet worden.

2. Verder onderzoek rond PANDORA en protocolontwerp

Vele van te maken opmerkingen over PANDORA zijn het gevolg van de niet juiste analyse van een model met meer processen. Deze opmerkingen worden hier niet vermeld.
De belangrijkste opmerkingen (beperkingen) over PANDORA zijn reeds in het afstudeerverslag van Onno de Ruiter [102] te vinden. Bij de meeste van zijn opmerkingen sluit ik mij aan. Een opmerking wil ik daar echter aan toe voegen.

De beperkte capaciteit van het huidige PANDORA systeem is bekend. Toch moet men geen al te hoge verwachtingen hebben van een PANDORA-implementatie op een krachtiger computer. De analyse zal zeer veel sneller verlopen, maar zal niet zeer veel soepeler verlopen. Er zullen dezelfde moeilijkheden zijn als bij de huidige PANDORA-implementatie; men dient

goede modellen op te stellen en er blijft veel handwerk te doen. Grote modellen blijven daardoor onaantrekkelijk. Een krachtiger computer zal wel veel gerbuikersvriendelijker zijn, de analysetijden zijn nu veelal onaanvaardbaar lang. Dit blokkeert de voortgang, gedurende de analysetijd moet men noodgedwongen iets anders oppakken en verliest op deze manier het gevoel voor het te analyseren model.

2.1 Toevoegingen

Er zijn enige dingen die aan PANDORA verandert of toegevoegd kunnen worden. Enige daarvan staan reeds uitvoerig beschreven in appendix F.

a. Naverwerkfilters
Filters die de uitvoer comprimeren tot aanvaardbare proporties (Appendix F, hoofdstuk 5).
I.Naverwerk-commando's voor loops.
Programma dat dynamische deadlocks opspoort en een programma dat meervoudig gemelde loops enkelvoudig doorgeeft ('filloops'-reeds gerealiseerd).
II.Naverwerk-commando's voor deadlocks
In hoofdstuk 5 van Appendix F wordt beschreven dat de naverwerking van deadlocks gedeeltelijk met de hand dient te geschieden. Er dient dan wel in een 'showdl' commando en een eventueel 'showcode' commando voor algemeen gebruik voorzien te worden. Uit tijdbesparing kan eventueel in een 'jan' (just analyses) commando voorzien worden (reeds gerealidseerd).

b. Facultatieve toestand definitie voor PSL
Modellen met (verzonnen) toestanden worden makkelijker interpreteerbaar indien toestandtransities aan de CHARTS toegevoegd worden (Appendix F, hoofdstuk 6).

c. Bepaalde parameters (MAXLENGTH, MAXDEPTH) van pan zouden buiten pan gedefinieerd moeten kunnen worden (zonder te hercompileren). Deze parameters zouden bijvoorbeeld in een voor iedere gebruiker te definieren 'pan.ini' file vermeld kunnen worden, welke ingelezen wordt bij elk gegeven 'pan' commando. Ook is het mogelijk deze waarden optioneel, in de vorm van een vlag, met pan mee te geven.

d. Indien bij het opstarten van de analyse niet snel genoeg gewone uitvoer in de file 'OUTPUT' verschijnt en de files PAN.LOOP en PAN.DEAD wel geschreven worden, dient een waarschuwing op het beeldscherm te verschijnen. De opstart procedure is dan niet goed verlopen, het model klopt niet.

e. Er zou voorzien kunnen worden in een commando dat de analyse kan pauseren, zonder dat er gegevens verloren gaan. Hiermee kan men tussendoor de analyse op verantwoorde manier stoppen. Dit kan overdag, als er meerdere gebruikers zijn, de computer aanzienlijk ontlasten, en biedt ook de

mogelijkheid om tussendoor wat sneller andere analyses uit te voeren. Dit commando moet dus vooral gezien worden in het licht van een beter computer-capaciteits beheer.

f. Als het in PSL gespecificeerde model symmetrisch is (twee gelijke processen), dan zullen alle volgorden, evenals de fouten, ook symmetrisch zijn. Men zou daarom kunnen volstaan met melding van de helft van het aantal volgorden. Dit reduceert de hoeveelheid volgorden met een factor twee. Bij de ontwikkeling van een dergelijk commando moet men uitgaan van speciaal aan de specificatie toe te voegen kenmerken en tekens.

g. De transitie tabellen van Yeu [103] kunnen verder gereduceerd worden met de theorie die in appendix C gegeven is [5]. (eventueel op beperkte schaal)

h. De programma's die transitietabellen en SDL-diagrammen produceren moeten bijgeschaafd worden. Beide kunnen geen grote modellen aan, beide zullen daartoe opgesplitste uitvoer moeten kunnen genereren. Voor de SDL-diagrammen is het in bepaalde gevallen bijvoorbeeld eenvoudige mogelijk een toestand per blad aan te nemen. De uitgevoerde SDL-diagrammen kunnen bovendien een beter aanzien gegeven worden (eventueel kunnen de toestanden (appendix F) daarbij verwerkt worden).

i. Er wordt op het moment van schrijven gewerkt aan een SDL-PSL compiler. Dit is een welkome aanvulling. PANDORA zal als ontwerpsysteem nog completer worden als daarnaast een goede SDL-editor ontworpen wordt.

2.2 Onderzoek PANDORA

De behoeften uit 3.1 zijn (relatief) gemakkelijk aan PANDORA toe te voegen. Er zijn echter ook enige behoeften die een fundamentelere of uitgebreidere aanpak nodig hebben (volledige ontwerpcyclus...). Deze worden in deze paragraaf vermeld.

a. Onderzoek naar de specificatietaal van PANDORA.
Om een protocol volledig op te ontwerpen is PSL niet zo krachtig, voornamelijk vanwege de niet gedefinieerde variabelen. Die variabelen behoeven niet noodzakelijkerwijs in het validatiemodel te worden opgenomen, het aantal volgorden zal er namelijk zeer snel door groeien, evenals de analyseduur. Bij de nieuwe specificatietaal kan gedacht worden aan een taal als LOTOS [44] of afgeleide van deze taal. Onderzoek kan ook gedaan worden naar toevoeging van variabelen aan het analysemodel (eventueel op beperkte schaal) en naar technieken die de variabelen op een verantwoorde manier buiten de analyse om kunnen leiden.
Een krachtiger specificatietaal zal de kracht van PANDORA als ontwerpsysteem sterk verhogen.

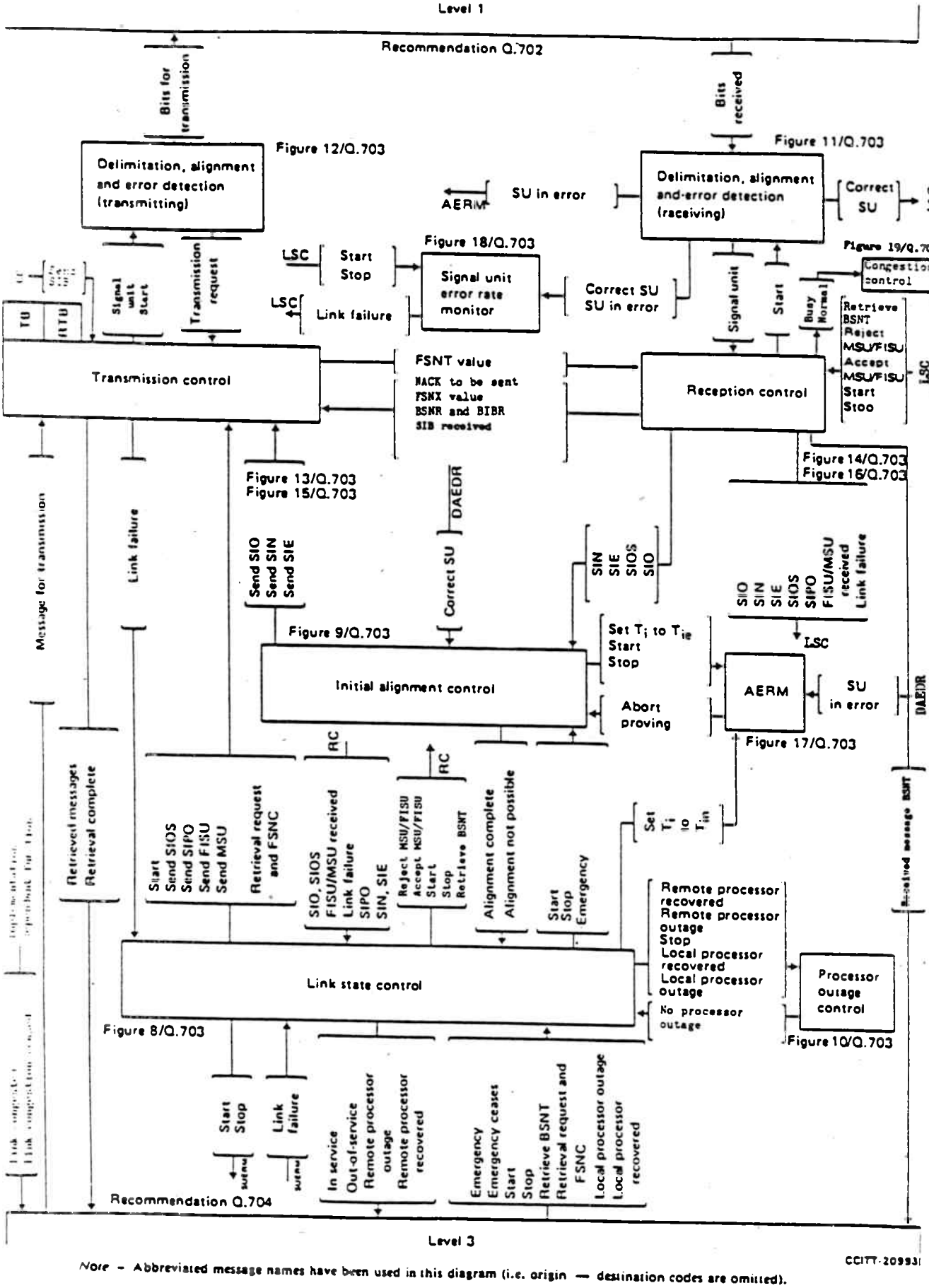
b. De eisen die er aan PANDORA als ontwerpsysteem gesteld moeten worden kunnen gevonden worden, door te pogen een iets complexer protocol op PANDORA te ontwerpen.

c. Er kan onderzoek gedaan worden naar de geschiktheid van specificaties met toestand/stimulus tabellen voor validatie met PANDORA.

d. De consistentie van de validatie-algebra kan onderzocht worden (afstudeeropdracht voor wiskundige o.i.d.). De gevonden foute aanname zou namelijk kunnen betekenen dat er meer verkeerde aannames in de algebra zijn gemaakt. Bij dit onderzoek kan ook de juistheid van het interne reductiemechanisme betrokken worden, evenals de gevolgen van de reductie bij loops (volgordes na loops niet voortgezet). Het analysedeel van pan kan dus resumerend in zijn geheel doorgelicht worden.

3 Onderzoek Protocol-ontwerp

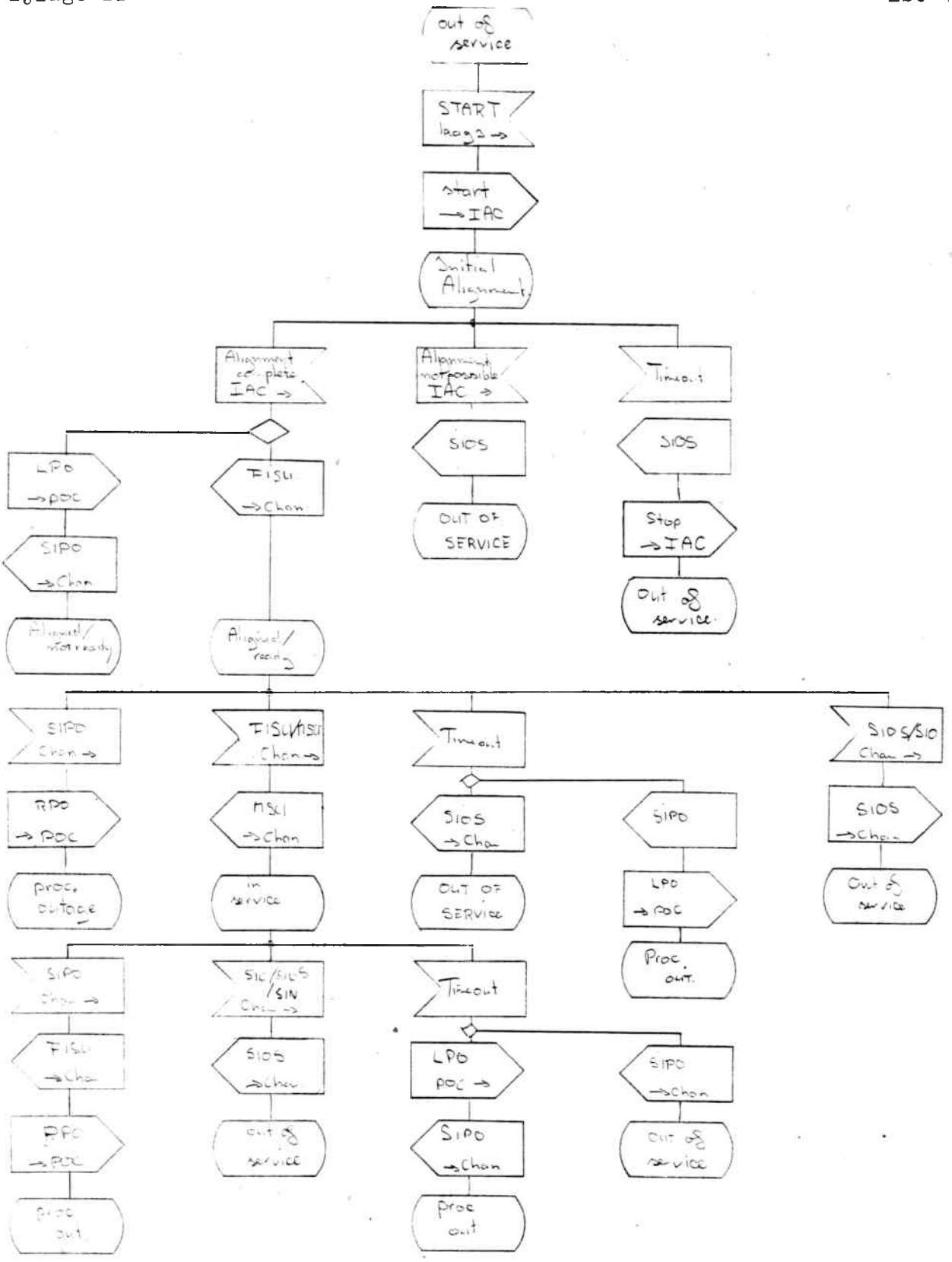
a. De verschillende functionele structuren binnen een laag kunnen op hun invloed op de betrouwbaarheid, en geschiktheid (zinnigheid) voor validatie onderzocht worden. Dit is een vrijwel nieuw vakgebied, waarin nog veel te publiceren is. Dit onderzoek kan zich tot het gehele ontwerpproces richten, en kan uiteindelijk leiden tot een ontwerpmethodiek of een protocolsynthesiser.
b. De relatie functionele structuur - bestek - vrijheid van toegepaste techniek, kan beschouwd worden. Dit zal onder meer tot een beter begrip van protocolspecificaties leiden.

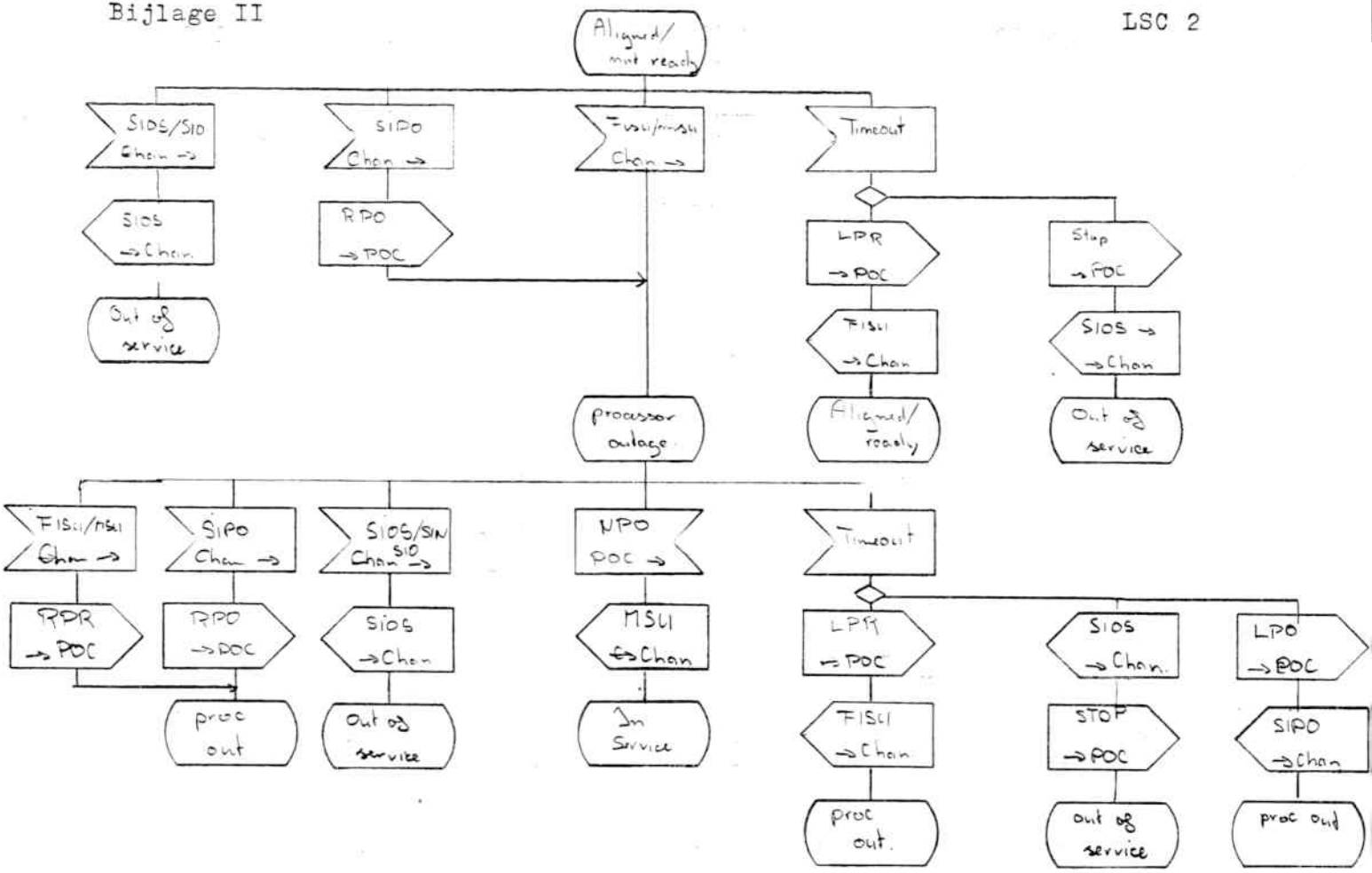


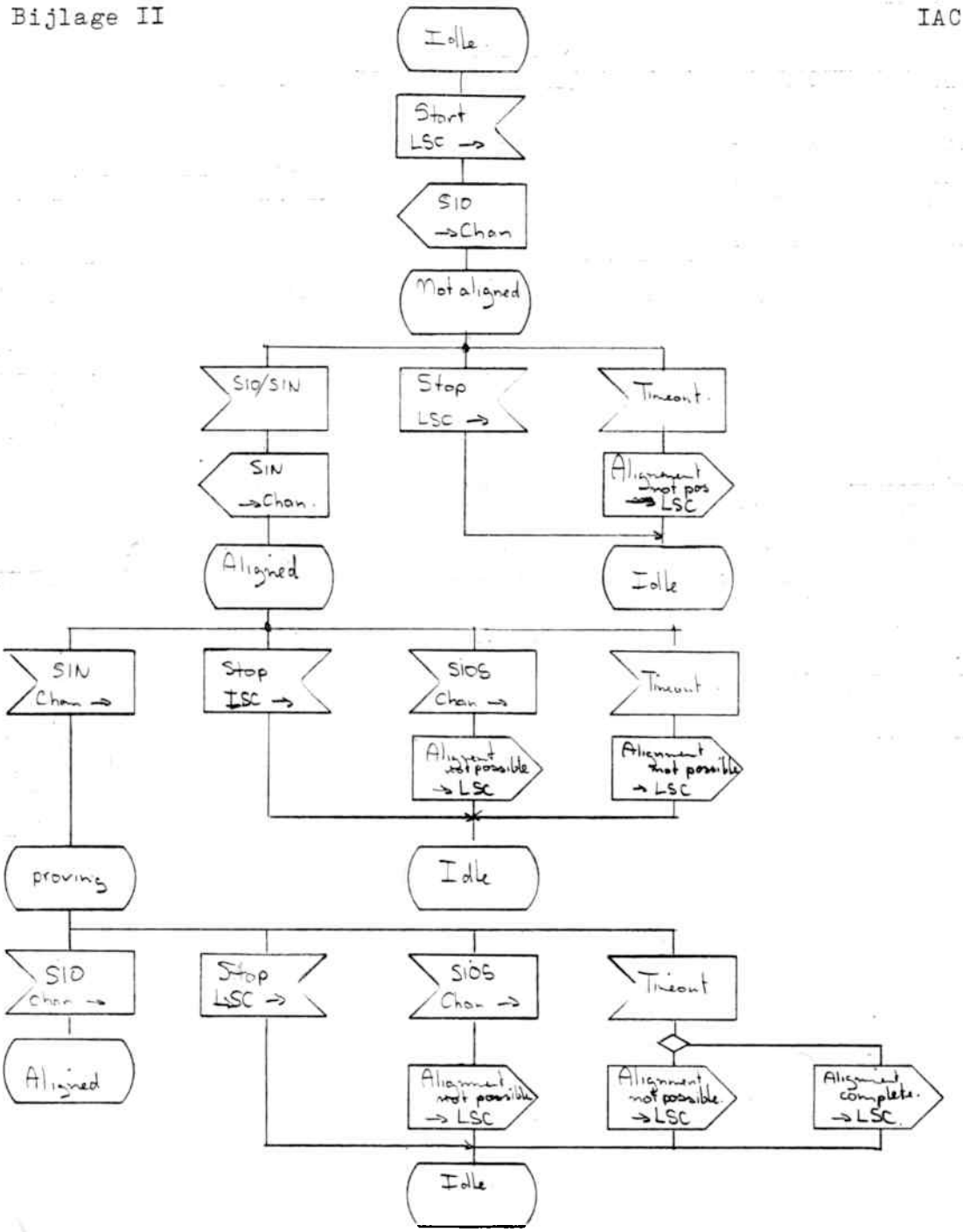
Note – Abbreviated message names have been used in this diagram (i.e. origin — destination codes are omitted).

FIGURE 7/Q.703

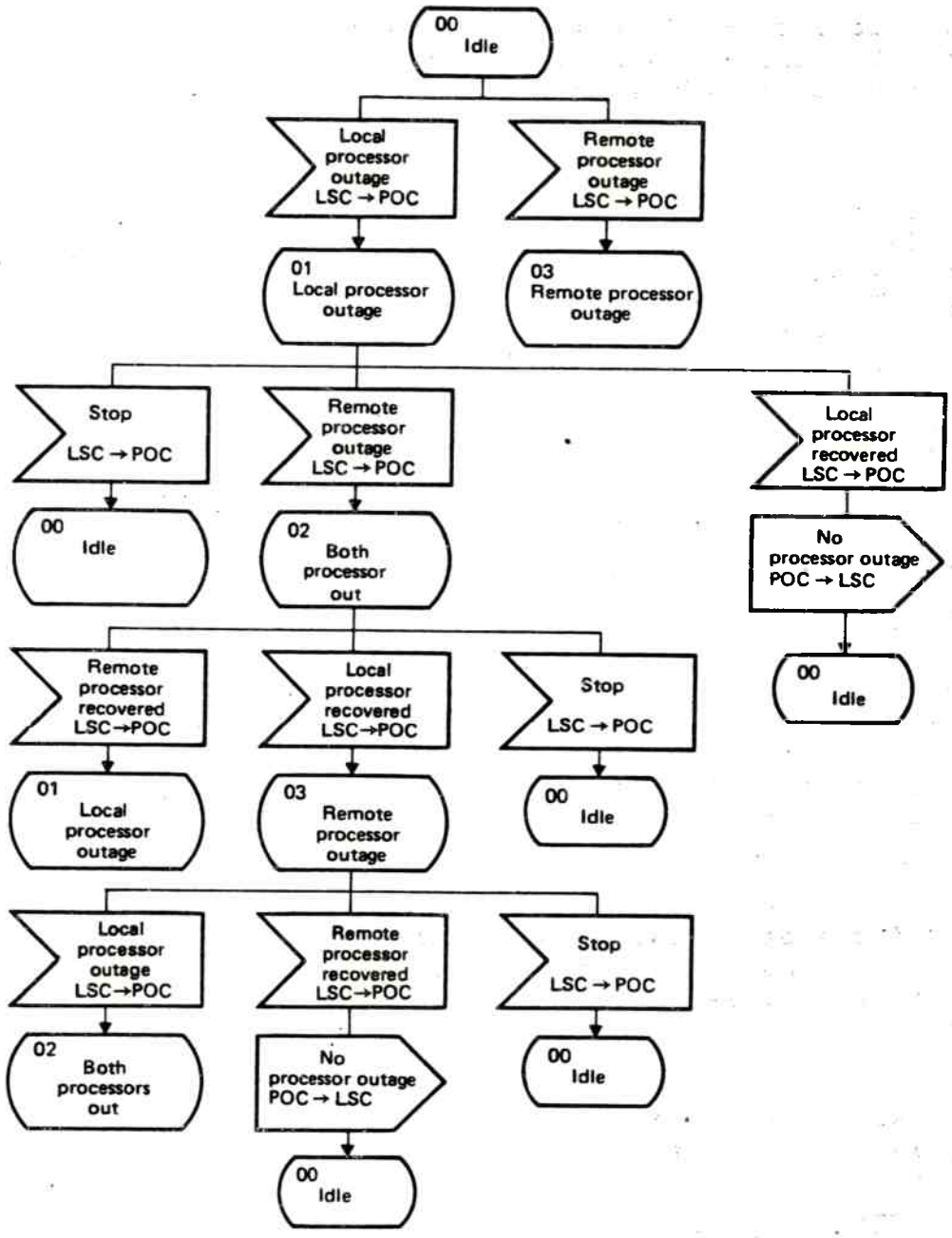
Level 2 - Functional block diagram


out of service
START
start → IAC
Initial Alignment
Alignment not possible IAC →
Timeout
SIOS
OUT OF SERVICE
SIOS
Stop → IAC
Out of service
SIPO
Chan →
SIPO → Chan
FISU → Chan
Aligned/ ready
Timeout
SIOS → Chan
OUT OF SERVICE
SIPO
LPO → POC
Proc. OUT.
SIOS/SIO Chan →
SIOS → Chan
Out of service
LPO → POC
SIPO → Chan
Out of service
Timeout
LPO POC →
SIPO → Chan
Proc out


Aligned/ not ready
SIOS/SIO Chan →
SIPO Chan →
FISU/MSU Chan →
Timeout
SIOS → Chan.
Out of service
RPO → POC
LPR → POC
Stop → POC
FISU → Chan
SIOS → Chan
Aligned/ ready
Out of service
Processor outage
FISU/MSU Chan →
SIPO Chan →
SIOS/SIN Chan →
SIO
NPO POC →
Timeout
RPR → POC
RPO → POC
SIOS → Chan
MSU ⇔ Chan
LPR → POC
SIOS → Chan.
LPO → POC
proc out
Out of service
In Service
FISU → Chan
STOP → POC
SIPO → Chan
proc out.
out of service
proc out

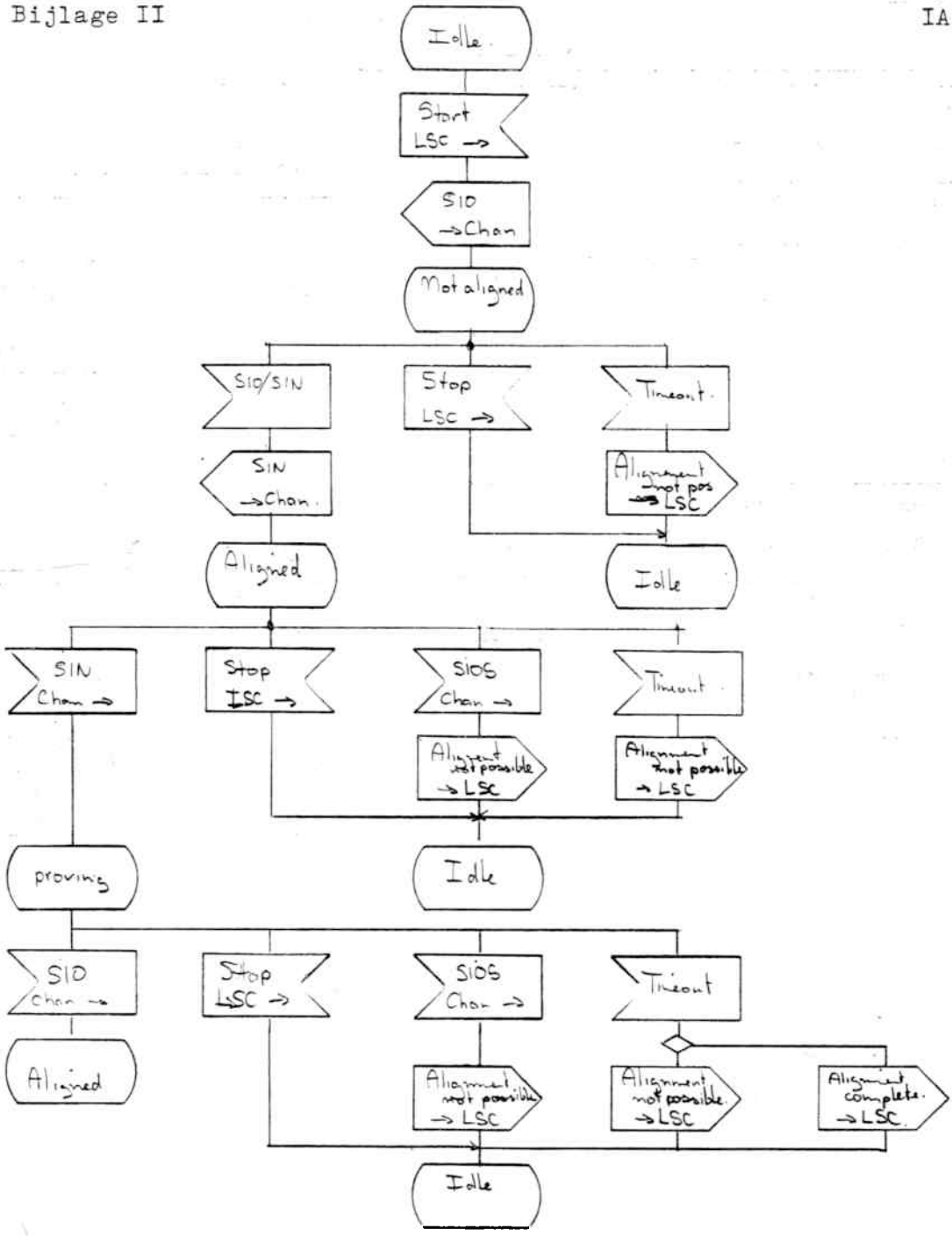

Idle
Start LSC →
SIO →Chan
Not aligned
SIO/SIN
Stop LSC →
Timeout
SIN →Chan.
Alignment not poss →LSC
Aligned
Idle
SIN Chan →
Stop LSC →
SIOS Chan →
Timeout
Alignment not possible →LSC
Alignment not possible →LSC
proving
Idle
SIO Chan →
Stop LSC →
SIOS Chan →
Timeout
Aligned
Alignment not possible →LSC
Alignment not possible →LSC
Alignment complete →LSC
Idle

```
proc POCa
STIDLE:
    do
    :: LSCa?aLPO -> break
    :: LSCa?aRPO -> goto STRPO
    :: default -> skip
    od;
STLPO:
    do
    :: LSCa?aRPO -> break
    :: LSCa?aLPR -> LSCa!aNPO -> goto STIDLE
    :: LSCa?aSTOP -> goto STIDLE
    :: default -> skip
    od;
STBPO:
    do
    :: LSCa?aRPR -> goto STLPO
    :: LSCa?aLPR -> break
    :: LSCa?aSTOP -> goto STIDLE
    :: default -> skip
    od;
STRPO:
    do
    :: LSCa?aLPO -> goto STBPO
    :: LSCa?aRPR -> LSCa!aNPO -> goto STIDLE
    :: LSCa?aSTOP -> goto STIDLE
    :: default -> skip
    od
end POCa;

proc LSCa
STOOS:
    do
    :: init?aSTART -> break
    :: default -> skip
    od;
/*STIA*/
    do
    :: timeout;
      if
      :: LSCb!fisua -> break
      :: POCa!aLPO ; LSCb!sipoa -> goto STANR
      :: LSCb!siosa -> goto STOOS
      fi
    :: default -> skip
     od;
STAR:
    do
    :: LSCb?fisub -> LSCb!fisua -> goto STIS
    :: LSCb?sipob -> POCa!aRPO -> goto STPO
    :: default -> skip
    :: timeout;
      if
      :: POCa!aLPO ; LSCb!sipoa -> break
      :: LSCb!siosa -> goto STOOS
      fi
    od;
```



LSC Link state control
POC Processor outage control

Processor outage control


Idle
Start LSC →
SIO →Chan
Not aligned
SIO/SIN
Stop LSC →
Timeout
SIN →Chan
Alignment not poss →LSC
Aligned
Idle
SIN Chan →
Stop LSC →
SIOS Chan →
Timeout
Alignment not possible →LSC
Alignment not possible →LSC
proving
Idle
SIO Chan →
Stop LSC →
SIOS Chan →
Timeout
Aligned
Alignment not possible →LSC
Alignment not possible →LSC
Alignment complete →LSC
Idle

```
proc POCa
STIDLE:
    do
    :: LSCa?aLPO -> break
    :: LSCa?aRPO -> goto STRPO
    :: default -> skip
    od;
STLPO:
    do
    :: LSCa?aRPO -> break
    :: LSCa?aLPR -> LSCa!aNPO -> goto STIDLE
    :: LSCa?aSTOP -> goto STIDLE
    :: default -> skip
    od;
STBPO:
    do
    :: LSCa?aRPR -> goto STLPO
    :: LSCa?aLPR -> break
    :: LSCa?aSTOP -> goto STIDLE
    :: default -> skip
    od;
STRPO:
    do
    :: LSCa?aLPO -> goto STBPO
    :: LSCa?aRPR -> LSCa!aNPO -> goto STIDLE
    :: LSCa?aSTOP -> goto STIDLE
    :: default -> skip
    od
end POCa;

proc LSCa
STOOS:
    do
    :: init?aSTART -> break
    :: default -> skip
    od;
/*STIA*/
    do
    :: timeout;
       if
       :: LSCb!fisua -> break
       :: POCa!aLPO ; LSCb!sipoa -> goto STANR
       :: LSCb!siosa -> goto STOOS
       fi
    :: default -> skip
     od;
STAR:
    do
    :: LSCb?fisub -> LSCb!fisua -> goto STIS
    :: LSCb?sipob -> POCa!aRPO -> goto STPO
    :: default -> skip
    :: timeout;
       if
       :: POCa!aLPO ; LSCb!sipoa -> break
       :: LSCb!siosa -> goto STOOS
       fi
    od;
```

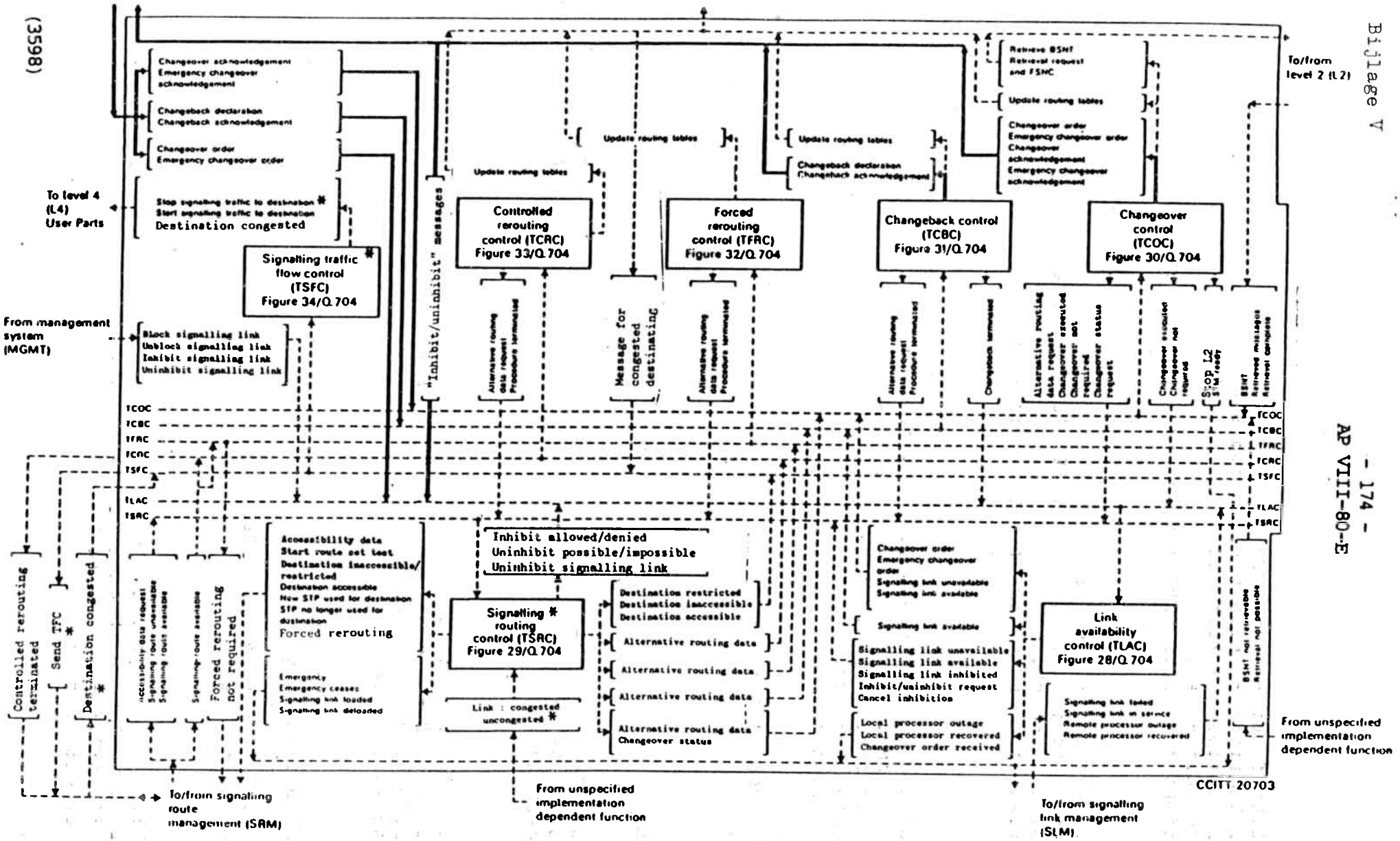
```
proc IACa
STIDLE:
     do
     :: ST?starta -> IACb!sioa -> break
     :: default -> skip
     od;
/*STINAL*/
     do
     :: IACb?siob -> IACb!sina -> break
     :: IACb?sinb -> IACb!sina -> break
     :: timeout ; IACb!sioa
     od;
STAL:
     do
     :: IACb?sinb -> break
     :: IACb?siob -> skip
     :: timeout;
          if
          :: IACb!sina
          :: skip -> goto STIDLE
          fi
     od;
/*STPR*/
     do
     :: IACb?siob -> goto STAL
     :: IACb?sinb -> skip
     :: timeout -> goto STIDLE
     od
end IACa;

proc IACb
STIDLE:
     do
     :: ST?startb -> IACa!siob -> break
     :: default -> skip
     od;
/*STINAL*/
     do
     :: IACa?sioa -> IACa!sinb -> break
     :: IACa?sina -> IACa!sinb -> break
     :: timeout ; IACa!siob
     od;
STAL:
     do
     :: IACa?sina -> break
     :: IACa?sioa -> skip
     :: timeout;
          if
          :: IACa!sinb
          :: skip -> goto STIDLE
          fi
     od;
/*STPR*/
     do
     :: IACa?sioa -> goto STAL
     :: IACa?sina -> skip
     :: timeout -> goto STIDLE
     od
end IACb.
```

=>

```
STANR:
    do
    :: LSCb?fisub -> goto STPO
    :: LSCb?sipob -> POCa!aRPO -> goto STPO
    :: default -> skip
    :: timeout ;
      if
      :: POCa!aLPR ; LSCb!fisua -> goto STAR
      :: LSCb!siosa ; POCa!aSTOP -> goto STOOS
      fi
    od;
STIS:
    do
    :: LSCb?sipob -> LSCb!fisua ; POCa!aRPO -> break
    :: default -> skip
    :: timeout;
      if
      :: POCa!aLPO ; LSCb!sipoa -> break
      :: LSCb!siosa -> goto STOOS
      fi
    od;

STPO:
    do
    :: POCa?aNPO -> LSCb!fisua -> goto STIS
    :: LSCb?fisub -> POCa!aRPR
    :: default -> skip
    :: timeout ;
      if
      :: POCa!aLPO ; LSCb!sipoa
      :: POCa!aLPR ; LSCb!fisua
      :: LSCb!siosa ; POCa!aSTOP -> goto STOOS
      fi
    od
end LSCa;

proc LSCb
STOOS:
    do
    :: init?bSTART -> break
    :: default -> skip
    od;
/*STIA*/
    do
    :: timeout;
      if
      :: LSCa!fisub -> break
      :: POCb!bLPO ; LSCa!sipob -> goto STANR
      fi
    :: default -> skip
     od;
STAR:
    do
    :: LSCa?fisua -> LSCa!fisub -> goto STIS
    :: LSCa?sipoa -> POCb!bRPO -> goto STPO
    :: LSCa?siosa -> goto STOOS
    :: default -> skip
    :: timeout -> POCb!bLPO ; LSCa!sipob -> break
    od;
```

```
STANR:
    do
    :: LSCa?fisua -> goto STPO
    :: LSCa?sipoa -> POCb!bRPO ; goto STPO
    :: LSCa?siosa -> POCb!bSTOP -> goto STOOS
    :: default -> skip
    :: timeout -> POCb!bLPR ; LSCa!fisub -> goto STAR
    od;
STIS:
    do
    :: LSCa?siosa -> goto STOOS
    :: LSCa?sipoa -> LSCa!fisub ; POCb!bRPO -> break
    :: default -> skip
    :: timeout -> POCb!bLPO ; LSCa!sipob -> break
    od;

STPO:
    do
    :: LSCa?siosa -> POCb!bSTOP -> goto STOOS
    :: POCb?bNPO -> LSCa!fisub -> goto STIS
    :: LSCa?fisua -> POCb!bRPR
    :: default -> skip
    :: timeout ->
      if
      :: POCb!bLPO ; LSCa!sipob
      :: POCb!bLPR ; LSCa!fisub
      fi
    od
end LSCb;

proc POCb
STIDLE:
    do
    :: LSCb?bLPO -> break
    :: LSCb?bRPO -> goto STRPO
    :: default -> skip
    od;
STLPO:
    do
    :: LSCb?bRPO -> break
    :: LSCb?bLPR -> LSCb!bNPO -> goto STIDLE
    :: LSCb?bSTOP -> goto STIDLE
    :: default -> skip
    od;
STBPO:
    do
    :: LSCb?bRPR -> goto STLPO
    :: LSCb?bLPR -> break
    :: LSCb?bSTOP -> goto STIDLE
    :: default -> skip
    od;
STRPO:
    do
    :: LSCb?bLPO -> goto STBPO
    :: LSCb?bRPR -> LSCb!bNPO -> goto STIDLE
    :: LSCb?bSTOP -> goto STIDLE
    :: default -> skip
    od
end POCb.
```

(3598)



Note – Abbreviated message names have been used in this diagram (i.e., origin → destination codes are omitted).

FIGURE 27/Q.704 (Sheet 1 of 2)

Level 3 Signalling traffic management (STM); functional block interactions

* Functions modified by Sheet 2 in case of multiple congestion states.

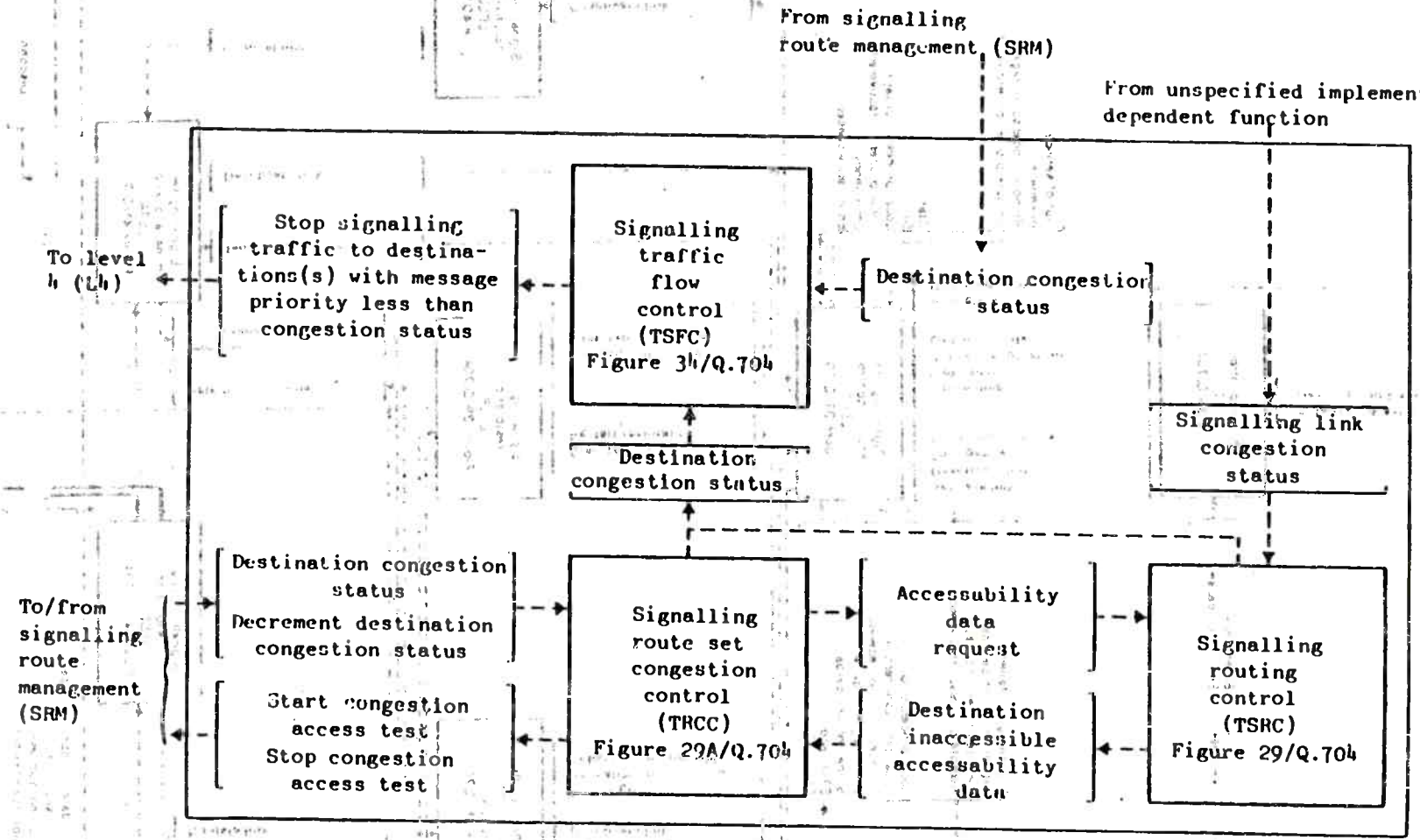
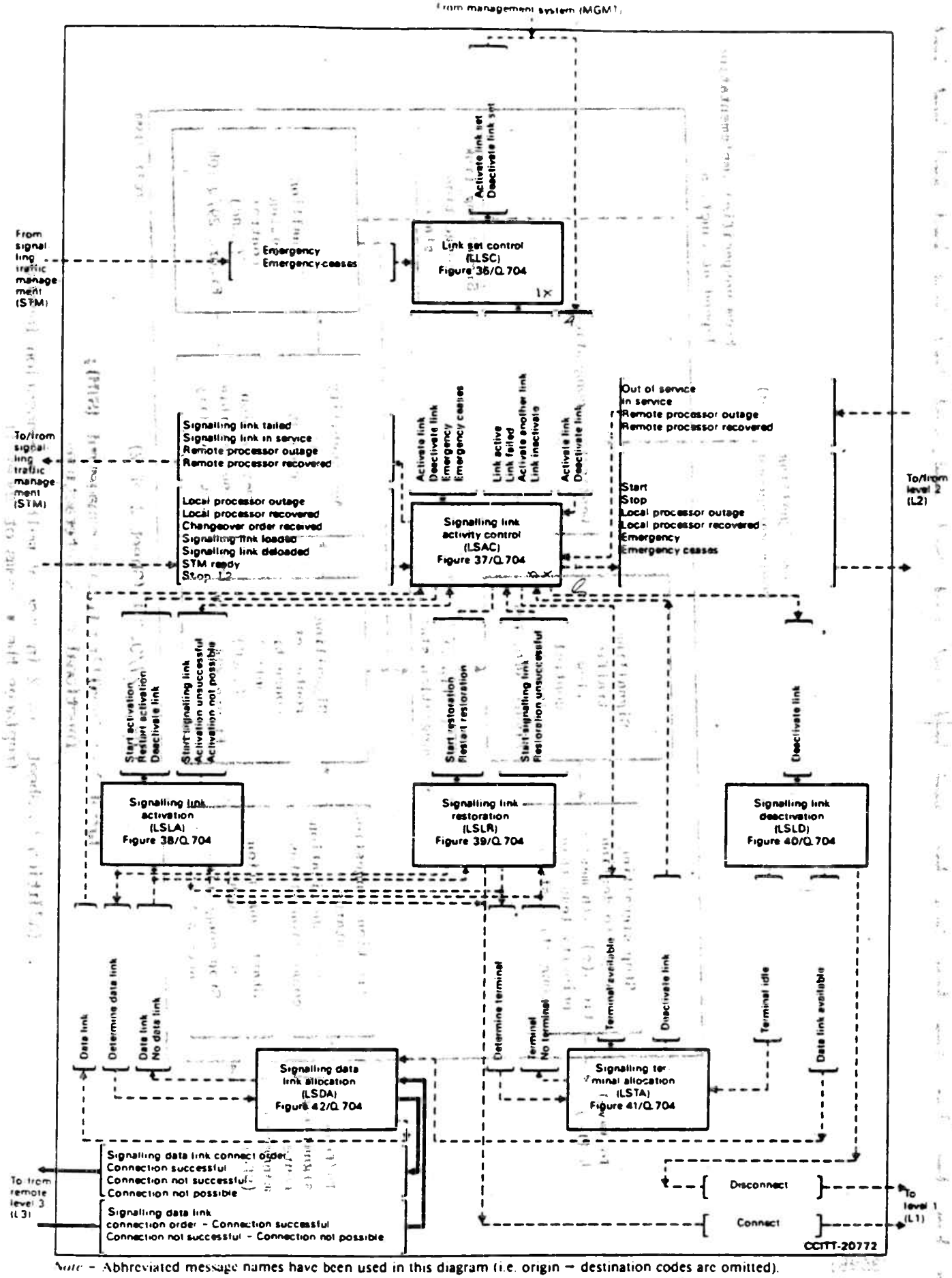


FIGURE 27/Q.704 (Sheet 2 of 2)

Level 3 - Signalling traffic management (SIM): functional block interactions

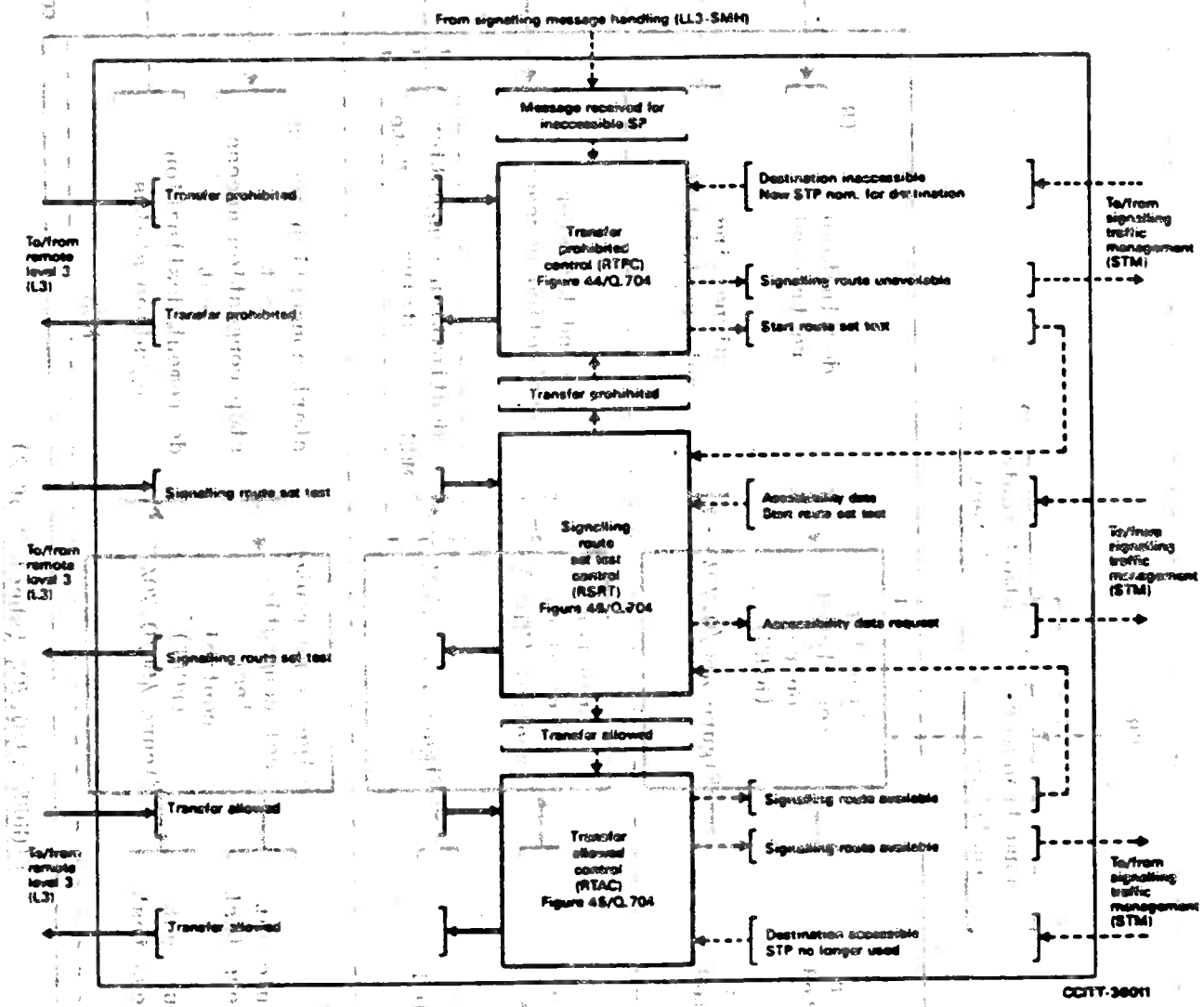
(addition to Sheet 1 of 2 in case of multiple congestion levels)
(replaces the * items of Sheet 1)



Note – Abbreviated message names have been used in this diagram (i.e. origin – destination codes are omitted).

FIGURE 35/Q.704

(3598) **Level 3 - Signalling link management (SLM); functional block interactions**



Note - Abbreviated message names have been used in this diagram (i.e. origin → destination codes have been omitted).

FIGURE 43/Q.704 (Sheet 1 of 2)

Level 3 - Signalling route management (SRM); functional block interactions

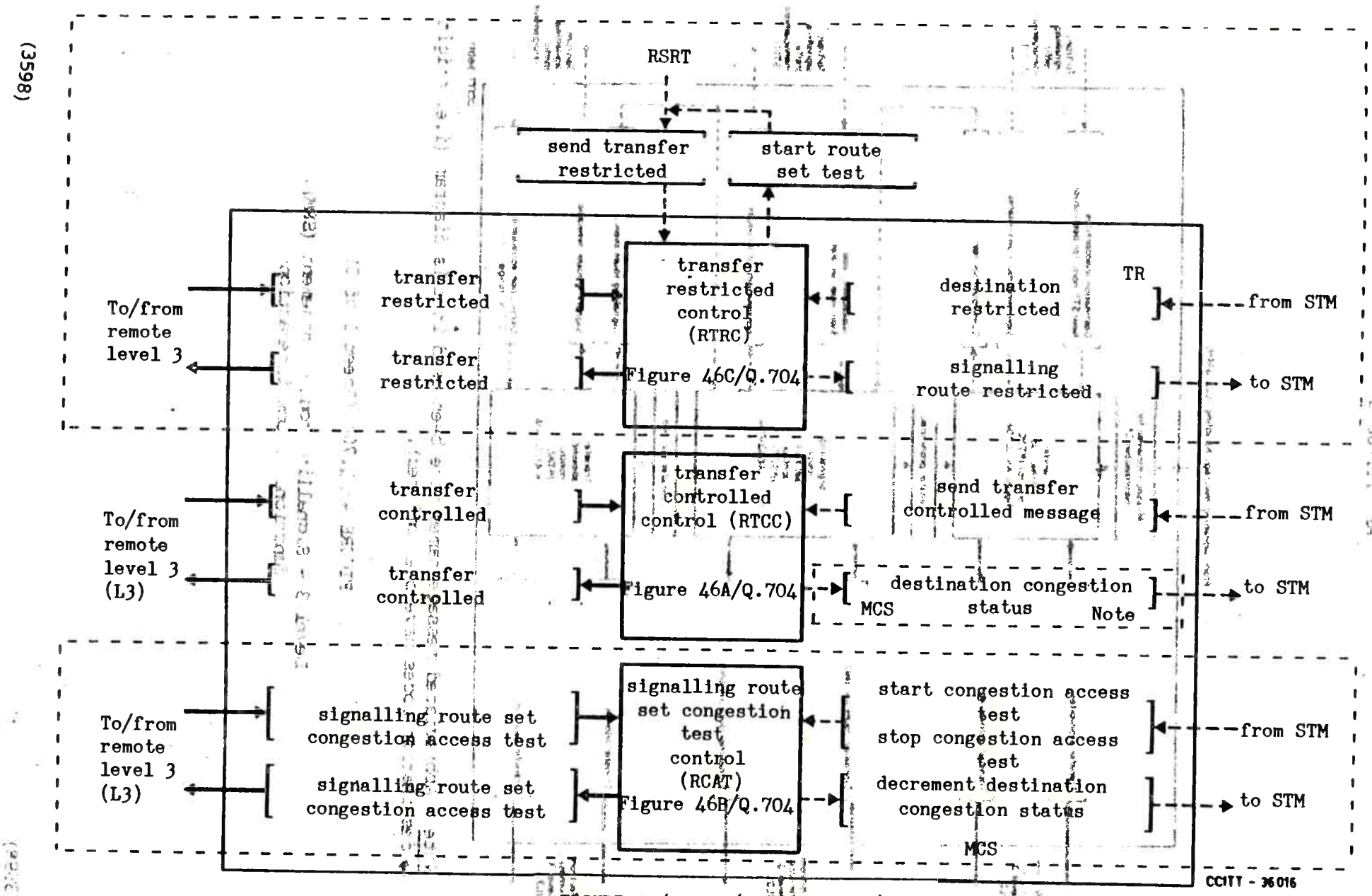


FIGURE 43/Q.704 (sheet 2 of 2)

Level 3 - signalling route management (SRM); functional block interactions

Note - Replaces † in case of multiple congestion states.

V. The Cross Operator

The cross operator can obviously be defined to take any positive number of operands. We can therefore also write a cross product as a function that takes individual process behaviors as arguments

$$\chi(P1, P2, \cdots, Pn)$$

where $P1, \cdots, Pn$ are protocol expressions. In the simplest case the cross can be applied to the single process that appends messages to and deletes them from its own mailbox. For instance

$$\chi\left[\frac{1}{a}\cdot\frac{1}{b}\cdot a\cdot\frac{1}{c}\cdot b\cdot(d+e)\right] = \frac{1}{a}\cdot\frac{1}{b}\cdot a\cdot\frac{1}{c}\cdot b\cdot\varnothing.$$

Neither the d message nor the e message can be received because the first message in the mailbox at that moment is a c. The cross operator thus implements the generalized reduction rule discussed above. The generalization is fairly simple when we use the formalized notion of soundness, defined in the next section.

If more than one process is considered the cross evaluator must distinguish between the mailboxes to verify that messages are received in the order in which they were sent. (A FIFO order is to be maintained for all messages addressed to the same mailbox. Two messages addressed to two distinct mailboxes, however, need not be received in the order in which they were sent.)

The cross operator then has, as parameters, the sets of messages that can be appended to each mailbox in the system: one per process. We call these sets the "mailbox sorts."

To simplify the discussion we will assume that every message used below uniquely identifies both its sender and its receiver. That is, if we say that process A sends message m to B, it is implied that no other process in the system considered will use the same message name m. This assumption quarantees that mailbox sorts are always disjoint. In practice, this "disambiguation" of message names is easily accomplished by extending every message name with the names of the sending and receiving processes.

All events in the process behaviors concerning messages that are not in any mailbox sort specified are treated as nonevents. These messages are not subject to the soundness rules that apply to the others. This convention is useful when we study message exchanges between processes that are part of a larger system. We can then concentrate on messages exchanged between a subset of the processes in the system and abstract from the interactions with the environment. Messages that are outside the parameter sets of the cross product (outside the mailbox sorts) are called "free" messages.

The following example will illustrate the use of mailbox sorts and free messages. Consider a system of three processes named P, Q, and R. The three mailbox sorts are specified as

$$S(P) = (a, c),\ S(Q) = (b), \text{ and } S(R) = (d).$$

Let the behavior of the processing P, Q, and R be given by

$$\frac{a}{b}\cdot c,\ \frac{1}{d\cdot a}\cdot b, \text{ and } \frac{d}{c}, \text{ respectively.}$$

Instead of considering all possible interactions in one single cross product, we can single out a subsystem of two processes and abstract from the third process. Let us, for instance, take the cross product of P and Q, with c and d as free messages. The mailbox sorts $S(P)$ and $S(Q)$ for the subsystem with a set of free messages F become

$$S(P) = (a),\ S(Q) = (b),\ F = (c, d).$$

For $(P\chi Q)$ we find

$$\left[\frac{a}{b}\cdot c\right]\chi\left[\frac{1}{d\cdot a}\cdot b\right] = \frac{1}{d\cdot a}\cdot\frac{a}{b}\cdot b\cdot c = \frac{1}{d}\cdot c.$$

The validity of the reduction implied by the second equal sign can be argued with the aid of the "soundness" concept to be discussed in the next section. In this case we can cross out the a's and the b's only because of the fact that symbol d is neither in $S(P)$ nor in $S(Q)$, and therefore the simple reduction rule (5) from Section IV applies.

It is now easy to calculate $(P\chi Q\chi R)$ as $((P\chi Q)\ \chi\ R)$

$$\left[\frac{1}{d}\cdot c\right]\chi\left[\frac{d}{c}\right] = \frac{1}{d}\cdot d\cdot\frac{1}{c}\cdot c = 1.$$

The fact that the cross product reduces to 1 proves that every message sent is eventually received. The expansion shows that the system P, Q, R is completely specified (there are no "unexpected" receptions) and cannot deadlock (there are no dead ends).

Reversing the order of the symbols in the first term of the product above gives us the trivial deadlock

$$\left[\frac{c}{d}\right]\chi\left[\frac{d}{c}\right] = \left[c\cdot\frac{1}{d}\right]\chi\left[d\cdot\frac{1}{c}\right] = \varnothing.$$

We will refrain from specifying the mailbox sorts for every cross product that occurs below. In most cases they can trivially be derived from the context. (The mailbox sort of a process is implicitly defined to be the set of all messages that occur in the numerators of the corresponding protocol expression.)

Here are some useful identities for the cross operator, again adopted as axioms:

$$\chi(A, 1) = \chi(A) \tag{I.06}$$

$$\chi(A, B) = \chi(B, A) \tag{I.07}$$

$$\chi(A, B, C) = \chi(\chi(A, B), C) \tag{I.08}$$

$$\chi(A, (B + C)) = \chi(A, B) + \chi(A, C). \tag{I.09}$$